NetBeans IDE Field Guide

IDE Project Fundamentals

Chapter Contents

NetBeans IDE has a comprehensive project system that provides a structure for your sources, tests, and outputs and simplifies your workflow. This project system is based on the Ant build tool, which provides added flexibility in the way you configure your projects.

This chapter provides an overview of the project system and how you can leverage it for your projects. In the Integrating Existing Ant Scripts With the IDE chapter, you can learn some more advanced techniques, which are particularly useful if you are developing applications with specific requirements that are not addressed in standard IDE projects.

Introduction to IDE Projects

The starting point for most work in the IDE is through the creation of a project. When you create a project, the IDE typically does the following things for you (depending on project type):

- Creates a source tree with a skeleton class inside.
- Creates a folder for unit tests.
- Sets classpaths for compilation, running, and (depending on type of project) testing. (The compilation classpath also determines the classes that the Source Editor is aware, for example, when you use code completion features.)
- Sets the Java platform that the project is to run in. By default, it is the same platform that the IDE runs on.
- Creates an Ant build script (`build.xml`), which contains the instructions that the IDE uses when you perform commands on your project, such as compiling source files, running the application, running tests, debugging, compiling Javadoc documentation, and building JAR files. In addition, you can use this build script to run tasks on your project from outside of the IDE.

You can have multiple projects open at the same time, and the projects can be linked through dependencies. Project-specific commands in the Build and Run menus act on the currently designated main project. The main project is marked in bold as shown in Figure Main project.
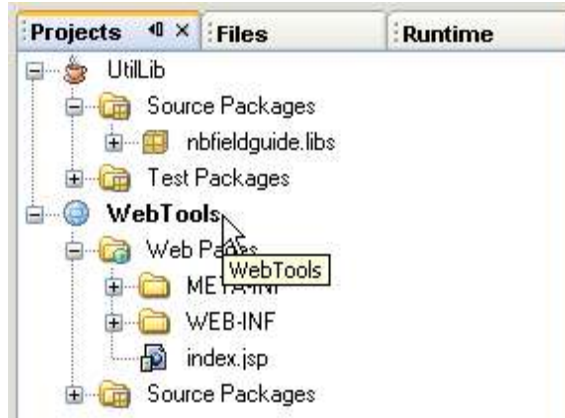
Illustration 1Main Project

NetBeans IDE Tip

Unlike in 3.x versions of NetBeans IDE, explicit creation of an IDE project is a primary step in your workflow in NetBeans IDE 4.1. In NetBeans IDE 3.x versions, "projects" were a peripheral paradigm with a limited feature scope.

Also, as opposed to the filesystem concept in NetBeans IDE 3.x, the folders included in the Projects and Files windows do not necessarily represent the classpath. In fact, it is OK to have multiple projects open, even if they have no relationship to the main project you are working with.

What Is Ant And Do I Need to Know Anything About It?

Ant is the mechanism that NetBeans IDE uses for running project-related commands. If you have no interest in Ant as such, you can completely ignore it, much as you would never bother decoding project metadata in another IDE. However, if Ant is already the lifeblood of your build process, you can set up NetBeans IDE to accommodate your existing build process, either by overriding specific Ant targets that the IDE generates or by providing your own Ant script.

Ant was developed by the Apache Software Foundation to automate routine developer tasks, such as compiling, testing, and packaging your application. Similar to Make, Ant has the advantage of being written in Java, so it works across multiple platforms. You can also use Ant to invoke other processes, such as checking out sources from version control, obfuscating classes, etc. In addition, you can write Java classes to extend Ant's functionality. On big development teams, Ant is often used as a production tool to compile and package the whole application for distribution.

Ant scripts themselves are written in XML. They are divided into high-level targets, which are collections of tasks that are run for specific purposes, such as cleaning the build directory, compiling classes, and creating packaged outputs.

Other IDEs provide integration with Ant to support writing and running of build scripts. NetBeans IDE takes this a step further by making Ant the

backbone for all project-related commands in the IDE. When you create an IDE project, the IDE generates an Ant script for you with targets for, among other things, compiling, running, debugging, and packaging your application. When you run project commands in the IDE (such as Build Main Project or Run Main Project), the IDE itself is calling the Ant scripts.

The fact that the IDE's project system is based on Ant provides another advantage: other developers do not have to use NetBeans IDE to build the project. It is possible to run an IDE-generated Ant script from another IDE or the command-line, which could be particularly useful for doing production builds of your team's application.

Choosing the Right Project Template

When you open the New Project wizard (File | New Project), you are presented with several templates, the use of which might not be immediately apparent.

Depending on the distribution of the IDE that you have, you might have several categories of templates. See Table 1.1 for the list:

Table 1.1

Template Category	Description
General	For desktop applications or Java libraries based on Java 2 Standard Edition.
Web	For web applications based on Java 2 Enterprise Edition.
Enterprise	For enterprise tier applications, such as those that include Enterprise JavaBeans components (EJBs) and Web services, based on Java 2 Enterprise Edition.
MIDP	For applications targeted toward handheld devices, based on Java 2 Micro Edition.
Samples	Sample applications that are ready to build and run from the IDE.

For each category, the IDE provides various templates based on the structure of the project and the layout of your files.

Standard project templates (all of the templates with the exception of "With Existing Ant Script" templates) provide maximum integration with the IDE's user interface. However, the use of those templates assumes that your project:

- Has one source root and one test root.
- Is designed to produce one distributable output (e.g. A JAR or WAR file).
- Will use the Ant script that the IDE has generated for you (though you can customize this Ant script).

If any of these things are not true of your project, you can:

- Restructure your sources to fit into this model.
- Create individual projects for each source root or output and declare dependencies between the projects.
- Modify the generated Ant script to add or override targets (though this option

does not alter the one source root per project limitation).
- Use a "With Existing Ant Script" (or "free-form") template (marked with a ⊞ icon to create your project).

The free-form templates offer you more flexibility in structuring your project. However, when you use a free-form template, you have to do some extra configuration and write build targets to get some IDE functionality (like debugging) to work.

This chapter mainly covers general Java projects created from standard templates. There is also some information on Web projects, though Web projects are covered in more detail in a separate chapter.

Creating a Project From Scratch

If you have want to start developing an application from scratch, start with the base template for the type of application you are developing.

For Java desktop applications or a standard Java library to be used by other applications:

1. Choose File | New Project.
2. In the Categories tree, select the General folder.
3. Select Java Application or Java Class Library.

   The Java Application template includes skeleton code for a main class. Use this template if you want this project to contain the entry point for you application.

   The Java Library template contains no main class. Use this template if you want to create a library that is used by other applications.
4. Click Next.
5. Optionally, fill in the following fields in the Name and Location panel of the wizard:

   Project Name. The name by which the project is referred to in the IDE's user interface.

   Project Location. The location of the project on your system.

   The resulting Project Folder field shows the location of the folder that contains folders for your sources, test classes, compiled classes, packaged library or application, etc.
6. Optionally, deselect the Set as Main Project checkbox if you have another project open which you want the IDE's main project commands (e.g. Build Main Project) to apply to.
7. Set the main class, using the fully-qualified class name, but without the `.java` extension (e.g. `com.mycompany.MyApp.MyAppMain`).
8. Click Finish.
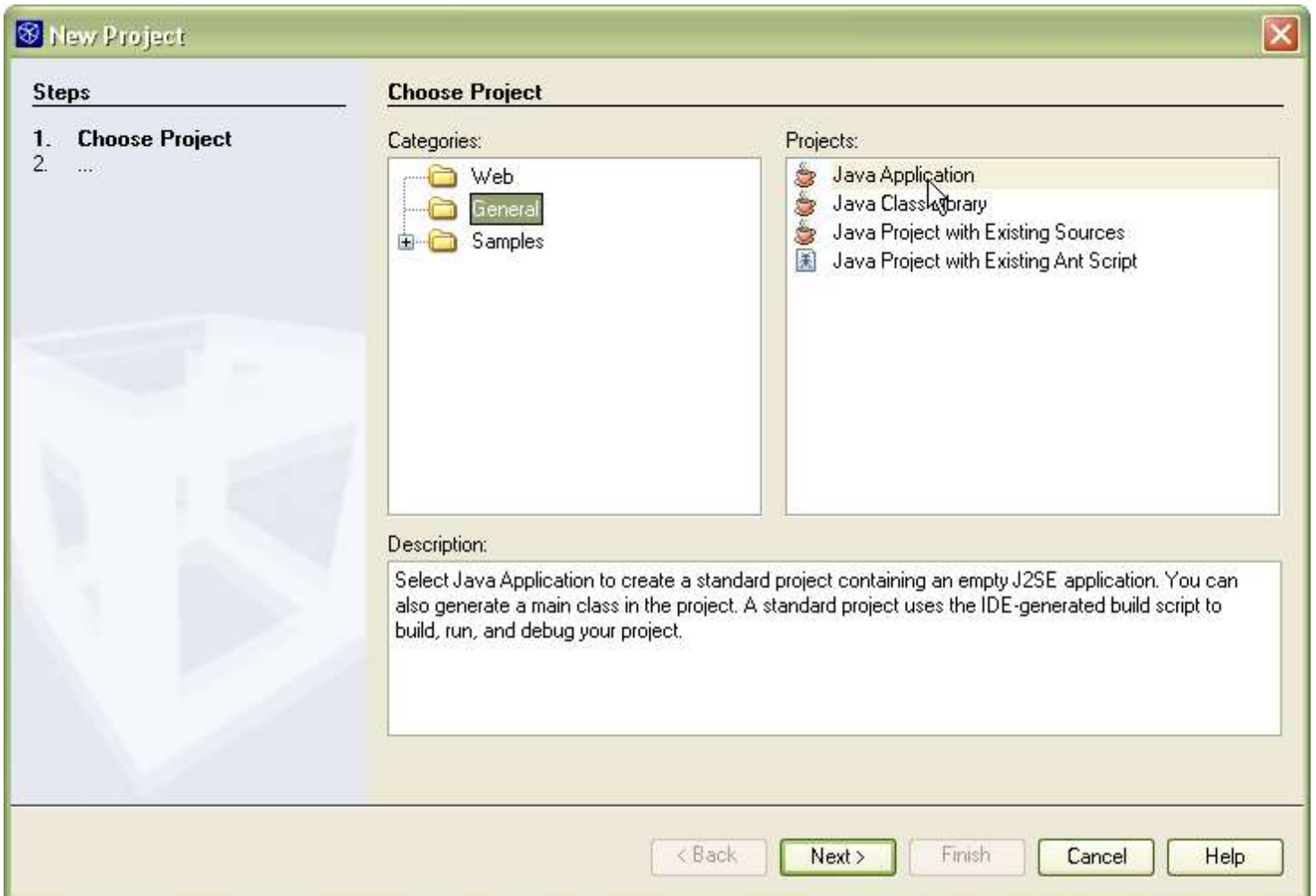
The generated project is viewable in both the Projects window and the Files window.

The Project window provides a "logical" view of your sources and other files you are likely to edit often (such as web pages in web application projects). Project metadata (including the project's Ant build script) and project outputs (such as compiled classes and JAR files) are not displayed here.

Java source files are grouped by package, instead of by folder hierarchy.

The Files window represents all of the files that are in your project in folder hierarchies as they appear on your system.

For web applications:

1. Choose File | New Project.
2. In the Categories tree, select the Web folder.
3. Select Web Application and click Next.
4. Fill in the following fields (or verify the generated values) in the Name and Location panel of the wizard:

   Project Name. The name by which the project is referred to in the IDE's user interface.

   Project Location. The location of the project on your system.

The resulting Project Folder field shows the location of the folder that contains folders for your sources, test classes, compiled classes, packaged library or application, etc.

5. Optionally, deselect the Set as Main Project checkbox if you have another project open which you want to the IDE's main project commands (e.g. Build Main Project) to apply to.

6. Optionally, adjust the following:

   Context Path. The URL namespace that the web app uses. For example, if the context property value is `MyWebApp,` the web application is accessed from a file within `http://`*`HostName:`*`PortNumber/MyWebApp/`.

   J2EE Specification Level. The version of the J2EE platform that your application will run against.

7. Click Finish.

If the application you are developing requires development in multiple source hierarchies, you can create multiple IDE projects to accommodate them. You can connect projects by declaring dependencies in a project's Properties dialog box. See the Structuring Your Projects topic for more information.

Importing a Project Developed in a Different Environment

If you have a project that you have been working on in a different development environment, you can "import" the project into NetBeans IDE using a "With Existing Sources" project template.

> **NetBeans IDE Tip**
>
> A standard NetBeans IDE project can only have one source root. You can get around this restriction by creating projects for each source root, declaring a main project as the entry point of the application, and setting dependencies between the projects. See Structuring Your Projects for more information.

To create an IDE project for a Java application that was created in a different environment:

1. Choose File | New Project.
2. In the Categories tree, select the General folder.
3. Select Java Application With Existing Sources and click Next.
4. In the Source Packages Folder field, enter the folder that contains the default package of your sources (e.g. the folder might be called `src`).
5. In the Test Packages Folder field, enter the folder that contains the default package of your unit tests (e.g. the folder might be called `tests`). You can leave this field blank if you have no unit tests.
6. Fill in the following fields (or verify the generated values) in the Name and Location panel of the wizard:

   Project Name. The name by which the project is referred to in the IDE's user interface.

   Project Folder. The location of the project's metadata. By default, the IDE places this folder with your project sources, but you can designate a different location.
7. Optionally, deselect the Set as Main Project checkbox if you have another project open which you want the IDE's main project commands (e.g. Build Main Project) to apply to.
8. Click Finish.

To create an IDE project for a web application that you have begun developing in a different environment:

1. Choose File | New Project.
2. In the Categories tree, select the Web folder.
3. Select Web Application With Existing Sources and click Next.
4. In the Location field, enter the folder that contains your sources and web pages.
5. Optionally, edit the following fields in the Name and Location panel of the wizard:

   Project Name. The name by which the project is referred to in the IDE's user interface.

   Project Folder. The location of the project's metadata. By default, the IDE places this folder with your project sources, but you can designate a different location.

6. Optionally, deselect the Set as Main Project checkbox if you have another project open which you want to the IDE's main project commands (e.g. Build Main Project) to apply to.

7. Click Next.

8. Make sure the Web Pages and Java Sources fields are filled in with the appropriate folders. The Web Pages field should point to the folder that contains all of the application's web pages. The Java Sources field should point to the folder that contains the default Java package for the project (e.g. the folder might be called `src`).

9. Optionally, if you have a folder that contains libraries specifically for this project, such as tag libraries, specify that folder in the Libraries field.

10. Click Finish.

NetBeans IDE Tip

For more complex projects that require a lot of classes, it might work best to use the web application project as the main project and configure it to depend on Java Library projects that handle most of the processing in your application. For applets and tag libaries, it is necessary to use Java Library projects to get them to work within your web app.

Navigating Your Projects

Once the project is set up, your files are accessible from both the Projects window and the Files window.

Projects Window

The Projects window is designed to be the center of operations for file creation, project configuration, and project building. The Projects window displays only the files that are likely to be regularly edited in a project, such as Java source files, web pages, and tests. Build outputs and project metadata are ignored.

Java sources are displayed according to package structure, which generally makes it easier to navigate to files since you do not have to navigate through nested folder hierarchies.

The main node for each project has commands on its contextual (right-click) menu for compiling, running, debugging, creating Javadoc, and other project-related actions.
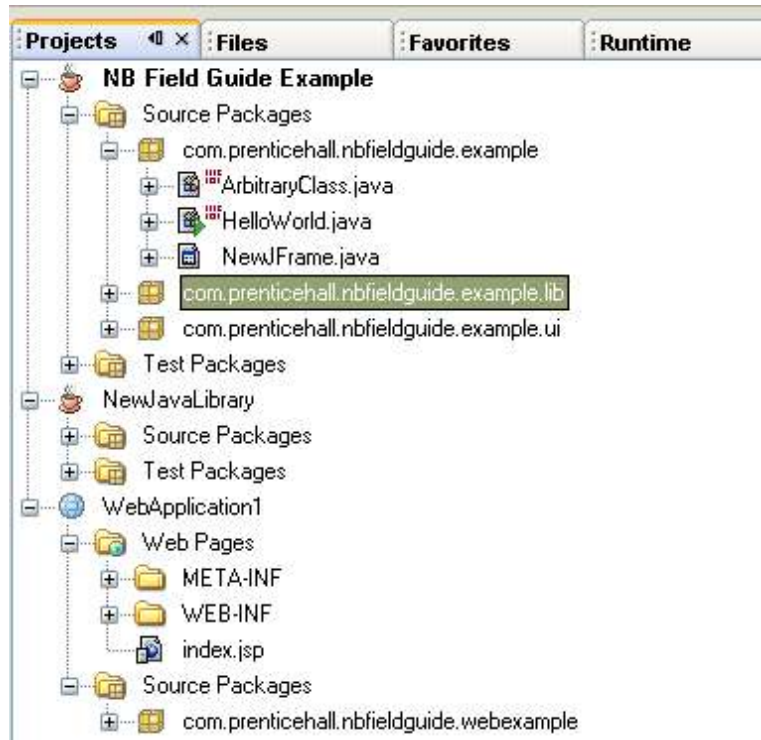
Files Window

The Files window displays your files organized according to file hierarchy, much as they appear in a file manager on your system. (One exception to this is that the .form files that are used to generate the design view of GUI classes that you create in the IDE's Form Editor are not displayed – just the .java file is displayed.) In addition to your sources, tests, and web files, project metadata and files produced when you build the project also appear. See Figure FilesView.

The Files window is useful if you need to browse the physical structure of your project, such as the contents of the JAR file (or other archive file) produced by the project. The Files window also provides direct access to project metadata files if you need to customize them. Otherwise, it is probably best to work in the Projects window, since project-related commands are not available in the Files window.
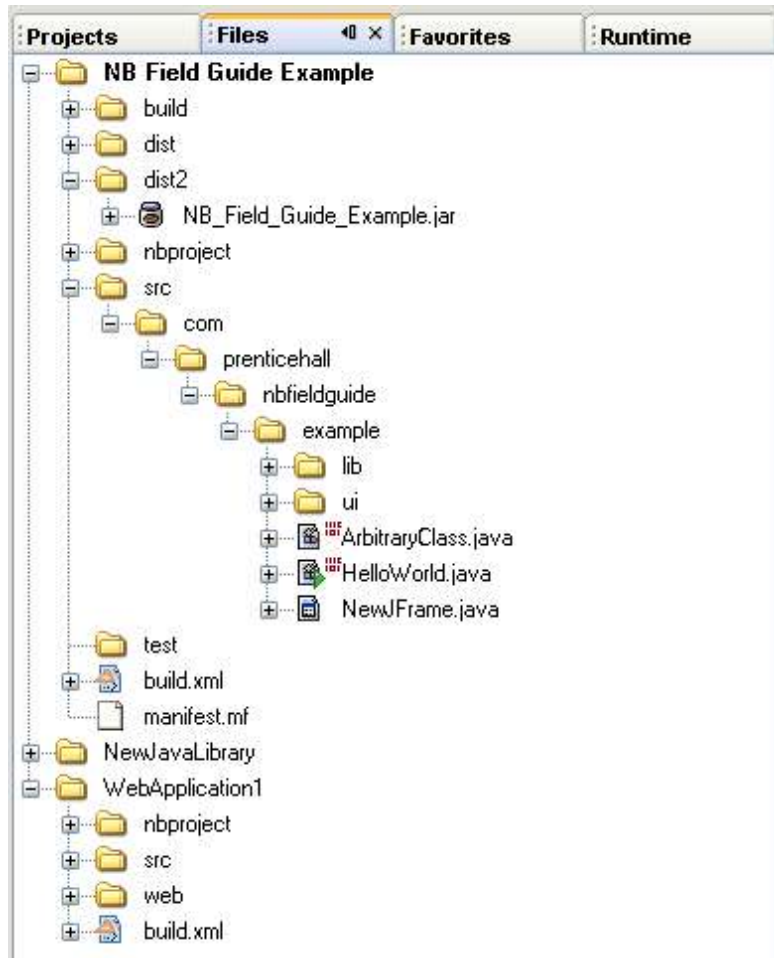
NetBeans IDE Tip

If you prefer a physical view of your files but you do not want to see the top levels of the file hierarchy that are necessitated by the Java package structure, you can use the Favorites window. See Working With Files Not In the Project.

You can also set the Projects window to display files according to file hierarchy instead of by package using an undocumented option in your `netbeans.conf` file. Open the IDE's installation directory, expand the `etc` folder, open `netbeans.conf` in a text editor, and add the following line into the `netbeans_default_options` section of the file:

```
-J-
Dorg.netbeans.spi.java.project.support.ui.packageView
.USE_TREE_VIEW=true
```

Fast Navigation Between the Projects and Files Windows

If you are working with a node in one window but want to switch to that node in a different window, there are several keyboard shortcuts available to speed navigation. For example, if you have been working in the Projects window and now want to inspect the build script (which, in standard projects, is viewable through the Files window), you can jump straight to the project's

node in the Files window without having to scroll to the node. The following are some of the shortcuts that are particularly useful for navigating between the "Explorer" style windows:

Table 1.2

Keyboard Shortcut	Action
Ctrl-Shift-1	Open the Projects window and display the node that corresponds to the currently selected file, package, or folder.
Ctrl-Shift-2	Open the Files window and display the node that corresponds to the currently selected file, package, or folder.
Ctrl-Shift-3	Open the Favorites window and display the node that corresponds to the currently selected file, package, or folder.
Ctrl-Shift-8	Open the Versioning window and isplay the node that corresponds to the currently selected file, package, or folder. This command only works if you have set up the IDE to use version control for that project.

You can also use these shortcuts in the Source Editor when you want to jump to the node for that file.

Physical Structure of IDE Projects

The Projects window provides a "logical" view of projects, displaying your sources according to the Java package structure. Output files (such as compiled classes and JAR files), project metadata, and version control metadata files (if you are using version control) are hidden.

If you want to see the actual files created in the project, open the Files window (by clicking the Files tab above the Projects window).

When you create a general project (using a standard template), the IDE creates the following folders and files:

- `nbproject` folder, which includes files that store data about your project and are used to create the build script for your project.
- `src` folder, which holds your source files. This folder corresponds with the Source Packages node that appears in the Projects window.
- `test` folder, which holds any unit tests that you create for your project. This folder corresponds with the Test Packages node that appears in the Projects window.
- `build.xml` file, which is the Ant build script that is used when compiling, cleaning, running, and debugging your project. This file actually imports the `build-impl.xml` file which is generated in the `nbproject` folder.

When you build your project, the following folders are added:

- `build` folder,which holds the compiled class files.
- `dist` folder,which holds the packaged output for the project (a JAR file).

For web applications created from standard templates, the IDE creates the following folders and files:

- `nbproject` folder, which includes files which store data about your project and are used to create the build script for your project.
- `src` folder, which holds Java source files.
- `web` folder, which typically holds any HTML pages, JSP pages, custom tag libraries, and other files for the web application.
- `build.xml` file, which is the Ant build script that is used when compiling, cleaning, running, and debugging your project.

When you build your project, the following folders are added:

- `build` folder,which holds the compiled class files.
- `dist` folder,which holds the compiled application in distributable form (a WAR file).

Working With Files Not In the Project

When you set up a project in the IDE, only the files most likely to need hand editing (such as your source files) are exposed in the Projects window. You can access all files in your project using the Files window, but sometimes there are other files you might want to access. If there are other files on your system that you want to regularly have access to, you can display them in the Favorites window.

To use the Favorites window:

1. Choose Window | Favorites to open the Favorites window as a tab in the area occupied by the Projects window.
2. In the Favorites window, right-click the Favorites node and choose Add Favorites.
3. In the file chooser, select the root folder that you want to add to Favorites.

You can add multiple folders to in Favorites window and you can add folders at any level of a file hierarchy. This provides a lot of flexibility in what files you make visible. For example, if you have a deep package hierarchy, you can save a lot of time spent on folder expansion by adding folders that directly contain your files.

You can run most IDE commands on nodes in the Favorites window. However, project-specific commands (such as Build Main Project) are not available in Favorites, so you must call those commands in the Projects window, through main menu, or through keyboard shortcuts.

NetBeans IDE Tip
If you merely want to open a file in the IDE one time, you can use the File | Open command.

Creating Packages and Files in the Project

Once the project is set up, you can create packages and files from the Projects window or the Files window. Right-click the node for package or folder where you would like to add a class or package and choose New | File/Folder to open the New File wizard. Or you can directly choose one of the templates listed below File/Folder.

To create a Java package:

1. Right-click the Source Packages node in the Projects window and choose New | Java Package.

   If Java Package is not one of the choices in the New submenu, choose New | File/Folder instead. In the New File wizard, select the Java Classes node, select Java Package, and click Next.

2. In the Package Name field of the wizard, type the name of the package, delimiting levels of the package with periods (e.g. `com.mydomain.myproject`) and then click Finish.

NetBeans IDE Tip

When creating subpackages, you can save yourself a few keystrokes by choosing New | Java Package from a package node. The base package name is already filled in for you, so you just need to add the last part of the new package name.

You can also enter a new package when using the New File wizard to create a new Java class.

To create a file:

1. Right-click the Source Packages node in the Projects window and choose New | File/Folder.

2. In the New File wizard, browse the templates available, select the one you want and click Finish.

NetBeans IDE Tip

You can also select a template straight from the New submenu, where a short list of templates commonly used for the selected project type is displayed. The list of files that are available there is updated to reflect the templates that you commonly use.

File Templates

File creation in the IDE begins with templates. The templates that are available depend on the features that you have installed in the IDE. The following are some of the available categories:

Java Classes

Several templates that provide skeleton code for basic types of classes, such as main classes, interfaces, and exceptions.

Java GUI Forms

Swing and AWT templates for developing visual desktop applications. When you create a file from one of these templates, the Form Editor opens, which enables you to build forms visually (dragging and dropping components from a palette to the Form Designer, changing properties in the Component Inspector, etc.).

JavaBeans Objects

Various templates for classes that adhere to the JavaBeans component architecture. Included are templates for a bean with a skeleton getter and setter, BeanInfo classes, a property editor, and a customizer class.

JUnit

Templates that provide skeleton code for unit tests of Java classes.

XML
    XML-related templates for XML documents, XML schemata, DTDs, XSL stylesheets, and cascading stylesheets,

Ant Build Scripts
    Provides a simple templates for a simple skeleton for an Ant script and a template for a custom Ant task with detailed comments. These scripts could be useful if you want to extend the default behavior of the IDE's project system, but are not necessary if the IDE's project system already provides all of the features you need.

Web
    Provides templates that are useful for web applications, including JSP files, HTML files, tag library files and tag library descriptors (TLD files). Also provides Java class templates for servlets, filters, tag handlers, and web application listeners.

Other
    Provides templates for HTML, properties, and empty files.

Starting With a Blank File

    If you want to start with a completely blank file without a pre-determined file extension, you can use the Other | Empty File template. If you give the file an extension that the IDE recognizes, the IDE will treat it as that type of file in the editor (complete with syntax highlighting and other Source Editor features).

Configuring the Project's Classpath

    You can manage the classpath for your application through the project's Properties dialog box, which you can open by right-clicking the project's node in the Projects window and choosing Properties.

    The IDE enables you to have different classpaths for compiling, testing, and running your application. When you set the compilation classpath, those items are automatically added to the other classpaths. You can then add items specific to compiling tests, running the application, and running tests on the application.

    The compilation classpath also affects some editor features, such as which classes are available in the code completion feature.

    See the following topics for instructions on specific tasks.

Changing the Version of the JDK That Your Project is Based On

    By default, IDE projects use the same JDK that the IDE runs on. To set up a project to run on a different JDK, you need to:

    1. Add the JDK in the Java Platform Manager
    2. Specify the JDK for the project to use in the project's Properties dialog box.

    By default, the IDE includes only the version of the JDK that the IDE is running on in the Java Platform Manager.

To make a JDK version available for projects in the IDE:

1. Choose Tools | Java Platform Manager.
2. Click Add Platform.
3. In the file chooser, navigate to and select the JDK version that you want to add to the Java Platform Manager.

   The JDK folder should be marked with the icon in the file chooser.
4. Close the Java Platform Manager.

To switch the JDK that a project uses:

1. Right-click the project's main node in the Projects window and choose Properties.
2. Select the General node.
3. Select the JDK that you want to use in the Java Platform combo box and click OK.

NetBeans IDE Tip

If you want to change the JDK that the IDE itself runs on, you can do so in the `netbeans.conf` file in the IDE's installation directory. In a file manager on your system, navigate to the IDE's installation directory, expand *NetBeansHome*`/etc` and open `netbeans.conf` in a text editor. Below the `#netbeans_jdkhome="/path/to/jdk"` comment line, type the `netbeans_jdkhome` option with the path to the JDK.

For example: `netbeans_jdkhome="C:/j2sdk1.4.2_04"`

Referencing JDK Documentation (Javadoc) From the Project

When you are coding in the IDE, it is often useful to have Javadoc documentation handy for the classes you are using. When you have Javadoc documentation on your system, you can freely browse it from within the IDE by jumping to a browser from the class you have currently selected in your code (Alt-F1).

The JDK documentation is not included with the standard JDK download. If you do not have the JDK documentation on your system, you can get from java.sun.com (for JDK 5.0, go to http://java.sun.com/j2se/1.5.0/download.jsp).

To make JDK documentation viewable in the IDE:

1. Choose Tools | Java Platform Manager.
2. Select the Javadoc tab.
3. Click the Add ZIP/Folder button. Then navigate to and select the JDK documentation zip file or folder on your system.
4. Close the Java Platform Manager.

NetBeans IDE Tip

You do not need to have the JDK's Javadoc on your system to see Javadoc when you are using code completion. The code completion box gets Javadoc comments straight from the JDK source code.

Adding Folders and JAR Files and to the Classpath

If your project relies on pre-built binaries or classes that are not part of a NetBeans IDE project, you can add these to your classpath as well.

To add a JAR file or a set of classes to your project classpath:

1. Right-click the project's node in the Projects window and choose Properties.
2. Select the Compiling Sources node.
3. Click Add JAR/Folder. In the file chooser, navigate to and select the folder of the project that you want to depend on and click Add Project JAR files.
4. Click OK to close the Project Properties dialog box.

Making External Sources Available in the IDE

You might want to associate documentation and source with classes that your project depends on. This is useful if you want to do any of the following:

- See the source if debugging into a class that your project depends on. You can solve this problem by using the IDE's "library" construct.
- Jump to the source of a referenced class from the Source Editor (Alt-O).
- Jump to the Javadoc documentation of a source file from within the IDE (Alt-F1).

The IDE has a Library Manager feature that enables you to declare these associations, which you can then take advantage of for all of your projects.

To create a library:

1. Choose Tools | Library Manager.
2. In the New Library dialog box, type a display name for the library and click OK.
3. In the Class Libraries list, select the new library.
4. Select the Classpath tab and click Add JAR/Folder. Then select the JAR file or the folder that contains the classes and click Add JAR/Folder.
5. If you want to associate sources with the library, select the Sources tab and Click Add JAR/Folder. Then select the JAR file or the folder that contains the sources and click Add JAR/Folder.
6. If you want to associate Javadoc documentation with the library, select the Javadoc tab and Click Add JAR/Folder. Then select the JAR file or the folder that contains the documentation and click Add ZIP/Folder.
7. Click OK to close the Project Properties dialog box.

To add a library to your project:

1. Right-click the project's node in the Projects window and choose Properties.
2. Select the Compiling Sources node.
3. Click Add Library and select the library from the dialog box.
4. Click OK to close the Project Properties dialog box.

NetBeans IDE Tip

When you designate a library through the Library Manager, the IDE automatically includes the documentation and sources that you have associated with the JAR file in the classpath when you add only the JAR to the classpath.

If the library you have designated consists of more than one JAR file, the other JARs are not duplicated on the classpath if you have added them to the classpath individually.

Structuring Your Projects

Standard IDE projects are essentially modular. If your application just needs to be built from one source hierarchy into one JAR file, then you can build it with a single project. If your application exceeds that scope, you can build your application from multiple IDE projects that are linked together with one of those projects declared as the main project.

Setting the Main Project

The main project is the one that project-specific commands (such as Build Main Project) in the main menu always act on. When an application is composed of many related projects, the main project serves as the entry point for the application for purposes of compiling, running, testing, and debugging. You can only have one main project set at a time.

To make a project the main project, right-click that project's node and choose Set Main Project.

NetBeans IDE Tip

If you find that your main project inadvertently gets changed from time to time, it might be because some project templates contain an option to make the new project the main project. If you create a new project that you do not want to be a main project, make sure to deselect the Set As Main Project checkbox in the New Project wizard.

Creating Subprojects

For more complex applications, you might need to create multiple IDE projects, where one project is the application entry point that depends on other projects. (Any project that has another project depending on it function as a subproject, though it is not specifically labeled as such in the IDE.) Each IDE project can create one distributable output (such as a JAR file), which in turn might be used by other projects. There is no particular limit to how long on a chain of dependencies might be, though the chain must not be circular. (For example, if project A depends on classes project B, project B cannot depend on classes in project A).

NetBeans IDE Tip

Though it might be a hassle at first, reorganizing your code to eliminate circular dependencies will probably pay off in the long run by making your code easier to maintain and extend.

To set a dependency between two projects:

1. Right-click the project's node in the Projects window and choose Properties.
2. Select the Compiling Sources node.
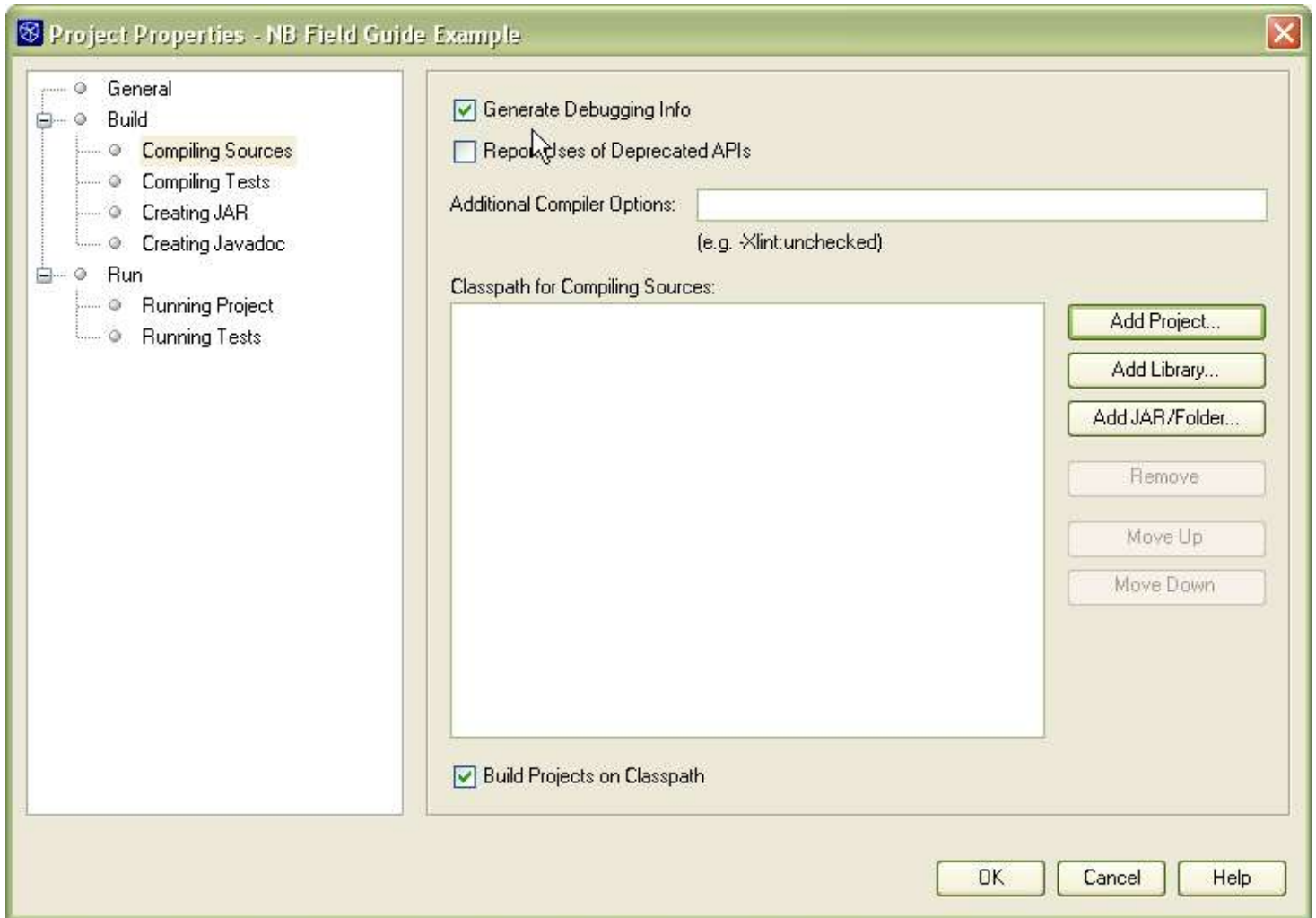
3. Click Add Project. In the file chooser, navigate to and select the folder of the project that you want to depend on and click Add Project JAR files.

4. Make sure the Build Projects on Classpath checkbox is selected.

5. Click OK to close the Project Properties dialog box.

NetBeans IDE Tip

When you add a project to another project's classpath, all sources and Javadoc documentation are added as well. .

Displaying and Hiding Projects

You can have multiple IDE projects open at once, whether or not they are connected to each other in any way. However, the only projects that you *need* to have open at any given time are the main project (which serves as an entry point for building, running, and debugging) and the projects that you are currently editing. Even if your main project depends on other projects, these projects do not need to be open if you are not actively working on them. Any dependencies that you have set up in a project's Properties dialog are honored whether the projects being depended on are open or closed.

To hide a project, right-click the project's node in the Projects window and choose Close Project.

To display a project, choose File | Open Project.

To open all of the projects that a project depends on, right-click the project's node in the Projects window and choose Open Required Projects.

Setting Up a Project to Work with Version Control

For projects of any size, using a version control (or "source control") system offers several advantages:

- You can save the history of versions of your files, which enables you to revisit previous versions of your files to help diagnose and fix problems in your programs.
- Multiple developers can work on the same source files and quickly incorporate changes that others make.
- Multiple users can work with the same project metadata and quickly incorporate changes that others make to the project metadata.

NetBeans IDE provides a user interface for the CVS, PVCS, and Visual Source Safe version control systems (VCSs). If you have one of these VCSs installed on your system, you can call version control commands on your files from within the IDE.

Though version control features are not tightly coupled with the IDE's project system, working with VCSs in the IDE is straightforward:

- Use the Versioning Manager to point the IDE to the version control project (or, in CVS parlance, working directory).
- Call version control commands on the files from the various "explorer" windows (Projects, Files, Versioning, and Favorites).
- Call IDE project commands from nodes in the Project window. (You have to explicitly set up an IDE project.)

NetBeans IDE Tip

When you create an IDE project, the IDE does not automatically recognize whether you are using version control. Likewise, when you set up a version control project in the IDE, the IDE does not set up an IDE project.

To start working with version-controlled sources:

1. Choose Versioning | Versioning Manager.
2. In the Versioning Manager dialog box, click Add.
3. Select one of the version control systems offered in the Profiles combo box and then fill in the requested information about your version control project, including where the version controlled sources are located and where the executable for the version control system is located.

If CVS is your version control system, you can have the IDE use a CVS executable that you have on your system or the IDE's own built-in CVS client.

Once you have your VCS set up with the IDE, the Versioning window opens as a tab in the space also occupied by the Projects and Files windows. In the Versioning window, you can run

any available version control commands, though other IDE commands are not available there. You probably will do most of your work in the Projects windows, where project-related commands are available. Version control commands are available in file nodes in the Projects window, but you need to use the Files window or Versioning window if you want to run a version control command on a folder.

> **NetBeans IDE Tip**
>
> If your project uses a version control system other than those named above, you can check http://vcsgeneric.netbeans.org/profiles/index.html to see if there is a profile for your VCS and download it.

Versioning and Sharing Project Metadata

If you are part of a team that is developing an application for which, in addition to sharing sources, you also share a build environment, you can put the project metadata under version control. For example, you could change a project's dependencies, check in those changes to your version control system, and have others on your team update those changes.

The metadata for NetBeans IDE projects is stored in XML and properties files, which makes it fairly easy to include in your version control project and share with other users.

To put your IDE project metadata under version control, add the following to your VCS repository:

- The project's `nbproject` folder and the files within it, such as `project.xml`, `build-impl.xml`, and `project.properties`. (The `private`, folder, which holds settings specific to your local copy of the IDE, is automatically ignored).
- The project's `build.xml` file.

Both the `nbproject` folder and the `build.xml` file are visible from the Files window.

If the project metadata files are in the same directory as your sources, you can simply add them within your current VCS setup. (For example, in CVS, you would merely run the Add and Commit commands.)

If the project metadata files are in a different directory (as they typically are when you create a project using existing sources) or not everybody on your team is using NetBeans IDE, you might need to create to a new area for them in your VCS repository (for example, in CVS, you would probably create a new CVS module).

> **NetBeans IDE Tip**
>
> If you create an IDE project within an existing VCS working directory and then establish dependencies in that project to other projects and libraries that are also within the VCS working directory, the IDE assumes that all other people working on these projects will be accessing all of these resources from the same relative location. As a result, the IDE stores all references to the locations of these resources as relative paths in the `project.properties` file, which makes this information sharable between all users who check out the projects from the VCS. This saves users who check out these projects from having to set these classpath dependencies themselves.
>
> If the projects (including sources and `nbproject` folder) are not all within a VCS working directory, the references are stored as absolute paths in the `private.properties` file, which is excluded from the VCS. Anybody who does a VCS check out of such a project (and then uses the File | Open Project command) is then prompted to use the

Resolve Reference Problems command to manually set the references to the resources.

Resolving Merge Conflicts in Project Metadata files

If your project's properties are modified by two or more developers simultaneously merge conflicts can occur when you update your project from your version control system.

If merge conflicts occur in `project.xml` or `project.properties`, you should be able to resolve them manually in the Source Editor.

Should a merge conflict occur in your `build-impl.xml` file (which itself would probably be a result of a merge conflict in the `project.xml` file), do the following:

1. Resolve the merge conflict in the `project.xml` file and commit the change.
2. Delete the `build-impl.xml` file.
3. In the Project's window, right-click the project's main node and choose Close Project.
4. Reopen the project by choosing New | Open Project. The `build-impl.xml` file will be regenerated open reopening the project.

Compiling a Project

Once you have set up your project, created your source files, and set up any necessary dependencies in the project's Properties dialog box, you can compile your project by choosing Build | Build Main Project (F11).

By default, when you build a standard Java project in the IDE, the following occurs:

- All modified files in the project are saved.
- Any uncompiled files (or files that have been modified since they were last compiled) under the Source Packages node are compiled.
- A JAR file (or other archive, depending on the project category) is created with your classes and other resources.
- Any projects that the main project depends on are built. (In fact, these projects are built first.)

When you build your project, output on the progress of the Ant script is printed to the Output window.

NetBeans IDE Tip

If you merely want to compile your project (without building JAR files or running any other post-compile steps), you can create a shortcut to the `compile` target in the project's build script. Open the Files window and expand the `build.xml` node. Right-click the `compile` target and choose Create Shortcut. In the wizard, you can designate a menu item, toolbar button, and a keyboard shortcut for the target. The shortcut only works for that particular project.

Compiling Selected Files or Packages

If you have a larger project, you might want to be able to compile a few files without rebuilding the entire project.

To compile a single file, right-click the file's node and choose Compile File.

To compile a package, right-click the package's node in the Projects window and choose Compile Package.

Doing a Fresh Build

The Build Project command produces its outputs incrementally. When you rebuild a project, any files that have changed or been added are recompiled and the JAR file (or other output file) is repackaged with these changed classes. As your project evolves, sometimes you need to explicitly erase your compiled classes and distributables, particularly if you have removed or renamed some source files. Otherwise, classes that are no longer part of your sources might linger in your compiled outputs and cause problems with running your program.

To clean your project, right-click the node for your project in the Projects window and choose Clean Project. This command deletes the `build` and `dist` folders in your project.

To do a fresh build, choose Build | Clean and Build Main Project (Shift-F11).

Stopping a Build

If you start a build and want to halt it before it completes:

1. Choose Window | Execution.
2. In the Execution window that has just opened up in the output area, right-click the node for the running build and choose Terminate Process.

The build will terminate, but any artifacts created by the build will remain.

Changing the Location of Compiled Classes and JAR Files

By default, the compiled class files and the packaged distributable (e.g. a JAR or WAR file) are placed in folders (named `build` and `dist`, respectively) parallel to the `src` folder in your project. If you would like the outputs to appear elsewhere or simply change the names of the folders, you can do so in the `project.properties` file.

NetBeans IDE Warning

If you want to have your class files placed in the same directory as their source files, you need to override the project's `clean` target so that only class files are deleted from the directory when the Clean command is run. You also need to adjust the IDE's Ignored Files property. See the Compiling Classes Into the Same Directories As Your Sources topic below.

To access the `project.properties` file:

1. Open the Files window by clicking the Files tab (or pressing Ctrl-2).
2. Expand the project's `nbproject` folder and open the `project.properties` file.

Table 1.3 lists properties that determine where your outputs are created.

Table 1.3

Property	Specifies the Directory . . .
`build.classes.dir`	Where compiled class files are created. You can change the name of the output folder here
`build.test.classes.dir`	Where unit test files are created (for general Java projects).

Property	Specifies the Directory . . .
`build.test.results.dir`	Where unit test results are stored (for general Java projects).
`build.generated.dir`	Where servlets and other dynamically generated files are placed when the project is built (for Web projects).
`build.web.dir`	Where subdirectories for class files and web files are when the project is built. (for Web projects).
`build.dir`	Where the above directories for the previously listed properties are placed. You can also change the value of this property. If you remove this property from the values of other properties, you must also override any targets that use this property (such as the `clean` target).
`dist.jar`	Where the project's JAR file is created. Also specifies the name of the JAR file. (for general Java projects).
`dist.war`	Where the project's WAR file is created. Also specifies the name of the WAR file (for Web projects).
`dist.javadoc.dir`	Where Javadoc for the project is generated when you run the Generate Javadoc for Project command.
`dist.dir`	Where the generated Javadoc documentation and the project's JAR file or WAR file are placed.

NetBeans IDE Tip

If you move either of these folders outside of the project folder, they will
no longer be visible in the Files window. You can remedy this by setting
up the Favorites window to display them. Choose Window | Favorites.
Then right-click the Favorites node and choose Add to Favorites.
Navigate to the folder that you want to display and click Add.

Compiling Classes Into the Same Directories As Your Sources

If your build environment requires that you compile your classes into the same directories as your sources, you must :

- Modify the properties that represent any build outputs that you want to have moved.
- Override the IDE's `do-clean` target so that it only deletes the class files in your source directory and not your sources as well.
- Optionally, update the IDE's Ignore Files property so that compiled class files are not displayed in the Projects window.

Here is the step-by-step procedure for having your classes compiled into the same directories as your sources for a general Java project:

1. Open the Files window by clicking the Files tab (or pressing Ctrl-2).
2. Expand the project's `nbproject` folder and open the `project.properties` file.
3. Change the `build.classes.dir` property to point to your source directory.
4. Open the `build-impl.xml` file and copy the `do-clean` target.
5. Paste the `do-clean` target into your `build.xml` file.
6. Modify the `do-clean` target so that only class files are deleted. For example, you might replace

```
<delete dir="${build.dir}"/>
```

with

```
        <delete includeEmptyDirs="true">
            <fileset dir=".">
                <include name="${build.classes.dir}/**/*.class"/>
            </fileset>
        </delete>
```

7. If you do not want compiled class files to display in the Projects and Files windows, choose Tools | Options, expand IDE Configuration | System, and select the System Settings node. Select the Ignored files property and add to the regular expression to designate the files to ignore. For example, you might add `|class$` to the end of the expression.

Investigating Compilation Errors

If you get an error when compiling a class, you can jump to the source of the error by double-clicking the hyperlinked error line in the Output window or pressing F12. If there are multiple errors, you can cycle through them by pressing F12 (next error) or Shift-F12 (previous error).

Saving Build Output

You can save build output to a file by right-clicking in the Output window and choosing Save As.

By default, every command that you run in the IDE that uses Ant re-uses the same tab in the Output window and thus the output from the previous command is cleared every time you run a new build command. If you would like to preserve the Ant output in the UI, you can configure the IDE to open up a new tab every time you run a build command.

To preserve the output from all build commands in the IDE:

1. Choose Tools | Options. Then navigate to and select the Building | Ant Settings node.
2. Deselect the Reuse Output Tabs property.

Running a Project

Once you have set up your project, created your source files, and set up any necessary dependencies in the project's Properties dialog box, you can run your project in the IDE by choosing Run | Run Main Project (F6).

By default, when you run a general Java project in the IDE, the following occurs:

- All modified files in the project are saved.
- Any uncompiled files (or files that have been modified since they were last compiled) under the Source Packages node are compiled.
- A JAR file with your classes and other resources is created (or updated).
- Any projects that the main project depends on are built. (In fact, these projects are built first.)
- The project is run inside of the IDE. For general Java projects, the IDE uses the designated main class as the entry point.

When you build your project, output on the progress of the Ant script is printed to the Output window.

Setting or Changing the Project Main Class

To run a general Java project, you have to have an executable class designated as the entry point. You can designate the main class:

- In the New Project wizard when creating your project from the Java Application project template.
- In the project's Properties dialog box. (In the Projects window, right-click the project's node and choose Properties. Then select the Run | Running Project node and fill in the Main Class field with the fully qualified name of the main class, without the `.java` extension.)

If you try to run a project that has no main class set, you will be prompted to choose one from a list of executable classes.

Setting Runtime Arguments

If you need to pass arguments to the main class when you are running your project, you can do it through the project's Properties dialog box.

1. Right-click the project's node in the Projects window and choose Properties.
2. Select the Run | Running Project node and add the arguments in the Arguments field.

Setting the Runtime Classpath

By default, the classpath for running the project inherits the compilation classpath. If you need to alter the classpath just for runtime, you can make adjustments in the Running Project section of your project's Properties dialog box.

1. Right-click the project's node in the Projects window and choose Properties.
2. Select the Run | Running Project node.
3. Use the Add Project (for other IDE projects), Add Library (for collections of JARs, sources and Javadoc that you have designated in the Library Manager), or Add JAR/Folder buttons to make additions to your classpath.

Writing Your Own Manifest for Your JAR File

When you build a general Java project, a JAR file is created with a simple manifest with entries for `Manifest-Version`, `Ant-Version`, and `Created-By`. If the project has a main class designated, that main class is also designated in the JAR manifest.

If you have other entries that you would like to add to the manifest of a project created from the Java Application Template or Java Project With Existing Sources template, you can add them directly in the `manifest.mf` file that sits next to the `build.xml` file. Go the Files window and double-click the the `manifest.mf` file's node to edit in the Source Editor.

You can also specify a different manifest file for the project to use. To specify a custom manifest:

1. Open the Files window by clicking the Files tab (or by pressing Ctrl-2).
2. Expand the project's `nbproject` folder and open the `project.properties` file.
3. In the `manifest.file` property, type the manifest's name. If the manifest is not in the same folder as the `build.xml` file, include the relative path from the `build.xml` file.

NetBeans IDE Tip

You can write the manifest in the IDE's Source Editor using the Empty File template. Open the Files window. Then right-click the project's main folder and choose New | Empty File.

For projects created from the Java Library template, a basic manifest is generated , but no editable copy of that manifest appears in the Files window. If you would like to specify a different manifest for a project created from the Java Library template, simply add the `manifest.file` property to the project's `project.properties` file and point it to the manifest that you have created.

Filtering Contents Packaged into Outputs

If you have any files that appear within your source packages, but which you do not want packaged in the project's distributable, you can have those files filtered out of the project's distributable.

By default, `.form` files and `.java` files are filtered from the output JAR file of general Java projects. In web projects, `.form`, `.java`, and `.nbattrs` files are filtered out by default.

To change the filter for the JAR file contents of a general Java project:

1. Right-click the project's node in the Projects window and choose Properties.
2. Select the Build | Creating JAR node and modify the regular expression in the Filter field.

For web projects,

1. Right-click the project's node in the Projects window and choose Properties.
2. Select the Build | Packaging Project node.
3. Use the Add Filters button to add individual filters in the form of regular expressions.

NetBeans IDE Tip

`.form` files are created for classes that you create with the IDE's Form Editor. The IDE uses these files to regenerate the design view of those classes, however they are not necessary for the packaged application. If you delete a `.form` file, the corresponding `.java` remains and you can edit the code, but you can no longer use the IDE's Form Editor to change the form.

`.nbattrs` files are files NetBeans IDE creates to hold information about given directories. Projects created in NetBeans IDE 4.0 and later are unlikely to have these files, but legacy projects might.

Running a Project From Outside of the IDE

Since the IDE's project commands are based on Ant scripts and properties files, you can run these targets from outside of the IDE.

Assuming you have Ant installed on your system, you simply can call the the the `build.xml` file or one of its targets from the command line by changing directories to the directory holding `build.xml` and typing `ant`.

If your project uses optional Ant tasks that are defined within the IDE, you might need to do some manual configuration. For example, if you have a target that depends on JUnit, you need to place the JUnit binary in your Ant classpath.

Setting Up a Headless Build Environment

If you have a large nested project structure that you want to run from outside of the IDE, such as when you are doing production builds of your application, you will probably need to make some adjustments to set up headless builds.

The following are some of the things you need to do:

- Set up Ant (version 1.6 or higher) on your system. You can either download Ant or use the Ant JAR file that is included with the IDE (`ide4/ant/lib/ant.jar` in the IDE's installation directory). Visit http://ant.apache.org/ to download or get more information on Ant. Make sure that the command-line version of Ant that you are using is running from the same version of the Java platform that your project is using. You

- Make JUnit available to Ant. You can do this by adding the JUnit JAR file to *AntHome*/lib. You can use the IDE's copy of JUnit, which is located in *NBHome*/ide4/modules/ext and is named according to version number (e.g. In NetBeans IDE 4.0, it is `junit3.8.1.jar`).

- Make sure that any libraries that the IDE uses when building your project are accessible from the build script that the build machine uses. These libraries are specified in the `build.properties` file that is located in your IDE user directory. You can find the IDE's user directory by choosing Help | About, clicking the Detail tab, and looking at the User Dir value.

Modifying the IDE-Generated Build Script

The build scripts that the IDE generates for you in standard projects are based on common scenarios that work for many development situations. But if the script does not do what you want to do, you can add to, override parts of, or change the script entirely.

When you create a standard project, the IDE generates two build scripts: `build-impl.xml` and `build.xml`.

The `build-impl.xml` file is generated based on the type of project template you started with and is regenerated based on any changes that occur in the project's associated `project.xml` file. Do not edit `build-impl.xml` directly since any changes you make there would be lost should the file ever need to be regenerated.

The `build.xml` file serves as the master build script. By default, it has no targets of its own. It imports `build-impl.xml`. You can freely edit `build.xml`.

To add a target to the build script of a standard project:

1. In the Files window, expand the project's main folder.
2. Double-click the `build.xml` file to open it in the Source Editor.
3. Below the `import` element, type any targets that you would like to add to the build script.

To override part of a standard project's build script:

1. In the Files window, expand the projects `nbproject` folder and open the `build-impl.xml` file.
2. Copy the target that you want to override, paste it into the project's `build.xml` file, and then make your modifications to it there.

When a target appears in both the `build-impl.xml` and `build.xml` files, the version in the `build.xml` file takes precedence.

If you are merely modifying the `depends` attribute of the target, you do not have to copy the whole body of the target. You can copy just the `target` element without its sub-elements. The sub-elements will be imported from the `build-impl.xml` file.

NetBeans IDE Tip

To make customization of build scripts easier, the generated build scripts include several empty targets that are called from main targets. For example, the compile target depends on `pre-compile` and `post-compile` targets. If you need to add any steps to the build process just before or after compilation of your files, you can customize these targets without having to add to the `depends` attribute of the `compile` target.

Running a Specific Ant Target from the IDE

You can run any Ant target within the IDE by expanding the build script's node in the Files window, right-clicking the target and choosing Run Target.

You can stop the target in the Execution window. Choose Window | Execution. In the Execution window, right-click the node that represents the running target and choose Terminate Process.

Completing Ant Expressions

The IDE has a "completion" feature to reduce the number of keystrokes needed when editing an Ant script. When you use the completion feature, the Source Editor gives you a choice of how to complete the current word with a popup dialog box.

Activate the completion feature by typing the first few characters of an element or attribute and then pressing Ctrl-Space.

If there is only one possible completion, the missing characters from the word are filled in. (For attributes, =" is generated as well.)

If there are multiple possible matches, you can choose a completion by scrolling through the list and then pressing the Enter key once the correct word is selected. Keep typing to narrow the number of selections in the list.

The order of the selection list is generally "smart", meaning that elements or attributes that are commonly used in the given context are put at the top.

For example, you can enter the following target:

```
<target name="default" depends="dist,javadoc" description="Build whole
project."/>
```

by doing the following:

1. Typing `<t`
2. Pressing Ctrl-Space and then Enter to select `target`
3. Pressing the spacebar
4. Pressing Ctrl-Space and then Enter to select `name` (`name="` is inserted into the script)
5. Typing `default"`
6. Pressing Ctrl-Space and then Enter to select `depends` (`depends="` is inserted into the script)
7. Typing `dist,javadoc"`
8. Pressing Ctrl-Space and then Enter to select `description` (`description="` is inserted into the script)
9. Typing `Build whole project."/>`
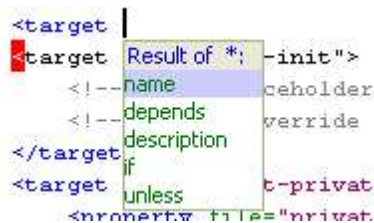
Making a Menu Item or Shortcut for a Specific Ant Target

If you have a target in your build script that you use often, but which is not represented by a menu item or keyboard shortcut, you can create such a menu item and a shortcut.

1. In the Files window, expand the projects `nbproject` folder and expand the `build-impl.xml` node.
2. Right-click the target that you want to map and choose Create Shortcut.

A wizard opens which enables you to set a menu item, toolbar item, and a keyboard shortcut for the target. Since the IDE records these custom menu items and shortcuts as Ant files, you can customize the way the shortcuts are called by selecting the Customize Generated Ant Code checkbox in the wizard.

Shortcuts set in this way are not global. They apply only to the build script on which they are set.

NetBeans IDE Tip

The wizard prevents you from overriding existing custom shortcuts, but it does not prevent you from overriding standard IDE shortcuts. If you inadvertently use a key combination for your new shortcut that is used elsewhere in the IDE, your new shortcut takes precedence.

Removing a Custom Menu Item or Shortcut

If you have added a menu item or shortcut for an Ant target and would like to remove it, you can do so manually. The IDE stores these customizations in your user directory in small XML files that work as mini-Ant scripts.

To remove a shortcut to an Ant target:

1. In your IDE's user directory, expand the `config` folder. (If you are not sure where your IDE user directory is, choose Help | About and click the Details tab to find out.)

2. Look in the following subfolders (and possibly their subfolders) for an XML file with the name of the Ant target or the keyboard:

    • Menu (if you have created menu items)
    • Shortcuts (if you have created keyboard shortcuts)
    • Toolbar (if you have added a toolbar item)

3. Manually delete the XML file that represents the shortcut.

Changing a Custom Menu Item or Shortcut

You can change custom menu item for an Ant target by either editing the mini Ant script for the target or by deleting the mini Ant script for the target and creating a new shortcut. For example, you can change the build script that a shortcut applies to by changing the `antfile` attribute in the shortcut's script.

To manually edit the file for the custom menu item or shortcut:

1. In your IDE's user directory, expand the `config` folder. (If you are not sure where your user IDE directory is, choose Help | About and click the Details tab to find out.)

2. Look in the following subfolders (and possibly their subfolders) for an XML file with the name of the Ant target or the keyboard:

    • Menu (if you have created menu items)
    • Shortcuts (if you have created keyboard shortcuts)
    • Toolbar (if you have added a toolbar item)

3. Double-click the node for the menu item or shortcut to open it in the Source Editor.