NetBeans IDE Field Guide

Table of Contents

NetBeans IDE Fundamentals

This chapter provides a general overview of both the workflow in the IDE and the key parts of the IDE. Once you finish this chapter, you should have a solid understanding of the IDE's principles and be able to take advantage of the IDE's central features.

If you are already familiar with NetBeans IDE (4.0 or higher), you can probably skim this chapter or skip it altogether. Subsequent chapters will revisit most of this material in greater depth to answer more involved questions and provide additional details that you can use to squeeze more productivity out of the IDE.

Creating a Project

Before you can do any serious work in the IDE, you need to set up a project. The project essentially sets up a context for you to write, compile, test, and debug your application. This context includes the classpath, folders your sources and tests, and a build script with targets for

compiling the application, running tests, building JAR files (or other types of distributable archive files).

You can choose from a variety of project template categories, which are grouped according to the technology you are basing your application on (e.g. standard Java, J2EE Web tier, J2EE Enterprise tier, J2ME).

Within the template categories, you have templates for new applications and for setting up an IDE project for existing applications you are working on. The New Project wizard provides a description for each template.

The "With Existing Sources" templates in each category enable you to set up standard IDE projects around applications you have been developing in a different environment.

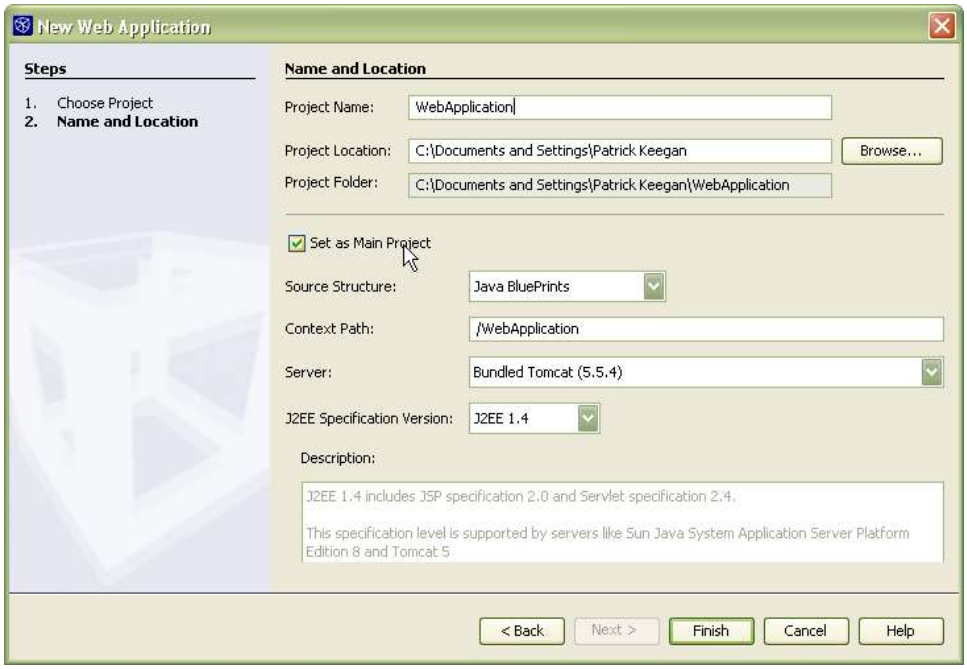
The "With Existing Ant Script" templates in each category take that a step further and enable you to set up a project based entirely on any existing Ant script. This approach requires some manual configuration to get some IDE features (such as debugging) to work with the Ant script, but the pay-off is that you can get the IDE to work with any project structure, even if it does not adhere to the conventions of a standard IDE project.

To set up a project:

1. Choose File | New Project.
2. In the wizard select a template for your project and complete the wizard.

   The fields that you are asked to fill in depend on the template. Typically you need to specify a location for the project (or, in the case of projects that use existing sources, where the sources are located). Web, Enterprise, and Mobility projects also include fields relevant for those specific types of applications.

Figure 2-1



New Project Wizard, Web Application template, Choose Project page

When you create a project, typically the IDE does the following things for you:

- Creates a source tree with a skeleton class inside.

- Creates a folder for unit tests.
- 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.

You can find more information on setting up projects in Chapter 3.

Projects Window

The Projects window is essentially the command center for your work. It is organized as a tree view of nodes that represent parts of your project. It provides an entry point for your files as well as configuration options for the application you are developing.

In addition to displaying nodes for the files in the application that you are developing, it also displays nodes for libraries relevant to your application. The Libraries node shows the version of the JDK you are developing against as well as any other libraries you are basing your project on.

The Projects window presents your project in “logical” form. That is, it represents the units of your application conceptually (rather than literally). For example, Java sources are grouped into packages without nodes for each level of file hierarchy. Files that you do not normally need to view, such as compiled Java classes and project metadata files, are hidden. This makes it easier to access the files you are most regularly work with. In addition, the Projects window provides a Libraries node, which gives you a view of your classpath.

If you want to browse the physical structure of the project, including the project metadata, compiled classes, JAR files, and other files created in builds, open the Files window.

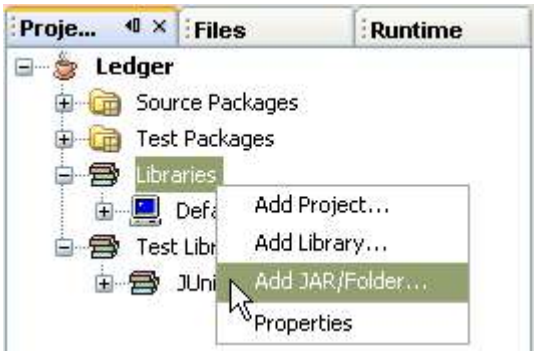
Configuring the Classpath

When you create a project, the IDE sets up a default classpath for you based on the project template you are using. If you have other things to add to the classpath you can do so through the Libraries node of the project.

In fact, the IDE distinguishes between several types of classpaths depending on project type, such as compilation classpath, test compilation classpath, running classpath, and test running classpath. The compilation classpath typically serves as a base for the other classpaths (i.e. other classpaths inherit what is in the compilation classpath).

To add an item to the compilation classpath (and thus the other classpaths as well), right-click the project's Libraries node and choose Add JAR/Folder.

Figure 2-2



Projects window. Adding a JAR file to the classpath.

NetBeans IDE Tip

When you right-click the Libraries node, you also can choose Add Project or Add Library. When you add a project, you add the project's output (such as a JAR file) to the classpath.

If you choose Add Library, you can add one of the “libraries” recognized by the IDE's Library Manager. In this context, libraries are essentially just a convenient grouping of one or more JAR files, sources, and/or Javadoc documentation. You can manage existing libraries and designate new ones in the Library Manager, which you can open by choosing Tools | Library Manager.

You can edit other classpaths in the Properties dialog box for a project. To open the Project Properties dialog box, right-click the project's node in the Projects window and choose Properties. In the dialog box, click the Libraries node and use the customizer in the right panel to specify the different classpaths.

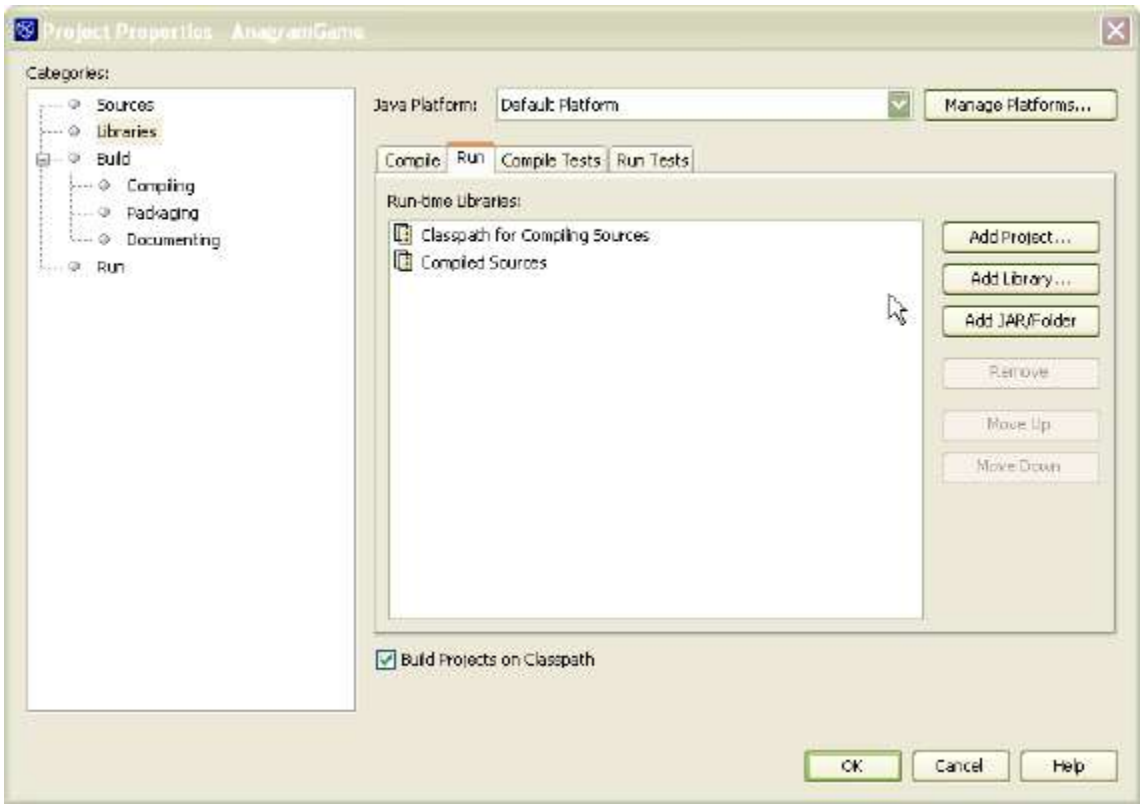


Figure 2-3
Project Properties dialog box, Libraries page.

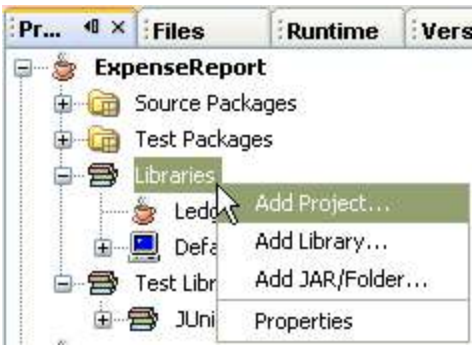
Creating a Sub-project

Though there is no explicit distinction in the IDE between a project and a "sub-project", you can create a hierarchy of projects by specifying dependencies between projects. For example, you might create an umbrella Web Application project that relies on one or more Java Class Library projects. For larger applications, you might have several layers of project dependencies.

To set dependencies between projects:

1. Right-click the project's Libraries node and choose Add Project.
2. In the file chooser that appears, navigate to the folder for the project you want to depend on. Project folders are designated with the  icon.

Once you have established this dependency, the distributed outputs (such as JAR files) of the "added" project become part of the other project's classpath.



NetBeans IDE Tip

There is no visual project/sub-project distinction in the IDE, but there is a concept of “main” project. The main project in the IDE is simply the one that the IDE treats as the entry point for the primary commands such as Build Main Project and Run Main Project. The current main project is indicated with bold font in the Projects window.

Figure 2-4
Projects window. Making one project depend on another.

There can be only one main project set at a time, though it is possible to have multiple projects open at the same time (including umbrella projects that serve as entry points for other applications you are developing).

You can make a project the main project by right-clicking its node in the Projects window and choosing Set Main Project.

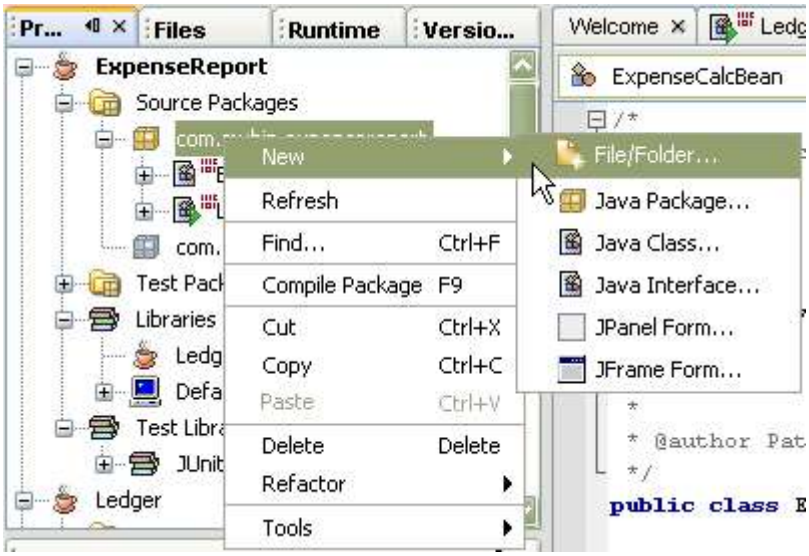
Creating and Editing Files

Once you have a project set up, you can add files to your project and start editing. You can add files to a project by creating them from the New File wizard.

To open the New File wizard, do one of the following:

- In the Projects window, right-click the Source Packages node (or one of the package nodes underneath it) and choose one of the templates from the New submenu. If none of the templates there suit you, choose File/Folder (as shown in Figure 2-5) to open up a wizard with a complete selection of available templates.
- Choose File | New File to open the New File wizard.

Figure 2-5
Projects window. Creating a new file



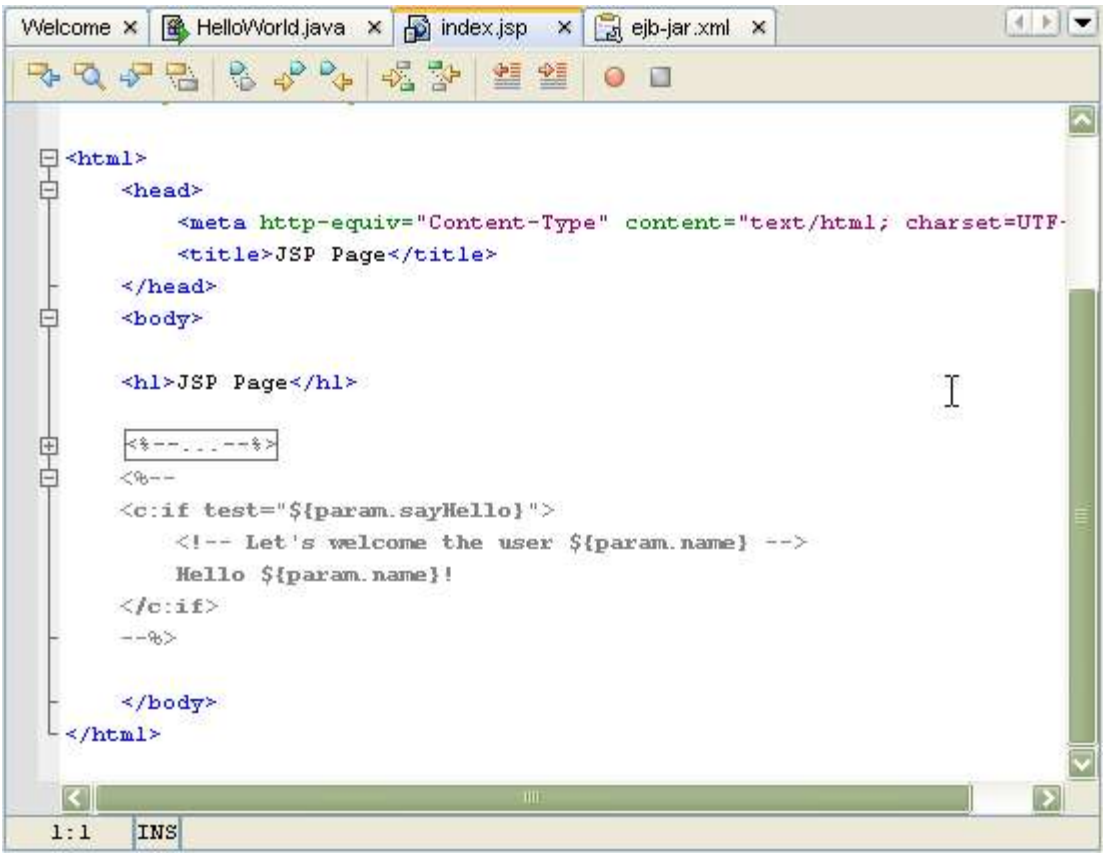
In the New File wizard, you can name the file and specify a folder. For Java classes, you can designate a period-delimited package name (as opposed to a slash-delimited folder name).

Once you complete the wizard, the file opens up in a tab in the area of the IDE to the right of the Projects window. For most templates, a Source Editor tab opens.

About the Source Editor

The Source Editor is the central area of the IDE where you write and generate code. The Source Editor is actually a collection of different types of editors with different purposes. There are text editors for different types of files, such as Java, JSP (as shown in Figure 2-6), XML, HTML, and plain text files.

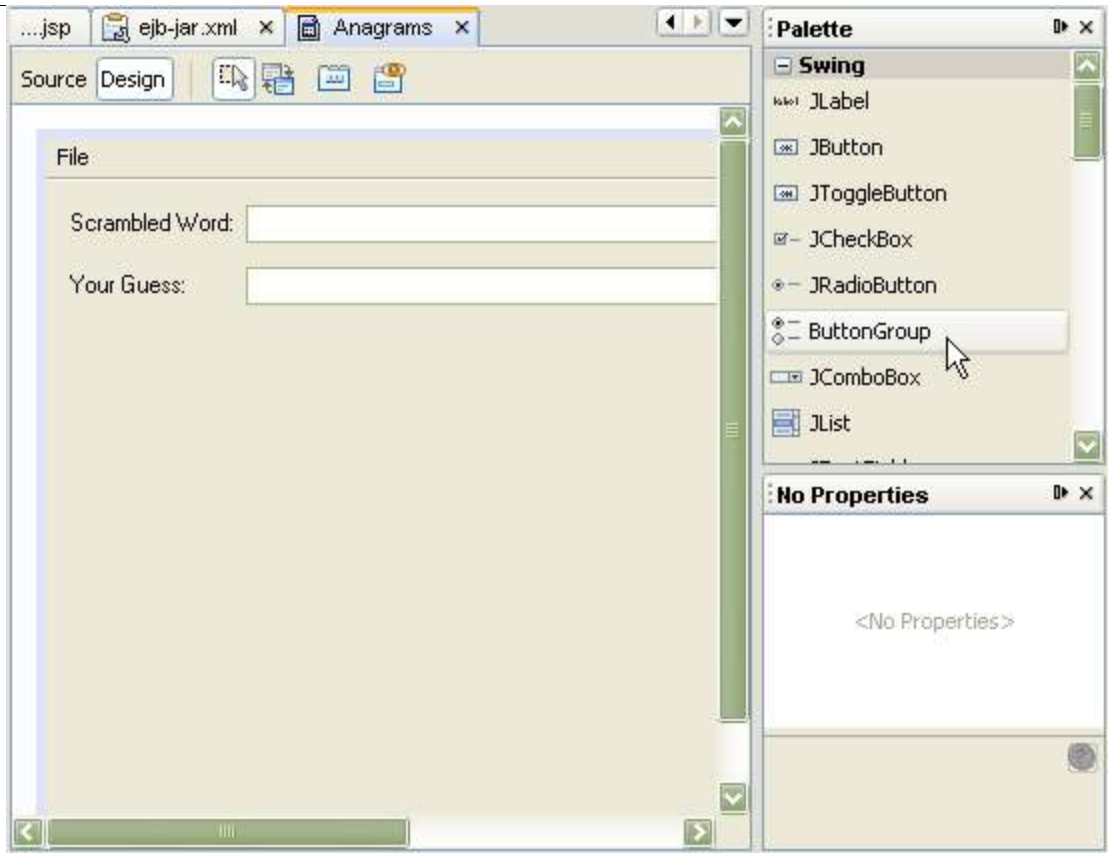
Figure 2-6
Source Editor window with JSP file open.



There are also visual editors for AWT and Swing forms, deployment descriptors, and other types of files, though it is also possible to edit the source of these types of files directly.

For example, GUI templates such as JPanel Form and JFrame Form open in a visual design area (as shown in Figure 2-7) along with Palette, Inspector, and Properties windows. You can click the Source button in design area's toolbar to access the file's source.

Figure 2-7
Form Editor Design View in the Source Editor window.



Setting Up and Modifying Java Packages

You can set up a Java package in the New Project and New File wizards. You can also create packages independently of these wizards.

To create a new package, right-click the Source Packages node within your project and choose New | Java Package. In the wizard, fill in a period-delimited package name (e.g. `com.mybiz.myapp`).

You can then move classes into this package by cutting and pasting or by dragging their nodes.

NetBeans IDE Tip

When you move classes, the Refactor Code for Moved Class dialog box opens and offers to update the rest of the code in the project to reflect the changed location of the class. Click Next to see a preview of the changes in the Refactoring window. Then click Do Refactoring to make the changes.

Compiling and Building

When you set up a project, the IDE provides a default classpath and compilation settings, so the project should be ready to compile as soon as you have added some classes to the project.

You can compile an individual files or packages by right-clicking its node and choosing Compile. But more typically you will "build" the entire project. Building, depending on project type, typically consists of compiling projects and sub-projects and creating outputs such as JAR files for each of those projects.

To build your project, right-click the project's node in the Projects window and choose Build Project. If that project is currently designated as the main project (the project name is bold in the Projects window), you can choose Build | Build Main Project or press F11. If you want to delete the products of previous builds before building again, choose Build | Clean and Build Main Project or press Shift-F11.

When you initiate a build, the IDE tracks the progress of the build in the Output window in the form of Ant output.

NetBeans IDE Tip

You can specify compiler options in the Project Properties dialog box. Right-click the project's node in the Projects window and choose Properties. Then click the Compiling node to enter the options.

Viewing Project Metadata and Build Results

In the Files window, you can view the physical structure of your project, including compiled class files, output JAR files, your build script, and other project metadata.

Project-related commands (such as Build Project) are not available from nodes in the Files window, but other "Explorer" type commands like Open, Cut, and Paste are.

The Files window is useful if you want to customize the build script for your project or you want to browse your project's outputs. You can also examine the contents of JAR files created by your project.

Figure 2-8 shows the structure of the HelloWorld application created in Chapter 1.

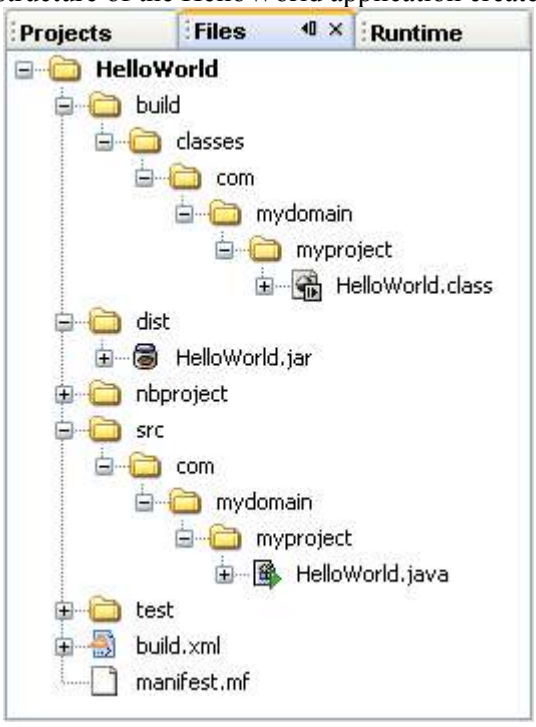


Figure 2-8
Files window "physical" view of the HelloWorld project

Navigating to the Source of Compilation Errors

If any compilation errors are reported when you compile or build, you can navigate straight to the source of the error by double-clicking the hyper-linked error in the Output window (as shown in Figure 2-9) or by pressing F12.

If you have multiple errors, you can use F12 (Next Error) and Shift-F12 (Previous Error) to navigate between the locations of the errors.

Figure 2-9

Output Window with compiler error showing.

Running

You can run the application you are developing from within the IDE by right-clicking the project's node and choosing Run Project or by pressing F6.

You can run an individual file by right-clicking the file in the Source Editor or the file's node in the Projects window and choosing Run File or pressing Shift-F6.

You can stop a running application by opening the Runtime window, expanding the Processes node, right-clicking the node for the running process, and choosing Terminate Process.

If you need to specify a main class for the project or you want to run the project with some arguments, you can specify these in the Project Properties dialog.

To for the project to run in the IDE. You can do so by right-clicking the project's node in the Projects window, choosing Properties, selecting the Run node, and entering

Creating and Running Tests

IDE project templates are set up with unit testing in mind. Most project types set up a folder next to the folder containing your sources for unit tests. You can have the IDE generate skeleton code for a class's unit test for a class and place it within the test folder with a package structure corresponding to that of the class the test is for.

To generate unit test code for a class:

1. In the Projects window, right-click the class you want to create a test for and choose Tools | JUnit Tests | Create Tests.
2. In the Create Tests dialog box, set a class name and location and specify the code generation options for the test.

   By default, the class name is filled in for you and corresponds to the name of the class being tested with `Test` appended to the name. The test classes is placed in a test folder that has the same package structure as your sources.

To run the selected project's tests, press Alt-F6 or choose Run | Test "*ProjectName*".

To run a test for a specific file, select the file in the Source Editor or Projects window and press Ctrl-F6 or choose Run | Run File | Test "*Filename*".

Debugging the Application

The IDE's debugger enables you to pause execution of your program at strategic points ("breakpoints") and check the values of variables, the status of threads, etc. Once you have paused execution at a breakpoint, you can step through code line by line.

To start debugging a program:

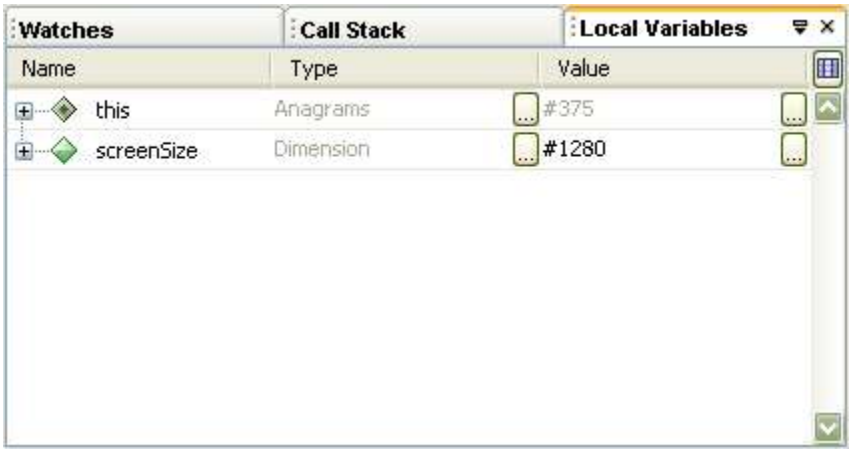
1. Make sure that the program you want to debug is currently set as the IDE's main project.

   The name of the main project is shown in bold font in the Projects window. You can make a project the main project by right-clicking its node and choosing Set Main Project.
2. Determine the point in your code where you want to start debugging and set a breakpoint at that line by clicking in the left margin of that line.

   The icon appears in the left margin to mark the breakpoint. In addition, the whole line is highlighted in pink.
3. Start the debugger by choosing Debug Main Project or pressing F5.

   The IDE builds (or rebuilds) the application and then opens up the Debugger Console in the bottom left of the IDE and the, Watches, Call Stack, and Local Variables windows in the lower right.
4. Click the Local Variables window (as shown in Figure 2-10) to view the values of any of the variables of the program that are currently in scope.

Figure 2-10



Debugger windows, with the Local Variables window in focus.

Integrating Version Control Commands

If you already use a version control system for your sources, you can easily integrate that system's commands into the IDE workflow. The IDE provides support for working with various version control systems.

The IDE acts as a graphical interface for version control client application you are already using. When you call version control commands from the IDE, the IDE passes those commands to the version control client, which then carries at the commands. The IDE also displays any output generated by the version control client.

In NetBeans IDE 4.1, setting up the IDE to work with versioned sources is separate from project setup. If you already have sources checked out from a version control system and want to make version control commands available within the IDE for a project, you need to register the versioned working directory with the IDE.

NetBeans IDE Tip

If you are using a post-4.1 version of the IDE, this process might be streamlined, so that the registration of the version control system in the IDE is coupled with the creation of the project.

To set up the IDE to work with your version control system:

1. Choose Versioning | Versioning Manager.
2. In the Versioning Manager dialog box, click Add.
3. Select the version control system you are using from the Profile combo box and point to the location of the working directory.

   If you have several projects within the same working directory, you can select the root directory to register version control for all of those projects at the same time.
4. Verify the server settings that the IDE fills in and add any missing settings.

   If you are using CVS as your version control system, you have the option of using using a client built-in the IDE instead of a separate CVS executable.
5. Click Finish to exit the wizard and then click Close to exit the Versioning Manager.
6. If you have not already done so, create an IDE project (or IDE projects) for your sources through the New Project wizard so that you can further develop these sources in the IDE.

NetBeans IDE Tip

If no profile is available in the wizard for the version control system you are using, you might be able to find a profile online at http://vcsgeneric.netbeans.org/profiles/index.html.

You can also create your own profile by choosing the Empty profile in the Versioning Manager and then customizing it to work with your version control system. See http://vcsgeneric.netbeans.org/doc/profiles/index.html for information on creating a profile for your version control system.

See Chapter 3: Setting Up a Project to Work with Version Control for more information on using version control with the IDE, including information on versioning your project metadata.

Once you have set up a version control working directory in this manner, the Versioning window appears in the area occupied by the Projects window. You can run version control

commands on the files from this window. However, you can not run project-related commands or do “Explorer” type things with files, such as open, copy, or paste.

If you already have set up an IDE project for those sources, a submenu with version control commands appears in the right-click menu of all of that project's nodes in the Projects window.

Managing IDE Windows

The IDE's window system is designed to provide a coherent and unobtrusive layout of the various window you need while enabling you to effortless adjust the layout as you work. These are some of the things you can do as you work:

- Resize windows by clicking on a window border and dragging it to the width or height you prefer.
- Maximize a window within the IDE by double-clicking on its tab. (You can revert to the previous window layout by again double-clicking on the tab.) You might find this feature particularly useful in the Source Editor.
- Move a window to a different part of the IDE by clicking on its tab and dragging it to a different part of the IDE.
- Use drag and drop to split a window.
- Make a window “sliding” by clicking its button. When you click this button, the window is minimized with a button representing that window placed on one of the edges of the IDE. You can mouse over the button to temporarily display the window, or you can click the button to open the window.