

# NetBeans IDE Field Guide

Copyright © 2005 Sun Microsystems, Inc. All rights reserved.

## Table of Contents

Integrating Existing Ant Scripts With the IDE.....	1
Creating a Free-Form Project.....	2
Mapping a Target to an IDE Command.....	3
Project Validation.....	4
Setting Up the Debug Project Command For a General Java Application.....	5
Mapping the Project's Sources to its Outputs.....	5
Creating the Debug Target .....	5
Mapping the Debug Target to the IDE Command.....	7
Setting Up the Debug Project Command For A Web Application.....	7
Creating the Debug Target .....	7
Mapping the Debug Target to the IDE Command.....	9
Setting Up Commands for Selected Files.....	9
Setting Up the Compile File Command.....	10
Creating the compile-selected-files Target.....	10
Mapping the compile-selected-files Target to the IDE Command.....	10
Handling Properties in the project.xml File.....	11
Setting Up the Run File Command.....	12
Creating the run-selected-file Target.....	12
Mapping the run-selected-file Target to the IDE Command.....	13
Setting Up the Debug File Command.....	13
Creating the debug-selected-file Target.....	14
Mapping the debug-selected-file Target to the IDE Command.....	14
Setting Up the Debugger's Fix Command.....	15
Creating the debug-fix Target.....	15
Mapping the debug-fix Target to the IDE Command.....	16
Changing the Target JDK for a Free-form Project.....	17
Making a Custom Menu Item for a Target.....	18
Debugging Ant Scripts.....	18

## Integrating Existing Ant Scripts With the IDE

The user interface for standard projects in NetBeans IDE is designed to handle common development scenarios, to be easy to use, and to encourage good programming practices (such as modular design with no circular dependencies). It is particularly well-suited to creating projects from scratch.

However, the standard user interface does not cover all scenarios, particularly for projects originally developed in other environments. If this is your case, you can take advantage of the IDE's tight integration with Ant to customize the IDE to work with your existing Ant build script.

If you already have your own build script and do not want to (or cannot) re-create it through the IDE, you can set up the IDE to use that build script by creating a free-form project. Free-form projects also might be preferable to standard projects if you create multiple outputs from individual source roots, or if there is anything too restrictive in standard projects.

#### **NetBeans IDE Tip**

Before committing to using your existing build script with the IDE, carefully consider whether you really need to use your own Ant script and make sure that it is not possible to replicate your existing build processes with a combination of standard IDE projects since standard IDE projects will likely be easier to maintain over a long-term basis.

The IDE's project system is based on Ant to the degree that even incremental commands (such as for compiling a single file) and other commands that you might specifically associate with IDE use (such as debugging) are defined in the build script. Build targets for these commands are generated by default in standard projects, but not in free-form projects. In free-form projects, it is left up to you to write the targets in whatever way will work with your project.

Using a build script that was created outside of the IDE entails the following steps:

- Creating a project in the IDE and using one of the free-form (“With Existing Ant Script”) templates.
- Mapping key existing build targets to the IDE commands that correspond to them (such as for compiling and running applications and running tests). You can create these mappings in the New Project wizard as you set up the project or later in the Project Properties dialog box.
- Registering classpath items (such as external source roots or JAR files) in the IDE project so that IDE-specific features such as code completion and refactoring work correctly. You can do this in the New Project wizard as you set up the project or later in the Project Properties dialog box.
- Writing new build targets for commands that are IDE-specific (such as debugging) and editing the project's `project.xml` file to map these targets to IDE commands.

---

## ***Creating a Free-Form Project***

1. Choose File | New Project.
2. Select a project category (such as General or Web) and then select the “With Existing Ant Script” template for that category.
3. In the Name and Location page of the wizard, fill in the Location field with the folder that contains the various elements of your project, such as your source folder, test folder, and build script.

If the build script is at the top level of the folder that you have specified, the rest of the fields are filled in automatically. If the build script is not found, fill in the Build Script field manually.

If you wish, you can change the other fields, such as Project Folder (which determines where the IDE stores metadata for the project).

4. In the Build and Run Actions page, specify targets for the listed IDE commands so that the IDE knows which target in your script to run when you choose the command in the IDE. You can click the combo box arrow next to each command to select from a list of all targets in the build script, or you can type a target into the combo box manually. If the IDE finds a likely target for the command, it is filled in automatically, though you can change it if you like. If you leave any of the commands blank, you can later fill them in manually outside of the wizard.

If the build script imports targets from other build scripts, those targets are not shown in the combo box list, though you can type one of those targets in manually.

Not all available IDE commands are given here. You can provide mappings for other IDE commands (such as Compile File) directly in the `project.xml` file. See “Mapping a Target to an IDE Command” below.

5. (For Web projects only) In the Web Sources page, specify the folder that contains your Web pages, fill in the context path for the application, and mark the J2EE Specification level.

6. In the Source Package Folders page, specify all of the folders that contain your top-level packages. For example, if the package structure of one of your source roots begins with `com`, choose the folder that contains `com` (e.g. `src`). Similarly, if you have any test packages, you can specify them here in the Test Package Folders area.

On this page, also be sure to set the Source Level to the appropriate JDK version. Even if this is already accounted for in your Ant script, you need to set the source level here so that IDE-specific features, such as proper Source Editor syntax highlighting and code completion work correctly.

7. In the Classpath page, specify any libraries or sources that each source root is compiled against. Doing this hooks up IDE features such as code completion and refactoring to your project. You do not have to specify the JDK on this page.

#### **NetBeans IDE Tip**

Most of the fields in the template wizard are also editable in the Project Properties dialog box, so you do not have to fill in each value immediately. For example, if you still have to write a target for an IDE command, you can later map the target to the command in the Build and Run pane of the Project Properties dialog box. To open up a Project Properties dialog box, open the Project's window, right-click the project's main node, and choose Properties.

---

## *Mapping a Target to an IDE Command*

When you use a general Java free-form template, the New Project wizard enables you to map specific build targets IDE commands, including the following:

- Build Project
- Clean Project
- Generate Javadoc
- Run Project

- Test Project

You can also use the Project Properties dialog box (Build and Run page) to map targets to these commands. Other commands (such as Debug Project, Compile File, Run File, Debug File, and Fix) need to have targets created for them and then be mapped in the `project.xml` file manually if you want them to work in the IDE.

See Table 13-1 for a list of commands that you can map to your build script. The IDE Action column gives the code name for the command that you use when mapping a build target to the command.

The next several topics provide examples of how to create Ant targets for specific commands and then map them to the IDE.

*Table 13-1*

Command	IDE Action
Build Project	build
Clean and Build Project	rebuild
Compile Selected Files	compile.single
Clean Project	clean
Run Project	run
Run Selected File	run.single
Redeploy Project (for web applications)	redeploy
Test Project	test
Test File	test.single
Debug Test For File	debug.test.single
Debug Project	debug
Debug File	debug.single
Fix	debug.fix
Step Into	debug.stepinto
Generate Javadoc	javadoc

## Project Validation

The IDE comes bundled with XML schemas for the `project.xml` files and automatically validates them every time you edit and save them. If you make invalid changes to a `project.xml` file, the IDE reports the errors in the Output window.

If you would like to inspect the schema yourself, you can view them online. See Table 13-2 for a list of the schema used.

*Table 13-2: Free-Form Project Schema*

Schema	Description
<a href="http://www.netbeans.org/ns/freeform-project/1.xsd">http://www.netbeans.org/ns/freeform-project/1.xsd</a>	Defines the <code>&lt;general-data&gt;</code> part of the <code>project.xml</code> file for all free-form project types.
<a href="http://www.netbeans.org/ns/freeform-project-java/1.xsd">http://www.netbeans.org/ns/freeform-project-java/1.xsd</a>	Defines the <code>&lt;java-data&gt;</code> part of the <code>project.xml</code> file for all free-form project types.
<a href="http://www.netbeans.org/ns/freeform-project-web/1.xsd">http://www.netbeans.org/ns/freeform-project-web/1.xsd</a>	Defines the <code>&lt;web-data&gt;</code> part of the <code>project.xml</code> file for web applications.

---

## Setting Up the Debug Project Command For a General Java Application

To get debugging to work with a free-form project, you need to:

- Make sure that the target you use for compiling specifies `debug="true"` when calling the `javac` task.
- Creating a mapping in the IDE between the project's sources and the project's outputs, so that the debugger knows which sources to display when you are stepping through the running program.
- Add a target to your Ant script for the command, making sure that all necessary path elements for the command are defined, either in the build script or in a `.properties` file that is called by the script.
- Map the target to the IDE through the project's `project.xml` file.

### Mapping the Project's Sources to its Outputs

In NetBeans IDE 4.1 free-form projects, the IDE does not automatically know which sources are associated with compiled classes that you run. Therefore, to get the IDE's debugging features to work, you need to create this mapping between the sources and the outputs.

To map a project's sources to its outputs:

1. In the Projects window, right-click the project's node and choose Properties.
2. In the Project Properties dialog box, select the Output node.
3. In the right pane, click the Add JAR/Folder button and navigate to the folder or JAR file that contains the compiled classes corresponding to the source root selected in the Source Packages Folder field.

If you have multiple source roots, repeat this step for each source root listed in the Source Packages Folder field.

#### NetBeans IDE Tip

When you write your debug target, the outputs you specify here will need to be referenced as part of the `classpath` attribute of the `jpda`start (or `nbjpdacconnect`) task.

### Creating the Debug Target

Below is a sample target for debugging that can be used when you want to start a general Java application in the debugger.

---

```
<target name="debug" depends="compile" if="netbeans.home"
description="Debug Project">
  <nbjpdastart name="Display-Name-for-Debugged-App"
    addressproperty="jpda.address" transport="dt_socket">
    <classpath refid="run.classpath"/>
    <sourcepath refid="debug.sourcepath"/>
  </nbjpdastart>
  <java fork="true" classname="fully-qualified-main-class-name">
    <jvmarg value="-Xdebug"/>
    <jvmarg value="-Xnoagent"/>
    <jvmarg value="-Djava.compiler=none"/>
    <jvmarg
      value="-Xrunjdpw:transport=dt_socket,address=${jpda.address}"/>
  </java>
</target>
```

```

    <classpath refid="run.classpath"/>
  </java>
</target>

```

This target can be used in your project with some customizations to fit your environment. Look at Table 13-3 for further details and some things to look out for.

*Table 13-3: Details of the Debug Target for a General Java Application*

Target, Task, Attribute or Property	Description
depends	Attribute where you specify targets that need to be run before the current target is run.
netbeans.home	Ant property that is loaded by any instance of Ant that runs inside of the IDE. The <code>if="netbeans.home"</code> attribute ensures that the target is only be run if it is called from within the IDE.
nbpjpdastart	A special task bundled with the IDE to debug programs within the JPDA debugger.
addressproperty	An attribute of <code>nbpjpdastart</code> that defines the property that holds the port that the debugger is listening on. (The IDE automatically assigns the port number to the property.) The value of the property that is defined there (in this case, <code>jpda.address</code> ) is passed as the value for the <code>address</code> sub-option of the <code>-Xrunjdpw</code> option.
transport	An attribute specifying the debugging transport protocol to use. You can use <code>dt_socket</code> on all platforms. On Windows machines, you can also use <code>dt_schem</code> .
classpath	An attribute of <code>nbpjpdastart</code> that represents the classpath used for debugging the application. In the given example, it is mapped to the property that holds the execution classpath.
sourcepath	An attribute of <code>nbpjpdastart</code> used to specify the explicit location of source files that correspond to JAR files in your classpath. If you have used the IDE's Library Manager to associate source with JAR files, you should not need to set this attribute.
fork	Attribute of the <code>java</code> element that determines whether or not the debugging process is launched in a separate virtual machine. For this target, the value must be <code>true</code> .
classname	Attribute of the <code>java</code> element. It points to the class that the debugger executes. For the Debug Project target, this attribute should be the fully qualified name of the main class of the project.
jvmarg	Parameter of the <code>java</code> element for providing arguments to the JVM. The arguments provided in the example are typical for debugging J2SE applications with the JPDA debugger.

The example uses the `refid` attribute to reference two path elements (`run.classpath` and `debug.sourcepath`) that need to be defined in elsewhere in your build script. For example, `run.classpath` could be defined as in the snippet below:

```

<path id="run.classpath">
  <pathelement path="{javac.classpath}">
    <pathelement path="{build.classes.dir}">
  </path>

```

The paths for `javac.classpath` and `build.classes.dir` would need to be defined in their own path elements. If the path element needs to reference a physical location, the `location`

attribute could be used to specify a directory relative to the Ant project's base directory. For example:

---

```
<path id="build.classes.dir">
  <pathelement location="classes">
</path>
```

---

## Mapping the Debug Target to the IDE Command

You need to provide a mapping for the Ant target so that the Debug Project command works on your project when that project is selected.

To map the target to the IDE command:

1. In the Files window, expand the project's folder, expand the `nbproject` folder, and then open the `project.xml` file.
2. Under the `<ide-actions>` line, add a mapping for the Debug Project command. The mapping might look something like the following example.

---

```
<action name="debug">
  <target>debug</target>
</action>
```

---

---

## Setting Up the Debug Project Command For A Web Application

To get debugging to work with a free-form Web project, you need to.

- Make sure that the target you use for compiling specifies `debug="true"` when calling the `javac` task.
- Make sure you have a target in your build script for deploying your Web application and that you have the target mapped to the Deploy command. You can provide this mapping in the wizard when creating the project or on the Build and Run page of the Project Properties dialog box.
- Creating a mapping in the IDE between the project's sources and the project's outputs, so that the debugger knows which sources to display when you are stepping through the running program.
- Add a target to your Ant script for attaching the debugger to a running Web application, making sure that all necessary path elements for the command are defined, either in the build script or in a `.properties` file that is called by the script.
- Map the target to the IDE through the project's `project.xml` file.

## Creating the Debug Target

Below is a sample target for debugging that can be used when you want to attach the debugger to your Web application to.

---

```

<target name="debug" depends="compile, deploy" if="netbeans.home"
description="Debug Project">
  <nbjpdaconnect name="Display-Name-for-Debugged-App"
    host="${jpda.host}" address="${jpda.address}"
    transport="dt_socket">
    <classpath refid="${run.classpath}"/>
    <sourcepath refid="${debug.sourcepath}"/>
  </nbjpdaconnect>
  <nbbrowse url="${client.url}"/>
</target>

```

---

This target can be used in your project with some customizations to fit your environment. Look at Table 13-4 for further details and some things to look out for.

*Table 13-4: Details of the Debug Target for a Web Application*

Target, Task, Attribute, or Property	Description
depends	Attribute where you specify targets that need to be run before the current target is run.
netbeans.home	Ant property that is loaded by any instance of Ant that runs inside of the IDE. The <code>if="netbeans.home"</code> attribute ensures that the target is only be run if it is called from within the IDE.
nbjpdaconnect	A special task bundled with the IDE to enable attaching the JPDA debugger to a running application.
host	An attribute of <code>nbjpdaconnect</code> that specifies the host name of the machine that the debugged application is running on.  In this example, the <code>jpda.host</code> property is used. The value of this property would need to be defined elsewhere in the build script or in a <code>.properties</code> file that is referenced by the build script.
address	An attribute of <code>nbjpdaconnect</code> that specifies the port that the debugger is listening on.  In this example, the <code>jpda.address</code> property is used. The value of this property would need to be defined elsewhere in the build script or in a <code>.properties</code> file that is referenced by the build script.
transport	An attribute specifying the JPDA debugging transport protocol to use. You can use <code>dt_socket</code> on all platforms. On Windows machines, you can also use <code>dt_schem</code> , though the IDE and the debugged application would both have to be running on the same machine. An attribute specifying the debugging transport protocol to use. You can use <code>dt_socket</code> on all platforms. On Windows machines, you can also use <code>dt_schem</code> .
classpath	An attribute of <code>nbjpdaconnect</code> that represents the classpath used for debugging the application. In the given example, it is mapped to the property that holds the execution classpath.
sourcepath	An attribute of <code>nbjpdaconnect</code> used to specify the explicit location of source files that correspond to JAR files in your classpath.
nbbrowse	Element that specifies a web page to be opened in the default browser that is specified by the IDE.  In this example, the <code>client.url</code> property is used. The value of this property would need to be defined elsewhere in the build script or in a <code>.properties</code> file that is referenced by the build script.

The example uses the `refid` attribute to reference two path elements (`run.classpath` and `debug.sourcepath`) that need to be defined in elsewhere in your build script. For example, `run.classpath` could be defined as in the snippet below:

---

```
<path id="run.classpath">
  <pathelement path="{javac.classpath}">
  <pathelement path="{build.classes.dir}">
</path>
```

---

The paths for `javac.classpath` and `build.classes.dir` would need to be defined in their own path elements. If the path element needs to reference a physical location, the `location` attribute could be used to specify a directory relative to the Ant project's base directory. For example:

---

```
<path id="build.classes.dir">
  <pathelement location="classes">
</path>
```

---

### Mapping the Debug Target to the IDE Command

You need to provide a mapping for the Ant target so that the Debug Project command works on your project when that project is selected.

1. In the Files window, expand the project's folder, expand the `nbproject` folder, and then open the `project.xml` file.
2. Under the `<ide-actions>` line, add a mapping for the Debug Project command. The mapping might look something like the following example.

---

```
<action name="debug">
  <target>debug</target>
</action>
```

---

---

### *Setting Up Commands for Selected Files*

To get file-specific commands (such as Compile File, Run File, and Debug File) to work in the IDE, you need to do the following:

1. Add a target to your Ant script for the command, making sure that all necessary path elements for the command are defined, either in the build script or in a `.properties` file that is called by the script.
2. Map the target to the IDE through the project's `project.xml` file, and include a `context` element to provide the IDE a way of passing the currently selected files to the Ant script.
3. Define any properties that needed in the project's `project.xml` file, either in the `project.xml` file or in a properties file that is referenced from the `project.xml` file.

See the next few topics for examples of how to set up the commands.

---

## Setting Up the Compile File Command

To be able to compile selected files or packages in a free-form project in the IDE, you need to write an Ant target for the command and then map that target in the project's `project.xml` file.

### Creating the compile-selected-files Target

Below is a sample target for compiling selected files:

---

```
<target name="compile-selected-files" depends="compile">
  <fail unless="selected-files">Must set
    property 'selected-files'</fail>
  <mkdir dir="${classes.dir}"/>
  <javac srcdir="${src.dir}" destdir="${classes.dir}" debug="true"
    includes="${selected-files}">
    <classpath refid="javac.classpath"/>
  </javac>
</target>
```

---

In this example, the `selected-files` property picks up the files or the package that you have selected in the IDE. The value of `selected-files` is passed from the `project.xml` file when you choose the Compile File command. If no files are selected in the IDE when Compile File is chosen (and therefore no value is passed to `selected-files` from the `project.xml` file), the target does not complete successfully.

This example build target uses the `refid` attribute to reference the `src.dir`, `classes.dir`, and `javac.classpath` path elements, which need to be defined in elsewhere in your build script. For example, `javac.classpath` could be defined as in the snippet below:

---

```
<path id="javac.classpath">
  <pathelement location="libs">
</path>
```

---

### Mapping the compile-selected-files Target to the IDE Command

1. In the Files window, expand the project's folder, expand the `nbproject` folder, and then open the `project.xml` file.
2. Within the `<ide-actions>` element, add a mapping for the Compile File command. The mapping might look something like the following example.

---

```
<action name="compile.single">
  <target>compile-selected-files</target>
  <context>
    <property>selected-files</property>
    <folder>${src.dir}</folder>
    <pattern>\.java$</pattern>
    <format>relative-path</format>
    <arity>
      <separated-files>,</separated-files>
    </arity>
  </context>
</action>
```

---

See Table 13-5 for a description of the parts of the `<action>` element in the `project.xml` file.

*Table 13-5: Details of the `project.xml` `<action>` Element*

Target, Task, or Property	Description
<code>context</code>	Parameter that the IDE uses to collect information about the files that the command is to be run on.
<code>property</code>	Parameter that defines the name of the property that is passed the names of the currently selected files or packages in the IDE when the Compile File command is chosen. A target can then reference this property to determine what files to run the command on. For example, the sample <code>compile-selected-files</code> target above references the <code>selected-files</code> property to determine which files are to be compiled. You can provide any value for this parameter that you wish.
<code>folder</code>	Parameter that enables you to specify the directory in which the <code>compile-selected-files</code> target is enabled. In this example, the value is provided as a reference to the <code>src.dir</code> property.
<code>pattern</code>	Parameter that contains a regular expression to limit the kinds of files that the target can be run on. In this example, only files with the <code>.java</code> extension are passed to the target.
<code>format</code>	Parameter that specifies the form in which the selected files are passed to the target. Possible values for this element are: <code>relative-path</code> – passes the file name with its path relative to the folder specified by the <code>folder</code> element <code>relative-path-noext</code> – like <code>relative-path</code> except that the file name is passed without its extension. <code>java-name</code> – like <code>relative-path-noext</code> except that periods (.) are used in instead of slashes to delimit the folders in the path. <code>absolute-path</code> – passes the file name with its absolute path <code>absolute-path-noext</code> - like <code>absolute-path</code> except that the file name is passed without its extension
<code>arity</code>	Parameter that specifies whether single or multiple files can be passed to the target. Possible values are: <code>&lt;separated-files&gt;delimiter&lt;/separated-files&gt;</code> - Multiple files can be passed. <code>&lt;one-file-only&gt;</code> - only one file can be passed.

### Handling Properties in the `project.xml` File

In the example above, the `src.dir` property is referenced from the `project.xml` file. This property needs to be defined, either in a file referenced by the `project.xml` file or directly in the `project.xml` file.

In the `project.xml` file, properties are defined in the `<properties>` element, which belongs between the `<name>` and `<folders>` elements. Within the `<properties>` element, use the `<property>` element and its `name` attribute to define an individual property, or use the `<property-file>` element to designate a `.properties` file. Note that this syntax is different than Ant's syntax for defining properties.

After completing the New Project wizard where you have specified the build script to use, something like the following is generated in your `project.xml` file.

---

```
<properties>
  <property name="project.dir">C:\MyNBProjects\SampleFreeForm</property>
  <property name="ant.script">${project.dir}/build.xml</property>
</properties>
```

---

You can add additional property or property file references within the `<properties>` element. Since the `src.dir` property in this example is likely a property that can also be used in your build script, it might be useful to set the property in one place and let both the build script and `project.xml` file use it. A reference to a `.properties` file from the `project.xml` file would look something like the following line:

```
<property-file>${project.dir}/MyProject.properties</property-file>
```

### NetBeans IDE Tip

File paths that are referenced from the `project.xml` file are relative to the project folder. For path references to work the same for both the `project.xml` file and the build script, the build script needs to be in the project folder (which is actually folder that *contains* the `nbproject` folder).

If the build script is in a different folder, you might solve the path discrepancy by moving the `project.dir` property in the example above to a properties file that is common for both the `project.xml` file and build script and use that property in the values of other properties that you define for your classpath, source path, etc. For example, you could create a property to specify the location for compiled class files and give it the value `${project.dir}/build/classes`.

---

## Setting Up the Run File Command

To be able to run a selected file in a free-form project in the IDE, you need to write an Ant target for the command and then map that target in the project's `project.xml` file.

### Creating the run-selected-file Target

Below is a sample target for running selected files:

---

```
<target name="run-selected-file" depends="compile-selected-files"
description="Run Single File">
  <fail unless="selected-file">Must set
    property 'selected-file'</fail>
  <java classname="${selected-file}">
    <classpath refid="run.classpath"/>
  </java>
</target>
```

---

In this example, the `selected-file` property picks up the file that you have selected in the IDE. The value of `selected-file` is passed from the `project.xml` file when you choose the Run File command.

This example also assumes you have a working `compile-selected-files` target (like the example in the Setting Up the Compile File Command on page XXX), though it is also possible to have the target depend on a different compile target you have set in your script.

The example uses the `refid` attribute to reference a run classpath that must be defined elsewhere in the script with `run.classpath` specified as the `id` attribute of a `path` element. For example, `run.classpath` could be defined as in the snippet below.

---

```
<path id="run.classpath">
  <pathelement path="{javac.classpath}">
  <pathelement path="{build.classes.dir}">
</path>
```

---

The paths for `javac.classpath` and `build.classes.dir` would need to be defined in their own path elements. If the path element needs to reference a physical location, the `location` attribute could be used to specify a directory relative to the Ant project's base directory. For example:

---

```
<path id="build.classes.dir">
  <pathelement location="classes">
</path>
```

---

## Mapping the run-selected-file Target to the IDE Command

1. In the Files window, expand the project's folder, expand the `nbproject` folder, and then open the `project.xml` file.
2. Within the `<ide-actions>` element, add a mapping for the Run File command. The mapping might look something like the following example, where `path-to-file-with-debug-target` is replaced with the name of the file that contains the debug target.

---

```
<action name="run.single">
  <target>run-selected-file</target>
  <context>
    <property>selected-file</property>
    <folder>${src.dir}</folder>
    <pattern>\.java$</pattern>
    <format>java-name</format>
    <arity>
      <one-file-only/>
    </arity>
  </context>
</action>
```

---

See Table 13-5 for a description of the parts of the `<action>` element in the `project.xml` file. See Handling Properties in the `project.xml` File on page XXX for information on calling properties from the `project.xml` file.

---

## Setting Up the Debug File Command

To be able to debug a selected file in a free-form project in the IDE, you need to write an Ant target for the command and then map that target in the project's `project.xml` file.

## Creating the debug-selected-file Target

Below is a sample target for debugging a selected file:

---

```
<target name="debug-selected-file" depends="compile-selected-files"
if="netbeans.home" description="Debug a Single File">
  <fail unless="selected-file">Must set
    property 'selected-file'</fail>
  <nbjpdastart name="${selected-file}" addressproperty="jpda.address"
    transport="dt_socket">
    <classpath refid="run.classpath"/>
    <sourcepath refid="debug.sourcepath"/>
  </nbjpdastart>
  <java fork="true" classname="${selected-file}">
    <jvmarg value="-Xdebug"/>
    <jvmarg value="-Xnoagent"/>
    <jvmarg value="-Djava.compiler=none"/>
    <jvmarg
      value="-Xrunjdpw:transport=dt_socket,address=${jpda.address}"/>
    <classpath refid="run.classpath"/>
  </java>
</target>
```

---

In this example, the `selected-file` property picks up the file that you have selected in the IDE. The value of `selected-file` is passed from the `project.xml` file when you choose the Debug File command.

This example also assumes you have a working `compile-selected-files` target (like the example in Setting Up the Compile File Command on page XXX), though it is also possible to have the target depend on a different compile target you have set in your script.

The example uses the `refid` attribute to reference two path elements (`run.classpath` and `debug.sourcepath`) that need to be defined in elsewhere in your build script. For example, `run.classpath` could be defined as in the snippet below:

---

```
<path id="run.classpath">
  <pathelement path="${javac.classpath}">
  <pathelement path="${build.classes.dir}">
</path>
```

---

The paths for `javac.classpath` and `build.classes.dir` would need to be defined in their own path elements. If the path element needs to reference a physical location, the `location` attribute could be used to specify a directory relative to the Ant project's base directory. For example:

---

```
<path id="build.classes.dir">
  <pathelement location="classes">
</path>
```

---

See Table 13-3 for a description of the various parts of the target.

## Mapping the debug-selected-file Target to the IDE Command

1. In the Files window, expand the project's folder, expand the `nbproject` folder, and then open the `project.xml` file.
2. Within the `<ide-actions>` element, add a mapping for the Debug File command. The mapping might look something like the following example.

---

```
<action name="debug.single">
  <target>debug-selected-file</target>
  <context>
    <property>selected-file</property>
    <folder>${src.dir}</folder>
    <pattern>\.java$</pattern>
    <format>java-name</format>
    <arity>
      <one-file-only/>
    </arity>
  </context>
</action>
```

---

See Table 13-5 for a description of the parts of the `<action>` element in the `project.xml` file. See [Handling Properties in the `project.xml` File on page XXX](#) for information on calling properties from the `project.xml` file.

---

## Setting Up the Debugger's Fix Command

To be able to use the debugger's Fix feature in a free-form project, you need to write a special Ant target for the command and then map that target in the project's `project.xml` file. The Ant target needs to call the IDE's custom `nbpjdareload` task that the IDE uses to reload the fixed code into the debugged program's JVM. See [Chapter 4, Fixing Code During a Debugging Session](#) for information on using Fix.

### Creating the debug-fix Target

Below is a sample target for running the Fix command:

---

```
<target name="debug-fix" description="Reload Fixed Code Into the
Debugger">
  <javac srcdir="${src.dir}" destdir="${classes.dir}" debug="true">
    <classpath refid="javac.classpath"/>
    <include name="${selected-file}.java"/>
  </javac>
  <nbpjdareload>
    <fileset dir="${classes.dir}">
      <include name="${selected-file}.class"/>
    </fileset>
  </nbpjdareload>
</target>
```

---

In this example, the `selected-file` property picks up the file that have selected in the IDE. The value of `selected-file` is passed from the `project.xml` file when you choose the `Run | Fix` command.

The example uses the `refid` attribute to reference the `javac.classpath` path element, which needs to be defined elsewhere in your build script. For example, `javac.classpath` could be defined as in the snippet below:

---

```
<path id="javac.classpath">
  <pathelement location="libs">
</path>
```

---

## Mapping the debug-fix Target to the IDE Command

1. In the Files window, expand the project's folder, expand the `nbproject` folder, and then open the `project.xml` file.
2. Within the `<ide-actions>` element, add a mapping for the Fix command. The mapping might look something like the following example.

---

```
<action name="debug.fix">
  <target>debug-fix </target>
  <context>
    <property>selected-file</property>
    <folder>${src.dir}</folder>
    <pattern>\.java$</pattern>
    <format>relative-path-noext</format>
    <arity>
      <one-file-only/>
    </arity>
  </context>
</action>
```

---

## Inside the Generated Build Scripts

Here is a look at all of the pieces of the project metadata and how they work together.

When you create a standard project, the IDE creates the files listed in Table 13-6.

Table 13-6

File	Description
<code>build.xml</code>	This script is the master build script for the project. When you call a project-related command from the IDE, the IDE calls a target in this file. You can freely edit this file if you want to make customizations to your build process. This file is generated when you create the project but is not regenerated afterwards. Any configuration that you do in the IDE that is relevant to the build script is reflected in the <code>build-impl.xml</code> file, which is imported by <code>build.xml</code> . If a target with the same name appears in both <code>build.xml</code> and <code>build-impl.xml</code> , the target in <code>build.xml</code> takes precedence.
<code>nbproject/build-impl.xml</code>	Included in standard projects (but not free-form projects), this file contains the meat of the build script and is imported by <code>build.xml</code> . It is generated based on the type of project and the contents of that project's <code>project.xml</code> file. Do not edit this file.
<code>nbproject/project.properties</code>	Included in standard projects (but not free-form projects), this file contains values that the build script uses when building your project. These values include things such as the name and location of the directory for your compiled files, and references to properties set elsewhere in the project. Changes that you make in your Project Properties dialog box are propagated here. You can also modify this file directly in the Source Editor.

File	Description
<code>nbproject/project.xml</code>	Provides the basic metadata that determines how the project works in the IDE. For standard projects, this file determines how <code>build-impl.xml</code> and <code>project.properties</code> are generated. For free-form projects, this file serves as the glue between your build script and the IDE's user interface. This file is generally editable but you are only likely to need to edit it for free-form projects.
<code>nbproject/genfiles.xml</code>	Used by the IDE to help keep track of the state of the build script (such as whether the <code>build-impl.xml</code> file needs to be regenerated, etc.). Do not edit this file.
<code>nbproject/private/private.properties</code>	Holds properties that are specific to your installation of the IDE. These properties are not to be versioned, but they can be used by headless builds run on your machine.

### NetBeans IDE Tip

There is also a `build.properties` file that is created in your IDE's user directory. This file holds properties for the location of libraries that are packaged with the IDE, any libraries that you specify with the IDE's Library Manager, and any versions of the Java platform that you register with the IDE's Java Platform Manager. The `private.properties` file references the `build.properties` file with its `user.properties.file` property.

---

## Changing the Target JDK for a Free-form Project

If you want to set a target JDK for your project that differs from the JDK that the IDE is running on, you must specify the JDK version in *both* of the following places:

- The Ant script for any pertinent tasks, such as `javac`. This ensures that the built targets (and the IDE commands that call them) work correctly.
- The Project Properties dialog box for the project. This ensures that IDE-specific functions, such as code completion and the Javadoc popup, work correctly.

For the `javac` task, you could do this by including the `source` and `target` options when you call `javac`. For example, the call to the task might look something like the example below. The `javac.source` and `javac.target` properties would need to be specified elsewhere in the script or in a `.properties` file with the values set to the appropriate JDK version (e.g 1.3, 1.4, or 1.5).

---

```
<javac srcdir="${src.dir}" destdir="${classes.dir}"
      debug="true" source="${javac.source}"
      target="${javac.target}"
      <classpath refid="javac.classpath"/>
</javac>
```

---

To change the target JDK in the project's properties:

1. In the Projects window, right-click the project's node and choose Properties.
2. In the Properties window, select the Sources node and select the target JDK from the Source Level combo box.

3. If you do not already have a JDK registered with the IDE's Platform Manager that matches the source level you have set, choose Tools | Java Platform Manager and add that JDK. See Chapter 3, *Changing the Version of the JDK That Your Project is Based On* for more information on the Java Platform Manager.

---

## *Making a Custom Menu Item for a Target*

If your build script has a target that does not exactly correspond with any of the available menu items, you can create a custom menu item for it. The menu item is then available when you right-click the project's node in the Projects window.

To create a custom menu item for a target:

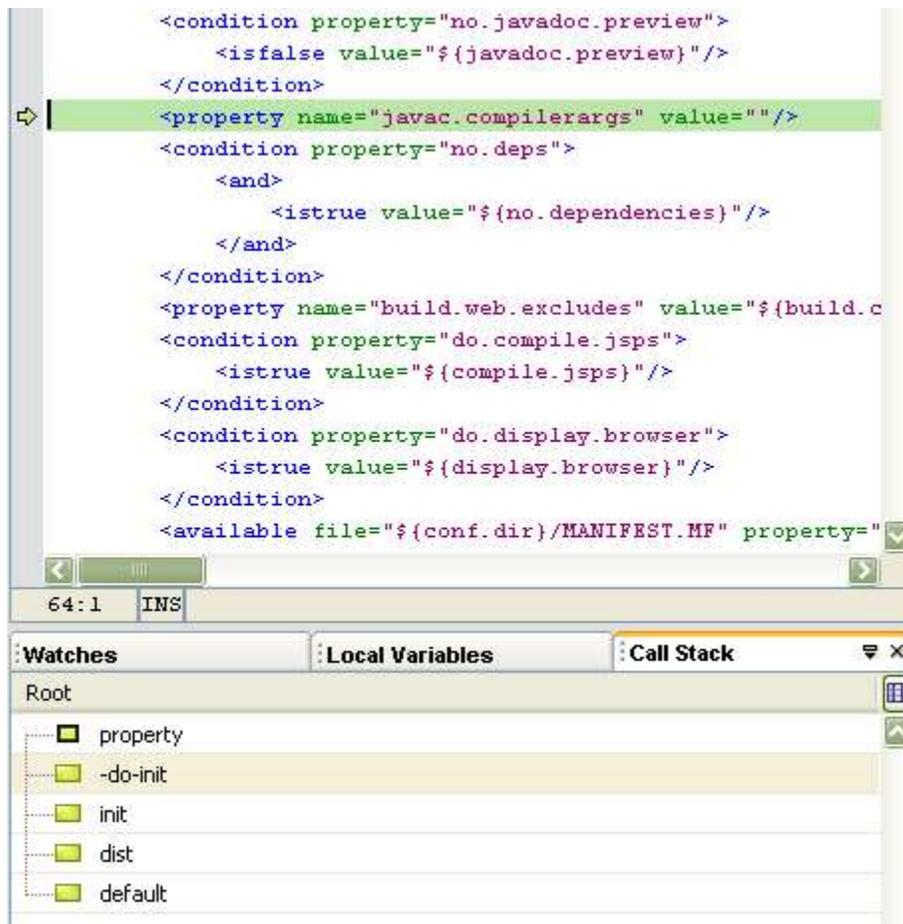
1. In the Projects window, right-click the project's node and choose Properties.
2. In the Properties window, select the Build and Run node.
3. Click the Add button next to the Custom Menu Items table to add a blank row to the table.
4. Fill in the target name in the Ant Target column and the name for the menu item in the Label column.

---

## *Debugging Ant Scripts*

If you need to troubleshoot an Ant script or you just would like a tool to help you make sense of a working script, you can use the IDE's new Ant Debugger module. The module essentially plugs into the IDE's visual debugging framework and provides most of the debugging features you are used to for Java files and applies them to Ant scripts. For example, you can:

- Use the Step Into, Step Over, Step Out, and Continue commands to trace execution of the script or a specific target. This feature is particularly useful to help you untangle the order in which nested targets (and even nested scripts) are called.
- Use the Call Stack window to monitor the current hierarchy of nested calls.
- Set breakpoints.
- View the values of properties in the Local Variables window.
- Set a watch on a property.



**Figure 13-1**  
*Ant Debugger*

The Ant Debugger is not included in the standard NetBeans IDE 4.1 download, but you can add it to your installation through the IDE's Update Center. Choose Tools | Update Center and then go through the wizard to pick and download the module. When presented with a list of “update centers” in the wizard, first look in the “Beta” update center.

Most likely, the module will be a standard part of the NetBeans IDE release in versions after 4.1.

Once the Ant Debugger module is installed, to start debugging a build script:

1. Open the Files window and navigate to the build script.
2. Right-click the build script and choose Debug Target and then the name of the target you want to debug.

The program counter goes to the line where the target is declared and stops. You can then step through the target with any of the normal debugging commands.