Make Ant easy with Eclipse

Write, build, and debug code in the Ant editor easily

Skill Level: Introductory

Prashant Deva (pdeva@placidsystems.com)
Founder
Placid Systems

18 Apr 2006

Eclipse can make working with Apache Ant easier. Discover the Ant integration features in the Eclipse integrated development environment (IDE), and learn how to write, build, and debug code in Eclipse through the Ant editor.

Section 1. Before we start

About this tutorial

Apache Ant is considered the Holy Grail of build tools in the Java™ development world. Most Java projects worth their salt have some sort of custom build process attached to them in the form of an Ant build script. Therefore, every worthwhile Java IDE must have some sort of support built in for Ant. Eclipse, a favorite IDE, doesn't disappoint: It provides extensive built-in support for Ant.

Learn how to make use of the features present in Eclipse to write and debug Ant files, and also how to use Ant files as builders.

Prerequisites

This tutorial assumes a basic knowledge of creating Ant scripts and working with the Eclipse platform.

System requirements

To try the examples, you need the Eclipse SDK from Eclipse.org running on Java Virtual Machine (JVM) V1.4 or later. Download Java technology from Sun Microsystems or from IBM.

Section 2. Working with Ant

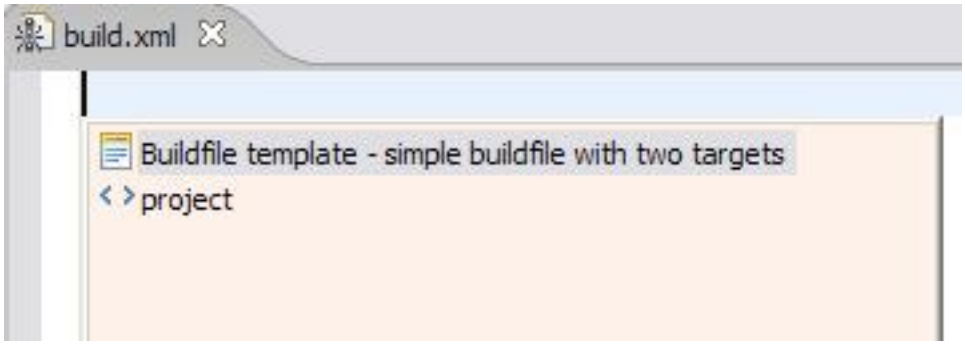
Create a new Ant buildfile

Begin by adding a new Ant file to your project, which you'll use for the rest of the tutorial:

1. Open the Package Explorer
2. Right-click any Java Project and click **New > File**
3. In the New File window, type `build.xml` as the file name

The file is created, and the Ant editor opens. Now, add some content to the file. Click anywhere in the editor and press **Ctrl+Space**. A completion proposal containing an option called Buildfile template -- simple buildfile with two targets appears (see Figure 1). Click this to add a sample project containing two targets to the file.

Figure 1. Using the Buildfile template



Now that we have some content, let's take a closer look at the Ant editor.

Section 3. The Ant editor

Discover the best of the Ant editor features: highlighting, code completion, folding, renaming, and marking occurrences and problems.

Highlighting

For ease of use, the editor shows each element of the buildfile in a different color. Comments appear in a different color from the attributes, and so on. You can even customize the color for each element type.

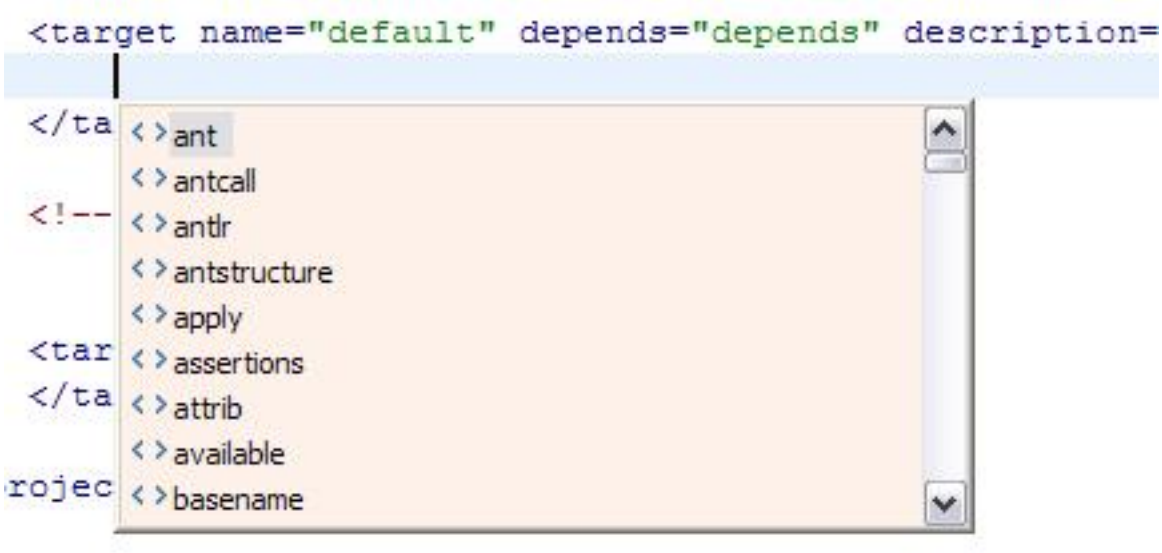
To change highlighting colors, complete these steps:

1. Click **Window > Preferences > Ant > Editor**
2. Click the **Syntax** tab
3. On the resulting page, select colors for each element type

Code completion

The Ant editor provides comprehensive code-completion functionality to help you type your Ant buildfiles quickly. Click inside a target definition, then press **Ctrl+Space** to see a list of all available tasks. After you select a task, the editor inserts the opening and closing tags automatically (see Figure 2).

Figure 2. The list of tasks



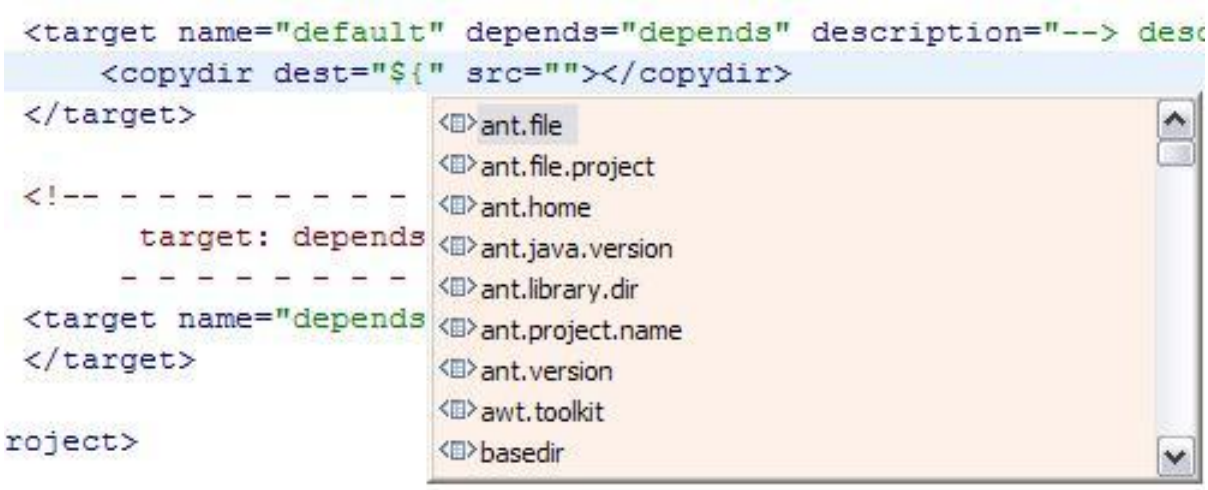
But that's not all. The Ant editor's code-completion abilities go far beyond automatic tag insertion. The editor is aware of the targets defined in our buildfile. So, for example, when we want to insert the name of a target -- say, typing the `default` attribute of a project or the `depends` attribute of a target, pressing **Ctrl+Space** shows a list of all available targets we can fill in (see Figure 3).

Figure 3. Available targets

The editor even knows the properties defined in our buildfile. So when we're typing

the value of an attribute, after typing the initial $ (dollar sign), we can press **Ctrl+Space** to see a list of all the properties defined in our buildfile *and* all the system properties (see Figure 4).

Figure 4. List of available properties



Another code-completion feature in the Ant editor is Code Templates (see Figure 5). We used this when we used the Buildfile template to add sample content to our buildfile. Several templates are available in the Ant editor, and with them, we can quickly enter target definitions, property definitions, and more. Note that after we apply a template, a box appears on portions of text that the editor has filled in automatically (see Figure 6). These boxes are essentially for performing a sort of *fill in the blanks*. They allow us to type text, such as the name of a target and its dependency. We can use the **Tab** key to cycle between blanks in the template.

Figure 5. Code templates in action

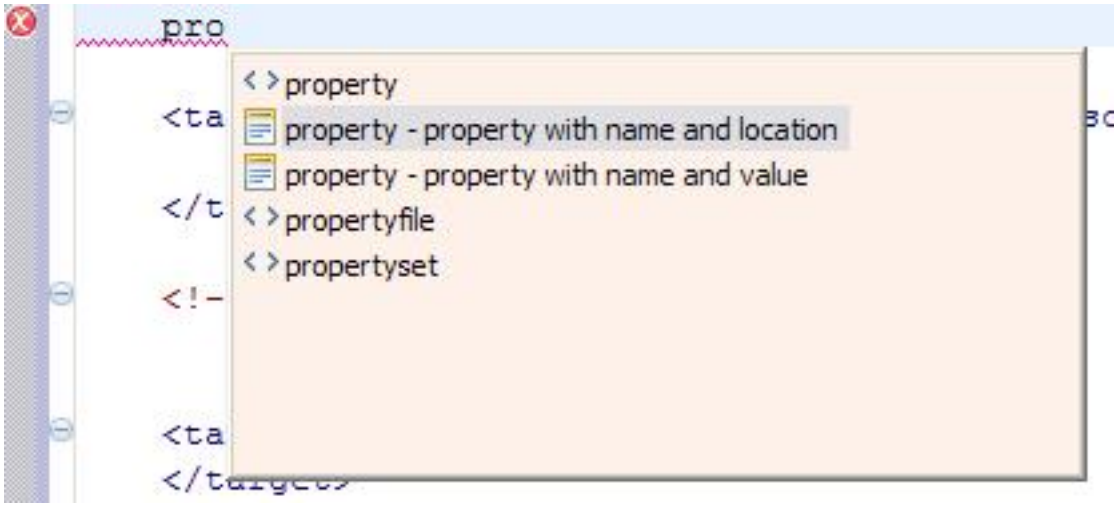


Figure 6. Applying templates

```
    <property name="myProperty" value="value"/>


    <target name="default" depends="depends" des
        <copydir dest="${" src=""></copydir>
```

Folding

The Ant editor can fold in all the Extensible Markup Language (XML) elements in our buildfile. Simply click the **+** or **-** buttons on the left to expand or collapse the various elements. This functionality is useful because with it, we can get a quick overview of the file's contents. If you hover your mouse over the **+** icon, a context window displays the contents of the element.

Renaming

One really great feature of the Ant editor is the Rename in File feature. Using this, we can rename target and property names throughout the file (see Figure 7). Say, for example, we want to rename a target. Right-click the name, then click **Rename in file**. Square boxes appear throughout the file where the target name has been referenced. Now we can edit the name of the target, and the change will be reflected throughout the file. This feature even works for property names.

Figure 7. Renaming a target

```
<target name="default" depends="dependaas"
    <mkdir dir="asdsa"/>
    <copydir dest="${myProperty}" src="./a
    <deltree dir="asdsa"/>
</target>


<target name="dependaas">
```

Mark occurrences

Clicking the **Toggle mark occurrences** button at the top turns the Mark Occurrences feature on or off. With this feature turned on, when we click the name of any target or property, all occurrences of that target or property throughout the file are highlighted (see Figure 8).

Figure 8. Marking occurrences of a target

```
<target name="default" depends="depends" de
    <copydir dest="${myProperty}" src=""></
</target>

<target name="depends">
</target>
```

Showing selected elements only

Clicking the **Show selected elements only** button (see Figure 9) shows only the element clicked. This functionality is especially useful when we must write a big target definition, and we don't want to look at any clutter. We can click this to make the rest of the file elements disappear -- thus, allowing us to focus on that target only.

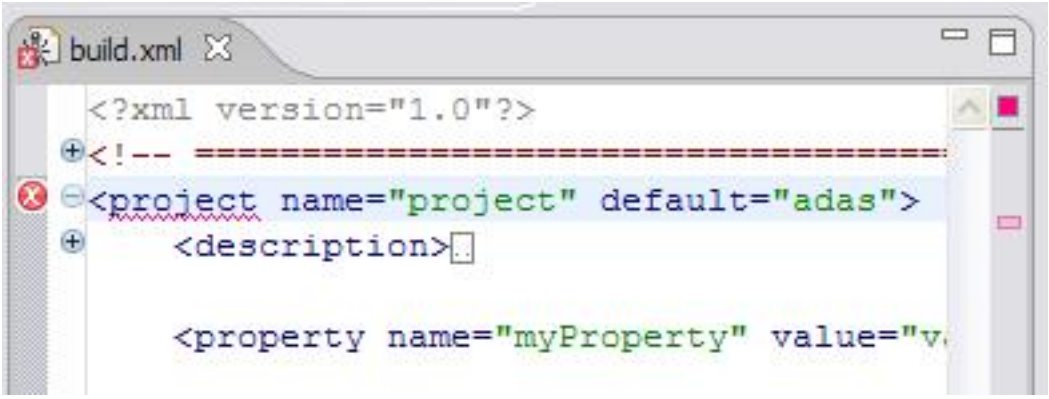
Figure 9. Current target only

Marking problems

The Ant editor can show errors and warnings in our buildfile while we type. This helps easily identify errors and potential mistakes in a buildfile early on, rather than figuring out later why we're getting a mysterious error during the build.

To see this feature in action, go to the `project` tag in build.xml. In the value of the `default` target, type the name of a target that doesn't exist in the buildfile. A `project` tag appears underlined with a red squiggly marker (see Figure 10). Hover your mouse over the marker, and a window appears, stating that the default target doesn't exist in the project. A red **X** icon appears to the left of the marker.

Figure 10. The Ant editor showing an error



Also notice the bar on the right side of the editor window. This shows all the errors and warnings in the file. As soon as an error or warning appears in the file, a corresponding red or yellow marker is placed at the approximate location on the bar. Click the marker to navigate to the location of the error. This gives us a nice overview of where and how many errors or warnings are present in your file, and we'll be able to navigate to them easily. There's also a square at the top of the bar that turns red if errors are present in the file. Thus, just by looking at the square, we can determine immediately whether the file is correct.

We can change the way the Ant editor handles problems by completing these steps:

1. Click **Window > Preferences**
2. Expand Ant, then expand Editor
3. In the Preferences window, click the **Problems** tab (see Figure 11).
   Figure 11. Configuring how problems appear in Ant

4. Select our options. Selecting the **Ignore all buildfile problems** check box disables error checking completely. By default, Eclipse thinks of every XML file as an Ant build file, so it tries to look for errors in them. However, if you have some XML files you don't want checked for errors, specify their names in the **Names** box.
   Below the **Names** box, we see the kinds of errors the Ant editor can detect, and we can set the severity level for each: Ignore, Warning, and Error. Choosing Warning from the list beside the error type signifies code that *might* create potential problems. Choosing Error indicates problem types in which there *definitely* is some problem with the code. If you find some problems a little too restrictive for the way you write code, you can choose Ignore, although I don't recommended doing so.

NOTE: The bar also works with the Mark Occurrences feature. Turn on Mark Occurrences and click any target name. The bar has little yellow markers corresponding to the location of each reference. Click the markers to navigate to the reference.

Section 4. Navigating the buildfile

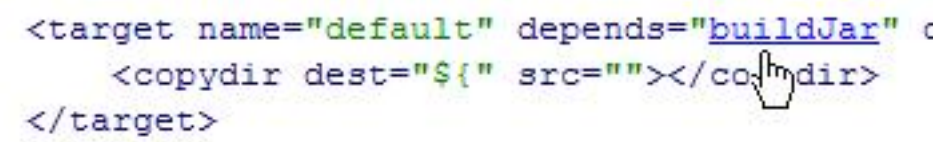
Eclipse offers several methods to help navigate huge buildfiles easily. Examples

include hyperlink and function key navigation, as well as two views: Outline and Ant.

Hyperlink and function key navigation

Press the **Ctrl** key and hover over the name of any target or property. The name turns into a hyperlink (see Figure 12). Clicking it takes us to the declaration of the target or property.

Figure 12. Target reference turns into hyperlink

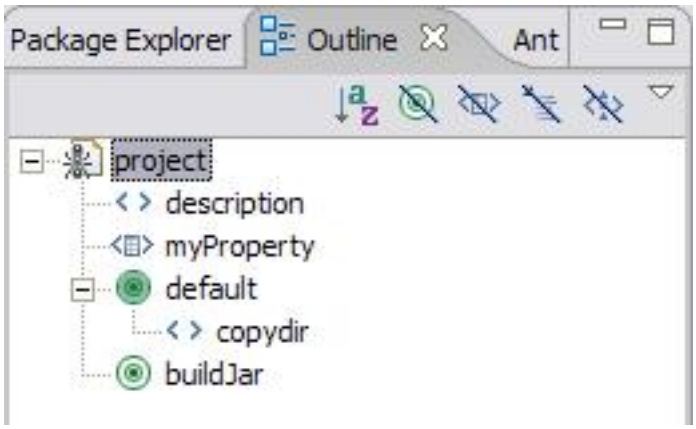


We can also press **F3** to go to the declaration of the target or property. We can change the shortcut key by expanding General, then expanding Keys to access the Keys preferences page.

The Outline view

As the name suggests, the Outline view shows the entire outline of the buildfile (see Figure 13). We can easily see all the targets and properties defined in the file. Internal targets and public targets have different icons, making them easy to differentiate. Even the default target is highlighted. Expanding any target shows all the tasks within it. Click any element in the Outline view to navigate to it directly. The view has a few buttons at the top by which we can filter -- sorting the items or hiding internal targets, properties, imported elements, and top-level elements.

Figure 13. The Outline view



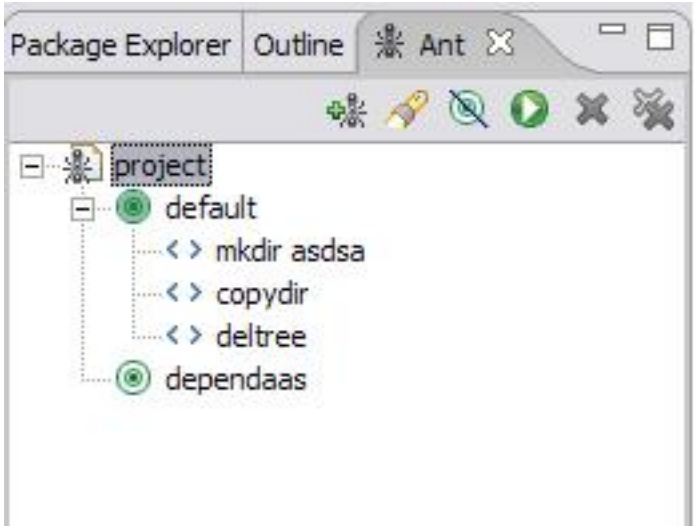
We can also run and debug targets right from the Outline view. To do so, right-click a target in the Outline view, then click **Run As** (or **Debug As**) **> ANT Build**.

The Ant view

Many times, you find yourself working with multiple scripts in multiple projects. So

instead of hunting for buildfiles in the Navigator or Package Explorer view, or digging through the External Tools toolbar list, the Eclipse folks created the Ant view to keep the whole mess straight (see Figure 14).

Figure 14. The Ant view



First, open the Ant view by clicking **Window > Show View > Other > Ant > Ant**. The view is blank the first time we open it, so we must add some buildfiles to it. Clicking the **+** button opens a window in which we can select our buildfiles from the projects open in the workspace. Run a target by selecting it and clicking **Run** or by right-clicking the target and clicking **Run as > Ant Build**.

We can also add buildfiles using Ant's search feature. Click the **Flashlight** icon on the toolbar, and a window appears in which we can specify the name of the file to search. Use special characters like * (asterisk) or ? (question mark) in the file name that stand for *any string* or *any character*, respectively. For example, typing `build*.xml` matches all XML file names that begin with the word *build*. If we don't select the **Include buildfile containing errors** check box, files containing errors won't be selected. Finally, we can choose whether to search within the entire workspace or just within a working set.

To remove a file from the Ant view, select it and click **Remove**. Click **Remove All** to clear the entire view.

The difference between the Ant view and the Outline view

Many people, when they first look at the Ant view, mistake it for an Outline view with multiple files. But several subtle differences exist between the views. While the Outline view is designed to help us navigate the current file, the Ant view allows us to manage the running and debugging of multiple targets in multiple build scripts that may be scattered throughout the workspace.

This fundamental difference becomes more apparent when we look closely at the features provided in the two views:

- We can add multiple files to the Ant view, while the Outline view shows only the outline of the current file.
- Double-clicking a target in the Outline view allows us to navigate to the corresponding declaration in the editor, but doing so in the Ant view actually runs the target.
- The Ant view doesn't show any properties or top-level elements because we can't "run" properties.
- Both the Outline view and the Ant view contain the **Hide internal targets** button, which we can click to hide all targets that aren't public, but the views provide this button for different purposes. So, while the Outline view provides this button solely as another way to filter the view, the Ant view provides it because as we will typically be running the public targets only, it makes sense to hide the internal targets from view.

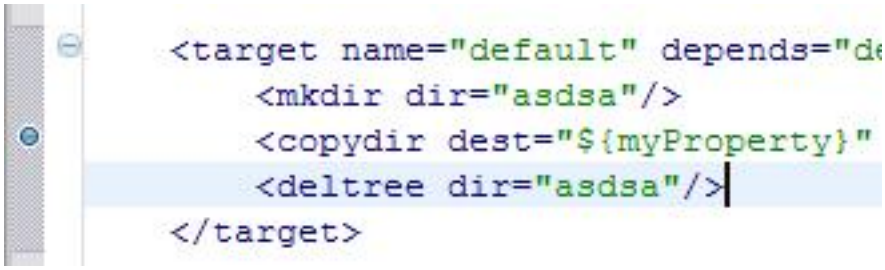
Section 5. Debugging Ant files

Yes, you read it correctly. You can actually debug Ant files in Eclipse just like you debug Java files, and have all the standard debugging features available. This is probably the best functionality in the Eclipse Ant integration.

Place breakpoints inside targets

Just as we always do with Java files, place breakpoints inside targets on the lines that call the task we're interested in stepping through. To put a breakpoint on a line, simply double-click beside the line on the gray bar at the left. A green ball appears, denoting that a breakpoint has been set (see Figure 15). Temporarily enable or disable breakpoints by clicking them or disabling them in the Breakpoints view. A disabled breakpoint appears as a white ball. Note that unlike Java breakpoints, we cannot set hit counts or conditions on breakpoints -- not that we'll need them while debugging Ant files, anyway.

Figure 15. Breakpoint set on a line in the buildfile

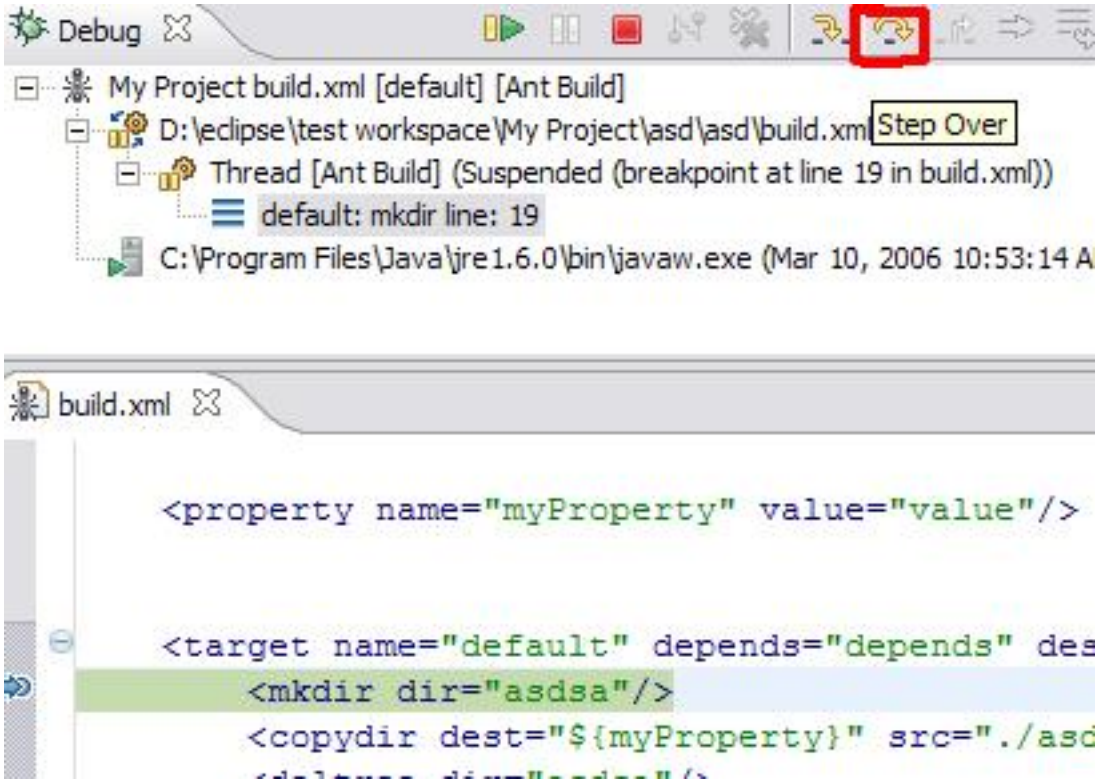


Debug the buildfile

Now, begin debugging. Right-click a target in the Ant view or the Outline view, then click **Debug As > Ant Build**. Just as with Java files, the buildfile pauses when the execution reaches the line on which we've set the breakpoint.

Here's the great part: Click the **Step Over** button in the Debug view to step through the lines in the buildfile, just as we step through Java statements (see Figure 16). As we step through each task, it will be executed and produce its output, which we can examine to see what's going wrong in the build process. The Run to Line functionality is available, too, so we can right-click a line and click **Run to Line** to cause buildfile execution to continue until it reaches that particular line. The process is similar to setting a temporary breakpoint on a line that's removed as soon as the line is reached.

Figure 16. Step through lines in a buildfile

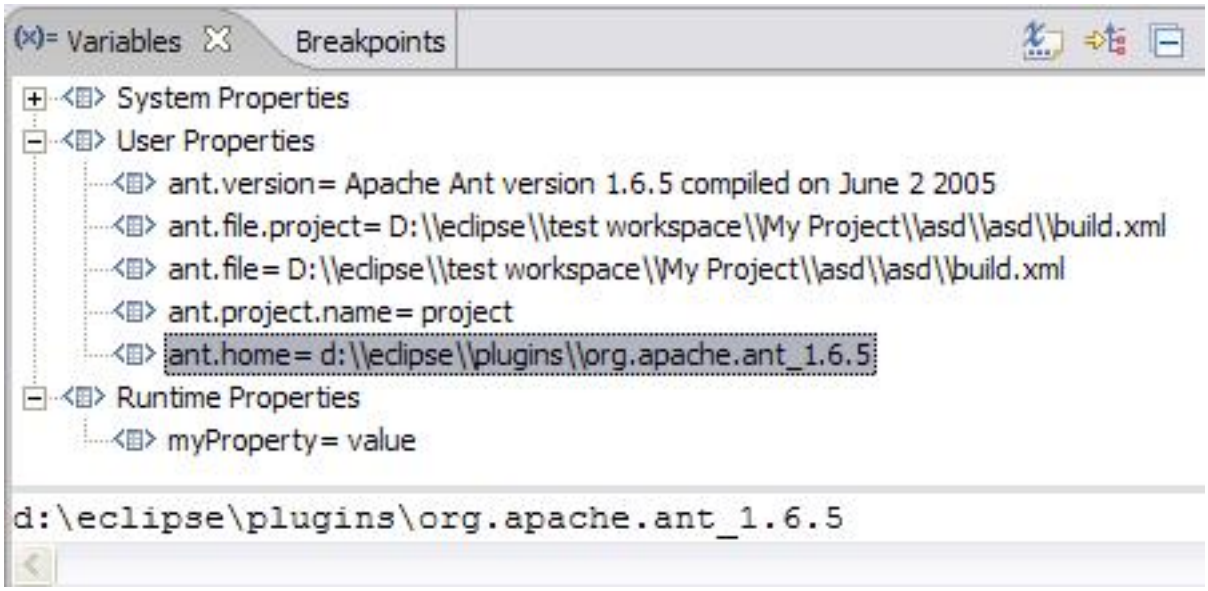


The Debug view shows the call stack of the tasks currently being executed. If a task calls another target -- say, **antcall** -- that target appears above the current task in the call stack.

A Variables view is also available (see Figure 17). Open this view to see all the Ant properties, which are the Ant equivalent of variables. The properties are shown grouped in three parts:

- **System properties:** Properties set from the system for the build
- **User properties:** Properties such as those set using the `-D` option
- **Runtime properties:** Properties defined in a buildfile that are set during runtime

Figure 17. The Variables view shows all properties



Note that unlike the Java debugger, the Ant debugger doesn't allow us to change the value of the properties shown in the Variables view.

Section 6. Using your Ant buildfile as a project builder

When we use the Eclipse Java IDE, we use the Java Builder unconsciously. The Java Builder is a silent beast that runs in the background each time we save files, and it compiles them instantly.

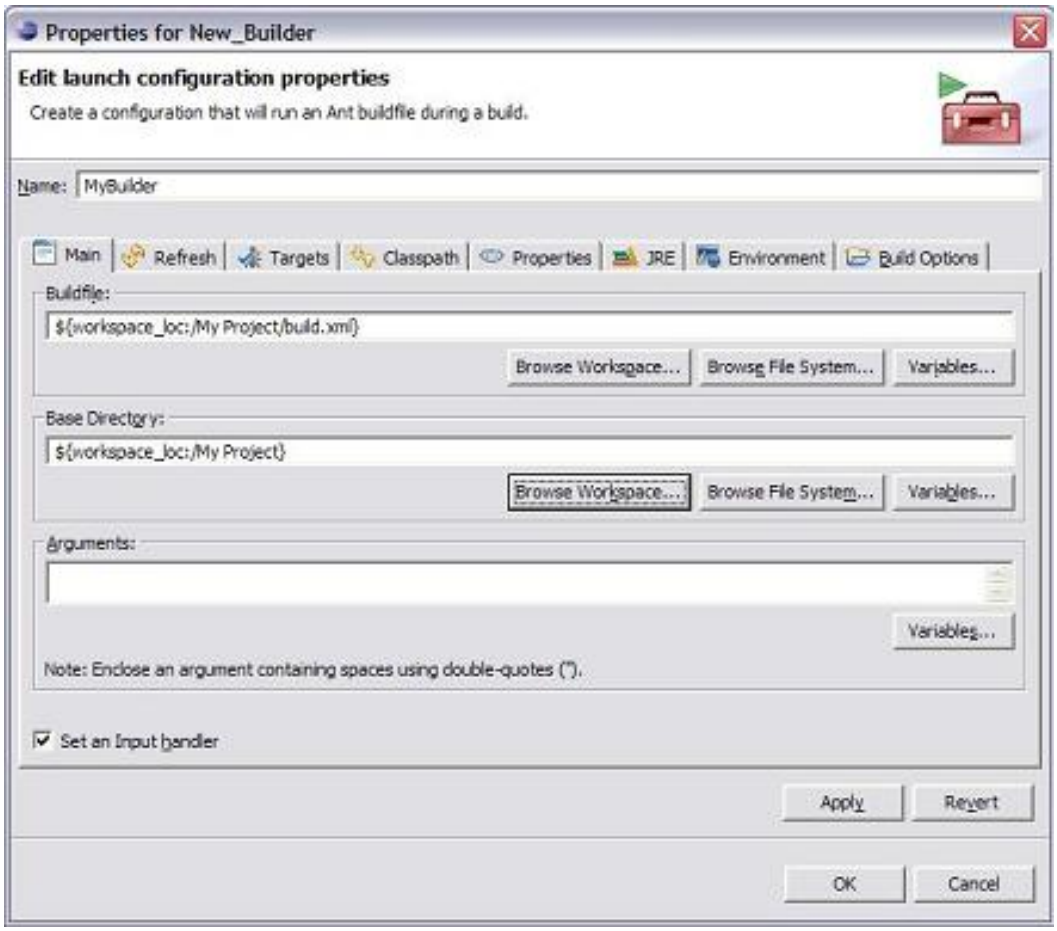
Although this may not seem like a big deal, it's one of the most amazing features of Eclipse: The Java Builder allows us to skip the compilation process altogether because our program is *always* in a compiled state, even though it might be full of errors. Thus, we can run our Java programs immediately after typing them without first going through a long and tedious compilation step. This functionality saves a lot of time and hassle for Eclipse users and is one of the reasons for Eclipse's huge popularity among programmers.

But what if we wanted to do something more than just compile files? What if we wanted to create a jar file for the entire project and copy it to a certain directory each time we make changes to a project? And what if we wanted all this to happen in the background without having to tell Eclipse each time? We could just sit, relax, write some code, sip of coffee, and let Eclipse handle the complex build process in the background without even needing to know that it's actually happening.

Sound like a dream? It isn't. We can actually make this happen. We simply need to add an Ant buildfile, which has all the complexity of the build process defined in it, to our project as a "builder." Do this, and the magic will start happening.

Why use Ant as a project builder?

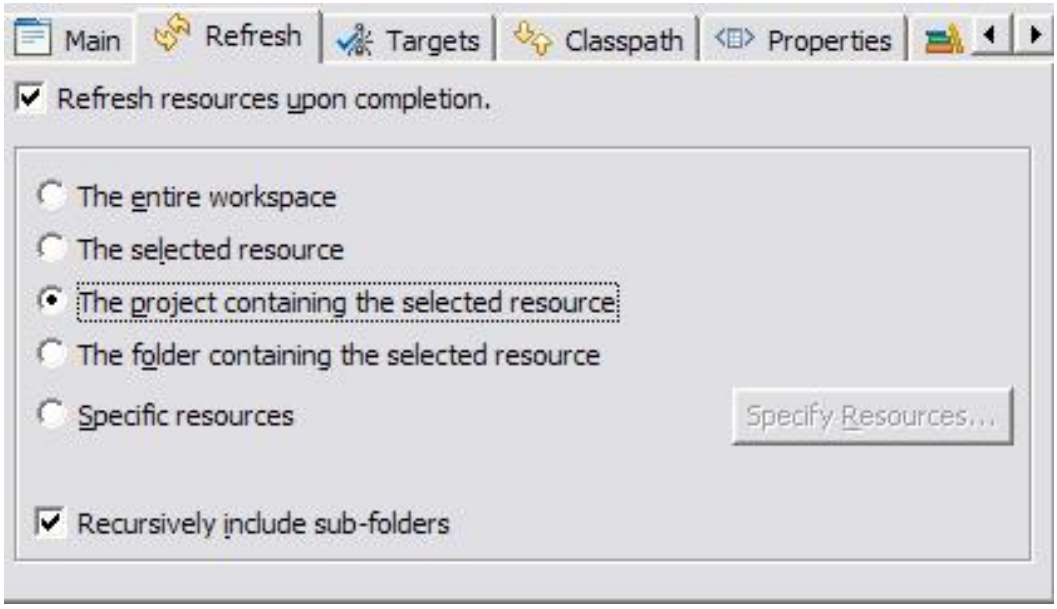
Assume we have an Ant buildfile that creates a jar file out of the class files in the project and places the jar file in the root of the project. (The exact contents of the buildfile are irrelevant for now.) We want this buildfile to run each time a Java file is modified, so the jar file always remains up to date. Complete these steps:

1. Right-click the project in the Package Explorer view and click **Properties**.
2. Expand Builders and click **New** to add a new builder to the project.
3. In the resulting window, select **Ant Build** and click **OK**.
4. The builder's Properties window appears (see Figure 18). Here, configure a builder.
   Figure 18. The builder configuration window
   

5. In the **Name** box, type `MyBuilder`.
6. Click **Browse Workspace** beneath **Buildfile** and select the buildfile from the project.

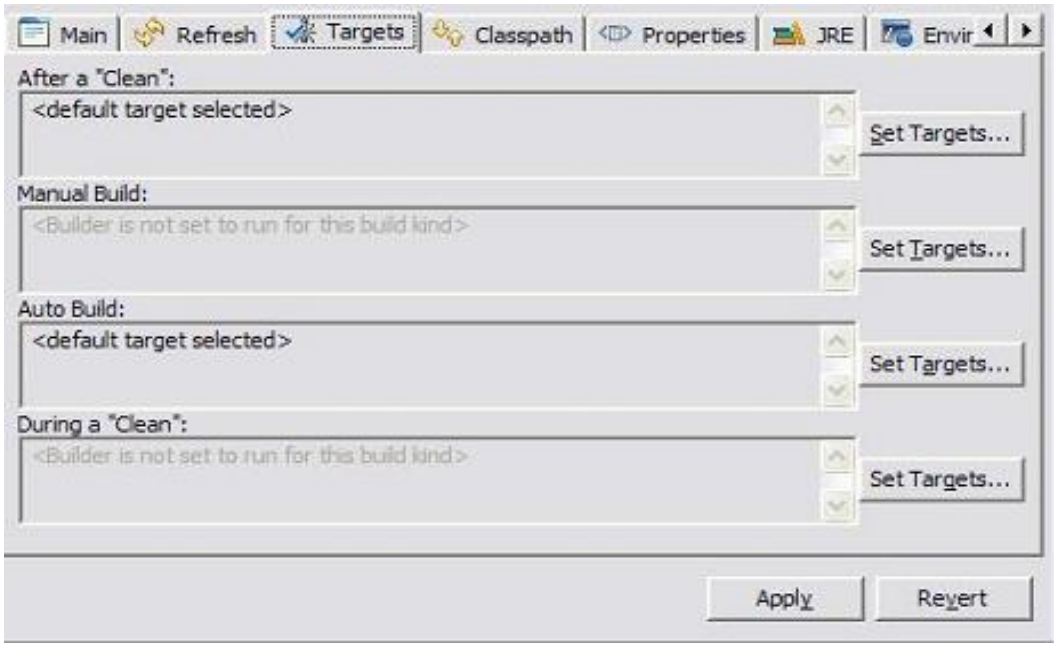
7. Click **Browse Workspace** beneath **Base Directory** and select the project containing the buildfile. Provide arguments to the buildfile, but because we don't need to provide any right now, leave it blank.

8. Click the **Refresh** tab (see Figure 19).
Refreshing a project instructs the Eclipse Workbench to look for any changes to the project made in the local file system by external tools such as Ant. So here, tell Eclipse whether to perform a refresh after a build script finishes and, if so, what parts of the workspace it should refresh.
Figure 19. Refresh tab



9. Select the **Refresh resources upon completion** check box. Doing so enables the options beneath it on the tab. Tell Eclipse how much of the workspace to refresh. Select the smallest possible entity sufficient for our Workbench to continue to run quickly. For this example, we just need to refresh the current project, so select the **The project containing the selected resource** option.

10. Click the **Targets** tab.
Figure 20. Targets tab



Here, we choose when the buildfile actually runs and, more specifically, which target is run. Four options are available:

- **After a "Clean"** -- The target runs each time we perform a clean operation on the project.
- **Manual Build** -- This option is used in case automatic builds are turned off. Whenever we perform a manual build, the specified target will run.
- **Auto-Build** -- The target runs each time an auto-build is performed. Typically, this is each time we save our Java files.
- **During a "Clean"** -- This option is different from After a "Clean" in that the target is called during the clean operation itself. Use this to perform some custom clearing up of files during the clean operation.

11. Set the target to be run. Each of the target options has a **Set Targets** button beside it with which we can set the target to be run during each operation. Generally, select the default target here, but we can select any target -- and even multiple targets, as well -- as the order in which they should run.

12. Define the targets to be run beside whichever operation we want the buildfile to run.
    In this case, because we want the jar file always to be current, set targets for the After a "Clean" and Auto Build operations. To do so, click **Set Targets**, then select the targets to be executed. If we see targets defined for any other operation, such as Manual Build, click **Set Targets** and clear those targets' check boxes to disable the buildfile from running during those operations.
    Also note that even though, for this example, we're choosing to run the target after each Auto Build operation, we should generally use this option

with caution because the Workbench can slow to a halt if the build process takes a long time. Generally, set the Manual Build and After a "Clean" options only.

13. Click **OK**.

Now it's time to test our newly added builder. Open any Java file in our project, make some modifications (for example, insert a space), and save it. The Auto Build will run, and we'll see in the console that the buildfile is running the target selected. The jar file is built, and appears in the Navigator and Package Explorer view. All this happens automatically each time.

Section 7. Conclusion

You have seen that Eclipse provides a powerful environment for writing, debugging, and navigating Ant build scripts. It even allows you to use Ant as a project builder so that your Ant files can execute automatically in the background. You're now ready to start churning out build scripts in Eclipse.

I suggest exploring the features described above for yourself by writing an Ant build script and using it as a project builder. Also, don't forget to keep the Ant reference manual by your side to take a look at descriptions of all the tasks available to you while writing your build scripts.

Resources

Learn

- Keep the Apache Ant 1.7.0 Manual by your side.
- The developerWorks article "Extending Ant to support interactive builds," by Anthony Young-Garner, demonstrates how to extend Ant to produce builds that are interactive at runtime.
- "Automate your build process using Java and Ant," by Michael Cymerman, is nice introduction to Ant.
- The developerWorks article "Incremental development with Ant and JUnit," by Malcolm Davis, offers another great introduction to using Ant.
- The Ant online documentation titled "Writing Your Own Task" explains the basics of developing custom tasks for Ant.
- Read the Ant support in the Eclipse documentation for Ant-related functionality within Eclipse.
- Check out the "Recommended Eclipse reading list."
- Browse all the Eclipse content on developerWorks.
- Users new to Eclipse should look at the Eclipse start here.
- Expand your Eclipse skills by checking out IBM developerWorks' Eclipse project resources.
- To listen to interesting interviews and discussions for software developers, check out check out developerWorks podcasts.
- For an introduction to the Eclipse platform, see "Getting started with the Eclipse Platform."
- Stay current with developerWorks' Technical events and webcasts.
- Check out upcoming conferences, trade shows, webcasts, and other Events around the world that are of interest to IBM open source developers.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- Get the latest Ant development news from Apache Ant.
- Check out the latest Eclipse technology downloads at IBM alphaWorks.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- Discuss Ant at the Ant mailing list.
- The Eclipse Platform newsgroups should be your first stop to discuss questions regarding Eclipse. (Selecting this will launch your default Usenet news reader application and open eclipse.platform.)
- The Eclipse newsgroups has many resources for people interested in using and extending Eclipse.
- Participate in developerWorks blogs and get involved in the developerWorks community.

About the author

Prashant Deva
Prashant Deva is the founder of Placid Systems and the author of the ANTLR Studio plug-in for Eclipse. He also provides consulting related to ANTLR and Eclipse plug-in development. He has written several articles related to ANTLR and Eclipse plug-ins, and he frequently contributes ideas and bug reports to Eclipse development teams. He is currently busy creating the next great developer tool.