Using NetBeans^{TM} IDE 4.1

Your Guide to Getting Work Done in NetBeans IDE

Welcome to the Using NetBeans^{TM} IDE 4.1 guide. This guide is designed to give you a more detailed introduction to the IDE than is available in the Quick Start guides by exploring the main aspects of the IDE. This guide is geared mostly to newcomers to NetBeans IDE 4.1, whether you are new to using IDEs or an experienced IDE user that is switching over from a different development environment. However, readers of this guide are assumed to have at least a basic understanding of the Java programming language and related technologies.

> **Note:** This guide does not cover the IDE's many new J2EE features. For more information about using NetBeans IDE 4.1 for developing J2EE applications, see the J2EE Tutorial for NetBeans IDE.

Using NetBeans IDE 4.1 covers the following topics:

Setting Up Projects

This section covers the basics of setting up your IDE to start developing your own projects. The process of managing project contents and properties is centered around the Projects window. The most common tasks in setting up a project are creating a project, setting the project's target JDK, and configuring the project's properties such that resource libraries are available to it.

This section covers:

Basic IDE Concepts

Before you start setting up your project, let's take a minute to get acquainted with some of the basic concepts involved with using the IDE.

Projects

In the IDE, you always work inside of a project. An IDE project is a group of Java source files and associated information about what belongs on the classpath, how to build and run the project, and so forth. You can create standard projects that use an IDE-generated Ant script to build the project, or create free-form projects that are based on your existing Ant scripts. The IDE stores project information in a project folder which includes an Ant build script and properties file that control the build and run settings, and a `project.xml` file that maps Ant targets to IDE commands.

Note: Though the IDE locates source directories within the project folder by default, your source directories do not necessarily need to be located in the project folder.

The following table summarizes the major differences between standard projects and free-form projects:

Standard Projects	**Free-Form Projects**
The IDE uses a NetBeans-generated Ant build script to build, run, clean, test, and debug your application.	The IDE uses targets in an existing Ant script to build, run, clean, test, and debug your application. If the Ant script does not contain targets for some of these functions, the functions are unavailable. You can write targets to implement these functions, either in your Ant script or in a secondary Ant script.

Some standard IDE projects (Java Application, Java Class Library, Web Application, Enterprise Application, and EJB Module) contain only one source folder and, for J2SE projects, a single test folder. Standard IDE projects with existing sources (Java Application, Web Application, Enterprise Application, and EJB Module) can have any number of source folders. Source folders can be added and removed after project creation. You can also create dependencies with other NetBeans projects. You can also use multiple source folders in standard projects by creating a separate project for each source folder and create dependencies between the projects.	Each project can have any number of source folders. Source folders can be added and removed after project creation. You can also create dependencies with other NetBeans projects.
The project classpath is controlled by the settings you set in the Libraries pane of the Project Properties dialog box. Any changes are immediately registered in the IDE-generated Ant script.	The project classpath is controlled by your Ant script. The classpath settings in the Classpath page of the Project Properties dialog box only tell the IDE which classes to make available for code completion and refactoring. When you change the classpath settings in the Ant script, you have to update the settings in the project's properties.
The build process is customized by setting basic options in the Project Properties dialog box or by overwriting targets in the NetBeans-generated Ant script.	All compilation and runtime options are set in the Ant build script.
The IDE builds one JAR file (for J2SE projects) or WAR file (for web projects) for the entire source folder.	The IDE builds as many output files as are specified in the project's Ant script.

The IDE contains the following standard project templates:

- **Java Application.** Template for creating a skeleton J2SE project with a main class.
- **Java Class Library.** Template for creating a skeleton Java class library without a main class.
- **Java Project with Existing Sources.** Template for creating a J2SE project based on your own Java sources.
- **Web Application.** Template for creating a skeleton web application.
- **Web Project with Existing Sources.** Template for creating a web project based on your own web and Java sources.
- **Enterprise Application.** Template for creating a skeleton enterprise application.
- **Enterprise Application with Existing Sources.** Template for importing an enterprise application into a standard IDE project.

- **EJB Module.** Template for creating an enterprise JavaBean module.
- **EJB Module with Existing Sources.** Template for importing an enterprise JavaBean module into a standard IDE project.

The IDE contains the following free-form project templates:

- **Java Project with Existing Ant Script.** Template for creating a J2SE project based on your own Java sources, built using your own Ant build script.
- **Web Project with Existing Ant Script.** Template for creating a web project based on your own web and Java sources, built using your own Ant build script.
- **EJB Module with Existing Ant Script.** Template for importing an EJB module into an IDE project that uses your own Ant build script.

Ant

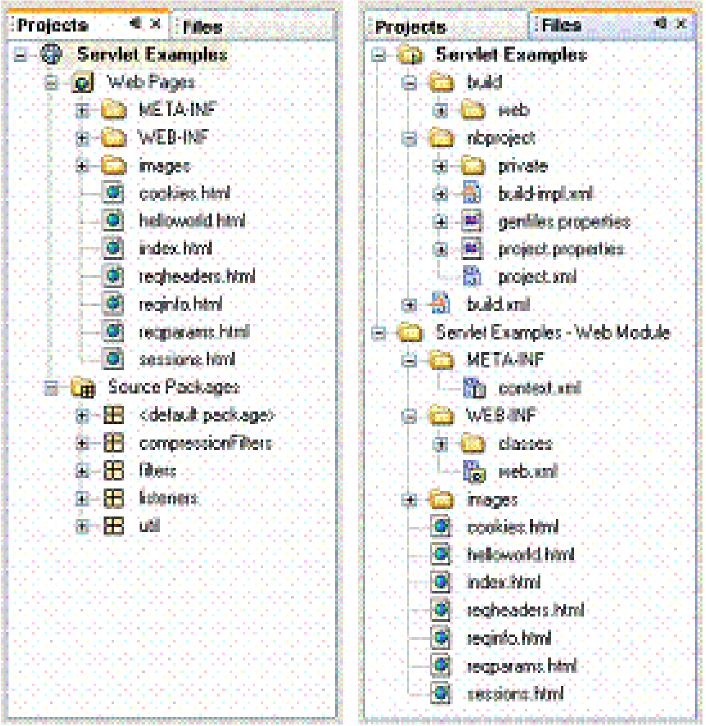
Apache Ant is a Java-based build tool used to standardize and automate build and run environments for development. The IDE's project system is built directly on top of Ant. All of the project commands, like Build Main Project or Debug Main Project, call targets in the project's Ant script. You can therefore build and run your project outside the IDE exactly as it is built and run inside the IDE.

It is not necessary to know Ant to work with the IDE. You can set all the basic compilation and runtime options in the project's Project Properties dialog box and the IDE automatically updates your project's Ant script. If you are familiar with Ant, you can customize a standard project's Ant script or write your own Ant script for a project.

If you are looking for resources on learning Ant, see http://ant.apache.org/resources.html.

Creating a Project

To create a new project, choose File > New Project (Ctrl-Shift-N). When the New Project wizard appears, simply select the right template for your project and complete the remaining wizard steps.



For instructions on using the New Project wizard, see the following documents:

- Quick Start Guide
- Quick Start Guide for Web Applications

Because NetBeans' post-3.6 projects structure is so different from the earlier versions, *automatic* importing of NetBeans IDE 3.6 projects is not implemented. To import the source code you were working on in version 3.6, create a new project for each source package root (or if you have your own Ant script, one free-form project for the entire application). For more information on importing source code into the IDE, see the following documents:

- Importing Existing Java Source Code into NetBeans IDE
- Importing Existing Web Applications into NetBeans IDE

When you finish creating a project, it opens in the IDE with its logical structure displayed in the Projects window and its file structure displayed in the Files window:

- The Projects window is the main entry point to your project sources. It shows a logical view of important project contents such as Java packages and Web pages. You can right-click any project node to access a contextual menu of commands for building, running, and debugging the project, as well as opening the Project Properties dialog box. The Projects window can be opened by choosing Window > Projects (Ctrl-1).

- The Files window shows a directory-based view of your projects, including files and folders that are not displayed in the Projects Window. From the Files window, you can open and edit your project configuration files, such as the project's build script and properties file. You can also view build output like compiled classes, JAR files, WAR files, and generated Javadoc documentation. The Files window can be opened by choosing Window > Files (Ctrl-2).

In addition, you can use the Favorites window to you access any location on your computer. This is handy for accessing files and directories that are outside of your project directories. The Favorites window does not know anything about project classpath and membership, so none of the project-related commands like Compile File are available. You can open the Favorites window by choosing Window > Favorites (Ctrl-3).

Setting the Main Project

When you develop a large application consisting of numerous source folders, it is common to split up your code into separate projects. Typically, one of these projects serves as the entry point for your application and, if it is a J2SE application, contains the application's main class. To tell the IDE which of your projects is the main entry point for your application, you set one project to be the main project. The IDE provides commands that act on the main project. For example, running the Build Main Project command builds both the main project and all of its required projects, thereby ensuring that all of your compiled classes are up-to-date. To set a project as the main project, right-click the project node in the Projects window, and choose Set as Main Project. Only one project can be the main project at any time.

Setting the Target JDK in a Project

By default, the IDE uses the version of the J2SE platform (JDK) with which the IDE runs as the default Java platform for compilation, execution, and debugging. You can view your IDE's JDK version by choosing Help > About and clicking the Detail tab. The JDK version is listed in the Java field.

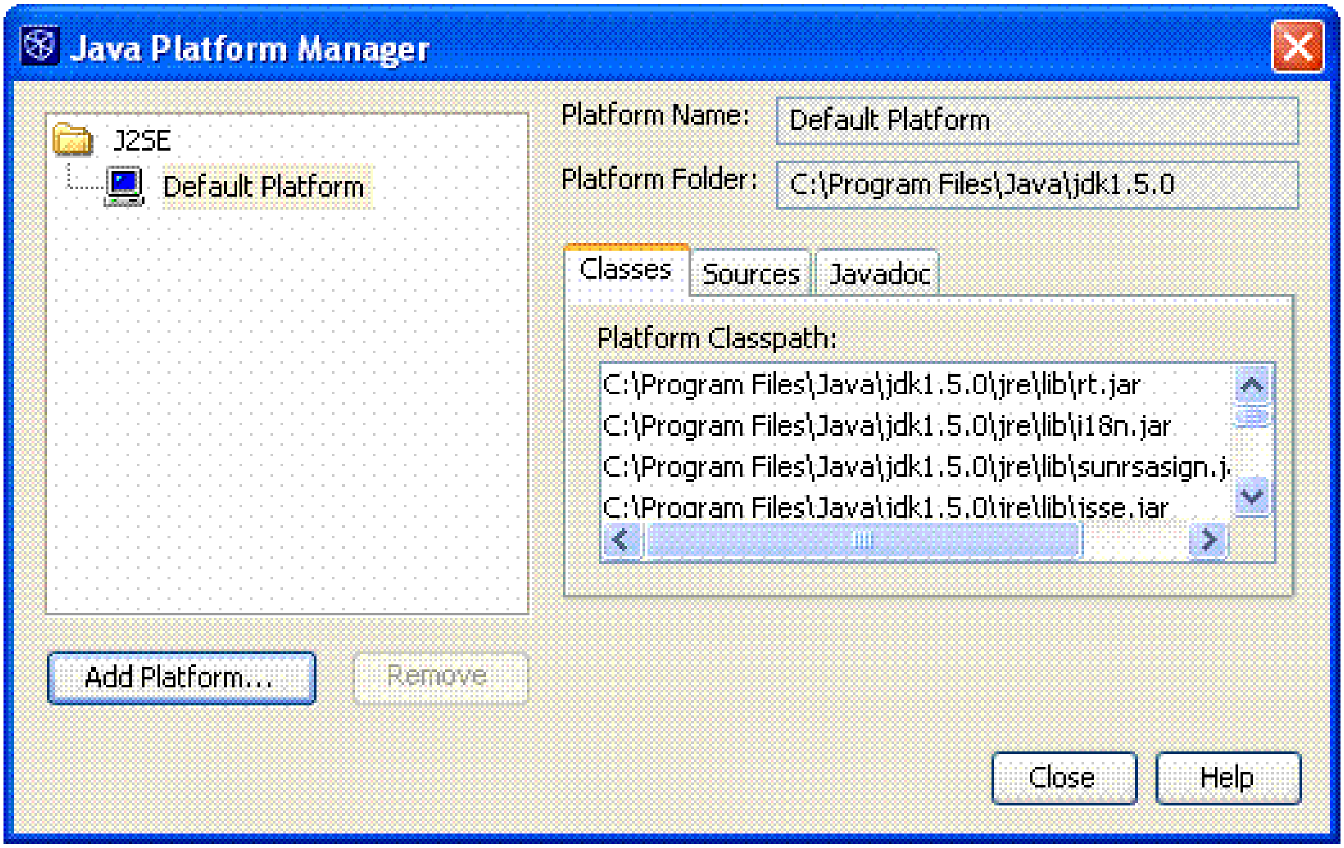
You can run the IDE with a different JDK version by starting the IDE with the `--jdkhome jdk-home-dir` switch from the command line or in your `IDE-HOME/etc/netbeans.conf` file. For more information, see Configuring IDE Startup Switches.

In the IDE, you can register multiple Java platforms and attach Javadoc and source code to each platform. Switching the target JDK for a standard project does the following:

- Offers the new target JDK's classes for code completion.
- If available, displays the target JDK's source code and Javadoc documentation.
- Uses the target JDK's executables (`javac` and `java`) to compile and execute your application.
- Compiles your source code against the target JDK's libraries.

In standard projects, you can switch the target JDK in the Libraries panel of the Project Properties dialog box. In free-form projects, you have to set the target JDK in the Ant script itself, then specify the source level in the Sources page of the Project Properties dialog box.

To register a new Java platform, choose Tools > Java Platform Manager from the main menu. Specify the directory that contains the Java platform as well as the sources and Javadoc needed for debugging.



Platform Name: Default Platform

Platform Folder: C:\Program Files\Java\jdk1.5.0

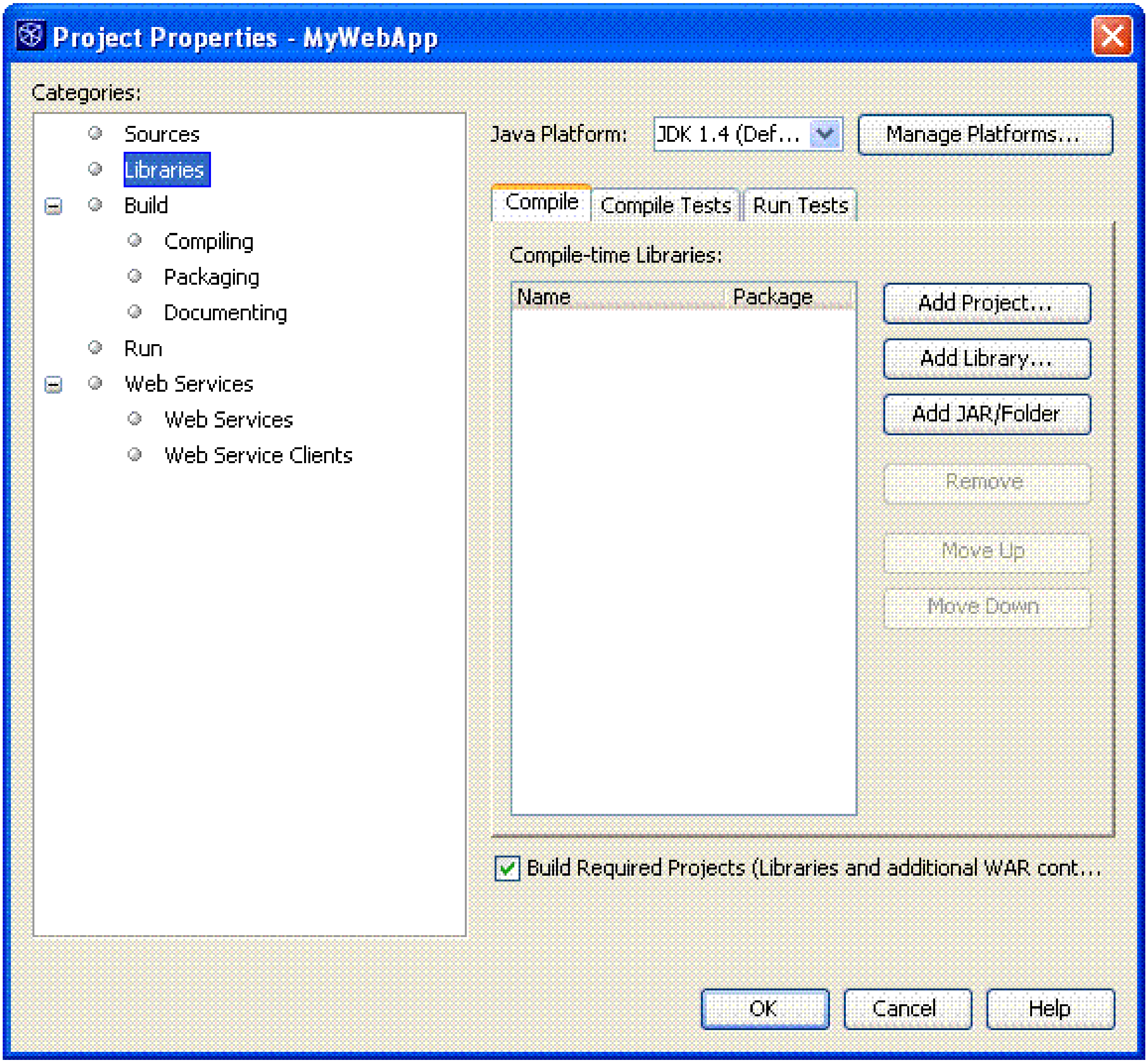
Platform Classpath:

```
C:\Program Files\Java\jdk1.5.0\jre\lib\rt.jar
C:\Program Files\Java\jdk1.5.0\jre\lib\i18n.jar
C:\Program Files\Java\jdk1.5.0\jre\lib\sunrsasign.j
C:\Program Files\Java\jdk1.5.0\jre\lib\jsse.jar
```

Managing a Project's Classpath

Adding a group of class files to a project's classpath tells the IDE which classes the project should have access to during compilation and execution. The IDE also uses classpath settings to enable code completion, automatic highlighting of compilation errors, and refactoring. For both standard projects and free-form projects, you first declare the classpath in the New Project wizard when creating the project. You can edit all classpath declarations for an existing project in the Project Properties dialog box, as explained below.

- In standard projects, the IDE maintains separate classpaths for compiling and running your project, as well as compiling and running JUnit tests (for J2SE applications). The IDE automatically adds everything on your project's compilation classpath to the project's runtime classpath. You can add JAR files, libraries, and dependent projects to the project's compilation classpath in the Compile tab of the Project Properties dialog box.



Project Properties - MyWebApp

Categories:

- Sources
- Libraries
- Build
  - Compiling
  - Packaging
  - Documenting
- Run
- Web Services
  - Web Services
  - Web Service Clients

Java Platform: JDK 1.4 (Def…

Compile-time Libraries:

Name	Package

☑ Build Required Projects (Libraries and additional WAR cont…

- In free-form projects, your Ant script handles the classpath for all of your source folders. The classpath settings for free-form projects only tell the IDE what classes to make available for code completion and refactoring. You can declare the classpath for free-form projects using the Java Sources Classpath panel in the Project Properties dialog box.

  Note: In free-form projects, declaring the classpath in the Project Properties dialog box does not change the actual compilation or runtime classpath of the source folders. The project's classpath declaration must exactly match the classpath used by your Ant script.

If you have attached Javadoc and source files to a JAR file in the Library Manager, the IDE automatically adds the Javadoc and source files to the project when you register the JAR file on a project's classpath. You can step into classes and look up Javadoc pages for the classes without configuring anything else.

Managing Dependencies Between Projects

When you create separate standard projects for each of your source roots, you have to set up the classpath dependencies between the projects. Typically you set up one main project, containing the project main class (in J2SE projects), and several required projects. A required project is a project that has been added to another project's classpath. When you clean and build a project, the IDE also cleans and builds its required projects. The required project's Javadoc and sources are also made available to the receiving project. In the Project Properties dialog box's Compile tab, click the Add Project button and select the project folder whose JAR files you want to add to the classpath (the file chooser displays IDE project folders with a 📁 icon). Then make sure that the Build Projects on Classpath checkbox is selected and click Add Project JAR Files.

If you want to add a free-form project to the classpath of a standard project, you have to add the free-form project's JAR file to the standard project's classpath. To do this, you must first declare all of the free-form project's output files in the Output panel of the free-form project's Project Properties dialog box.

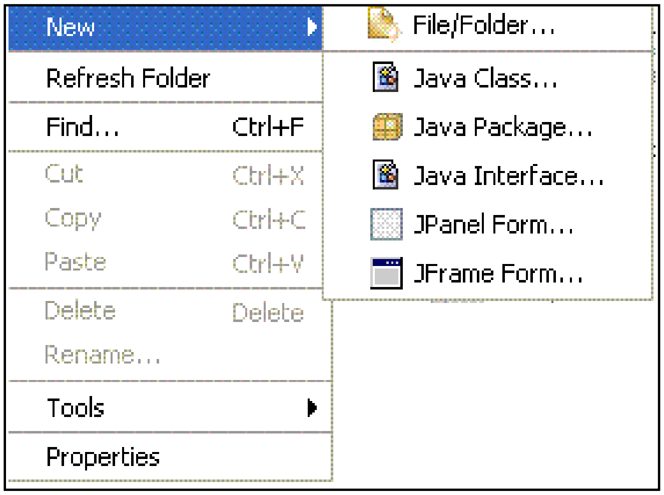
Creating and Editing Java Source Code

Creating and editing Java source code is the most important function that the IDE serves, since that's what developers generally spend most of their day doing. NetBeans IDE provides a wide range of tools that can compliment any developer's personal style, regardless of whether you prefer to code everything by hand or want the IDE to generate large chunks of code for you.

This section covers the following topics:

Creating Java Files

NetBeans IDE contains templates and wizards that you can use to create all kinds of source files, from Java source files to XML documents to resource bundles. You can



Perhaps the easiest way to create a file (once you have already created a project) is to right-click the project node of the project for which you want to create the file in the Projects window. You can then choose desired file type from the New submenu in the node's contextual menu. The New submenu contains shortcuts to commonly-used templates and a File/Folder command that you can use to access all provided NetBeans templates.

To demonstrate some of the IDE's source creation and editing features, let's create a class called `ColorPreview`. Choose File > New Project. In the Categories pane, select the General node. Under Projects, select Java Class Library and click Next. On the Name and Location page of the wizard, enter `ColorPicker` for the Project Name. Next, change the Project Location to any directory on your computer and click Finish. The IDE creates the ColorPicker folder on your system containing all of your sources and project metadata, such as the project Ant script. The project is displayed in both the Projects window and the Files window.

Now right-click the project and choose New > Java Class. Name the file `ColorPreview` and the package `Color`. Click Finish. The package is created in the Source Packages folder, the file is created in the package and opens in the Source Editor.

GUI Templates and Java Templates

If you want to visually edit a Java GUI form using the IDE's Form Editor, you have to create the form's source file using the IDE's Java GUI Forms templates. This template group contains templates for AWT and SWING forms. For example, you cannot create a normal Java class file and then change it to extend `JPanel` and edit it in the Form Editor.

For more information about creating Java GUIs in the IDE, see GUI Building in NetBeans IDE 4.1.

Editing Java Files in the Source Editor

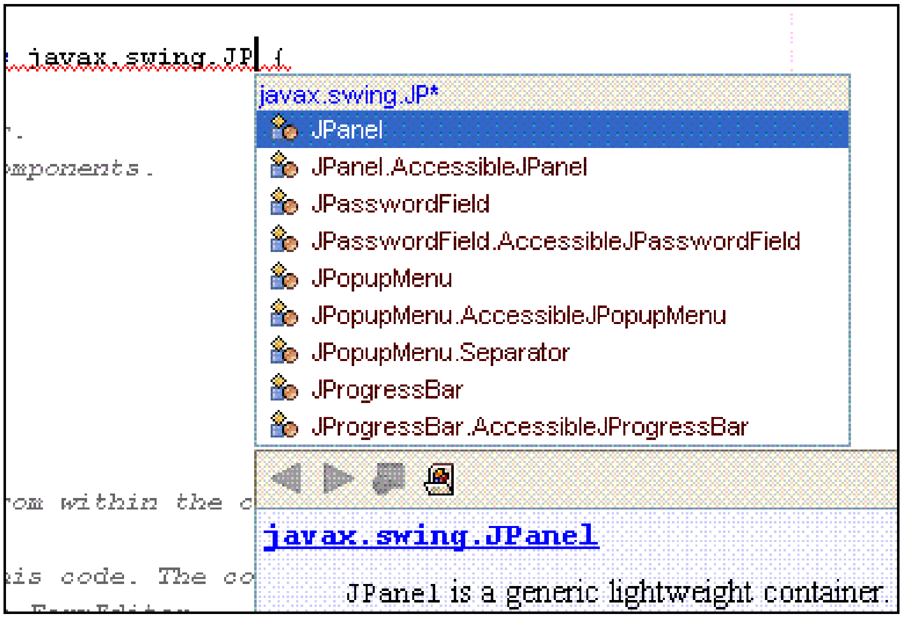
The Source Editor is your main tool for editing source code. It provides a wide range of features that make writing code simpler and quicker, like code completion, highlighting of compilation errors, syntax highlighting of code elements, as well as other advanced formatting and search features.

Although the Source Editor can be considered a single IDE component, it is really a collection of editors. Each type of source file has its own editor that provides different functionality. In this section we'll be dealing with the Java editor, but many of the same concepts apply to other editors. To open a Java source file in the Source Editor, double-click the file's node in the Projects window or Files window.

Note: *Double-clicking a Java form node ( ) in the Projects window or Files window opens the file in the Form Editor. In the Form Editor's toolbar there are two buttons, one for the design view and one for the source view which you can use to conveniently switch between views. If, however, you want to edit the source code for a Java form without opening the Form Editor, right-click its node and choose Edit from the contextual menu.*

Using Abbreviations, Word Matching, and Code Completion

The Source Editor provides many features that spare you from having to enter long Java class names and expressions manually. The most commonly used of these features are abbreviations, code completion, and word matching.



javax.swing.JP*
JPanel
JPanel.AccessibleJPanel
JPasswordField
JPasswordField.AccessibleJPasswordField
JPopupMenu
JPopupMenu.AccessibleJPopupMenu
JPopupMenu.Separator
JProgressBar
JProgressBar.AccessibleJProgressBar

```
 javax.swing.JP {
```

javax.swing.JPanel

`JPanel` is a generic lightweight container.

Code completion in the Java Source Editor lets you type a few characters and then choose from a list of possible classes, methods, variables, and so on to automatically complete the expression. The Source Editor also includes a Javadoc preview window that displays the Javadoc documentation for the code completion box's current selection, if any exists. The Javadoc is drawn from the project's compilation classpath.

Abbreviations are short groups of characters that expand into a full word or phrase when you press the space bar. For example, if you enter `psfs` and press the space bar, it expands into `public static final String`. For a full list of the IDE's default abbreviations, choose Keyboard Shortcuts Card in the Help menu or click here.

Note: You can also add, remove, and edit your own custom abbreviations for each of the IDE's editors. This can be done in the Options window by selecting Editing > Editor Settings, choosing the appropriate editor, and opening the property editor for the Abbreviations property.

Word matching is a feature that lets you type a few characters of a word that appears elsewhere in your code and then have the Source Editor generate the rest of the word automatically. To use this feature, type a few characters in the Source Editor and press Ctrl-L to generate the next matching word or Ctrl-K to generate the previous matching word.

As a quick exercise to show these features in action, let's make `ColorPreview` extend `JPanel`. Place the insertion point after `ColorPicker` in the class declaration, then type `ex` and press the space bar to expand the abbreviation into `extends`. Then type the first few letters of `javax`. The code completion box should pop up after a few seconds. If it does not, you can always manually open it by pressing Ctrl-Space. Use the code completion box to enter `javax.swing.JPanel`.

Configuring Code Completion

The IDE uses the classes on your compilation classpath to provide suggestions for code completion and other features. Classes from the target JDK version, other commonly used project-specific APIs like the Servlet, JSP, JSTL and XML APIs, as well as the sources you have manually added to the classpath can be used in code completion. For details, see Managing a Project's Classpath.

In the Options window, you can disable and enable code completion as well as set the length of the pause before the code completion box appears in the Source Editor. To do this for the Java editor, select Editing > Editor Settings > Java Editor and set the Auto Popup Completion Window property and the Delay of Completion Window Auto Popup property accordingly. You can also turn off the Javadoc preview box for code completion by selecting Java Editor and unchecking the Auto Popup Javadoc Window property.

Refactoring

Refactoring is the restructuring of code, using small transformations, in which the result does not change any program behavior. This type of code rewriting has gone on from the beginning of programming, but it did not get the official name of refactoring until recently. Just as you factor an expression to make it easier to understand or modify, you refactor code to make it easier to read, simpler to understand, and faster to update. Just as a refactored expression must produce the same result, the refactored program must be functionally equivalent with the orginal source. In the next section we'll take a look at an example of refactoring to help illustrate this functionality.

The IDE provides the following features to facilitate code refactoring:

- **Find Usages.** Finds where classes, variables, and methods are used throughout the source code in your project.
- **Rename.** Enables you to change the name of a class, variable, or method to something more meaningful. In addition, it updates all source code in your project to reference the element by its new name.
- **Change Method Parameters.** Enables you to add parameters to a method and change the access modifier.
- **Encapsulate Fields.** Generates a getter method and and a setter method for a field and optionally updates all referencing code to access the field using the getter and setter methods.
- **Move Class.** Enables you to move a class into another class or package and enables you to move a static field or a static method from one class to another. In addition, it updates all effected source code in your project to reference the element in its new location.
- **Undo Refactoring.** Enables you to roll back all the changes in all the files that were affected by the refactoring.

Some common motivations for refactoring code include:

- Making the code easier to change or easier to add a new feature
- Reducing complexity to promote understanding

- Removing unnecessary repetition
- Enabling use of the code for alternate or more general needs

Adding Fields, Bean Properties, and Event Listeners

Even if you prefer to write your code the old-fashioned way, the NetBeans IDE Java editor has some helpful code generation features that you may find handy, especially when dealing with bean properties and event listeners.

Let's start by adding some of the fields for our colors in `ColorPreview`. Go to the first line after the class declaration and type in the following code:

```
    int red;
```

Now let's turn this ordinary field into a bean property by making some getter and setter methods for it. Right-click in the field name and choose Refactor > Encapsulate Fields from the contextual menu. Click Next and then click Do Refactoring. The following code is generated in the file:

```
    public int getRed() {
        return red;
    }

    public void setRed(int red) {
        this.red = red;
    }
```

The methods now show up under the Methods node. The Bean Patterns node now also contains a bean property node for `red`.

Now let's add both the field and the get and set methods at the same time. In the Projects window, right-click the Bean Patterns node for `ColorPreview` and choose Add > Property. In the dialog, enter `green` for the name and `int` for the type, then check Generate Field, Generate Get Method, and Generate Set Method and click OK. The following code is added to the file:

```
    private int green;

    public int getGreen() {
        return this.green;
    }

    public void setGreen(int green) {
        this.green = green;
    }
```

So far, so good. But to fully generate a working bean that can get and set the value of each of the color bean properties and notify the caller of its changes, we have to add event listeners to each of the set methods. There are two ways to do this. First, you could right-click the Bean Patterns node and choose Add > Multicast Event Source to add the `java.beans.propertyChangeListener` methods, then enter the rest of the source by hand. An easier way is to generate all of the necessary code when you create the bean properties.

Ok, let's get rid of all of the methods and fields we have created so far. You can do so by deleting their nodes in the Projects window or Files window or just by deleting the code in the Source Editor. Next, right-click the Bean Patterns node and choose Add > Property. Enter `red` for the name, `int` for the type, and select the Bound checkbox. Notice that selecting this checkbox activates the Generate Property Change Support checkbox, which enables you to generate the property change support code in addition to the field and methods for the property. Select the Generate Property Change Support checkbox and click OK to generate the following code in the Source Editor:

```
    private int red;

    private java.beans.PropertyChangeSupport propertyChangeSupport =  new
java.beans.PropertyChangeSupport(this);

    public void addPropertyChangeListener(java.beans.PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(java.beans.PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public int getRed() {
        return this.red;
    }

    public void setRed(int red) {
        int oldRed = this.red;
        this.red = red;
        propertyChangeSupport.firePropertyChange("red", new Integer(oldRed), new Integer(red));
    }
```

All you have to do now is repeat the process for the `green` and `blue` properties and change the `ColorPreview` constructor to the following:

```
    public ColorPreview() {
        propertyChangeSupport = new java.beans.PropertyChangeSupport(this);
    }
```

And that's it! You've got a nice working bean ready to be used by the `ColorPicker` program.

Working With Import Statements

In the IDE, you can add import statements for Java classes in two different ways:

- For the whole file, by pressing Alt-Shift-F (Fix Imports) when the insertion point is in the file in the Source Editor.
- Individually, by pressing Alt-Shift-I (Fast Import) when the insertion point is in the referenced class name in your code.

The IDE's Fix Imports command adds import statements that are needed by your code and removes unused import statements. It does not, however, remove fully-qualified class names from code and replace them with import statements. The Fast Import command, on the other hand, enables you to choose how you want the import handled in your code.

The IDE's Fast Import command enables you to:

- Generate an import statement for the class.
- Generate an import statement for the package.
- Generate a fully qualified name in the code.

Search and Selection Tools

When you are dealing with a large group of files, the ability to quickly find, navigate to, and select certain strings or files is critical to your productivity. The following list gives you a quick overview of the search and selection tools that are available in the Source Editor:

Keyboard Shortcut	**Description of Command**
Ctrl-Shift-O	Switch to the Search Results window.
Ctrl-Shift-P	Find in Projects.
Ctrl-F	Search for text in the currently selected file. The Source Editor jumps to the first occurrence of the string and highlights all matching strings.
Ctrl-H	Replace text in the currently selected file.
F3	Find the next occurrence of the word you searched for.
Shift-F3	Find the previous occurrence of the word you searched for.
Ctrl-F3	Search for the next occurrence of the word that the insertion point is on.
Alt-Shift-H	Toggle on/off search result highlighting.
Alt-Shift-O	Open the Fast Open dialog box, which lets you quickly open a file. Start typing a class name in the dialog box. As you type, all files that match the typed prefix are shown.
Alt-G	Go to declaration. Similar to the previous shortcut, this opens the file where the variable at the insertion point is declared.
Ctrl-G	Go to line. Enter any line number for the current file and press Enter to jump to that line.
Ctrl-F2	Add a bookmark (🔖) to the line of code that the insertion point is currently on. If the line already contains a bookmark, this command removes the bookmark.
F2	Go to the next bookmark.
Shift-F2	Go to the previous bookmark.
Alt-L	Go to the next location in the jump list for the currently selected file. The jump list is a history of all locations where you made modifications in the Editor.
Alt-K	Go to the previous location in the jump list for the currently selected file.
Alt-Shift-L	Go to the next jump list location in all files (not the currently selected file).
Alt-Shift-K	Go to the previous jump list location in all files (not the currently selected file).

Formatting Java Source Code

The IDE automatically formats your code as you write it. You can also reformat specific lines of code or even entire files. The following table lists some common formatting commands.

Keyboard Shortcut	**Description of Command**
Ctrl-Shift-F	Reformat the entire file or whatever text is selected in the Source Editor.
Ctrl-T	Shift the current line or selection one tab to the right.
Ctrl-D	Shift the current line or selection one tab to the left.
Ctrl-E	Remove the current line.
Ctrl-Shift-T	Comment out the current line or all selected lines with line comments ("//").
Ctrl-Shift-D	Remove comments. This command only works for lines that begin with line comments ("//").

Navigating Between Documents

The Source Editor makes it easy to manage large number of open documents at one time. Each open document is represented by its own tab in the area directly below the IDE's toolbar. The tabs appear in the order in which you opened the documents, however, you can change a tab's position by simply grabbing and dragging it to the desired location along the row of tabs. Use the left and right buttons in the top-right corner to scroll through the row of tabs.

To switch between open files, do any of the following:

- Use the drop down list at the top-right of Source Editor. The drop down list displays all of your open files in alphabetical order.
- Press Alt-Left and Alt-Right to move one editor tab to the left or right.
- Press Ctrl-Tab or Ctrl-` to open the IDE window manager, which contains icons for each open document in the Source Editor as well as all open windows like the Projects window.

Other useful IDE features that assist you in navigating your documents include:

- **Maximize the Source Editor**. Double-click any document tab or press Shift-Escape to hide all other IDE windows. If you have split the Source Editor, only the partition you maximize is displayed.
- **Clone a document.** Right-click the document in the Source Editor and choose Clone Document.
- **Split the Source Editor.** Grabbing any document tab and drag it to the left or bottom margin of the Source Editor. A red box shows you where the new Source Editor partition will reside once you drop the document. Source Editor panes can be split any number of times.
- **Move documents between Source Editor partitions.** Grab the document tab and drag it to the row of tabs in the destination partition.

Configuring the Editor

To configure Source Editor settings, open the Options window and expand Editing > Editor Settings. The Editor Settings node has subnodes for each of the editors used for different file types. In this section, we will focus on configuring the Java editor, but many of the settings are the same for all editors.

Here is a quick overview of some of the more common customizations to the Source Editor:

- **View or change abbreviations.** Open the property editor for the Abbreviations property and make any changes to the list.
- **View or change all keyboard shortcuts for the IDE.** Open the property editor for the Key Bindings property.

- **View or change all recorded macros.** Open the property editor for the Key Bindings property.
- **Turn off code completion.** Set the Auto Popup Completion Window property to False.
- **Set the font size and color for code.** Use the Font Size property to quickly change the font size for all Java code in the Source Editor. Open the property editor for Fonts and Colors to change the font and color of each type of Java code, like method names or strings.
- **Change the indentation used in your code.** You can switch between indentation engines by choosing a new engine from the Indentation Engine property. You can also configure each indentation engine by opening the property editor for the property.
- **Set how many spaces are inserted for each tab in your code.** Set the Tab Size property accordingly.
- **Turn off Javadoc for code completion.** Go to the Expert tab and set the Auto Popup Javadoc Window to False.

Building Applications

The IDE uses an Ant build script to build your applications. In standard projects the IDE generates the build script based on the options you enter in the New Project wizard and the project's Project Properties dialog box. In free-form projects, the IDE relies on your existing Ant script to provide targets for IDE actions, such as building, running, and debugging. For more information on writing an Ant target that compiles the currently selected file and maps it to the IDE's Run File command, see Using Ant Scripts in the IDE.

In this section you will learn about the following:

- Building Projects, Packages, and Files
- Fixing Compilation Errors
- Customizing the Build Process
    - Regular Expressions for Filtering Output Files

Note: To customize the build process for web applications and to redefine WAR files, see Developing Web Applications.

Building Projects, Packages, and Files

Compilation in the IDE is simple. Once you have ensured that your project's compilation classpath is set correctly, you need only select the project, package, or file you want to compile and choose the appropriate Build or Compile command. The IDE then compiles the files.

To compile a project, package, or file in the IDE, select it in the Projects window and do one of the following:

- In the main menu, choose **Build > Build Main Project (F11)** to build the main project. Alternately, you can use the button in the toolbar.
- In the main menu, choose **Build > Clean and Build Main Project (Shift-F11)** to clean and build the main project. Alternately, you can use the button in the toolbar.
- In the Projects window, right-click the project node and choose **Build Project** to build the project.
- In the Projects window, right-click the project and choose **Clean Project** to clean the project.
- In the Projects window, right-click the package and choose **Compile Package (F9)** to compile a package.
- In the Projects window, right-click the file and choose **Compile File (F9)** to compile a file. Alternatively, choose **Build > Compile File (F9)**. Note that if you are using a free-form project, this command is disabled by default. You have to write an Ant target for compiling the currently selected file in the IDE and map it to the Compile File command.

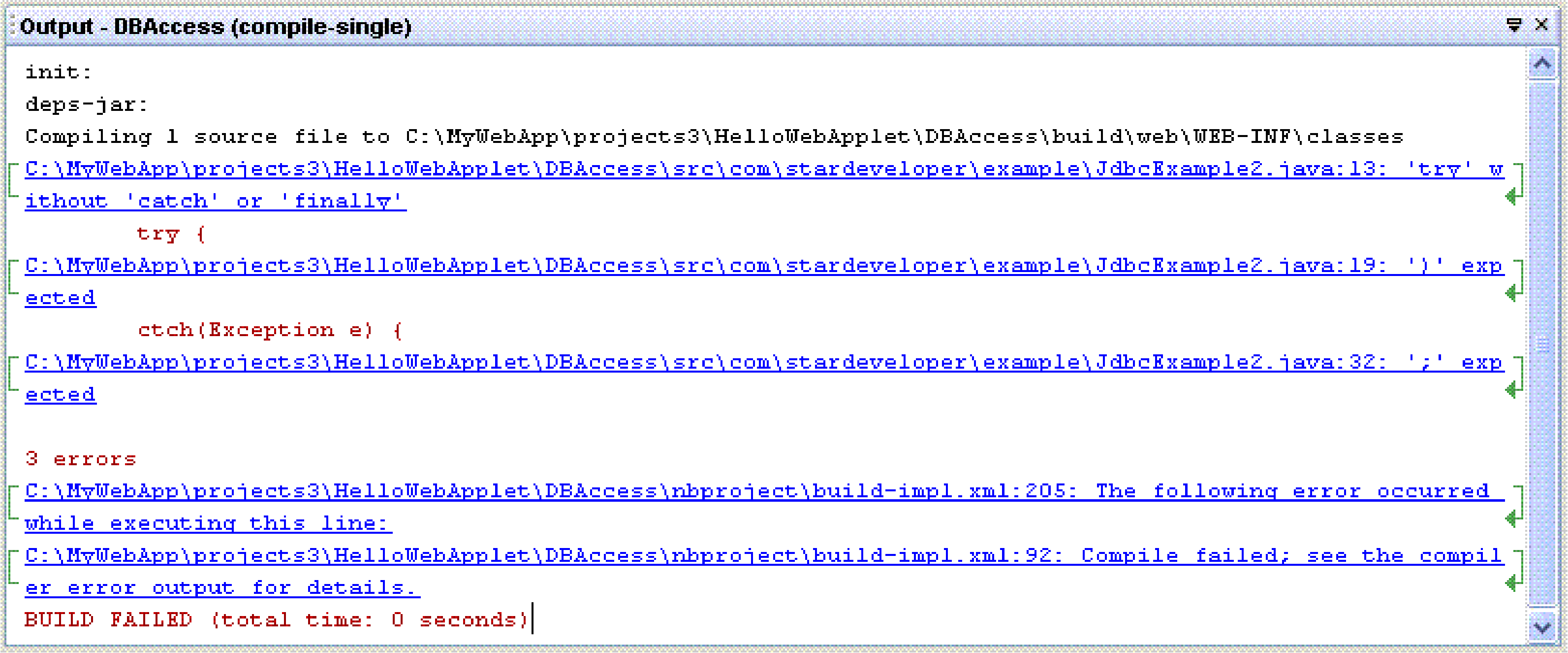
Whenever you invoke compile commands, the IDE displays the output including any compilation errors encountered in the Output window, as described in the following section.

If you expand a standard project's project directory node in the Files window, you will notice that the IDE compiles classes to the `build` folder. In addition, the IDE builds a JAR file for Java projects from your project sources automatically. The JAR file is generated to the `dist` directory of your project folder. In free-form projects, your Ant script controls output file creation.

Fixing Compilation Errors

As mentioned earlier, the IDE displays output messages and any compilation errors in the Output Window. This multi-tabbed window is displayed automatically whenever you generate compilation errors, debug your program, generate Javadoc documentation, and so on. You can also open this window manually by choosing Window > Output (Ctrl-4).

One important function of the Output window is to notify you of errors found while compiling your program. The error message is displayed in blue underlined text and is linked to the line in the source code that caused the error, as illustrated in the image below. The Output window also provides links to errors found when running Ant build scripts. Whenever you click an error link in the Output window, the Source Editor jumps to the line containing the error automatically. You can also use the F12 and Shift-F12 keyboard shortcuts to move to the next and previous error in the file.



Output - DBAccess (compile-single)

```
init:
deps-jar:
Compiling 1 source file to C:\MyWebApp\projects3\HelloWebApplet\DBAccess\build\web\WEB-INF\classes
C:\MyWebApp\projects3\HelloWebApplet\DBAccess\src\com\stardeveloper\example\JdbcExample2.java:13: 'try' w
ithout 'catch' or 'finally'
        try {
C:\MyWebApp\projects3\HelloWebApplet\DBAccess\src\com\stardeveloper\example\JdbcExample2.java:19: ')' exp
ected
        ctch(Exception e) {
C:\MyWebApp\projects3\HelloWebApplet\DBAccess\src\com\stardeveloper\example\JdbcExample2.java:32: ';' exp
ected

3 errors
C:\MyWebApp\projects3\HelloWebApplet\DBAccess\nbproject\build-impl.xml:205: The following error occurred 
while executing this line:
C:\MyWebApp\projects3\HelloWebApplet\DBAccess\nbproject\build-impl.xml:92: Compile failed; see the compil
er error output for details.
BUILD FAILED (total time: 0 seconds)
```

Every action that is run by an Ant script, such as compiling, running, and debugging files, sends its output to the same Output window tab. If you need to save the messages displayed in the Output window, you can copy and paste it to a separate file. You can also set Ant to print the command output for each new target to a new Output window tab by choosing Tools > Options, selecting Ant Settings, and selecting the checkbox in the Reuse Output Tabs property.

Customizing the Build Process

With standard projects, you can customize the build process by doing any of the following:

- Entering basic options, like classpath settings and JAR filters, in the New Project wizard when you create a project, or afterwards in the Project Properties dialog box.
- Customizing existing Ant targets.
- Creating new Ant targets.
- Editing properties in `project.properties` to change the name and location of build output folders and files.

The following table lists some common tasks for redefining a JAR file that you may find useful:

To perform this task	Follow these steps
Specify which files are added to a JAR file.	Right-click the project node in the Projects window and choose Properties. Click the Packaging subnode (under Build) and configure the filter and compression settings using the Exclude from JAR File field. See the Regular Expressions table below for details.
Change a JAR file's name and location.	In the Files window, go to the `nbproject` folder in your project folder and open `project.properties` in the Source Editor. Enter the full path to the JAR file in the `dist.jar` property.
Specify the manifest file for a JAR file.	In `project.properties`, type the name of the manifest file in the `manifest.file` property. The file name must be specified relative to the project's `build.xml` file. Note that if you are using the Java Application template, the IDE creates a manifest file for you.

Disable the generation of a JAR file for a project.	In the Files window, open your project folder and open `build.xml`. Override the `jar` target to have no contents and no dependencies. For example, add the following to `build.xml`: `<target name="jar" />`

Note: To customize the build process for web applications and to redefine WAR files, see Developing Web Applications.

Regular Expressions for Filtering Output Files

When you create a JAR file or a WAR file, you usually want to include just the compiled `.class` files and any other resource files located in your source directory, such as resource bundles or XML documents. The default filter does this for you by excluding all `.java`, `.nbattrs`, and `.form` files from your output file. Here are some additional regular expressions you can use:

<table>
<tr><th>Regular Expression</th><th>Description</th></tr>
<tr><td><code>\.html$</code></td><td>Exclude all HTML files</td></tr>
<tr><td><code>\.java$</code></td><td>Exclude all Java files</td></tr>
<tr><td><code>(\.html$)|(\.java$)</code></td><td>Exclude all HTML and Java files</td></tr>
<tr><td><code>(Key)|(\.gif$)</code></td><td>Exclude all GIF files and any files with <code>Key</code> in their name</td></tr>
</table>

For a guide to regular expression syntax, click here.

Running Applications

Because the IDE is built entirely on top of Ant, it uses an Ant script to run your applications. If you are using a standard project, the IDE generates the build script based on the options you enter in the project's Project Properties dialog box. For Java applications, you can set the project's main class, runtime arguments, VM arguments, and working directory in the Project Properties dialog box. If you are using a free-form project, the IDE uses your existing Ant script to run your application. You can write a target that executes the currently selected file in the IDE and map it to the Run File command.

This section covers the following topics:

- Running Projects and Files
- Customizing Runtime Options
    - Setting the Runtime Classpath
    - Setting the Main Class and Runtime Arguments
    - Setting JVM Arguments

To set runtime options for web applications, see Developing Web Applications.

Running Projects and Files

For Java projects, you typically set the project that contains the program's main class as the main project. For web projects, the main project is the project that is first deployed. You then run the entire application with the Run Main Project command (F6). You can also run any runnable class by choosing Run > Run File > Run *filename* (Shift-F6). You can also run any project by right-clicking its project node in the Projects window and choosing Run Project from the contextual menu. Note that for Java projects, the project must have a main class.

To run a project, package, or file, select it in the Projects window and choose one of the following:

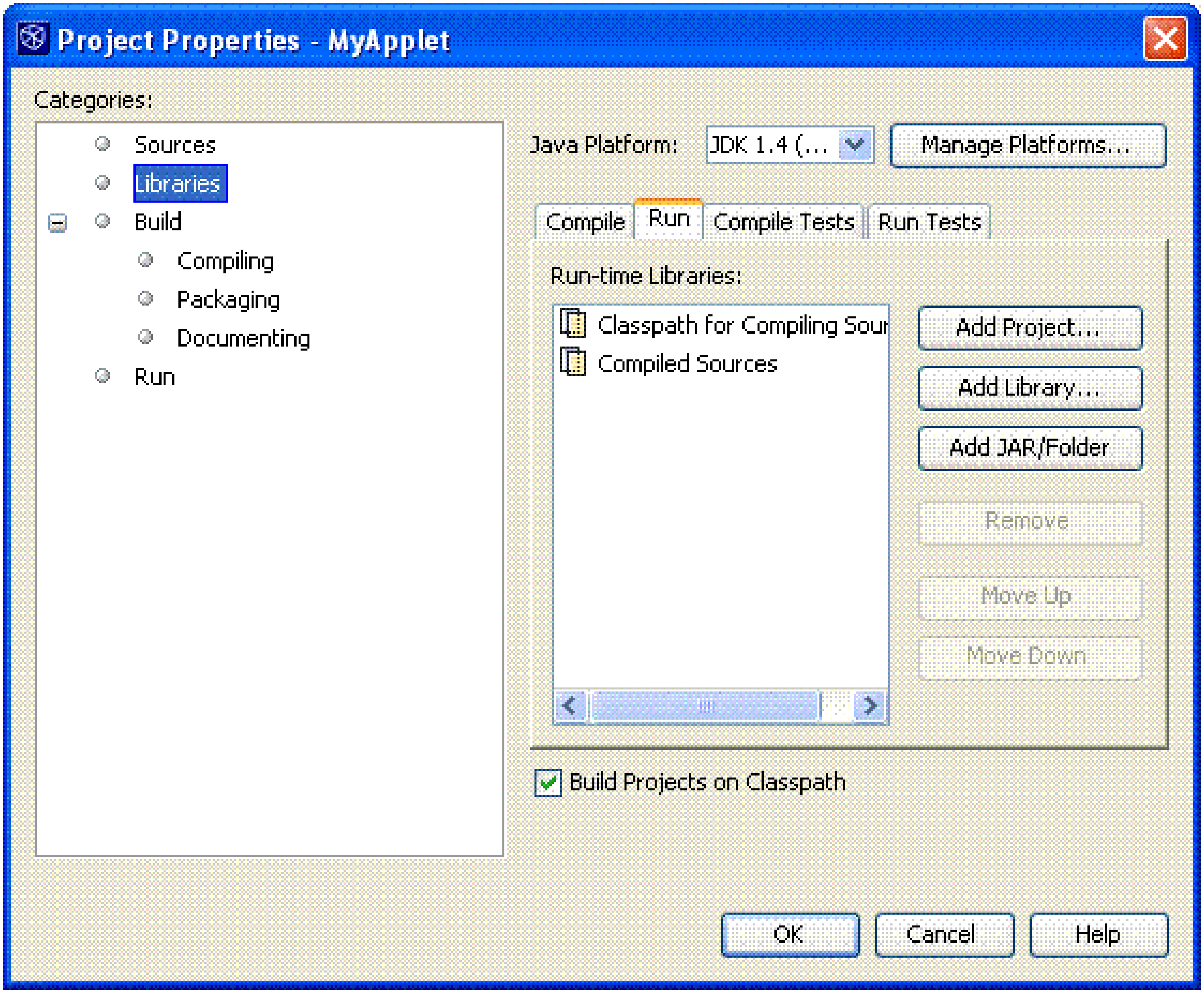
- In the main menu, choose **Run > Run Main Project (F6)** to run the main project. Alternately, you can use the ▶ toolbar button.
- In the Projects window, right-click the project and choose **Run Project** to run a project.
- In the Projects window, right-click the file and choose **Run File (Shift+F6)** to run a file. Alternatively, choose **Run > Run File > Run *filename* (Shift+F6)** in the main menu. Note that if you are using a free-form project, this command is disabled by default. You have to write an Ant target for running the currently selected file in the IDE and map it to the Run File command.

Any compilation errors and output are displayed in the Output window.

Customizing Runtime Options

By default, the IDE does not specify a main class, runtime arguments, and JVM arguments. The runtime classpath of each standard project contains the project's compiled classes and everything in the project's compilation classpath. You can view the project's compilation classpath by selecting the Libraries node in the Categories pane and then clicking the Compile tab in the right pane of the Project Properties dialog box.

To change runtime options, open the Project Properties dialog box by right-clicking the project node in the Projects window and choosing Properties. Next, select the Libraries node in the Categories pane and click the Run tab in the right pane of the dialog box as illustrated in the image below. Note that to access settings for the main class, program arguments, the working directory for program execution and VM options in NetBeans 4.1, you have to select the Run node. In the next section we'll take a closer look at how to configure the runtime classpath.



Project Properties - MyApplet

Java Platform: JDK 1.4 (...

Run-time Libraries:

Classpath for Compiling Sour
Compiled Sources

☑ Build Projects on Classpath

Setting the Runtime Classpath

To add projects, libraries, JAR files, and folders to the project's runtime classpath, use the buttons on the right side of the Run-time Libraries list in the dialog box.

If your project uses special libraries dynamically at runtime through an indirect interface or reflection (like JDBC drivers or JAXP implementations), you have to add these libraries to the runtime classpath. You also have to adjust your runtime classpath if the runtime dependencies between your projects do not match the compilation dependencies between the projects. For example, imagine that project A compiles against project B, and project B compiles against project C, but project A does not compile against project C. This means that project A only has project B on its runtime classpath. If project A requires both project B and project C during execution, you have to add project C to project A's runtime classpath.

In free-form projects, your Ant script handles the classpath for all of your source folders. The classpath settings for free-form projects in the Project Properties dialog box only tell the IDE what classes to make available for code completion and refactoring.

Setting the Main Class and Runtime Arguments

To set the project's main class, select the Run node in the Categories pane and type its fully qualified name in the Main Class field (for example, `org.myCompany.myLib.MyLibClass`). The main class must exist in the project or in one of the JAR files or libraries in the project's runtime classpath. Afterwards, type any necessary runtime arguments in the Arguments field.

If you use the Browse button to choose the project main class, the file chooser only shows classes in your project source directory. If you want to specify a class in one the libraries on the classpath, you have to type the fully-qualified name of the class in the Main Class field.

Setting JVM Arguments

Type a space-separated list of JVM arguments in the VM Options field.

You can set system properties by typing the following in the VM Options field:

```
-Dname=value
```

Debugging Applications

Example Code:

- ArrayFill.zip

Debugging is the process of examining your application for errors. You debug by setting breakpoints and watches in your code and running it in the debugger. You can execute your code one line at a time and examine the state of your application in order to discover any problems.

The IDE uses the Sun Microsystems JPDA debugger to debug your programs. When you start a debugging session, all of the relevant debugger windows appear automatically at the bottom of your screen. You can debug an entire project, any executable class, and any JUnit tests. The IDE also lets you debug applications that are running on a remote machine by attaching the debugger to the application process.

When you run or debug web applications, JSP pages, or servlets, you can also use the HTTP Monitor to monitor data flow. The HTTP Monitor appears by default. The HTTP Monitor gathers data about HTTP requests that the servlet engine processes. For each HTTP request that the engine processes, the monitor records data about the incoming request, the data states maintained on the server, and the servlet context. You can view data, store data for future sessions, and replay and edit previous requests. For details on the HTTP Monitor, choose Help > Help Contents in the main menu.

For free-form projects, you have to write an Ant target for the Debug Project command. You can also write targets to debug specific files and map these targets to the project's commands.

In this section you will learn about:

Basic Debugging

In this section, we will use a simple example to demonstrate how to start a debugging session, step through your code manually, and monitor variables and method calls. We will leave more advanced functions like setting breakpoints and watches for the following sections.

Our example for this section is the Array Fill application. This application is very simple. It creates an array of `sampleBeans`, each one of which has two properties, `firstName` and `lastName`. It then assigns values to the properties of each bean and prints out the values.

The first thing you want to do is run the application to see if it throws any exceptions. Download and extract the ArrayFill example .zip archive. To open the ArrayFill project in the IDE, press CTRL-Shift-O, locate the extracted ArrayFill folder and click Open Project Folder. The ArrayFill project opens in the IDE and the logical strucure of the project is visible in the Projects window.

In the Projects window, expand the `arrayfill` package under the Source Packages. The arrayfill package contains two classes: `ArrayFill` and `SampleBean`. Right-click `ArrayFill.java` and press Shift-F6 to execute it. The following output should appear in the Output window:

```
java.lang.NullPointerException
            at arrayfill.ArrayFill.loadNames(arrayFill.java:27)
            at arrayfill.ArrayFill.main(ArrayFill.java:34)
            Exception in thread "main"
            Java Result: 1
```

Starting a Debugging Session

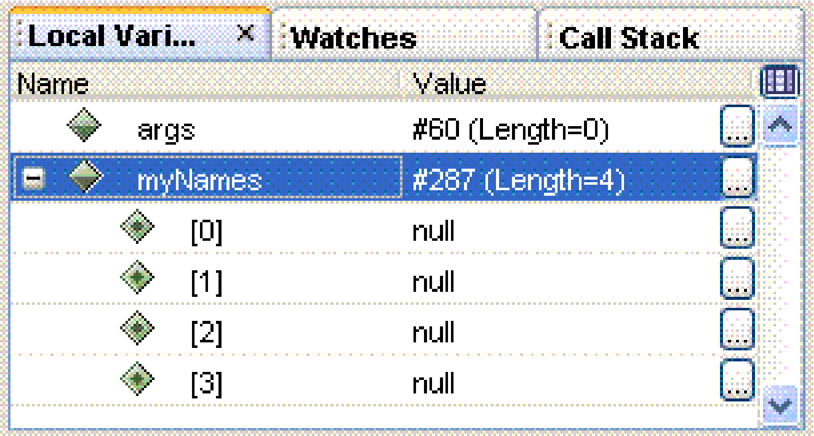
When you start a debugging session in the IDE, the IDE compiles the files that you are debugging, runs them in debug mode, and displays debugger output in the Debugger windows. To start a debugging session, select the file that you want to debug and choose one of the following commands from the Run menu:

- **Debug Main Project (F5)**. Runs the main project until the first breakpoint is encountered.
- **Step Into (F7).** Starts running the main project's main class and stops at the first executable statement.
- **Run to Cursor (F4).** Starts a debugging session, runs the application to the cursor location in the Source Editor, and pauses the application.

If more than one project is open in the IDE, make sure that Array Fill is set as the main project by right-clicking the ArrayFill node in the Projects window and choosing Set Main Project from the contextual menu. Press F7 to step into the main project's main class. If the main class for the project is not set, the IDE prompts you to set it. Then the IDE opens the file in the Source Editor, displays the Output window and Debugger windows, and stops just inside the `main` method.

Debugger Windows

Let's take a minute to look at the Debugger windows. The Debugger windows automatically open whenever you start a debugging session and close when you finish the session. By default, the IDE opens three Debugger windows: the Local Variables window, Watches window, and Call Stack window.



Name	Value
args	#60 (Length=0)
myNames	#287 (Length=4)
[0]	null
[1]	null
[2]	null
[3]	null

You can open other Debugger windows by choosing from the Window > Debugging menu. When you open a Debugger window during a debugging session, it closes automatically when you finish the session. If you open a Debugger window when no debugging session is open, it stays open until you close it manually. You can arrange Debugger windows by dragging them to the desired location.

The following table lists the Debugger windows.

Name	**Shortcut**	**Description**
Local Variables	Alt-Shift-1	Lists the local variables that are within the current call.
Watches	Alt-Shift-2	Lists all variables and expressions that you elected to watch while debugging your application.
Call Stack	Alt-Shift-3	Lists the sequence of calls made during execution of the current thread.
Classes	Alt-Shift-4	Displays the hierarchy of all classes that have been loaded by the process being debugged.
Breakpoints	Alt-Shift-5	Lists the breakpoints in the current project.
Sessions	Alt-Shift-6	Lists the debugging sessions currently running in the IDE.
Threads	Alt-Shift-7	Lists the thread groups in the current session.
Sources	Alt-Shift-8	Lists the source directories on your project classpath. You can set whether to step into or step over classes by deselecting their source folders here. The IDE automatically steps over JDK classes; if you want to step into them, select the JDK sources in this window.

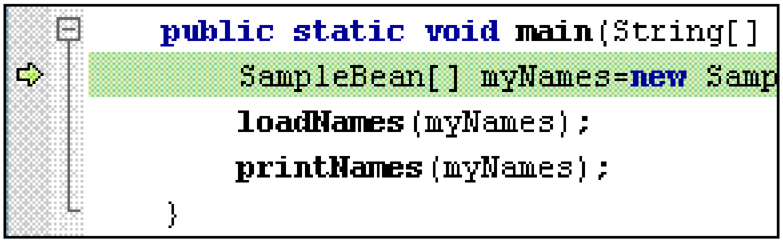
Stepping Through Your Code

You can use the following commands in the Run menu to control how your code is executed in the debugger:

- **Step Over (F8).** Executes one source line. If the source line contains a call, executes the entire routine without stepping through the individual instructions.

- **Step Into (F7).** Executes one source line. If the source line contains a call, stops just before executing the first statement of the routine.
- **Step Out (Alt-Shift-F7).** Executes one source line. If the source line is part of a routine, executes the remaining lines of the routine and returns control to the caller of the routine.
- **Pause.** Pauses application execution.
- **Continue (Ctrl-F5).** Continues application execution. The application will stop at the next breakpoint.
- **Run to Cursor (F4).** Runs the current session to the cursor location in the Source Editor and pauses the application.

In our example, use the F7 key to step through the code one line at a time. The `NullPointerException` occurred in the `loadNames` call, so when you step to that call, watch the value of the `names` array in the Local Variables view. Each of the beans have a value of null. You can continue stepping through the `loadNames` method - the `names` beans are null throughout.



```
    public static void main(String[]
        SampleBean[] myNames=new Samp
        loadNames(myNames);
        printNames(myNames);
    }
```

The problem here is that while the line

```
        SampleBean[] myNames=new SampleBean[fnames.length];
```

initiates the array that holds the beans, it does not instantiate the beans themselves. The individual beans have to be instantiated in the `loadNames` method by adding the following code:

```
        names[i]=new SampleBean();
```

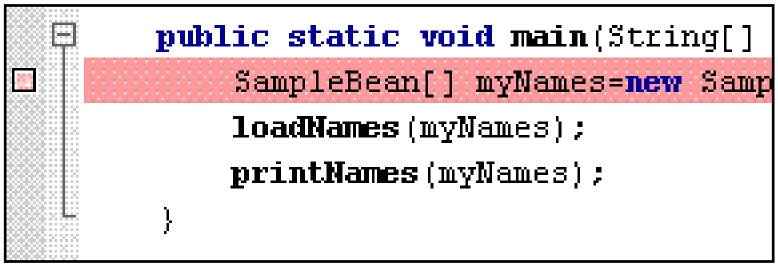
before the line `names[i].setLastName(lnames[i]);` in the `LoadNames` method.

Working With Breakpoints

Most applications are far too big to examine one line at a time. More likely, you set a breakpoint at the location where you think a problem is occurring and then run the application to that location. You can also set more specialized breakpoints, such as conditional breakpoints that only stop execution if the specified condition is true or breakpoints for certain threads or methods.

In this section, we will use the `ArrayFill` class from the last example, so you will have to recreate the bug by commenting out the code you added above.

Setting a Breakpoint



```
public static void main(String[]
    SampleBean[] myNames=new Samp
    loadNames(myNames);
    printNames(myNames);
}
```

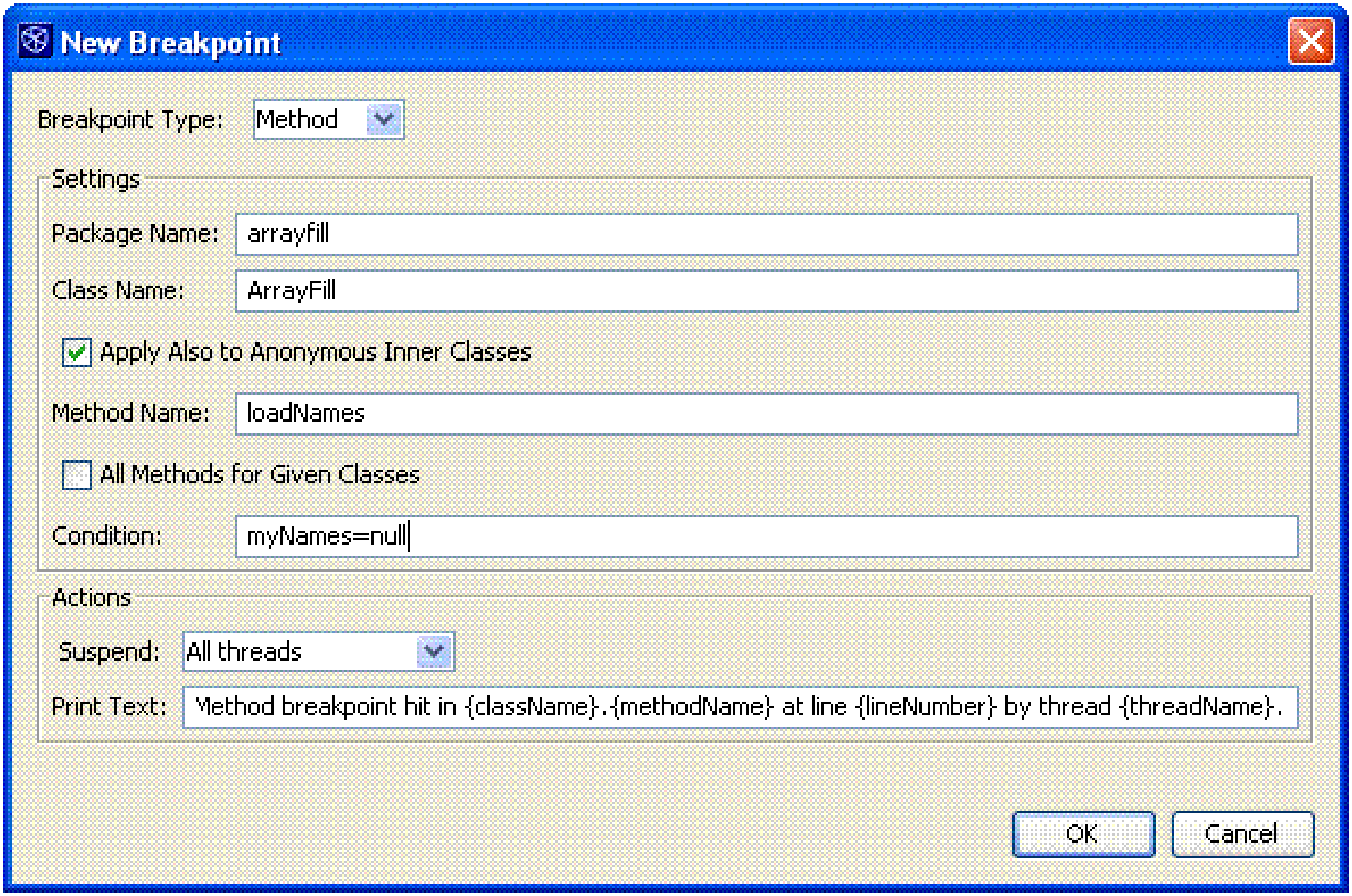
If you just want to set a simple line breakpoint, you can click the left margin of the desired line. A line breakpoint icon (□) appears in the margin. You can remove the line breakpoint by clicking it again.

For more complex breakpoints, use the New Breakpoint (Ctrl-Shift-F8) command in the Run menu. The New Breakpoint dialog box lets you choose the type of breakpoint you want to create and set breakpoint options such as conditions for breaking or the information that the breakpoint prints to the Output window.

Setting Conditions for a Breakpoint

Conditional breakpoints only stop execution if a specified boolean expression is true. If you want to set a conditional breakpoint, open the New Breakpoint dialog box and enter an expression in the Condition field.

For example, open `ArrayFill.java`, set the insertion point in the `loadNames` method call in the `main` method, and press Ctrl-Shift-F8. In the dialog box, enter `myNames=null` in the Condition field and click OK. Then press F5 to start debugging the project. The execution should break at the `loadNames` method call.



Breakpoint Type: Method

Settings

Package Name: arrayfill

Class Name: ArrayFill

☑ Apply Also to Anonymous Inner Classes

Method Name: loadNames

□ All Methods for Given Classes

Condition: myNames=null

Actions

Suspend: All threads

Print Text: Method breakpoint hit in {className}.{methodName} at line {lineNumber} by thread {threadName}.

Customizing the Output for a Breakpoint

In the New Breakpoint dialog box, you can also specify what information is printed when a breakpoint is reached. Enter any message in the Print Text field at the bottom of the dialog box. You can use variables to refer to certain types of information you want displayed.

Breakpoint Types

The following table lists the different breakpoint types that are available.

Type	**Description**
Line	You can break execution when the line is reached, or when elements in the line match certain conditions.
Method	When you set a breakpoint on a method name, application execution stops every time the method is executed.

Exception	You have several options for setting a breakpoint on an exception. You can break whenever a specific exception is caught, whenever a specific exception is not handled in the source code, or whenever any exception is encountered regardless of whether the application handles the error or not.
Variable	You can stop execution of your application whenever a variable in a specific class and field is accessed (for example, the method was called with the variable as an argument) or modified.
Thread	You can break application execution whenever a thread starts, stops, or both.
Class	When you set a breakpoint on a class, you can stop the debugger when the class is loaded into the virtual machine, unloaded from the virtual machine, or both.

Setting Watches

A watch enables you to track the changes in the value of a variable or expression during application execution. To set a watch, select the variable or expression you want to set a watch on in the Source Editor, then right-click and choose New Watch (Ctrl-Shift-F7).

You can also create fixed watches in the Watches view. While a normal watch describes the content of a variable, a fixed watch describes the object that is currently assigned to the variable. To create a fixed watch, right-click any item in the Local Variables or Watches view and choose Create Fixed Watch.

Developing Web Applications

A web application is a term used in the JavaServer Pages Specification and Java Servlet Specification. It roughly corresponds to the J2EE term "web module". A web module is the smallest deployable unit of web resources in a J2EE application. It corresponds to a "servlet context" as defined in the J2EE specification.

A web application typically consists of:

- Presentation logic, including HTML, JSP, and text files
- Programming logic, including JavaBeans components, servlets, and utility classes
- Information on how to tie them all together for deployment in the form of a deployment descriptor

In this section you will learn about the following:

- Building Web Applications
- Customizing the Build Process
- Running Web Applications
- Customizing Runtime Options

Building Web Applications

To compile a project, package, or file, select it in the Projects window and choose one of the following from the main window:

- In the main menu, choose **Build > Build Main Project (F11)** to build the main project.
- In the main menu, choose **Build > Clean and Build Main Project (Shift+F11)** to clean and build the main project.
- Right-click the project in the Projects window and choose **Build Project** to build a project.
- Right-click the project in the Projects window and choose **Clean Project** to clean a project.
- Right-click the package in the Projects window and choose **Compile Package (F9)** to compile a package.
- Right-click the file in the Projects window and choose **Compile File (F9)** to compile a file. Alternatively, choose **Build > Compile File (F9)**. If you are using a free-form project, this command is disabled by default. You have to write an Ant target for compiling the currently selected file in the IDE and map it to the Compile File command.

You can also use the following toolbar buttons:

- Build the main project
- Clean and build the main project

For details, see Building Applications.

Customizing the Build Process

For redefining a WAR file, the following table lists some common tasks:

To perform this task	Follow these steps
Specify which files are added to a WAR file.	Right-click the project node in the Projects window and choose Properties. Click Packaging and configure the filter and compression settings.
Change a WAR file's name and location.	In the Files window, go to the `nbproject` folder and open `project.properties` in the Source Editor. Enter the full path to the WAR file in the `dist.dir` property and the WAR file's name in the `war.name` property. Notice that these two properties are concatenated to form the `dist.war` property.
Disable the generation of a WAR file for a project.	In the Files window, double-click the `build.xml` node so that it opens in the Source Editor. Override the `do-dist` target to have no contents and no dependencies. For example, add the following to `build.xml`: `<target name="do-dist" />`
Perform some checks before or after compilation or before or after a WAR file is built.	In the Files window, double-click the `build.xml` node so that it opens in the Source Editor. Create targets with the following names: • `-pre-compile` • `-post-compile` • `-pre-dist` • `-post-dist` These targets are executed before or after the main task in question. Other similar targets exist, such as `-pre-init` and `-post-init`.

Any compilation errors and output are displayed in the Output window.

For general information on customizing the build process, see Customizing the Build Process.

Running Web Applications

To compile a project, package, or file, select it in the Projects window and choose one of the following from the main window:

- In the main menu, choose **Run > Run Main Project (F6)** to run the main project.
- Right-click the project in the Projects window and choose **Run Project** to run a project.
- Right-click the file in the Projects window and choose **Run File (Shift+F6)** to run a file. Alternatively, choose **Run > Run File > Run File (Shift+F6)**. If you are using a free-form project, this command is disabled by default. You have to write an Ant target for running the currently selected file in the IDE and map it to the Run File command.

You can also use the toolbar button below:

- Run the main project

Any compilation errors and output are displayed in the Output window.

For general information on running web applications, see Running Applications.

Customizing Runtime Options

Setting the Context Path

When you execute a web application on the server, the server uses a context path setting to derive the path to the web application. For example, if the context path is `/MyWebApp`, then you can access a file named `index.html` under the web application's document base (root directory) by using the URL `http://host:port/MyWebApp/index.html`.

You should set a context path for a web application if you plan to execute different web applications on the same server. Otherwise, files with the same name will overwrite files from other web applications. When you create a web application from the New Project wizard, the default context path is derived from the name of the document base (root directory).

To set the context path, right-click the project node in the Projects window and choose Properties from the contextual menu. In the Project Properties dialog box, select the Run page, and type the Context Path. The path must begin with a forward slash (/), such as `/MyWebApp`.

When you change the context path in the Project Properties dialog box, the IDE updates the context descriptor (`web/META-INF/context.xml` for the Tomcat Web Server or `web/WEB-INF/sun-web.xml` for the Sun Java System Application Server) to match.

Setting the Web Server

During development and testing, you can execute your web application using any server installed in the IDE that implements a J2EE-compliant web container. By default, the IDE executes web applications using the Server node's default web server. To change the default web server, right-click the project node in the Projects window and choose Properties. Click Run in the Project Properties dialog box and then choose the appropriate web server from the Server drop-down.

Setting the Web Browser

The IDE uses the IDE's default web browser to run a web application. You can set the default web browser to be one of the supported browsers, such as the supported versions of Mozilla, Netscape, and Internet Explorer, or to be the IDE's internal web browser. You can also configure the IDE to use other browser types.

Choose Tools > Setup Wizard from the main menu. Select a browser from the Web Browsers drop-down. You can choose from the following web browsers:

- Default System Browser.
  The browser that is registered as your operating system's default browser.
- Swing HTML Browser.
  A simple HTML browser based on a Swing component. The internal Swing HTML browser provides a higher level of integration with the IDE than an external browser. You can embed the Swing HTML browser into another IDE window or an MDI frame.
- Any other browser installed on your computer.

Setting Parameters

You can pass request parameters in URL query string format to JSP pages and servlets from the IDE. Specifying input data in this fashion provides a useful aid in testing for expected JSP or servlet output.

To specify parameters for a JSP page, right-click the JSP file in the Projects window and choose Properties. In the Request Parameters dialog box, type the parameters in the URL query string format. A URL query string is a string appended to the URL for passing parameter values in a GET request. The query string must begin with a question mark (?). The parameters must be in name/value pairs, with the pairs separated by ampersands (&). The names and values must be URL-encoded. For example, white space is encoded as +. The following URL shows an example of a query string:

```
http://www.myapp.com/sayhello?name=Fido+Filbert&type=Dog
```

To specify parameters for a servlet, select the servlet in the Projects window, and choose Tools > Set Servlet Execution URI in the main menu. Alternatively, right click the servlet and choose Tools > Set Servlet Execution URI from the contextual menu. Type the execution URI and parameters.

The IDE saves the parameters and automatically passes them to the JSP or servlet the next time you run it. Note that the HTTP Monitor tool enables you to edit and resend request parameters.

Connecting to Databases

Example Code:

- CustomerNameAndCity.zip

This section explains the basics of using the IDE to create a connection from a web application to a database. The web application is deployed using a Sun Java System Application Server or a Tomcat Web Server and data from a Pointbase database or a MySQL database are displayed on a JSP page.

This section covers the following topics:

Setting Up Your Resources

You use Java™ Database Connectivity (JDBC™) technology to connect to a database. The JDBC application programming interface (API) is Sun Microsystems' API for connecting to databases that support Structured Query Language (SQL). The JDBC API is a package of object-oriented objects that includes `Connection`, `ResultSet`, and `Statement`. Each object contains various API methods, for example, `connect()`, `close()`, and `prepareStatement()`. You will use these objects and methods later in this section. To do so, you need to use a database that supports SQL.

A database runs in a database *server*. If you don't have a database server, you can download one from the Internet. A *driver* supporting the JDBC API ("JDBC driver") translates JDBC calls into the network protocols that are used by SQL databases. Since many of these network protocols are proprietary, the MySQL vendor itself is the primary source for the driver that works with its database server. If the MySQL JDBC driver is made available to your IDE, you can make direct calls from the IDE to the database. If the MySQL JDBC driver is made available to your web application, you can make direct calls from your web application to the database using your web server.

- **Pointbase.** Pointbase is bundled with the IDE, if your IDE is bundled together with the Sun Java System Application Server. To use it with the instructions that follow, all you need to do is start it. Choose Tools > Pointbase Database > Start Local Pointbase Database. When you do this, you will see something similar to the following in the Output window:

  ```
  Starting Server C:\Program Files\Java\jdk1.5.0_01\bin\java
  ```

  Server started, listening on port 9092, display level: 0 ...

- **MySQL.** MySQL is not bundled with the IDE, but you can download it for free from the Internet. Go to its vendor's site and follow the instructions to download the MySQL database server. Once you've downloaded and installed your SQL database server, follow the vendor's instructions to start it. A MySQL driver can be downloaded for free from its vendor's site. Note that you are recommended to get a *type 4* driver, because it is a pure Java driver. You can search for the database driver of your choice by using Sun's JDBC Data Access API site.

Once you have your server resources, make them available to your IDE and your web applications, when appropriate, according to the instructions that follow.

Using the Database Explorer

The IDE provides the Database Explorer so that you can perform from the IDE the following simple operations that are related to JDBC-compliant databases:

- Connect to a database
- Choose a driver for your database
- Fill in the appropriate driver for your database
- Create, browse and edit database structures
- Enter SQL queries and see the results immediately
- Connect to multiple databases concurrently
- Migrate table schemas across databases from different vendors

In this section, you will use a simple example to demonstrate how to make a database driver available to the IDE, use the database explorer to create a a database, connect to the database, create a table, populate it with values, and view the values. We will explain how to use the data in a web application in subsequent sections. Note that using the database explorer is not a prerequisite for making a database connection from a web application. Our example for this section is a database called `sample`. It contains a table called CUSTOMER_TBL, with the fields NAME and CITY. It is a subset of the sample databases provided by Pointbase, so if you are using this database server, you do not need to do anything to create the database. However if you are using MySQL or any other database server, follow the instructions below to set everything up.

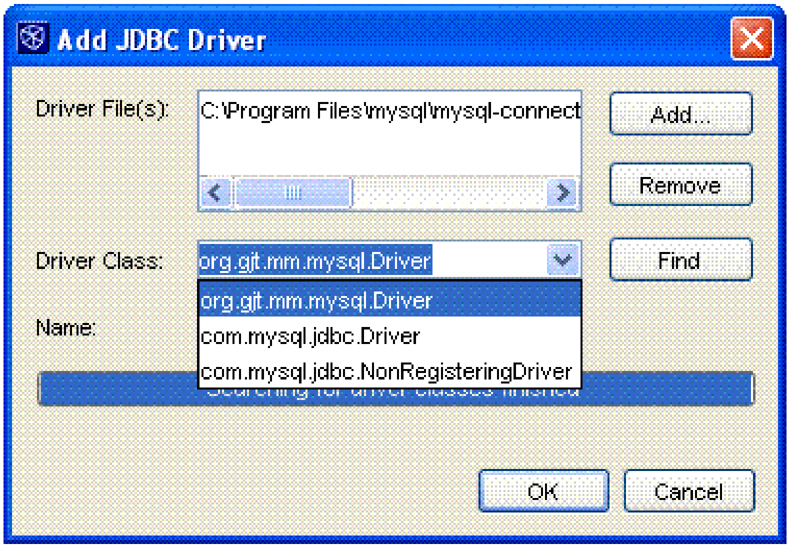
Access the Database Explorer

To get started, view the Runtime window. If the Runtime window is not displayed by default, choose Window > Runtime (Ctrl-5) from the main menu. Expand the Databases node. The Database Explorer is displayed.

Make the Database Driver Available to the IDE

Before you can use the Database Explorer to work with your database, you must make your database's driver available to the IDE. Because Pointbase is bundled with the IDE, this step is unnecessary if you are using Pointbase. However, for every other server, right-click the Drivers node and choose Add Driver from the contextual menu. The Add JDBC Driver dialog box appears. Click Add and browse to where you downloaded your database driver and select the database driver's JAR or ZIP file. The Database Explorer fills the other fields in the dialog box.

If the Driver Class field is empty or incorrect, click Find. The IDE searches the JAR file that you selected and finds all classes that implement the JDBC API Driver interface (`java.sql.Driver`). The Driver Class combo box lists the classes that the IDE finds. Use the combo box to select the driver class that you need and click OK.



Add JDBC Driver

Driver File(s): C:\Program Files\mysql\mysql-connect

Driver Class: org.gjt.mm.mysql.Driver

Name:

Add...
Remove
Find
org.gjt.mm.mysql.Driver
com.mysql.jdbc.Driver
com.mysql.jdbc.NonRegisteringDriver
Searching for driver classes finished
OK
Cancel

Now, when you expand the Drivers node in the Database Explorer, a new node is displayed for the new driver you added. The nodes under the Drivers node are all the registered drivers you can use. This is what the icons mean:

Icon	Description
⚙	The driver can be loaded by the IDE and you can connect to the database.
⊘	The IDE cannot connect to the database using this driver because the driver's JAR or ZIP file isn't placed in the specified location. You can correct the location by right-clicking the driver node and choosing Customize.

Note that when you have made your database driver available to the IDE, you have not made it available to your web application. At this stage, you can use the IDE to access and modify your database. You cannot access and modify your database from your web application yet. To be able to do so, your database driver's JAR file needs to be in your web application's `WEB-INF/lib` folder or, for Tomcat connection pooling, in Tomcat's `common/lib` folder. There is no need to do this yet. For details, see the instructions that apply in the sections that follow.

Create the Database

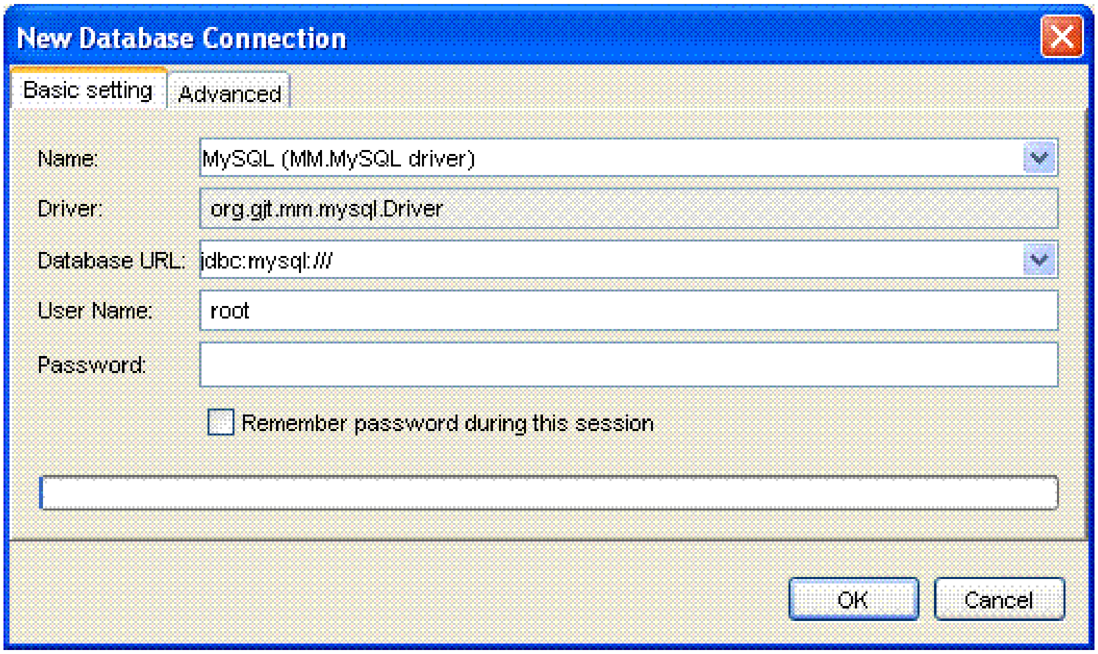
Before going further, make sure your database server is up and running. For example, for MySQL you can run `mysqld` in the database server's `bin` directory.

The database server that you downloaded provides functionality and instructions for creating a database. In the case of MySQL, you can use the IDE to create a database. To do so, right-click your driver's node and choose Connect Using from the contextual menu.

Now type the following URL in the Database URL text box:

```
    jdbc:mysql:///
```

Next, specify the username and password that you defined when you installed your database server. If you do not specify these, you will not be able to execute the SQL commands required to create the database. The New Database Connection dialog box should now look as follows:



Name: MySQL (MM.MySQL driver)

Driver: org.gjt.mm.mysql.Driver

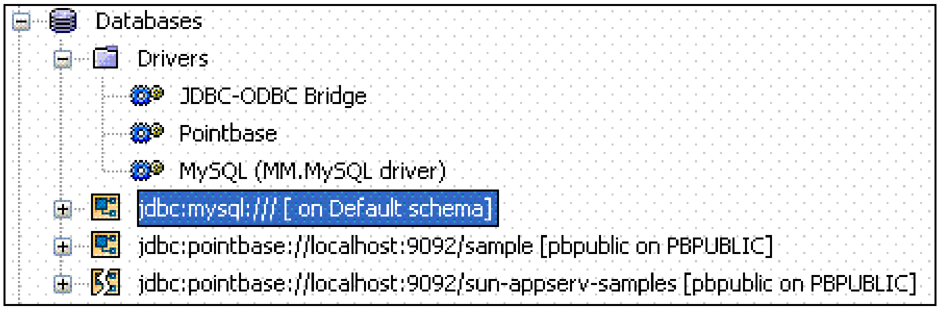
Database URL: jdbc:mysql:///

User Name: root

Password:

□ Remember password during this session

Click OK. A blue bar containing the text "Connection established" should now appear at the bottom of the New Database Connection dialog box. A database connection node appears in the Database Explorer, as shown below:


Databases
Drivers
JDBC-ODBC Bridge
Pointbase
MySQL (MM.MySQL driver)
jdbc:mysql:/// [on Default schema]
jdbc:pointbase://localhost:9092/sample [pbpublic on PBPUBLIC]
jdbc:pointbase://localhost:9092/sun-appserv-samples [pbpublic on PBPUBLIC]

Right-click the new database connection node. Choose Execute Command from the contextual menu. The `jdbc:mysql:///` tab is displayed. Type the following SQL command in the Command text box:

```
create database sample
```

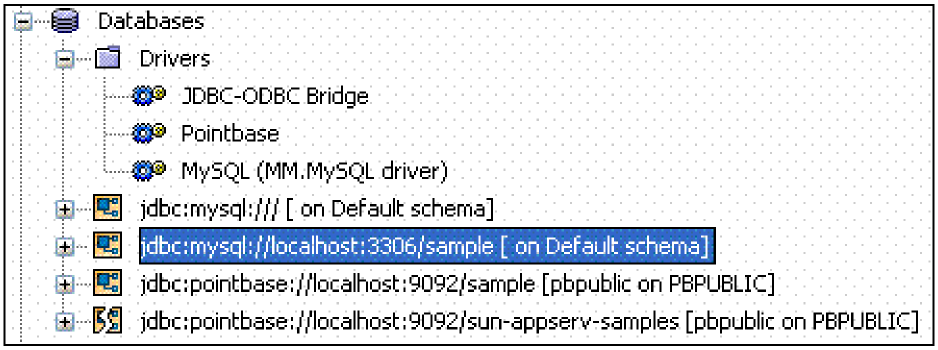
Click Execute. Below the Command history drop down, you should now see the message "Command successfully executed." You have now created a database from the IDE.

Connect to the Database

To connect to your database, right-click your driver's node and choose Connect Using from the contextual menu again. Each database server has different requirements for the syntax of a database URL. For MySQL, type the following URL in the Database URL text box:

```
jdbc:mysql://localhost:3306/sample
```

Specify the username and password that you defined for your database server and click OK. When you expand the Databases node, you should now see a node with a URL that includes the name of your database, as shown below:


Databases
Drivers
JDBC-ODBC Bridge
Pointbase
MySQL (MM.MySQL driver)
jdbc:mysql:/// [on Default schema]
jdbc:mysql://localhost:3306/sample [on Default schema]
jdbc:pointbase://localhost:9092/sample [pbpublic on PBPUBLIC]
jdbc:pointbase://localhost:9092/sun-appserv-samples [pbpublic on PBPUBLIC]

Create the Table

You can create the table in one of two ways. In the first, you execute an SQL command in the IDE. The SQL command creates the table. In the second, you use the Create Table dialog box to create the table. If you use the Create Table dialog box, you cannot set the `auto_increment` property, which means that you have to add a new value for the primary key (`CUST_ID`) manually whenever you add values to populate the table. For this reason, using the SQL command is preferable.

- **Using the SQL Command.** Right-click the node for your `sample` database, select Execute Command from the contextual menu and execute the following command:

  ```
  CREATE TABLE CUSTOMER_TBL
  (
    CUST_ID int unsigned not null auto_increment primary key,
    NAME varchar(30),
    ADDR_LN1 varchar(30),
    CITY varchar(25)
  );
  ```

- **Using the Create Table Dialog Box.** Expand the new node for your sample database, right-click the Tables node, and select Create Table. Type CUSTOMER_TBL in the Table name text field and define the table as shown in the illustration below. After filling out the first row, click Add Column to display a new row for input.

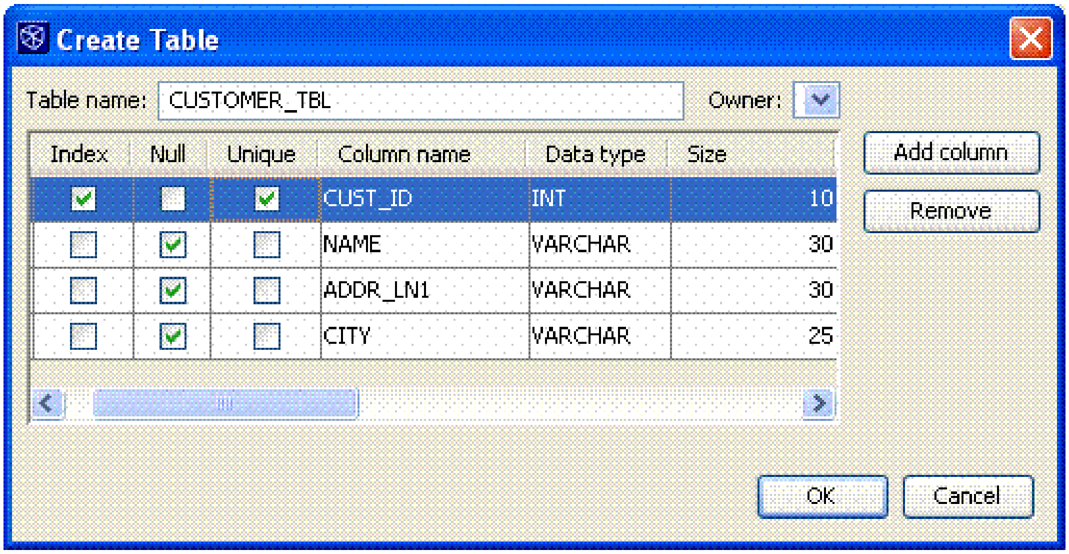


Table name: CUSTOMER_TBL

Owner:

Index	Null	Unique	Column name	Data type	Size
☑	☐	☑	CUST_ID	INT	10
☐	☑	☐	NAME	VARCHAR	30
☐	☑	☐	ADDR_LN1	VARCHAR	30
☐	☑	☐	CITY	VARCHAR	25

Populate the Table

To populate the table, right-click the node for your sample database and choose Execute Command from the contextual menu. Now execute the following command:

```
    INSERT INTO CUSTOMER_TBL
    (NAME, ADDR_LN1, CITY)
    values
    ("SuperCom", "490 Rivera Drive", "Miami");
```

Note: Remember that if you used the Create Table dialog box to create the table, you must include a value for the CUST_ID field.

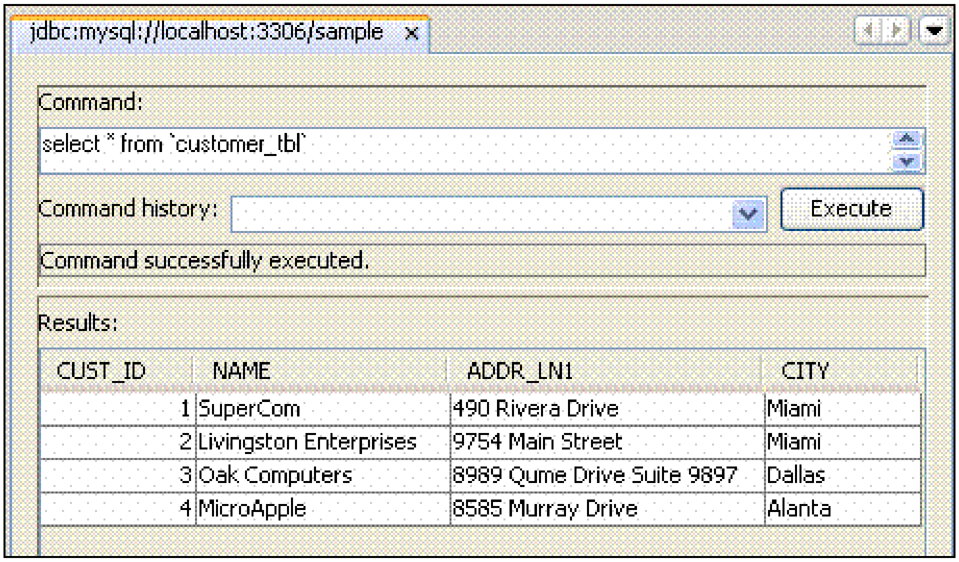
Execute the above command again a few times, but keep changing the values.

View the Values

Expand the node for your sample database, expand the Tables node, and right-click the CUSTOMER_TBL table in the Database Explorer and choose View Data from the contextual menu. The following command is executed to retrieve all the data in your CUSTOMER_TBL table:

```
    select * from `CUSTOMER_TBL`
```

The values that you inserted into the table are displayed:



Command:

```
select * from `customer_tbl`
```

Command successfully executed.

CUST_ID	NAME	ADDR_LN1	CITY
1	SuperCom	490 Rivera Drive	Miami
2	Livingston Enterprises	9754 Main Street	Miami
3	Oak Computers	8989 Qume Drive Suite 9897	Dallas
4	MicroApple	8585 Murray Drive	Alanta

Note that when you expand the CUSTOMER_TBL node in the Database Explorer, you can right-click each field and choose View Data, to view the values of each field in the table.

Connecting to a Database from a Web Application

MVC is a programming paradigm in which the application is separated into the following distinct parts:

- The *model* maintains the data that describes the state of the application.
- The *view* provides one or more views of the data.
- The *control* provides a conduit for data to enter the view.

It is good programming practice to access data using the control (a servlet or a JavaBeans component) only. However, for small applications or for testing during the development phase, you may want to quickly query your database by using SQL commands from a JSP page, which is the Java implementation of the view. This section outlines two strategies for doing this. The first shows you the simplest way of connecting to a database. Here, you use a JSP page to make a direct connection to your database and query your data. However, if you create a new connection to the database for each of your requests, you are wasting your web application's resources. For this reason, this method should be used for testing purposes only. The second strategy shows you how you can pool your connections to enhance data access efficiency.

Test the Database Connection from a JSP Page

In this example, you will use tags from the JSTL 1.1 library to access and display the data in your databse. This library is bundled with the IDE. You also need the database driver's JAR file to be on the application's classpath. Do this by right-clicking the project's Libraries node in the Projects window and then choosing Add Library. Select the JSTL 1.1 library and click Add Library. Right-click the Libraries node again and choose Add JAR/Folder. Next, browse to the folder that houses the database driver and select it. Click Open. When you expand the Libraries node, you should now see three new nodes: one for the database driver, one for the JSTL library's `standard.jar` file and another for the JSTL library's `jstl.jar` file.

Next, you need to let the server know that the JSP page uses tags from the JSTL library's CORE component and from the JSTL library's SQL component. To do so, add the following `taglib` directives after the `page` directives, which are at the top of your JSP file:

```
    <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    <%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

Next, create a data source inside your JSP page by adding the following code below the JSP file's `taglib` directives (in the next section, you will remove this data source and connect to one on the server):

- For Pointbase, add the following:

  ```
      <sql:setDataSource var="datasource"
  ```

```
        url="jdbc:pointbase://localhost:9092/sample" driver="com.pointbase.jdbc.jdbcDataSource"
        user="pbpublic" password="pbpublic"/>
```

- For MySQL, add the following:

```
    <sql:setDataSource var="datasource"
        url="jdbc:mysql://localhost:3306/sample" driver="org.gjt.mm.mysql.Driver"
        user="root" password=""/>
```

Note: You might need to specify a different user or password.

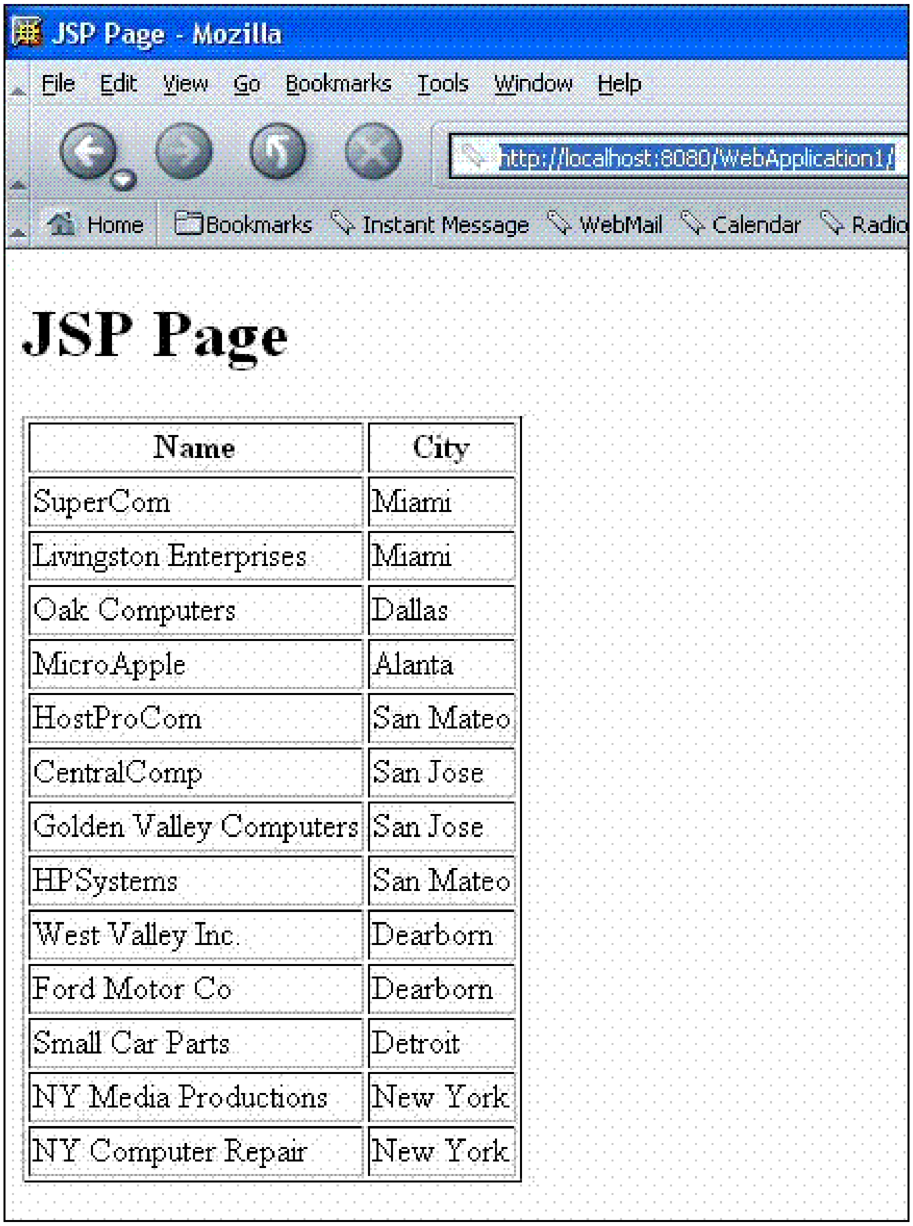
Next, you must enable the JSP file to select database records. Add this below the data source that you defined above:

```
    <sql:query var="queryresults" dataSource="${datasource}">
        SELECT * FROM CUSTOMER_TBL
    </sql:query>
```

Finally, to make the web application display the retrieved data in a table, add the following to the JSP page:

```
    <table border=1>
      <tr>
         <th>Name</th><th>City</th>
      </tr>
      <c:forEach var="row" items="${queryresults.rows}">
         <tr>
             <td><c:out value="${row.NAME}" /></td>
             <td><c:out value="${row.CITY}" /></td>
         </tr>
      </c:forEach>
    </table>
```

Right-click the project, choose Properties, click Run, and select the appropriate server for your database server. If your data is in a Pointbase database, choose the Sun Java System Application Server as your target server. If your data is in a MySQL database, choose the Tomcat Web Server. Then, when you run the project, the web application is deployed in the IDE's default browser and displays the data in a page similar to the illustration below:


JSP Page - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost:8080/WebApplication1/
Home
Bookmarks
Instant Message
WebMail
Calendar
Radio

JSP Page

Name	City
SuperCom	Miami
Livingston Enterprises	Miami
Oak Computers	Dallas
MicroApple	Alanta
HostProCom	San Mateo
CentralComp	San Jose
Golden Valley Computers	San Jose
HPSystems	San Mateo
West Valley Inc.	Dearborn
Ford Motor Co	Dearborn
Small Car Parts	Detroit
NY Media Productions	New York
NY Computer Repair	New York

Set Up a Database Connection Pool

A database connection pool is a group of reusable connections that a server maintains for a particular database. Web applications requesting a connection to a database obtain that connection from the pool. When an application closes a connection, the connection is returned to the pool. Connection pool properties may vary with different database vendors. Some common properties are the URL for the database name, user name, and password.

The first step in working with a database connection pool is to create a JDBC resource (also called a data source). A JDBC resource provides applications with a connection to a database. Typically, there is at least one JDBC resource for each database accessed by an application. It is possible to have more than one JDBC resource for a database. You can create a JDBC resource manually. For example, for Tomcat, you can use the `server.xml` file. However, be aware that you hand-edit files such as the `server.xml` file at your own risk; the IDE cannot repair it if it is damaged. Therefore, you are strongly encouraged to use the tools that the Sun Java System Application Server and the Tomcat Web Server provide for creating data sources. They can be accessed from the IDE. The usage of each is described below.

- **Connection Pooling with the Sun Java System Application Server.** To create a connection pool on the Sun Java System Application Server, you must first make sure that the server is registered in the IDE. If your version of the IDE is bundled with the Sun Java System Application Server, it is automatically registered in the IDE. If it is not bundled with the IDE, check that it is registered by going to the Runtime window, expanding the Servers node, and seeing whether there is a node for the Sun Java System Application Server. If the node is there, the server is registered in the IDE. If it is not there, go to Tools > Server Manager in the main menu and register the server.

Next, start the Pointbase database server. To do so, go to Tools > PointBase Database in the main menu and choose Start Local PointBase Database. You can view the databases available on the Pointbase database server by connecting to the database server from the IDE. To do so, go to the Runtime window, expand the Databases node, right-click the jdbc:pointbase://localhost:9092/sample node, and click connect. You can now expand the Tables node, right-click the CUSTOMER_TBL node, and choose View Data. Note the NAME and CITY fields, which you will work with in the examples that follow.

If you do not already have a web application, create one. Do this by choosing File > New Project (Ctrl-Shift-N). Once you have a web application, right-click the project node, choose New > File/Folder > Sun Resources > JDBC Resource. Click Next, click "Create New JDBC Connection Pool", type `jdbc/poolDB` in the JNDI field, click Next and Next again. In the Choose Database Connection page, click "Extract from Existing Connection", select the jdbc:pointbase://localhost:9092/sample, click Next and then Finish. You now have a connection pool with a data source called `jdbc/poolDB`.

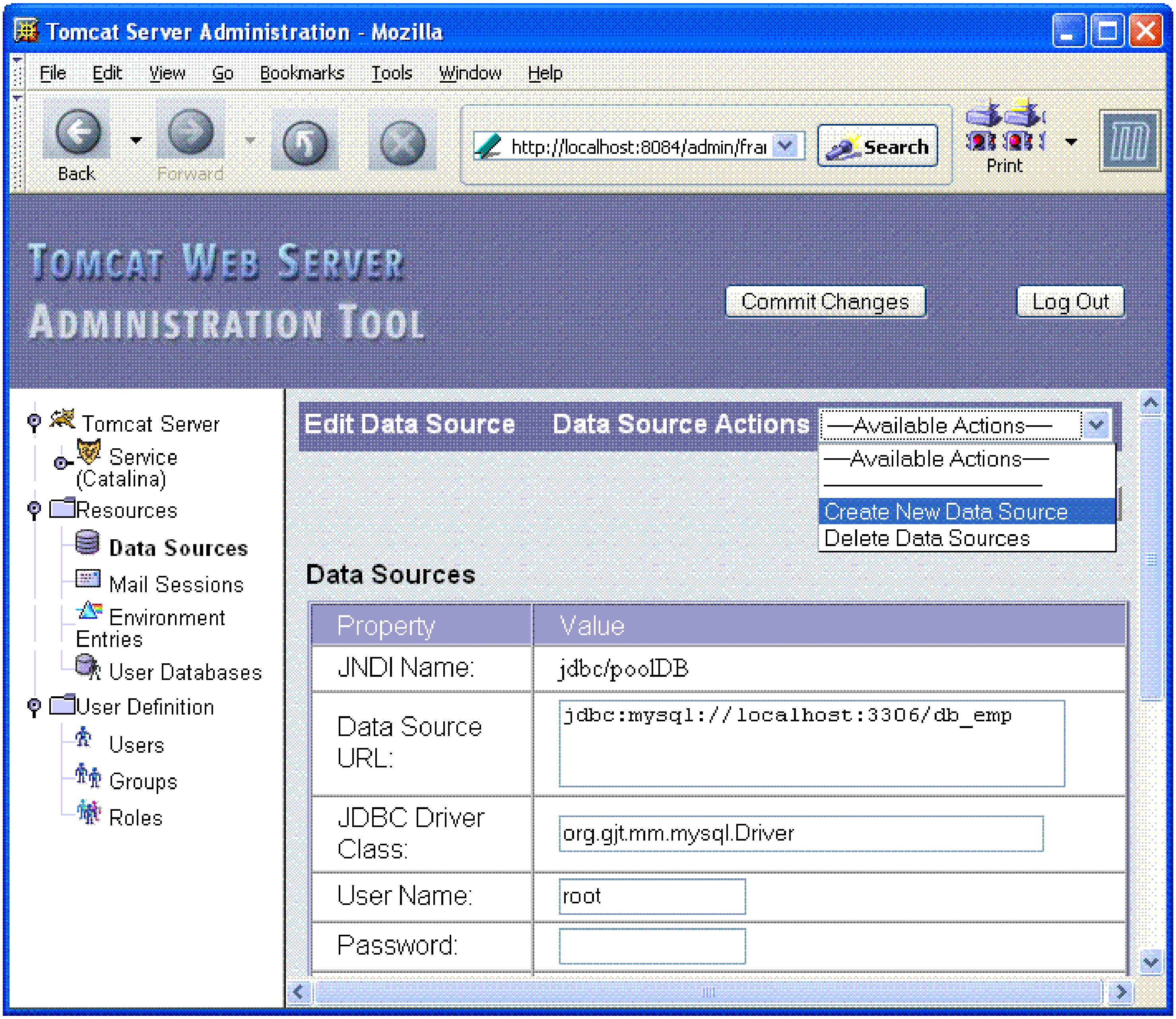
Start the Sun Java System Application Server 8.1 by right-clicking the server in the Runtime window and choosing Start/Stop Server. Now bo back to the Projects window, expand the Server Resources node, right-click the "jdbc/poolDB" and "connectionPool" nodes separately and choose Register for each. Your connection pool and data source are now registered with the server and ready to be accessed by your application.

- **Conection Pooling with the Tomcat Web Server.** The Tomcat Admin tool is bundled with the IDE. You can access it from the IDE in the Runtime window by expanding the Server Registry node and the Tomcat 5 Servers node. If the text to the right of the Tomcat instance node does not end with "[running]", start the server by right-clicking the Tomcat instance node, choosing Start/Stop Server, and clicking Start Server. When the server is running, expand the Web Applications node. Right-click the node for the `/admin` context and choose Open in Browser. The Tomcat Server Administration Tool opens in the IDE's default web browser.

  Type a User Name and Password for a Tomcat user to whom the "admin" role has been assigned. If the Admin tool does not open when you click Login, or if you don't have a user name or password, check the `tomcat-users.xml` file in Tomcat's base directory. To identify the location of this directory, right-click the Tomcat server instance node in the Runtime window and select Properties. In the Properties dialog box, the Base Directory property points to Tomcat's base directory. In the `tomcat-users.xml` file in Tomcat's base directory, check that a role called "admin" exists and that the role is assigned to a user. If necessary, define the "admin" role, assign the "admin" role to a user, save the file, and, using the Runtime window, stop and restart the Tomcat instance.

  In the Admin tool, click Data Sources. Choose Create New Data Source from the Data Source Actions combo box on the right. Type values such as the following to define your data source:

  - JNDI Name: `jdbc/poolDB`
  - Data Source URL: `jdbc:mysql://localhost:3306/db_emp`
  - JDBC Driver Class: `org.gjt.mm.mysql.Driver`
  - User Name: `root`
  - Password:



Data Sources

Property	Value
JNDI Name:	jdbc/poolDB
Data Source URL:	jdbc:mysql://localhost:3306/db_emp
JDBC Driver Class:	org.gjt.mm.mysql.Driver
User Name:	root
Password:	

Click Save. Then click Commit Changes. Then click Log Out. Your connection pool and data source are now registered with the server and ready to be accessed by your application.

Access a Database Connection Pool

- **Test the Data Source from a JSP page.** First, add the bundled JSTL 1.1 library to your web project's compilation classpath. Do this by right-clicking the project's Libraries node in the Projects window and then choosing Add Library. Select the JSTL 1.1 library and click Add Library. Right-click the Libraries node again and choose Add JAR/Folder. When you expand the Libraries node, you should now see two new nodes: one for the JSTL library's `standard.jar` file and another for the JSTL library's `jstl.jar` file. Next, you need to let the Tomcat web server know that the JSP page uses tags from JSTL's CORE component and JSTL's SQL component. To do so, add the following `taglib` directives after the `page` directives, which are at the top of your JSP file:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

Now modify the JSP file to enable the selection of database records via the connection pool. In the code below, make sure to specify the same JNDI name for the data source as the one used to name the data source above:

```
<sql:query var="queryresults" dataSource="jdbc/poolDB">
  SELECT * FROM CUSTOMER_TBL
</sql:query>
```

Next, to make the web application display the retrieved data in a table, add the following to the JSP file:

```
<table border=1>
  <tr>
     <th>Name</th><th>City</th>
  </tr>

  <c:forEach var="row" items="${queryresults.rows}">
     <tr>
         <td><c:out value="${row.NAME}" /></td>
         <td><c:out value="${row.CITY}" /></td>
     </tr>

  </c:forEach>
</table>
```

Finally, you need to provide deployment information to the server, using the web application's deployment descriptors. For both the Sun Application Server and the Tomcat Web Server, add the following to the `web.xml` file:

```
<resource-ref>
  <description>jdbc:pointbase://localhost:9092/sample [pbpublic on PBPUBLIC]</description>
  <res-ref-name>jdbc/poolDB</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
  <res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
```

Next, depending on the server, you need to provide server-specific information in the server's server-specific deployment descriptor:

- **Sun Java System Application Server.** When you create and save the `<resource-ref>` in the `web.xml`, as described above, the related `<resource-ref>` is automatically created for you in the `WEB-INF/sun-web.xml` file that the IDE creates for you when you choose the Sun Java System Application Server as your target server, either in the New Project wizard or in the Project Properties dialog box:

  ```
  <resource-ref>
    <res-ref-name>jdbc/poolDB</res-ref-name>
    <jndi-name>jdbc/poolDB</jndi-name>
  </resource-ref>
  ```

- **Tomcat Web Server.** Reference the data source in your application's `context.xml` file. This file is the Tomcat Web Server's server-specific file, just as the `sun-web.xml` file is the Sun Java System Application Server's server-specific file. The IDE creates the `context.xml` file for you when you make the Tomcat Web Server the application's target server, either in the New Project wizard or in the Project Properties dialog box. To access and change the `context.xml` file, expand the web project's META-INF node. Right-click the `context.xml` node, choose Edit from the contextual menu and use the Source Editor to add the following resource link between the `<context>` tags in the `context.xml` file:

  ```
     <ResourceLink name="jdbc/poolDB" type="javax.sql.DataSource" global="jdbc/poolDB"/>
  ```

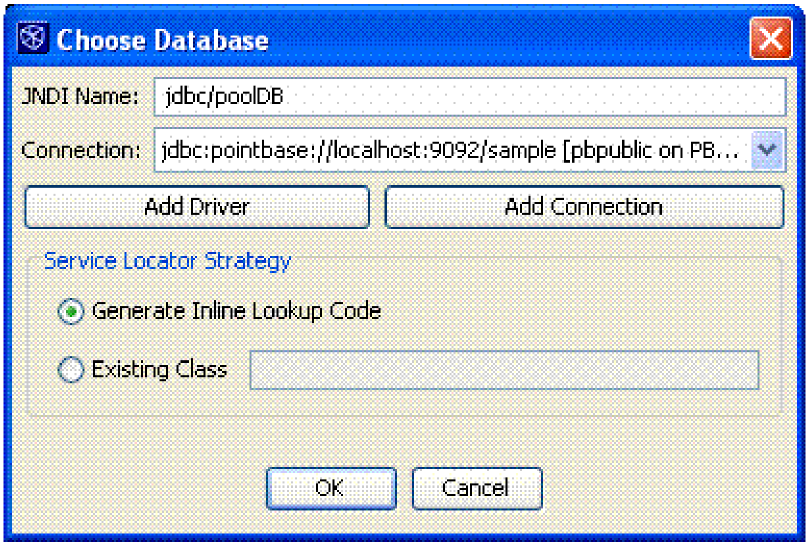
  Note: Do not double-click the `context.xml` file. If you do so, the IDE opens the `context.xml` file in the Context Editor instead of the Source Editor. You cannot add a resource link in the Context Editor. As you cannot open the `context.xml` file in both the Source Editor and the Context Editor at the same time, the IDE disables Edit in the contextual menu if the `context.xml` file is opened in the Context Editor.

  Your web application's `context.xml` file should now look similar to the following:

```
  <Context path="/MyWebApp">
      <ResourceLink name="jdbc/poolDB" type="javax.sql.DataSource" global="jdbc/poolDB"/>
      <Logger className="org.apache.catalina.logger.FileLogger" prefix="MyWebApp" suffix=".log"
timestamp="true"/>
  </Context>
```

In addition, you need to make the database driver's JAR file available for connection pooling. If you are using the version of Pointbase that is bundled with the Sun Java System Application Server, you do not need to do anything. But, if you are using MySQL with the Tomcat Web Server, for example, copy the database driver's JAR file into the server's `lib` folder. This folder is found within the Jakarta subfolder's `common/lib` folder, which is within the IDE's installation folder. If you have already started the server, make sure that you restart it after you copy the database driver's JAR file, so that the server can load the JAR file.

- **Use a Java Class to Access the Data Source.** The IDE can generate the code that establishes the connection to the data source. To generate this code, open a Java source file in the Source Editor, right-click anywhere in the body of the class, and choose Enterprise Resources > Use Database. In the Choose Database dialog box, type the `jdbc/poolDB` JNDI name for the data source and specify the connection string applicable to your database server. For example, choose the following for Pointbase:



JNDI Name: jdbc/poolDB

Connection: jdbc:pointbase://localhost:9092/sample [pbpublic on PB...

● Generate Inline Lookup Code

○ Existing Class

When you click OK, the following code is generated:

```
    private javax.sql.DataSource getPoolDB() throws javax.naming.NamingException {
        javax.naming.Context c = new javax.naming.InitialContext();
        return (javax.sql.DataSource) c.lookup("java:comp/env/jdbc/poolDB");
    }
```

In addition, the IDE adds resource references to your application's deployment descriptors:

  - **In the `web.xml` file:**

```
  <resource-ref>
    <description>jdbc:pointbase://localhost:9092/sample [pbpublic on PBPUBLIC]</description>
    <res-ref-name>jdbc/poolDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Shareable</res-sharing-scope>
  </resource-ref>
```

  - **In the `sun-web.xml` file:**

```
  <resource-ref>
    <res-ref-name>jdbc/poolDB</res-ref-name>
  <jndi-name>jdbc/poolDB</jndi-name>
</resource-ref>
```

Now that this code has been generated for you, you need to use it. You could, for example, generate the database connection code as described above and then implement it as shown in the Java class below. (The generated database connection code is shown in red.)

```
package org.me.customer;

public class CustomerDAO {

   private DataSource dataSource;

   /** Creates a new instance of CustomerDAO */
   public CustomerDAO() throws DAOException{
       try{
           dataSource = getPoolDB();
       }catch(NamingException e){
           throw new DAOException(e.getMessage());
       }
   }

   public Collection getAllRows() throws DAOException {
       ArrayList rows = new ArrayList();
       Connection conn = null;
       PreparedStatement prpStmt = null;
       ResultSet rs;
       try{
           conn = dataSource.getConnection();
           String sqlQuery = "SELECT * FROM CUSTOMER_TBL";
           prpStmt = conn.prepareStatement(sqlQuery);
           rs = prpStmt.executeQuery();
           while(rs.next()){

               rows.add(new Row(rs.getString(4), rs.getString(7)));
           }
       }catch(SQLException e){
           throw new DAOException(e.getMessage());
       }finally{
           try{
               conn.close();
               prpStmt.close();
           }catch(SQLException e){}
       }
       return rows;
   }

   private javax.sql.DataSource getPoolDB() throws javax.naming.NamingException {
       javax.naming.Context c = new javax.naming.InitialContext();
       return (javax.sql.DataSource) c.lookup("java:comp/env/jdbc/poolDB");
   }
}
```

To use the Java class above, you would need to create a Java class called Row that defines the fields name and city, together with getter and setter methods, as well as a DAOException class for DAO exceptions. To generate the applicable import statements, right-click anywhere in the Source Editor and choose Fix Imports (Alt-Shift-F). (The completed project is attached to the top of this file for your convenience.)

Next, to access and display the data, create a JSP file and add the following near the top to access the CustomerDAO class:

```
  <jsp:useBean id="queryresults" class="org.me.customer.CustomerDAO" scope="page" />
```

Then, add the bundled JSTL 1.1 library to your web project's compilation classpath. Do this by right-clicking the project's Libraries node in the Projects window and then choosing Add Library. Select the JSTL 1.1 library and click Add

Library. Right-click the Libraries node again and choose Add JAR/Folder. When you expand the Libraries node, you should now see two new nodes: one for the JSTL library's standard.jar file and another for the JSTL library's jstl.jar file.

Add the following near the top of the JSP file:

```
   <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

Finally, to make the web application display the retrieved data in a table, add the following to the JSP file:

```
    <table border=1>

      <tr>
         <th>Name</th><th>City</th>
      </tr>

      <c:forEach var="row" items="${queryresults.allRows}">
         <tr>
             <td><c:out value="${row.name}" /></td>
             <td><c:out value="${row.city}" /></td>
        </tr>
      </c:forEach>

    </table>
```

When you run the project, the application is deployed in the IDE's default browser and the JSP page is displayed in the same way as in the previous example.

- **Use Enterprise Beans to Access the Data Source.** Accessing a data source from an enterprise application is described in detail in the NetBeans IDE 4.1 Quick Start Guide for J2EE Applications.

Troubleshooting

- Error message:
  `javax.servlet.ServletException: Unable to get connection, DataSource invalid: "org.apache.commons.dbcp.SQLNestedException: Cannot create JDBC driver of class '' for connect URL 'null'"`

  Check the following:

  - Did you correctly add your data source to the `server.xml` file? For example, if you did not click Commit Changes in the Tomcat Admin tool, the data source is saved in Tomcat's Admin tool, but not in the `server.xml` file. Open the `server.xml` file and check that it includes the data source you defined in Tomcat's Admin tool.
  - Did you correctly add a resource link in the `context.xml` file? Without a resource link, your web application cannot find your data source, which is in your `server.xml` file. Check your `context.xml` file.
  - Does your resource link reference the data source correctly? Check that the resource link has the same name as the data source. To do so, open your `context.xml` file and your `server.xml` file and make sure that the name of the resource link is the same as the name of your data source.
- Error message:
  `javax.servlet.ServletException: Unable to get connection, DataSource invalid: "org.apache.commons.dbcp.SQLNestedException: Cannot load JDBC driver class 'org.gjt.mm.mysql.Driver'"`

  Check the following:

  - Is your database driver's JAR file in Tomcat's `common/lib` folder? If your database driver is not in Tomcat's `common/lib` folder, your Tomcat connection pool is unable find it and your web application is unable to connect to the database.
  - Did you stop and restart the Tomcat Server after you copied the database driver's JAR file in in Tomcat's `common/lib` directory? If you did not restart the Tomcat Server, the Tomcat Server is unable to load your database driver's JAR file.

Using Ant Scripts in the IDE

Apache Ant is a Java-based build tool used to standardize and automate build and run environments for development. Ant build scripts are XML files that contain targets, which in turn contain tasks. Ant tasks are executable bits of code that handle the processing instructions for your source code. For example, you use the `javac` task to compile code, the `java` task to execute a class, and so forth. You can use Ant's built-in tasks, use tasks written by third parties, or write your own Ant tasks.

If you are looking for resources on learning Ant, see http://ant.apache.org/resources.html.

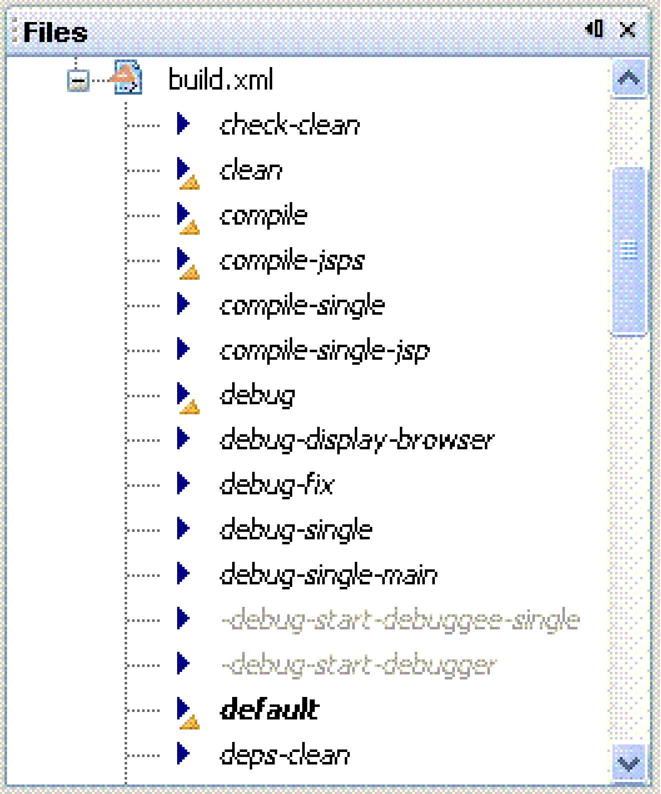
The IDE's project system is built directly on top of Ant. All of the project commands, like Build Project or Run File in Debugger, call targets in the project's Ant script. You can therefore build and run your project outside the IDE exactly as it is built and run inside the IDE.

You do not need to know Ant to work with the IDE. You can set all the basic compilation and runtime options in the project's Project Properties dialog box and the IDE automatically updates your project's Ant script. If you know how to work with Ant, you can customize a standard project's Ant script or write your own Ant script for your project.

This section covers the following topics:

Ant Scripts in the IDE

The IDE automatically recognizes Ant scripts and displays them as Ant script nodes ( ) rather than as normal XML files. As with all file types in the IDE, you can right-click Ant scripts in the Projects window, Files window, or Favorites window to access a contextual menu of commands. You can also expand the Ant script node to see an alphabetical list of subnodes representing the Ant script's targets. Each of these subnodes also has a contextual menu of commands.



In the Projects, Files, and Favorites windows, an Ant script's subnodes are flagged in the following ways:

- An emphasized target. These targets include a description attribute, which is displayed as a tooltip. You define the target's description attribute in the Source Editor.
- A target without a description attribute.

When you create a target that you want to run from the command line, give the target a description attribute. Then, if you forget the names of the targets or what they do, you can run the `ant -projecthelp <script>` command from the command line. With this command, Ant lists only those targets that have a description attribute, together with their descriptions. Especially when there are many targets in your Ant build script, emphasizing some and de-emphasizing others can be a useful way to distinguish between those that you use a lot and those that you use less often.

The font style of a subnode's label in the Projects, Files, and Favorites windows indicates the following:

- Normal
  A target that is defined within the current Ant script.
- *Italics*
  A target that is imported from another Ant script.
- Greyed out
  An internal target that cannot be run directly. Internal targets have names beginning with '-'.
- **Bold**
  The default target for the script, if there is one. The default target is declared as an attribute of the project, together with other project attributes, such as its name. You define the project's default attribute in the Source Editor.

Targets that are imported from another script but are *overridden* in the importing script are not listed. Only the overriding target is listed.

Build Files in Standard Projects

In standard projects, Ant scripts are stored in your project folder. The main Ant script for a standard project is `build.xml`. The IDE calls targets in `build.xml` whenever you run IDE commands. This file contains a single import statement that imports targets from `build-impl.xml`. In `build.xml`, you can override any of the targets from `build-impl.xml` or write new targets. In standard projects, `build-`

`impl.xml` is the Ant script that contains all of the instructions for building, running, and debugging the project. You should never edit this file. You can, however, open it to examine the Ant targets that are available to be overridden.

In free-form projects, there is no `build-impl.xml`. The IDE directly calls targets in the free-form project's Ant script.

Installing Ant Documentation

Even if you are an expert at using Ant, you probably still need to look in the Ant manual every once in a while. You can install the Ant manual directly in the IDE help system by going to the NetBeans Update Center and installing the Ant Documentation module. See Installing New Modules from the Update Center for more information on using the Update Center.

Editing an Ant Script

The easiest way to edit an Ant script is in the Source Editor. All of the normal XML search tools, selection tools, and keyboard shortcuts are available for editing Ant scripts. Double-click any of the Ant script's subnodes to jump to that target's location in the Source Editor. The IDE provides code completion for all standard Ant tasks. To enter an end tag for any empty beginning tag, type </.

Customizing Ant Scripts in Standard Projects

To edit an Ant script in standard projects, do any of the following:

- Add instructions to be processed before or after an Ant target is run. Each of the main targets in `build-impl.xml` also has a `-pre` and `-post` target that you can override in `build.xml`. For example, to get RMI working with regular projects, type the following in `build.xml`:

  ```
  <target name="-post-compile">
    <rmic base="${build.classes.dir}" includes="**/Remote*.class"/>
  </target>
  ```

- Change the instructions in an Ant target. Copy the target from `build-impl.xml` to `build.xml` and make any changes to the target.
- Create new targets in `build.xml`. You can also add the new target to the dependencies of any of the IDE's existing targets. Override the existing target in `build.xml` then add the new target to the existing target's `depends` property. For example, the following adds the `new-target` target to the `run` target's dependencies:

  ```
  <target name="new-target">
      <!-- target body... -->
  </new-target>

  <target name="run" depends="new-target,myprojname-impl.run"/>
  ```

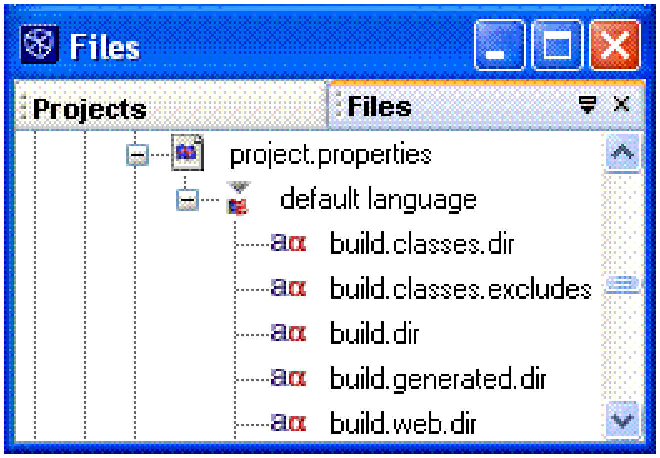
  Notice that you do not need to copy the body of the `run` target into `build.xml`.

Note that there is no `build-impl.xml` in free-form projects. In these projects, the IDE directly calls targets in the project's Ant script.

Editing an Ant Script's Properties

If you are working in a team, it is important that all of your build scripts be portable and usable on each developer's computer. Local details like the location of various libraries and executables can vary from computer to computer. One way to deal with this is to define these locations as properties.

Customizing Property Files in Standard Projects

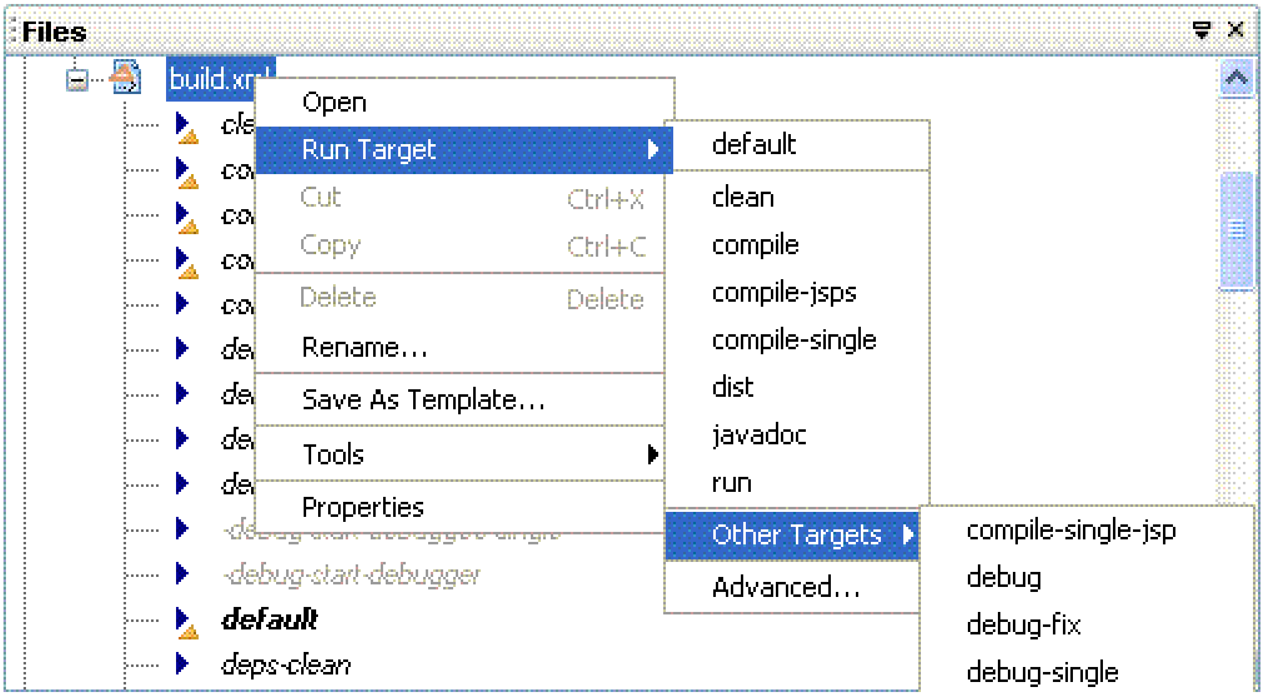


For standard projects, the following property files are available in your project folder:

- **`nbproject/project.properties`**. The Ant properties file that contains important information about your project, such as the location of your source and output folders. You can override the properties in this file. Be careful when editing this file. For example, the output folder is deleted every time you clean your project. You should therefore never set the output folder to the same location as your source folder without first configuring the `clean` target to not delete the output folder.
- **`nbproject/private/private.properties`**. The Ant properties file that holds properties that are defined for you only and are specific to your installation of the IDE. For example, if you are sharing the project over VCS, any properties you set in `private.properties` are not checked into the repository. You can copy a property from `project.properties` into `private.properties` and give the property different definitions in each file. The definitions in `private.properties` take precedence over those in `project.properties`.
- **`nbproject/genfiles.properties`**. An IDE-generated metadata file. You should never edit this file.

Note that you create your own property files in free-form projects.

Running an Ant Script



You can run targets in an Ant script from the Ant script's node in the Projects window, Files window, or Favorites window. To do so, right-click the Ant script node and choose a target from the Run Target submenu. Targets are sorted alphabetically. Only emphasized targets are listed. Choose Other Targets to run a target that has not been emphasized with a description attribute. Internal targets are excluded from these lists because they cannot be run independently.

You can specify runtime parameters for each target. Choose Advanced from the Ant script node's Run Target submenu, select a target to run, and specify its properties. Specify properties as name-value pairs to pass to Ant, one pair per line. For example:

```
build.compiler.emacs=true
```

Initially, only the global properties that are defined under the Ant Settings in the Options dialog are shown. You can also change the verbosity level, which is defined globally under the Ant Settings in the Options dialog. The changes you make to the parameters and verbosity level are saved in your user directory. If you open the advanced dialog again in subsequent IDE sessions, the previously used changes are used to initialize the fields. This makes it easy to test targets with particular properties several times in a row without needing to type the same values each time.

Instead of running a target by using the Ant script node's contextual menu, you can simply right-click the target's node and choose Run Target.

Writing Custom Ant Targets

In standard projects, you may want to extend your application's functionality by writing new Ant targets or by customizing the Ant targets provided by the IDE. In a free-form project, however, you have to write all Ant targets yourself. For example, you have to write an Ant target to run a project in the debugger. An example of a debug target for a J2SE application is as follows:

```
<target name="debug" depends="compile" description="Debug Project">
  <fail unless="netbeans.home">This target can only run inside the NetBeans IDE.</fail>
  <nbjpdastart name="My App" addressproperty="jpda.address" transport="dt_socket">
    <classpath path="run.classpath"/>
    <!-- Optional - If source roots are properly declared in project, should
                work without setting source path.
    <sourcepath path="debug.sourcepath"/> -->
  </nbjpdastart>
  <java fork="true" classname="com.me.myapp.Main">
    <jvmarg value="-Xdebug"/>
    <jvmarg value="-Xnoagent"/>
    <jvmarg value="-Djava.compiler=none"/>
    <jvmarg value="-Xrunjdwp:transport=dt_socket,address=${jpda.address}"/>
    <classpath refid="run.classpath"/>
  </java>
</target>
```

This Ant target does the following:

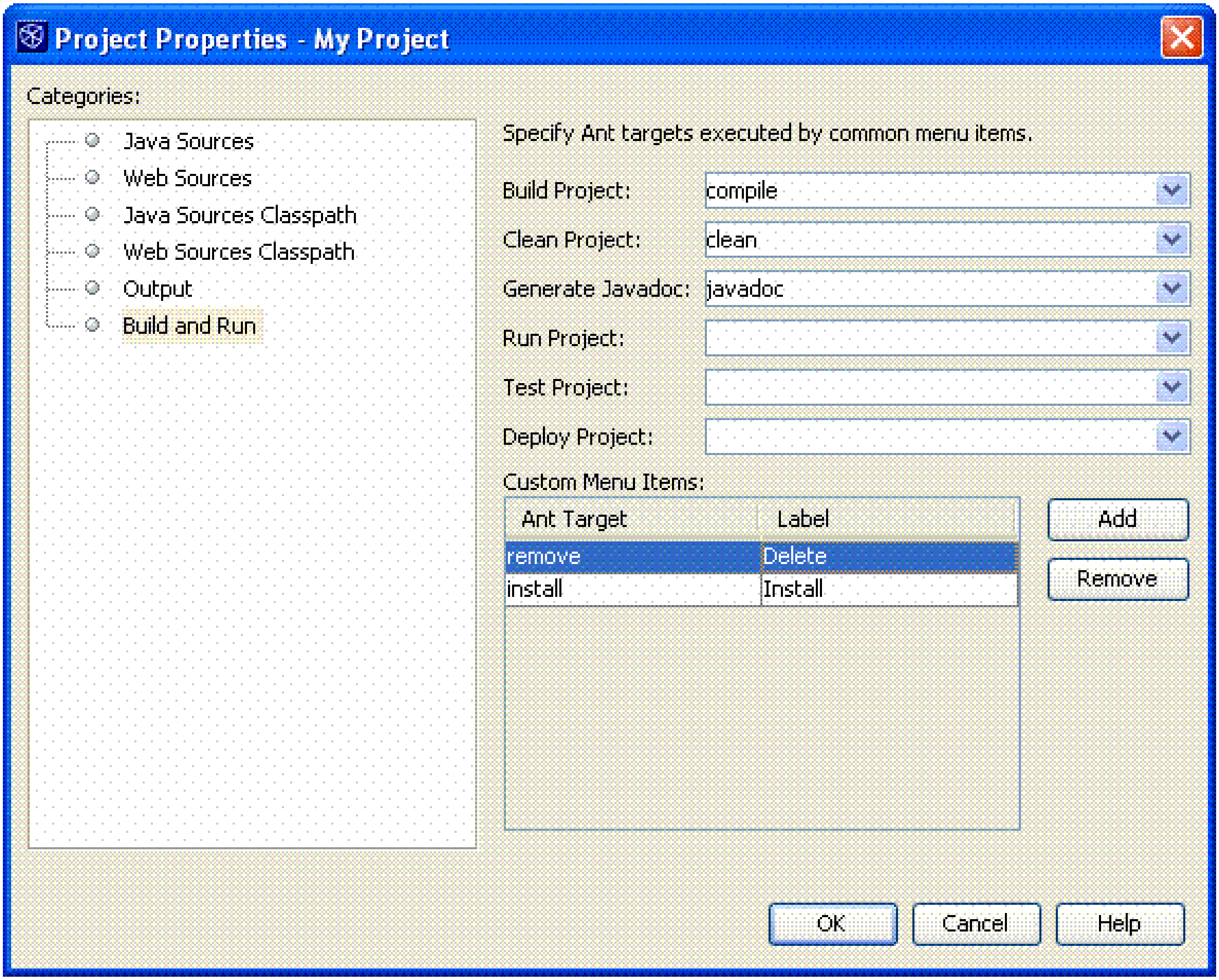
- Checks whether Ant is running inside the IDE (`fail unless="netbeans.home"`).
- Starts the debugger with the `nbjpdastart` task.
- Stores the address at which the debugger listens for the application in the `jpda.address` property (`addressproperty="jpda.address"`). You do not have to define the `jpda.address` property in your Ant script or properties file. It is defined by the IDE.
- Sets the runtime classpath with `run.classpath`. You should change the classpath definition to use the same classpath that is used for execution.
- Runs the application in debug mode, passing the `jpda.address` property as the address at which to connect to the debugger.

For examples of other targets, see debug a web application and compile/run/debug a single file.

Mapping Custom Ant Targets to Project Commands

In free-form projects, you map IDE commands to targets in your Ant script. By doing so, you can, for example, add an item for a Debug Project target to the project's contextual menu.

Click Build and Run in the left panel of the Project Properties dialog box.



Specify Ant targets executed by common menu items.

Build Project: compile

Clean Project: clean

Generate Javadoc: javadoc

Run Project:

Test Project:

Deploy Project:

Custom Menu Items:

Ant Target	Label
remove	Delete
install	Install

For each command, choose an Ant target from the drop-down. The drop-down contains each of the targets in your Ant script. However, if your Ant script uses an `<import>` statement to import targets from another Ant script, the targets do not show up in the drop-down list in the Project Properties dialog box. To map commands to these targets, type the names of the targets into the list.

You can also add a shortcut to an Ant target by right-clicking a target's node in the Projects window, Files window, or Favorites window, and choosing Create Shortcut from the contextual menu. Use the wizard to create a shortcut to the target. For example, if you have several runnable classes that you need to run often, you can write targets to run these classes and then run them using a shortcut.

You map other commands, like those that run on individual files, by editing the `project.xml` file manually. For details, see Advanced Free-form Project Configuration.

Writing Custom Ant Tasks

You can use custom Ant tasks to expand on the functionality provided by Ant's built-in tasks. Custom tasks are often used to define properties, create nested elements, or write text directly between tags using the `addText` method.

To create a custom Ant task in the IDE, press Ctrl-N and select the Custom Task template from the Ant Build Scripts folder. The Custom Task contains sample code for many of the common operations performed by Ant tasks. After each section of code, the template also shows you how to use the task in an Ant script.

For example, let's make our custom task create a simple option and a simple property and echo their values. Remove the comments from the following lines of code in MyTask:

```
private boolean opt;
public void setOpt(Boolean b) {
    opt = b;
}

private String myprop;
public void setMyprop(String s) {
    myprop = s;
}
```

Then go to the execute method and enter the following:

```
log("myprop=" + myprop + " opt=" + opt);
```

Now let's add a target to our `build.xml` script that give values to the option and property in `MyTask.java`. Add the following to `build.xml`:

```
<target name="mytask">
    <javac classpath="${ant.location}" destdir="build/taskclasses" srcdir="tasksource"/>
    <taskdef classname="examples.MyTask" classpath="." name="mytask"/>
    <mytask myprop="hello" opt="true"/>
</target>
```

You should define the `ant.location` property with the path to your Ant JAR, either in the build script itself or in a `properties` file. You can also enter the path to the Ant JAR directly in the `javac` task. Then expand the `build.xml` node in the Projects window, Files window, or Favorites window, right-click the `mytask` target, and choose Run Target from the contextual menu. The target is run and the Output window displays the values of `myprop` and `opt`.

Including Custom Ant Tasks in an Ant Script's Classpath

By default, the IDE ignores your environment's `CLASSPATH` variable whenever it runs Ant. For your Ant script to use customs tasks, you must add the tasks to the Ant's classpath in the IDE.

You can add custom tasks to Ant's classpath within the IDE by:

- Providing an explicit classpath to the tasks in your build script. This is the recommended method.
- Configuring the Additional Classpath property under Ant Settings in the Options window.

Build Scripts With an Explicit Classpath

Using an explicit classpath is the recommended method, as it ensures that your build scripts will be fully portable. You can write your tasks and include instructions to compile them and produce a JAR file in the build file. To use these tasks, include the long form of `taskdef`, which includes a classpath. Here is a simple example of such a task:

```
<project name="test" default="all" basedir=".">
    <target name="init">
        <javac srcdir="tasksource" destdir="build/taskclasses"/>
        <jar jarfile="mytasks.jar">
            <fileset dir="build/taskclasses"/>
        </jar>
        <taskdef name="customtask" classname="com.mycom.MyCustomTask">
            <classpath>
                 <pathelement location="mytasks.jar"/>
            </classpath>
        </taskdef>
    </target>
</project>
```

The advantage of this method is that no special preparation is needed to begin using the script. The script is entirely self-contained and portable. This method also makes it easier to develop your tasks within the IDE, as the script compiles them for you automatically.

To make the your build scripts even more robust, use a property instead of a hard-coded location to specify the classpath to your tasks. You can store the property in the build script itself or in a separate `ant.properties` file. You can then change the classpath setting throughout your script by simply changing the value of the specified property.

Adding Binaries to an Ant Script's Classpath in the IDE

If you cannot declare a classpath in your build script, or you are using third-party build scripts which you cannot alter, you can add the tasks to Ant's classpath in the IDE. Open the Options window, select Building > Ant Settings, and type the full path to the binary in the Additional Classpath property.

Configuring the IDE

One of the main strengths of the IDE is its versatile configurability. You can customize your working environment to fit your needs and personal development style. Having all of these options can, however, make it difficult to find the exact setting you are looking for.

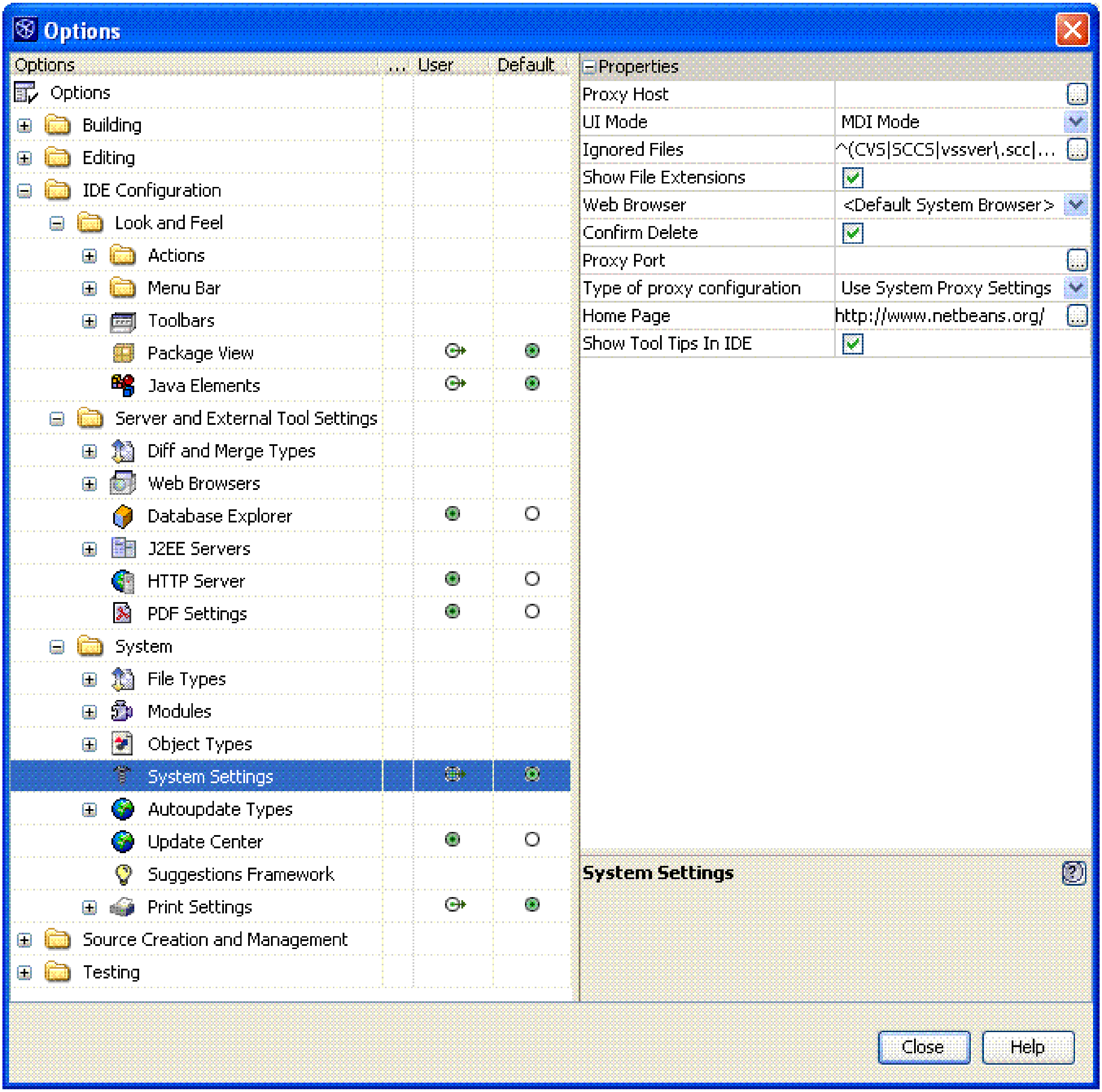
In this section you will learn about the following:

- Setting IDE default settings
    - Configuring IDE Startup Switches
    - Configuring General Java Settings
    - Working With File Types
- Enabling and disabling functionality
    - Disabling Modules
    - Installing New Modules from the Update Center
- Boosting NetBeans performance
    - Tuning JVM Switches for Performance

To configure project-level settings such as a project's properties and JDK level, see Setting Up Projects in the IDE.

Setting IDE Default Settings

The main tool for configuring default settings in the IDE is the Options window. You can open the Options window by choosing Tools > Options.



Options	User	Default
Options		
⊞ Building		
⊞ Editing		
⊟ IDE Configuration		
⊟ Look and Feel		
⊞ Actions		
⊞ Menu Bar		
⊞ Toolbars		
Package View	→	●
Java Elements	→	●
⊟ Server and External Tool Settings		
⊞ Diff and Merge Types		
⊞ Web Browsers		
Database Explorer	●	○
⊞ J2EE Servers		
HTTP Server	●	○
PDF Settings	●	○
⊟ System		
⊞ File Types		
⊞ Modules		
⊞ Object Types		
System Settings	→	●
⊞ Autoupdate Types		
Update Center	●	○
Suggestions Framework		
⊞ Print Settings	→	●
⊞ Source Creation and Management		
⊞ Testing		

⊟ Properties

Proxy Host	
UI Mode	MDI Mode
Ignored Files	^(CVS|SCCS|vssver\.scc|...
Show File Extensions	☑
Web Browser	<Default System Browser>
Confirm Delete	☑
Proxy Port	
Type of proxy configuration	Use System Proxy Settings
Home Page	http://www.netbeans.org/
Show Tool Tips In IDE	☑

System Settings

IDE settings are grouped under Options window nodes in the left panel of the window. The left panel also indicates whether the settings are stored at the user level or if they are default IDE settings. When you select a node in the left panel of the window, the properties you can modify are displayed in the right panel. As with regular property sheets, many properties in the Options window contain special property editors. An ellipsis button indicates that a property editor is available for a property, as in the Ignored Files property in the figure above.

You can revert a node's settings to the IDE's defaults by clicking the node's cell in the Default column, and choosing Revert Def. If the User and Default columns are not visible, click the << at the top of the Options window or press + to reveal (or hide) the columns.

Configuring IDE Startup Switches

Another tool for configuring the IDE is Java startup switches. You can add startup switches to the IDE on the command line or by entering them in a special file called netbeans.conf, which is located in the etc folder. You can enter IDE-specific startup switches and pass arguments directly to the JVM in which the IDE runs.

For example, to set the -Xmx (maximum heap size) for the JVM in which the IDE runs, either add the line -J-Xmx64m to your netbeans.conf file or launch the IDE from the command line by typing

```
./netbeans.sh -J-Xmx64m
```

on UNIX systems, or, on Windows systems

```
netbeans.exe -J-Xmx64m
```

The netbeans.conf file can have the various JVM switches separated either by spaces or on separate lines. Note that the JVM does not start when switches are passed that it does not understand. When this error occurs, the JVM returns a message pointing out the switch that caused the problem, as with the following example:

```
java -foo
Unrecognized option: -foo
Could not create the Java virtual machine.
```

The following table lists the available startup switches.

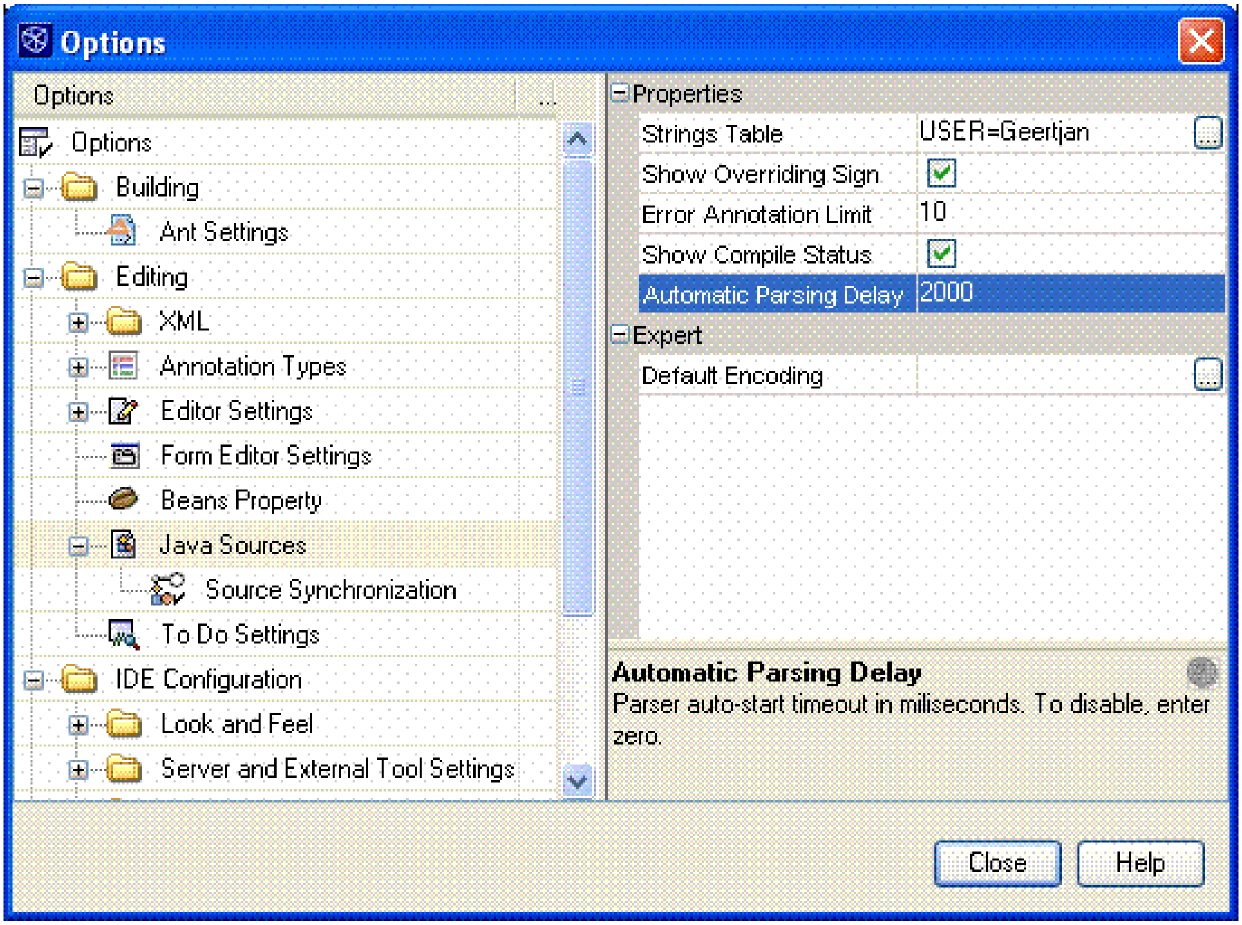
Startup Switch	**Description**
-h **--help**	Print descriptions of common startup parameters.
--cp:p *additional_classpath*	Prefix the specified classpath to the IDE's classpath.
--cp:a *additional_classpath*	Append the specified classpath to the IDE's classpath.
--fontsize *size*	Set the font size, expressed in points, in the IDE's user interface. If this option is not used, the font size is 11 points.
--jdkhome *jdk_home_dir*	Use the specified version of the Java™ 2 SDK instead of the default SDK. By default on Microsoft™ Windows systems, the loader looks into the registry and uses the latest SDK available. You should back up your user directory before you upgrade the SDK that the IDE uses. If you later need to revert to the previous JDK, switch to the backed up user directory to ensure that you do not lose any settings. To switch the IDE's user directory, use the -userdir switch detailed below.
-J*jvm_flag*	Pass the specified flag directly to the JVM.
-J-Dsun.java2d.noddraw=true	Prevent the use of DirectX for rendering. This switch might prevent problems that occur on some Microsoft Windows systems with faulty graphics cards.
-J-Dnetbeans.debugger.jpda.transport=dt_shmem *userdir*	Force the IDE to use the shared memory connection when starting a debugging session with the Debug > Start command. This parameter has no effect when you attach the debugger to an already running process.
-J-Dorg.netbeans.spi.java.project.support.ui.packageView.USE_TREE_VIEW=true	Display packages in the Project window in a directory-based view rather than a package-based view.
--laf *UI_class_name*	Selects the given class as the IDE's look and feel. The following are two examples of look and feel classes: • com.sun.java.swing.plaf.motif.MotifLookAndFeel • javax.swing.plaf.metal.MetalLookAndFeel
--locale *language[:country[:variant]]*	Use the specified locale.

--open *file*	Open the file in the Source Editor.
--open *file:line number*	Open the file in the Source Editor at the specified line.
--userdir *userdir*	Explicitly specify the userdir, which is the location in which user settings are stored. If this option is not used on a UNIX system, the location is the *HOME* directory by default. On Microsoft Windows systems, the default is the one you specified when you first launched the IDE. You can determine the current user directory in the About dialog box. Choose Help > About and then click the Detail Tab. The tab lists the location of the User Dir as well as other product details.

Configuring General Java Settings

The first thing you should do to configure the IDE is make sure it is using the correct Java Standard Development Kit (JDK) version. The JDK in which the IDE runs is important because it is automatically used as the platform against which all of your sources are compiled and executed. See Setting the Target JDK in a Project for details.

To see which JDK your IDE is running on, choose Help > About and click the Detail tab. The location of the JDK is listed as Java Home. By default, the IDE uses the JDK that is specified in your system's registry as the most recent. If you only have one version of the JDK installed on your computer, this is not a problem. If you have multiple JDK versions installed, it can be a good idea to explicitly specify which JDK the IDE should use. You can do so by using the `--jdkhome` *JDK folder* switch (for example, `--jdkhome c:\j2sdk1.5\`) on the command line or in your `netbeans.conf` file.



Properties	
Strings Table	USER=Geertjan
Show Overriding Sign	☑
Error Annotation Limit	10
Show Compile Status	☑
Automatic Parsing Delay	2000
Expert	
Default Encoding	

Automatic Parsing Delay
Parser auto-start timeout in miliseconds. To disable, enter zero.

Another important tool in defining Java settings is the Java Sources node in the Options window. The Java Sources node, which is located under the Editing node, contains general settings for how the IDE handles Java source files. The following settings are available in the Java Sources node:

- **Strings Table.** Specifies the _USER_ macro in templates and enables you to create your own macros for use in templates.
- **Error Annotation Limit.** Sets the amount of errors that are highlighted in the Source Editor for each open file. Set this property to zero to disable error annotation for Java files.
- **Show Compile Status.** If True, displays the compile status badge on the node for a Javaclass file when that file needs to be compiled.
- **Automatic Parsing Delay.** Specifies the time (in milliseconds) between a pause in typing or moving around in the Source Editor and the time that internal parsing information is refreshed. The default is two seconds. The IDE uses the internal parser to automatically update information about the current Java source file. You can disable automatic parsing by setting this property to zero. If this property is disabled, a Java file is parsed only when the file is saved or compiled.
- **Default Encoding.** Specifies the default encoding that the IDE uses to display and save `.java` files. Type an encoding name or leave blank to use your system's default encoding. This setting does not affect the encoding used to compile classes in the IDE.

Working With File Types

The IDE recognizes the standard file extensions for most types of files. For example, it knows that files with the extensions `.htm`, `.html`, and `.shtml` should all be treated as HTML files. Many file types, like XML, can have nonstandard file extensions that the IDE does not recognize.

If you want to treat all files with a certain file extension as a certain type of file, go to the Options window and expand IDE Configuration > System > Object Types. The Object Types node contains all of the file types that the IDE is currently configured to work with. You can use the Extensions and MIME Types property to specify which file extensions should be treated as a given type of file.

For example, JavaHelp™ map files are XML documents that have a `.jhm` extension. You can treat all JavaHelp map files as XML documents by adding `.jhm` to the list of Extensions and MIME Types for the XML Object object type.

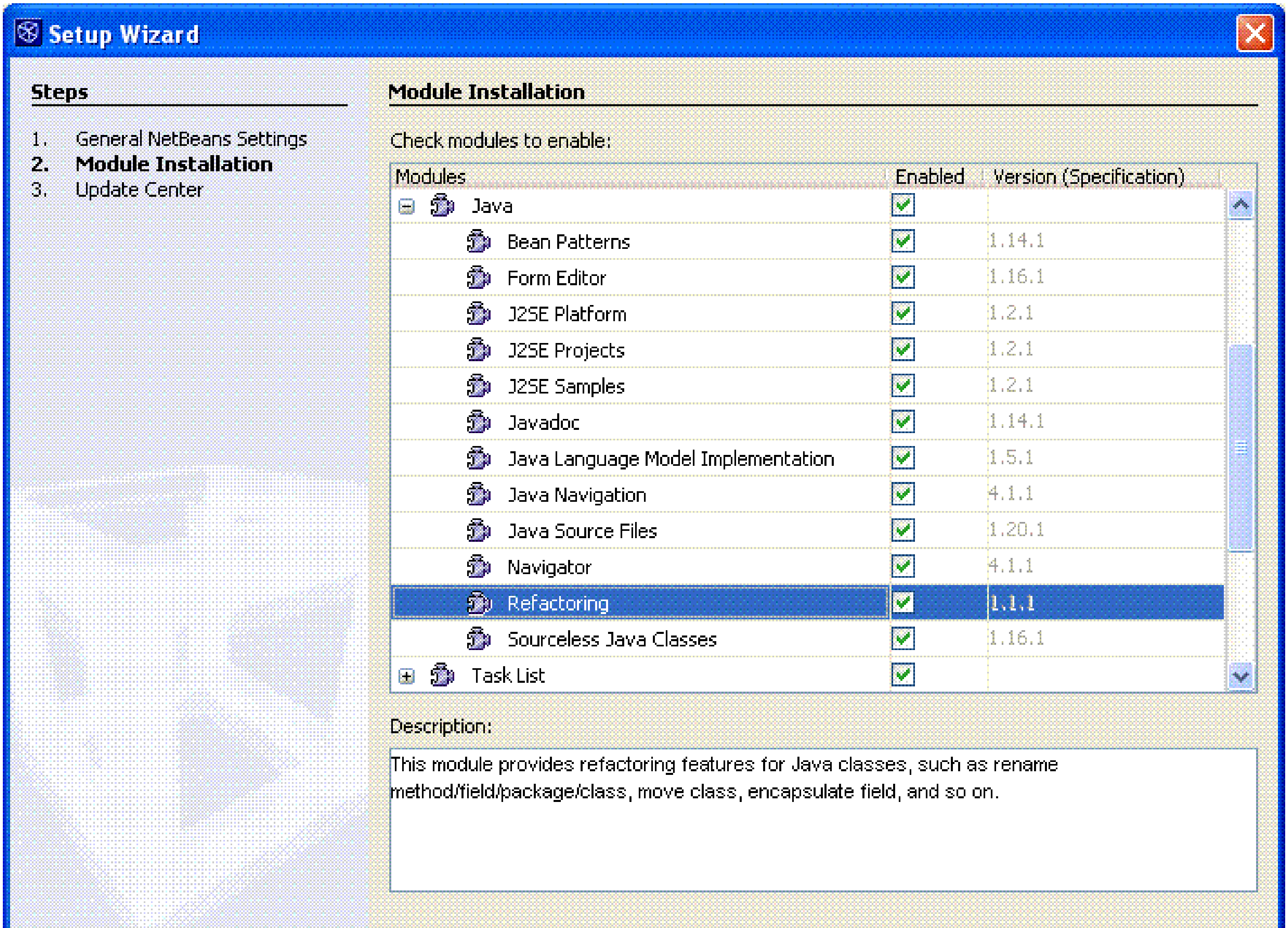
Enabling and Disabling IDE Functionality

NetBeans IDE is a fully modular IDE, meaning that its functionality is provided by modules that plug into the core NetBeans infrastructure. If you do not use the functionality provided by certain modules, you can turn those modules off. Turning off unused modules helps improve your IDE's startup time and performance. You can also add functionality to your IDE by downloading new modules from the Update Center.

Disabling Modules

The Setup wizard is the most convenient tool for enabling and disabling modules. Disabling a module only causes the IDE to ignore the module. The module is not deleted and it can be enabled again at any time.

To open the Setup wizard, choose Tools > Setup Wizard, then click Next to go to the Module Installation page of the wizard.



Setup Wizard

Steps

1. General NetBeans Settings
2. **Module Installation**
3. Update Center

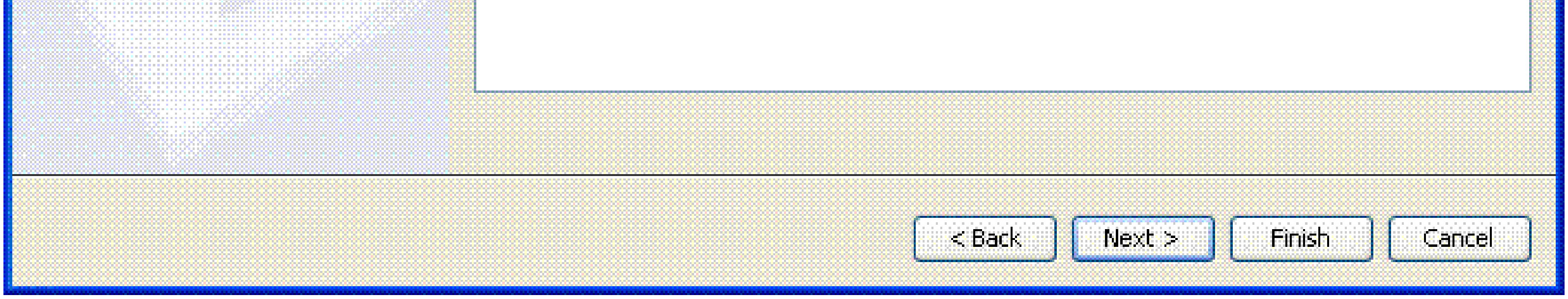
Module Installation

Check modules to enable:

Modules	Enabled	Version (Specification)
⊟ Java	☑	
Bean Patterns	☑	1.14.1
Form Editor	☑	1.16.1
J2SE Platform	☑	1.2.1
J2SE Projects	☑	1.2.1
J2SE Samples	☑	1.2.1
Javadoc	☑	1.14.1
Java Language Model Implementation	☑	1.5.1
Java Navigation	☑	4.1.1
Java Source Files	☑	1.20.1
Navigator	☑	4.1.1
Refactoring	☑	1.1.1
Sourceless Java Classes	☑	1.16.1
⊞ Task List	☑	

Description:

This module provides refactoring features for Java classes, such as rename method/field/package/class, move class, encapsulate field, and so on.



The Module Installation page groups related modules into module groups. For example, the Java group contains all of the modules that deal with Java development. You can disable all modules in a module group by unchecking the checkbox in its Enabled column, or expand the module group node to disable individual modules. When only certain modules in a module group are disabled, the Enabled column for the group is checked [boolean]. When you are finished, click Finish to activate your changes.

Certain modules depend on other modules to function properly. Disabling or enabling one of these modules may require you to also disable or enable the modules upon which it depends. If this is the case, the IDE displays a dialog that tells you which modules will also be disabled or enabled and asks for your confirmation.

Installing New Modules from the Update Center

You can add functionality to your IDE by downloading new modules from the NetBeans Update Center. To connect to the Update Center, choose Tools > Update Center from the main window. In the Update Center wizard, select the Update Centers that you want to connect to. Make sure that your proxy information is properly configured and that you can connect to the Internet. You can edit your proxy configuration using the Proxy Configuration button on the wizard page. Click Next when you are ready to proceed.

The second page of the wizard shows you all of the modules that are available on the Update Centers. The wizard only displays modules that are not already installed in your IDE or newer versions of modules that are already installed. Newer versions of modules that you already have are marked with an upward arrow. Select any module to see detailed information about the module, including a description, the version number of the module on the Update Center, and the version number of the module already installed on your system.

To schedule a module for installation, select it in the window on the left and click the right arrow button. To install an entire group of modules, select the module group node and click the double right arrow. When you are ready to proceed, click the Next button to view the modules' certificates and install the modules.

Boosting NetBeans Performance

You can monitor your IDE's performance with the Memory toolbar. To view the Memory toolbar, right-click anywhere in the toolbar area and select the Memory in the contextual menu. The toolbar has a slide that shows you how much of the IDE's memory is currently being used and how close it is to automatically performing garbage collection. You can manually initiate garbage collection by clicking the Memory toolbar button.

You can boost NetBeans performance by adjusting the JVM switches with which you start the IDE, as outlined in the next section.

Tuning JVM Switches for Performance

JVMs offer a variety of standard and non-standard switches that tune memory allocation and garbage collection behavior. Some of these settings can benefit the performance of the IDE.

Note that -X and especially -XX JVM switches are officially "unsupported" because they are often JVM or JVM-vendor specific. The switches discussed in this section are available for Sun Microsystems J2SE 1.4.2 and J2SE 1.5. Users of other JVM implementations may need to remove these switches in order to run the IDE.

The following settings should produce better-than-factory setting performance on most systems. With the exception of setting the "permanent area" size, these switches have been the defaults for the IDE for some time, and should already be present in your `netbeans.conf` file.

- **-J-Xverify:none** - This switch turns off Java bytecode verification, making classloading faster and eliminating the need for classes to be loaded during startup solely for the purposes of verification. This switch improves startup time, and there is no reason not to use it.
- **-J-Xms32m** - this setting tells the Java virtual machine to set its initial heap size to 32 megabytes. By telling the JVM how much memory it should initially allocate for the heap, we save it growing the heap as the IDE consumes more memory.
- **-J-Xmx128m** - this setting specifies the maximum amount of memory that the the Java virtual machine should use for the heap. Placing a hard upper limit on this number means that the Java process cannot consume more memory than physical RAM available. This limit can be raised on systems with more memory - the 128 megabyte setting helps to ensure that the IDE performs tolerably on 256Mb systems. ***Note:*** *Do not set this value to near or greater than the amount of physical RAM in your system or it will cause severe swapping during major collections.*
- **-J-XX:PermSize=20m** - this is a more exotic JVM switch, but one which also improves startup time. This setting sizes the "permanent area" of memory, where classes are kept. Since we know that all of IDE's classes take up a specific amount of memory, we give the JVM a hint as to how much memory it will need. This setting eliminates major garbage collection events during startup on many systems. Users of SunONE Studio or other IDEs that include more modules may want to set this number higher.

Listed below are some additional JVM switches which have either anecdotally or measurably impacted NetBeans performance on some, not all, systems. Your mileage may vary, but they may be worth a try.

- **-J-XX:CompileThreshold=100** - This switch will make startup time slower, by instructing HotSpot to compile many more methods down to native code sooner than it otherwise would. The reported result is snappier performance once the IDE is running, since more of the UI code will be compiled rather than interpreted. This value represents the number of times a method must be called before it will be compiled.
- **-J-XX:+UseConcMarkSweepGC -J-XX:+UseParNewGC** - Try these switches if you are having problems with intrusive garbage collection pauses. This switch causes the JVM to use different algorithms for major garbage collection events (also for minor collections, *if* run on a multiprocessor workstation), ones which do not "stop the world" for the entire garbage collection process. If you are using the PermSize switch, you should also add the line `-J-XX:+CMSClassUnloadingEnabled` to your `netbeans.conf` file so that class unloading is enabled (it isn't by default when using this collector). *Note: It is unclear as yet if this collector helps or hurts performance on uniprocessor machines.*
- **-J-XX:+UseParallelGC** - Some tests have shown that, at least on systems fairly well equipped with memory, the durations of minor garbage collections is halved when using this collection algorithm, on uniprocessor systems. Note that this is paradoxical - this collector is designed to work best on multiprocessor systems with gigabyte heaps. No data is available on its effect on major garbage collections. ***Note:*** *this collector is mutually exclusive with `-J-XX:+UseConcMarkSweepGC`* . The measurements supporting the use of this algorithm can be found on the performance web site.

Quick Reference

This section lists common tasks for application development, keyboard shortcuts for use in the IDE, and abbreviations for coding in the Source Editor. For more detailed information, click the links in the right column.

This section covers:

- Window Navigation Shortcuts
- Project Tasks
- VCS Tasks
- Configuring Tasks
- Source Editor Tasks
  - Abbreviations for Java Files
  - Abbreviations for JSP and Servlet Files
  - Abbreviations for XML and DTD Files
- Building Tasks
- Running J2SE Application Tasks
- Running Web Application Tasks
- Debugging Tasks
- JUnit Tasks

Window Navigation Shortcuts

Keys	**Action**
Ctrl-0	Switches focus to the Source Editor.
Ctrl-1/Ctrl-Shift-1	Switches focus to the Projects window.
Ctrl-2/Ctrl-Shift-2	Switches focus to the Files window.
Ctrl-3/Ctrl-Shift-3	Switches focus to the Favorites window.
Ctrl-4	Switches focus to the Output window.
Ctrl-5	Switches focus to the Runtime window.
Ctrl-6	Switches focus to the To Do window.
Ctrl-7	Switches focus to the Navigator window.
Ctrl-8/Ctrl-Shift-8	Opens the Versioning window.
Ctrl-9	Opens the VCS Output window.
Alt-Shift-1	Opens the Local Variables debugger window.

Alt-Shift-2	Opens the Watches debugger window.
Alt-Shift-3	Opens the Call Stack debugger window.
Alt-Shift-4	Opens the Classes debugger window.
Alt-Shift-5	Opens the Breakpoints debugger window.
Alt-Shift-6	Opens the Sessions debugger window.
Alt-Shift-7	Opens the Threads debugger window.
Alt-Shift-8	Opens the Sources window.
Ctrl-Tab	Toggles through the open windows in the order that they were last used. The dialog box displays all open windows and each of the open documents in the Source Editor.
Shift-Escape	Maximizes the Source Editor or the present window.
Ctrl-F4	Closes the current tab in the current window. If the window has no tabs, the whole window is closed.
Ctrl-Shift-F4	Closes all open documents in the Source Editor.
Shift-F4	Opens the Documents dialog box, in which you can save and close groups of open documents.
Alt-right	Displays the next tab in the current window.
Alt-left	Displays the previous tab in the current window..

Project Tasks

To perform this task	**Follow these steps**
Create a project.	1. Choose File > New Project (Ctrl-Shift-N). 2. Select the right template for your project.

Add a JAR file to a standard project's classpath.	1. In the Projects window, right-click the node for the project and choose Properties. 2. In the Project Properties dialog box, select the Libraries node in the Categories pane and ensure the Compile tab is selected. 3. Click Add JAR and select the JAR file in the file chooser. Note: If you also want to attach source code and Javadoc for the JAR file, click Add Library instead.
Set up compilation dependencies between projects.	1. In the Projects window, right-click the node for the project and choose Properties. 2. In the Project Properties dialog box, select the Libraries node in the Categories pane and ensure the Compile tab is selected. 3. Click Add Project and select the project folder for the project that you want to add to the classpath.
Open required projects.	• In the Projects window, right-click the project node and choose Open Required Projects.
Build a project.	• Choose Build > Build Main Project (F11) or right-click any project node and choose Build Project.
Clean a project.	• Right-click the project node and choose Clean Project.
Clean and build a project.	• Right-click the project node and choose Clean and Build Main Project (Shift-F11).
Run a project.	• Choose Run > Run Main Project (F6) or right-click any project node and choose Run Project.
Debug a project.	• Choose Run > Debug Main Project (F5) or right-click any project and choose Debug Project.

Specify sources for a JAR file on the project classpath.	1. Choose Tools > Library Manager from the main window. 2. If the JAR file is not already registered in the Library Manager, create a new empty library using the Add Library button. 3. Select the library in the left panel of the Library Manager. 4. In the Classpath tab, click Add JAR/Folder and specify the location of the JAR file containing the compiled class files. Note: A library can contain multiple JAR files. 5. In the Sources tab, add the folder or archive file containing the source code.
Specify Javadoc for a JAR file on the project classpath.	1. Choose Tools > Library Manager from the main window. 2. If the JAR file is not already registered in the Library Manager, register the JAR file as described above. 3. In the Javadoc tab, click Add ZIP/Folder and specify the location of the Javadoc files.
Set the main project.	• Right-click the project node and choose Set Main Project.

VCS Tasks

To perform this task	**Follow these steps**
Check out sources.	1. In the Versioning window, right-click the VCS working directory's top level node and choose CVS > Checkout. 2. Specify the modules you wish to check out. ○ To check out the entire repository select All. ○ To check out a specific directory, click the Select button to choose from a list of all of the modules in the repository. 3. Click Finish to check out the files.
Update local file versions.	1. Right-click the file's node that you wish to update. 2. Choose CVS > Update.

Diff files.	1. Right-click the appropriate revision node in the Versioning window. 2. Choose Diff Graphical or Diff Textual from the contextual menu.
Commit local changes.	1. Ensure that your local copies of the files are up-to-date by right-clicking and choosing CVS > Update before proceeding. 2. Right-click the files or directories you wish to commit and choose CVS > Commit.
Merge revisions.	1. In the Versioning window, expand the file's node to reveal the subnodes for each revision and branch. 2. Right-click the revision you wish to merge into your working copy and choose Merge With Selected Revision.
Configure a VCS working directory.	1. Choose Versioning > Versioning Manager. 2. Select the appropriate versioned working directory and click Edit. 3. Edit the necessary VCS settings in the working directory's Customizer.
Configure global VCS options.	1. Choose Tools > Options, expand the Source Creation and Management node, then select the Version Control Settings node. 2. Edit the desired properties in the Option window's right pane.

Configuring Tasks

To perform this task	Follow these steps
Configure general options	1. Choose Tools > Options from the main menu. 2. Select the option node for the properties you wish to edit. 3. Specify the desired settings in the option's property sheet.
Set the IDE's web browser	1. Choose Tools > Setup Wizard from the main menu. 2. In the wizard, select the desired browser from the Web Browser combo box.

Configure proxy settings	1. Choose Tools > Setup Wizard from the main menu. 2. In the wizard, select the Use HTTP Proxy Server check box. 3. Enter the Proxy Server Name and Port.
Configure the Auto Update feature	1. Choose Tools > Setup Wizard from the main menu. 2. Advance to the Update Center page of the Setup wizard. 3. Specify the update check frequency and any options.
Install additional modules	1. Choose Tools > Update Center from the main menu. 2. Designate the update center location you want to connect to in the Update Center wizard and click Next. 3. Select the modules you wish to install. 4. Review the licensing agreement and click Accept.
Customize the IDE's Menus	1. In the Options window, expand IDE Configuration > Look and Feel. 2. Right-click the Menu Bar node and choose Add > Menu. 3. In the New Menu dialog box, type a name for the menu and click OK. The IDE adds an empty menu to the main window. 4. Expand the Actions node and find the command you want to add to the menu, then right-click the command node and choose Copy. 5. Expand the Menu Bar node, right-click the node of the menu you just created, and choose Paste > Copy.
Configure General Java Settings	1. From the main menu, choose Tools > Options. 2. Expand the Editing node and select Java Sources. 3. Set properties as desired in the right pane of the window.

Source Editor Tasks

To perform this task	Follow these steps
Open a file that is not available in the Projects window or the Files window.	1. Choose File > Open File (Ctrl-O). 2. In the file chooser, navigate to the file and then click Open.

Maximize a tab in the Source Editor.	Do one of the following: - Double-click a file's tab in the Source Editor. - Make sure that the Source Editor window has focus and then press Shift-Escape. - Choose Window > Maximize.
Revert a maximized Source Editor to its previous size.	Do one of the following: - Double-click a file's tab in the Source Editor. - Press Shift-Escape. - Choose Window > Restore.
Display line numbers.	Choose View > Show Line Numbers.
View two files simultaneously.	1. Open two or more files. 2. Click the tab of one of the files and drag it to the side of the window where you want the file to be placed. Once the red preview box appears indicating the correct location for the window, release the mouse button to drop the window. The window can be split horizontally or vertically, depending on where you drag the tab.
Split the view of a single file.	1. Right-click the document's tab in the Source Editor and choose Clone Document. 2. Click the tab of the cloned document and drag it to the part of the window where you want the copy to be placed.
Format code automatically.	- Right-click in the Source Editor and choose Reformat Code. If any text is selected, only that text will be reformatted. If no text is selected, then the whole file is reformatted.

Source Editor Abbreviations for Java Files

Abbreviation	**Expansion**

En	Enumeration
Ex	Exception
Ob	Object
Psf	public static final
Psfb	public static final boolean
Psfi	public static final int
Psfs	public static final String
St	String
ab	abstract
bo	boolean
br	break
ca	catch (
cl	class
cn	continue
df	default:
ex	extends
fa	false
fi	final
fl	float
fy	finally
ie	interface
im	implements
iof	instanceof
ir	import
pe	protected
pr	private
psf	private static final

psfb	private static final boolean
psfi	private static final int
psfs	private static final String
pst	printStackTrace();
pu	public
re	return
serr	System.err.println ("
sout	System.out.println ("
st	static
sw	switch (
sy	synchronized
tds	Thread.dumpStack();
th	throws
tw	throw
twn	throw new
wh	While (

Source Editor Abbreviations for JSP and Servlet Files

Abbreviation	**Expansion**
ag	application.getValue("
ap	application.putValue("
ar	application.removeValue("
cfgi	config.getInitParameter("
jg	<jsp:getProperty name="
jspf	<jsp:forward page="
jspg	<jsp:getProperty name="
jspi	<jsp:include page="

jspp	<jsp:plugin type="
jsps	<jsp:setProperty name="
jspu	<jsp:useBean id="
oup	out.print("
oupl	out.println("
pcg	pageContext.getAttribute("
pcgn	pageContext.getAttributeNamesInScope("
pcgs	pageContext.getAttributesScope("
pcr	pageContext.removeAttribute("
pcs	pageContext.setAttribute("
pg	<%@ page
pga	<%@ page autoFlush="
pgb	<%@ page buffer="
pgc	<%@ page contentType="
pgerr	<%@ page errorPage="
pgex	<%@ page extends="
pgie	<%@ page isErrorPage="
pgim	<%@ page import="
pgin	<%@ page info="
pgit	<%@ page isThreadSafe="
pgl	<%@ page language="
pgs	<%@ page session="
rg	<request.getParameter("
sg	session.getValue("
sp	session.putValue("
sr	session.removeValue("
tglb	<%@ taglib uri="

Source Editor Abbreviations for XML and DTD Files

Abbreviation	Expansion
?xm	<?xml version="1.0" encoding="UTF-8"?>
!do	<!DOCTYPE>
!cd	<![CDATA[|]]>
!at	<!ATTLIST |>
!el	<!ELEMENT |>
!en	<!ENTITY |>
pu	PUBLIC "|"
sy	SYSTEM "|"
!at	<!ATTLIST |>
!el	<!ELEMENT |>
!en	<!ENTITY |>
!no	<!NOTATION |>
pu	PUBLIC "|"
sy	SYSTEM "|"
cd	CDATA
em	EMPTY
en	ENTITY
ens	ENTITIES
fi	#FIXED
im	#IMPLIED
nm	NMTOKEN
nms	NMTOKENS
nn	NOTATION
pc	#PCDATA

Build Tasks

To perform this task	**Follow these steps**
Add a JAR file to a project's classpath.	1. In the Projects window, right-click the node for the project and choose Properties. 2. In the Project Properties dialog box, select the Libraries node in the Categories pane and ensure the Compile tab is selected. 3. Click Add JAR and select the JAR file in the file chooser. Note: If you also want to attach source code and Javadoc for the JAR file, click Add Library instead.
Add an IDE project to a project's classpath.	1. In the Projects window, right-click the node for the project and choose Properties. 2. In the Project Properties dialog box, select the Libraries node in the Categories pane and ensure the Compile tab is selected. 3. Click Add Project and select the project folder for the project that you want to add to the classpath.
Build a project.	• Choose Build > Build Main Project (F11) or right-click any project in the Projects window and choose Build Project.
Compile a single file.	• Choose Build > Compile File (F9).
Clean a project.	• In the Projects window, right-click the project node and choose Clean Project.
Clean and build the main project.	• Choose Build > Clean and Build Main Project (Shift-F11).

View build products, such as JAR files and generated Javadoc.	1. In the Files window, expand the project folder node. 2. Open the `build` folder to view the project's compiled classes. Note: Javadoc files and built libraries, like JAR files and WAR files, are in the `dist` folder.
Correct compilation errors.	• In the Output window, double-click any Java syntax error to jump to the location in the source code where the error occurred. • In the Output window, double-click any Ant error to open the Ant script in the target that failed.

Running J2SE Application Tasks

To perform this task	**Follow these steps**
Set the runtime classpath.	1. Right-click the project node in the Projects window and choose Properties. 2. Select the Libraries node in the Categories pane. 3. Use the Add Project, Add Library, or Add JAR/Folder buttons to add the required file type to the project's classpath. Note: By default, the project's runtime classpath contains the project's compiled sources and everything on the compilation classpath.
Set the project main class	1. Right-click the project node in the Projects window and choose Properties. 2. Select the Run node in the Categories pane. 3. Type the fully qualified name of the class in the Main Class field.
Set the runtime arguments.	1. Right-click the project node in the Projects window and choose Properties. 2. Select the Run node in the Categories pane. 3. Enter the arguments in the Arguments field.

Set JVM arguments.	1. Right-click the project node in the Projects window and choose Properties. 2. Select the Run node in the Categories pane. 3. Type a space-separated list of arguments in the VM Arguments field.
Set the working directory for execution.	1. Right-click the project node in the Projects window and choose Properties. 2. Select the Run node in the Categories pane. 3. Type the full path to the working directory in the Working Directory field.
Run a project.	• Choose Run > Run Main Project (F6). • Right-click any project in the Projects window and choose Run Project.
Run a single file.	1. Select one or more files in the Projects window, Files window, or Source Editor. 2. Choose Run > Run File > Run *your_filename* from the main menu.

Running Web Application Tasks

To perform this task	**Follow these steps**
Set request parameters for JSP files.	1. In the Projects or Files window, navigate to the project's JSP file. 2. Right-click the JSP file and choose Properties. 3. Type the necessary parameters in the Request Parameters property field in the URL_query_string format.

Set execution URIs and parameters for servlets.	1. Select the servlet in the Projects or Files window. 2. Choose Tools > Set Servlet Execution URI in the main menu. 3. Type the execution URI and parameters. 4. Click OK. Note that you can only set a request URI for a servlet if it has servlet mappings in the deployment descriptor (`web.xml` file). The default URI is defined in the deployment descriptor.
Set the runtime classpath.	1. In the Projects window, right-click the Libraries node and choose 2. Add Project, Add Library, or Add JAR/Folder, to add the necessary item to the project's classpath. Note that by default, the project's runtime classpath contains the project's compiled sources and everything on the compilation classpath.
Set the web server.	1. In the Projects window, right-click the project node and choose Properties. 2. Select the Run node in the Categories pane. 3. Select the web server from the Server drop-down.
Set the context path and relative URL.	1. In the Projects window, right-click the project node and choose Properties. 2. Select the Run node in the Categories pane. 3. Type the desired path in the Context Path field and, optionally, a relative URL.
Run a project.	• Choose Run > Run Main Project (F6) • Right-click any web project in the Projects window and choose Run Project.
Run a single file.	1. Select one or more files in the Projects window, Files window, or Source Editor. 2. Choose Run > Run File > Run *your_filename* from the main menu.

Debugging Tasks

To perform this task	Follow these steps
Start a local debugging session.	- To debug the main project, choose Run > Debug Main Project (F5). - To debug any individual project, right-click the project and choose Debug Project.
Start a remote debugging session.	1. On the computer where program is located, start the program in debugging mode. 2. On the computer where the IDE is running, open the projects that contain the source for the program. 3. Choose Run > Attach Debugger. 4. Select the connector type, enter any required process information, and then click OK. Note: See your VM documentation for information about the connectors it provides.
Debug a single file.	- Select any runnable file in the Projects window and choose Run > Run File > Debug *filename*.
Finish a debugging session.	- To finish the current session, choose Run > Finish Debugger Session (Shift-F5). - To finish any session, open the Sessions window (Alt-Shift-6), right-click the session, and choose Finish.
Set a breakpoint.	- To set a line breakpoint, open the file in the Source Editor and click in the left margin on the desired line (Ctrl-F8). - In the Source Editor, select the element of code on which you want to set a breakpoint and choose Run > New Breakpoint (Ctrl-Shift-F8). Then set the breakpoint type and additional options in the New Breakpoint dialog box.
Modify breakpoint properties.	- Open the Sessions window (Alt-Shift-6), right-click the session, and choose Customize.

Set a watch.	- Right-click a variable or expression in the Source Editor and choose New Watch (Ctrl-Shift-F7).
Suspend and resume a thread.	- Open the Threads window (Alt-Shift-7), right-click the thread, and choose Suspend or Resume.
Manage which JDK classes the debugger steps into.	- Open the Sources window (Alt-Shift-8) and select the checkbox for the archive file or directory containing the JDK sources. - Open the Sources window (Alt-Shift-8) and uncheck the checkbox for the source directories you do not want to step into.
Pop a call from the call stack.	- To pop the most recent call from the call stack, choose Run > Stack > Pop Topmost Call. - To pop multiple calls, open the Call Stack window (Alt-Shift-3), right-click the call that you want to remain at the top of the call stack, and choose Pop to Here.
View information for a call on the call stack.	- To move one level away from the `main` routine, choose Run > Stack > Make Callee Current (Ctrl-Alt-up arrow). - To move one level toward the `main` routine, choose Run > Stack > Make Caller Current (Ctrl-Alt-down arrow). - To make a call current, double-click the call in the Call Stack window.

JUnit Tasks

To perform this task	**Follow these steps**
Create a test for existing sources.	1. Right-click the class or package node for which you wish to generate a test and choose Tools > JUnit Tests > Create Tests (Ctrl-Alt-J). 2. Set the parameters for the test skeleton generator in the Create Tests dialog box and click OK.

Create an empty test.	1. Choose File > New File. 2. Select JUnit in the Categories pane and Empty Test in the File Types pane. Click Next. 3. Enter a name for the test class in the Class Name field. 4. Select the target package from the Package combo box and set any required options. 5. Click Finish.
Run a test for a class.	1. Select the node of the class you wish to test. 2. From the Main menu, choose Run > Run File > Test *filename* (Ctrl-F6).
Run tests for an entire project.	• Right-click the node of the project for which you wish to run tests and choose Test Project (Alt-F6).
Edit a test.	1. Right-click the node for the source whose test you wish to edit. 2. Choose Tools > JUnit Tests > Open Test (Ctrl-Alt-K) from the contextual menu.
Debug a test.	1. Select the class for whose test you wish to debug. 2. Choose Run > Run File > Debug Test for *filename* (Ctrl-Shift-F6).
Configure JUnit.	1. Choose Tools > Options. 2. Expand the Testing node and select JUnit Module Settings. 3. Edit the necessary properties and then click Close.