

Plug-in Development

This document describes how to set up an environment for developing plug-ins. It assumes you have already read the introductory plug-in document that describes a plug-in's internal directory structure.

Environment

Within your application directory, you will need to create a **plugins** directory, then subdirectories for each type of plug-in you plan to develop, e.g. **handlers**, **connectors**, etc.

Each plug-in you develop will have its own subdirectory within the folder of the associated plug-in type. For example:

```
~/plugins/connectors/HelloWorld/
~/plugins/connectors/FooBar/
```

Ant

If you plan to use the Ant build script provided by Augur Systems to build your plug-in (highly recommended!), then you need to place it in the **plugins** directory.

To use the Ant script, you may need to install the **ant** command-line utility separately, if it is not already on your machine. We recommend you use **ant** version 1.7 or later. You can get the latest version here: <http://ant.apache.org/>

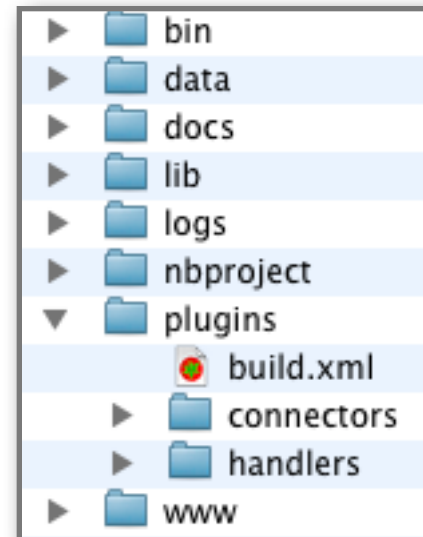
Alternatively, your Java editor may include a built-in implementation of Ant.

Skeleton

Contact Support to obtain a ZIP file containing a *skeleton* plug-in. It is a simplistic plug-in that just contains a plug-in's basic directory structure and a sort of "Hello World" source code implementation.

Alternatively, you might consider unzipping an existing plug-in as a template. Just be sure to first create a new subdirectory at `~/plugins/<type>/<newPluginName>` then copy the template's `*.api` file to that directory, and unzip it. Lastly, you'll need to change the source code file name(s), and the plug-in's class name where it is referenced in

```
~/plugins/<type>/<newPluginName>/Plugin.properties.
```



Building

The following describes how to compile and package a plug-in using the provided Ant script. If you do not use this script, it is up to you to compile your plug-in code and to construct the plug-in `*.api` file.

Ant Script

When you develop a new plug-in you will need to add its name to the build script at `~/plugins/build.xml`. Follow the editing instructions embedded in that file.

Basically, you will just have to copy a line in the Ant target named `build`. In that line, you will edit your plug-in's name (the same as its folder name), and decide if you want to include your plug-in's source code in the `*.api` file (recommended default) or keep it private.

Building

During development, you just need to run the `build` target. From the command line, move into the `~/plugins` directory, then execute: `ant` (Ant will automatically find the `build.xml` file in the current directory because that is the default Ant script name. And, since `build` is configured to be the default target, you do not need to specify the target name either.)

You can usually do the same thing from your IDE. The advantage of running at from within your IDE is that you can usually click on any error message to focus your editor on the offending source code line.

If the build is successful, it will deposit a `*.api` file in the associated `~/plugins/` subdirectory, for example: `~/plugins/connectors/HelloWorld.api`

You can then install that plug-in in your application for testing:

- For Augur, install via the Configuration Editor.
- For Socket Station, just stop/start Socket Station.

IDE Setup

An integrated development environment (IDE) helps you develop software faster and easier. An IDE usually provides color-coded source editing, syntax-checking, code-completion, and other code development features. You do not need to use an IDE to develop plug-ins (you could use a plain text editor instead), but it is highly recommended!

To setup any IDE for developing your plug-in you will have to tell it three things:

1. Where are your Java source files located?

This should be the `src` subdirectory within your plug-in's development directory.

2. What is the Java classpath?

The *classpath* tells your IDE where to find 3rd-party Java classes that may be referenced by your software. The classpath comprises directories, ZIP files, and/or JAR files. The classpath can include your plug-in's `lib` directory, if you need to include any libraries within your plug-in. However, the minimal classpath must include the application's core JAR files:

- For Augur: `~/lib/augur/*.jar`
- For Socket Station: `~/lib/ss/*.jar`

3. What version of Java?

Currently, your plug-in's Java code must target version 1.5 (also known as "Java 5" in marketing terms). It is OK to develop with a newer version of Java (e.g. your IDE might run within Java 6). However, you need to target version 1.5 during compilation. If you use the provided `ant` script, then it will automatically take care of compiling your plug-in correctly. However, you still need to configure the target in your IDE so that the IDE can best help you edit your software.

IDE Examples

The next section of this document describes the configuration for the NetBeans IDE. NetBeans is one of many Java editors you can use to create plug-ins. The basic setup should be similar for other IDEs.

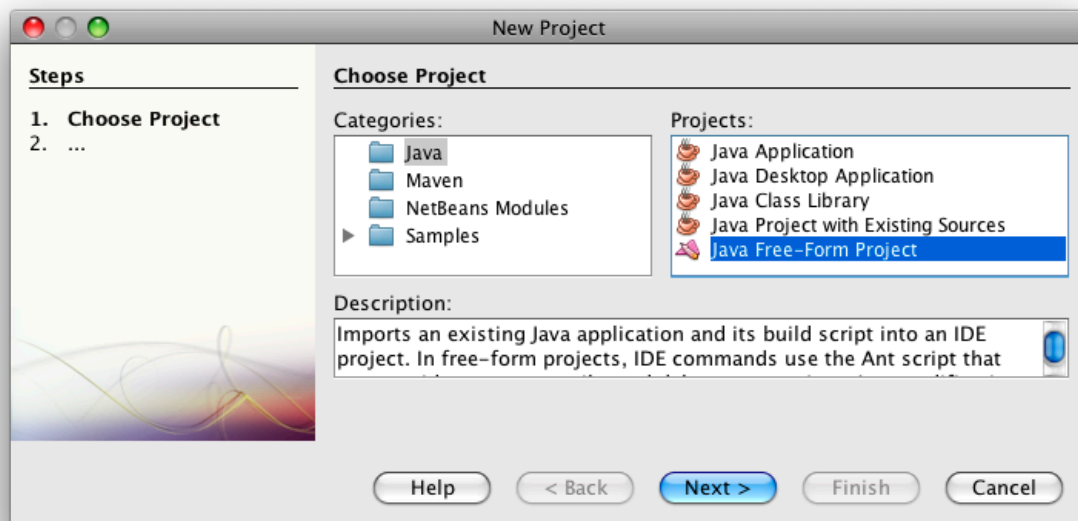
NetBeans Setup

This section describes how to set up the NetBeans IDE to edit your plug-in code. You can download the latest NetBeans from <http://www.netbeans.org/>. NetBeans bundles the IDE in several ways, depending on what type of development you intend to do with it. For our plug-ins, you only need the “Java SE” bundle, which is usually the smallest download.

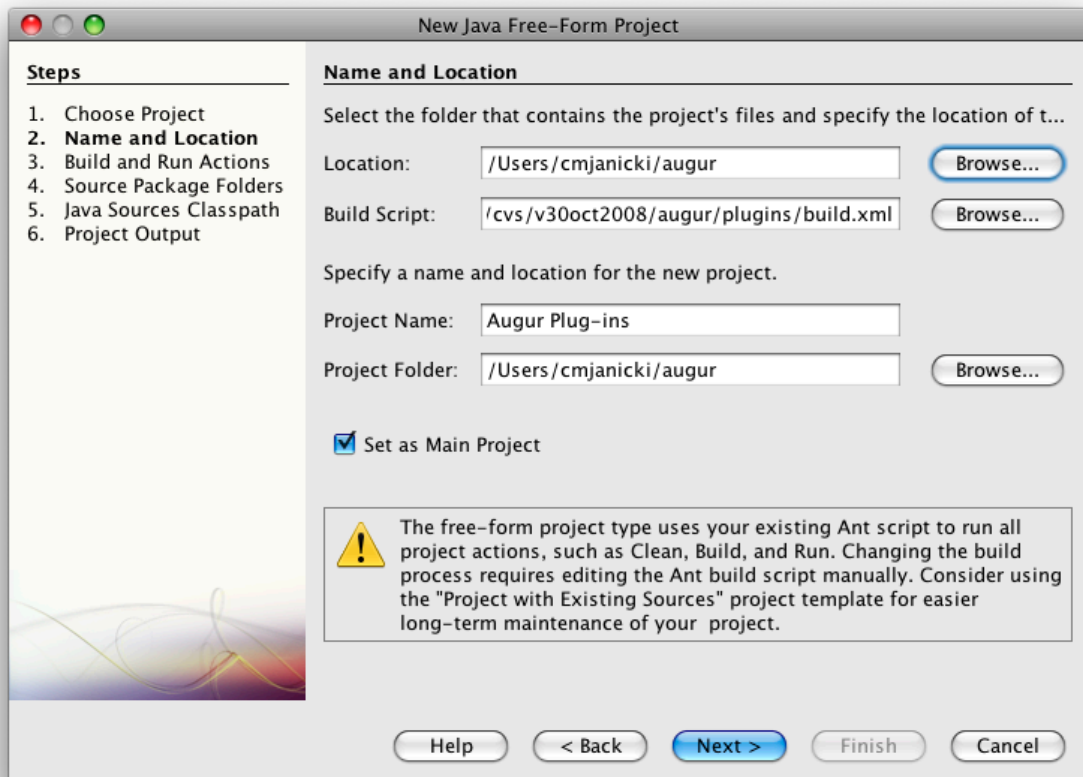
Creating a New Project

You will only need to create a project once. From within that project, you can edit multiple plug-ins.

- I. From the menu, select: **File > New Project**. A new window will pop up.



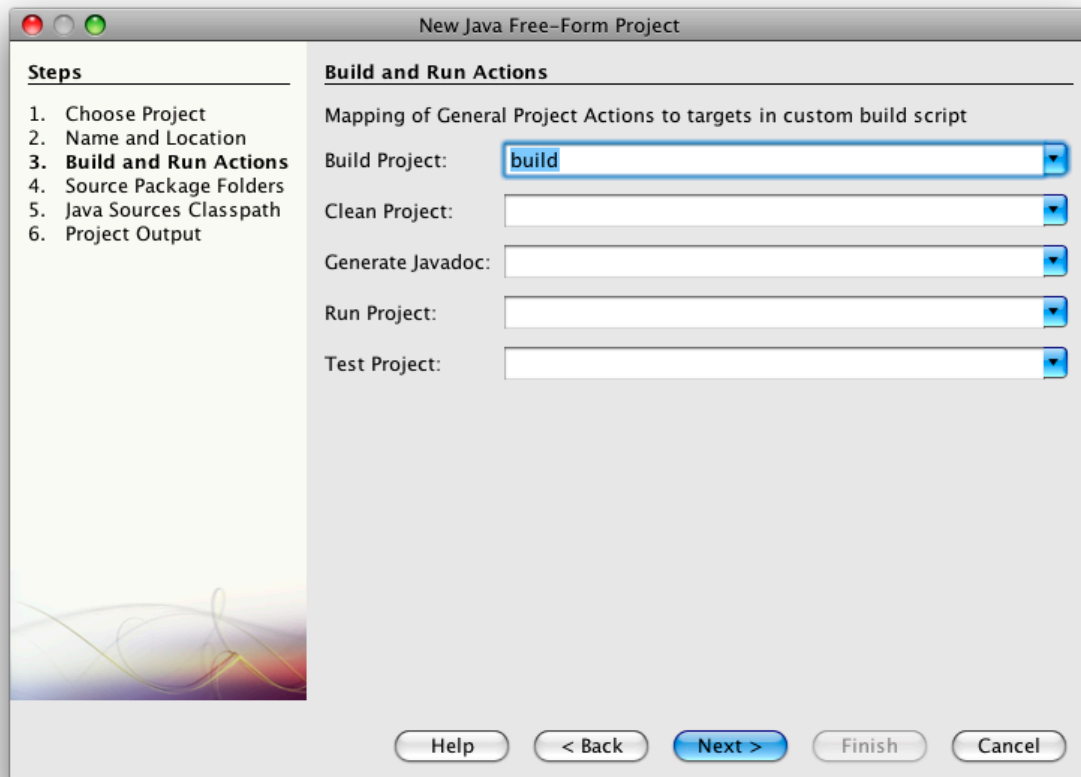
2. Choose a **Free-Form** project.



3. Set the following fields, then press the **Next** button:

- **Location:** Augur's home directory
- **Build Script:** ~/plugins/build.xml (within Augur's home directory)
- **Project Name:** Augur Plug-ins (or whatever)
- **Project Folder:** Augur's home directory

4. Leave the defaults for the build script mapping, then press the **Next** button.



5. Leave the defaults blank for the Java source and test panels, then press the **Finish** button.

Creating a New Plug-in

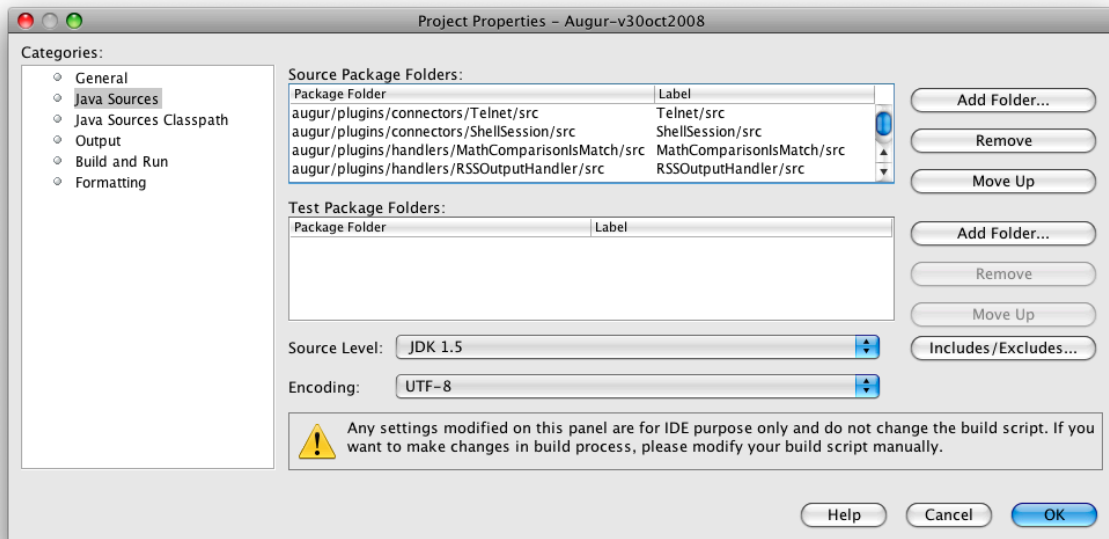
For each plug-in, you will need to create the plug-in's directory environment. Within NetBeans, this is as easy as copying/pasting an existing plug-in directory structure. NetBeans will copy the entire directory. You then need to rename a few files: 1) the new directory name; 2) the name of your plug-in Java source file(s), and 3) the name of your main class within the Plugin.properties file.

Alternatively, you could manually create the new directory, then unzip a template plug-in in that directory, and follow steps 2 and 3 above.

The following instructions tell NetBeans about the new `src` directory, and what NetBeans should use as a classpath for evaluating the Java source files found there:

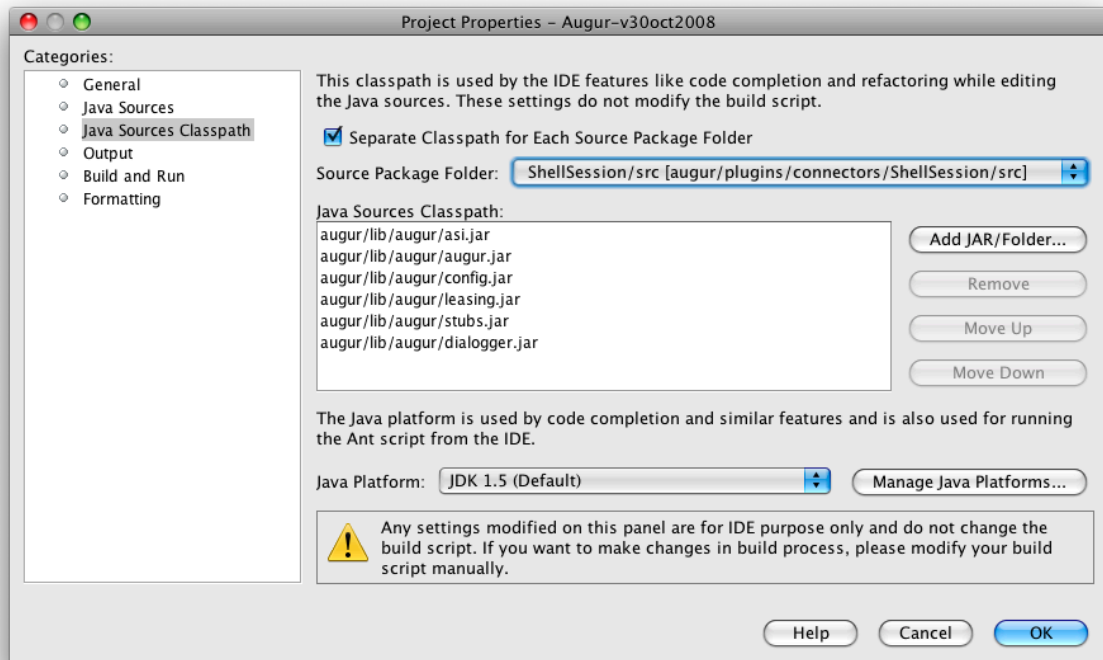
6. Open the projects window (or pop it into focus if it is already open) via the menu:
Window > Projects
7. Right-click on your project's root node, then select **Properties** in the pop-up context menu. (A new window will pop up to configure your project.)

- On the left “Categories” panel of the Project Properties window, select the **Java Sources** category. (This is where you tell NetBeans about the directories that may contain your java source files. You may have more than one, if you are developing multiple plug-ins.)



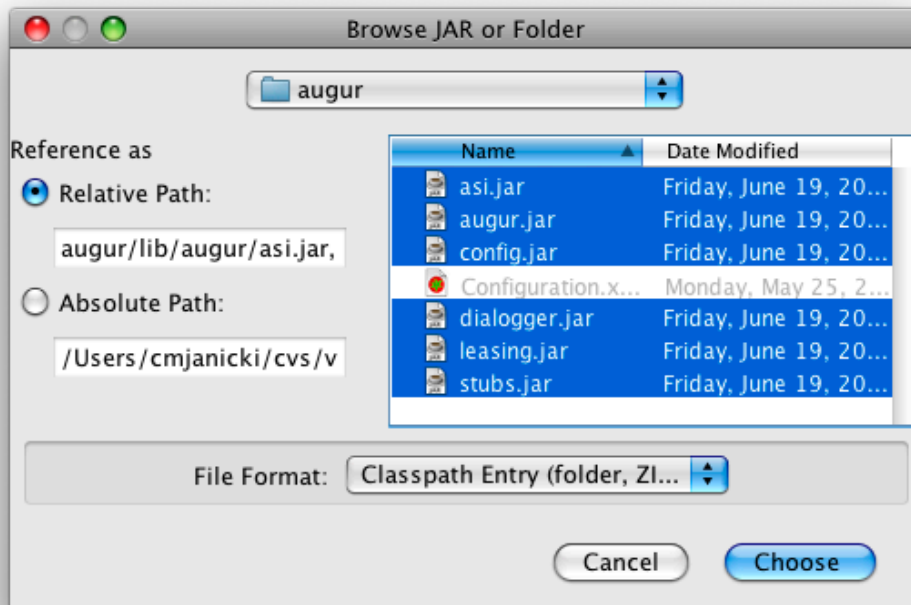
- Press the **Add Folder...** button. A file browser will appear.
- Select the **src** folder for your plug-in. (Make sure the folder is highlighted. If you double-click it will open the folder instead.) Press the **Open** button to confirm.
- Optional: Double-click in the **Label** column in the table, then shorten the label. We recommend deleting everything before the plug-in name.
- On the left “Categories” panel of the Project Properties window, select the **Java Sources Classpath** category. (This is where you tell NetBeans about *classpath* to be associated with each **src**

directory you specified in the previous steps.)



13. Select the **Separate Classpath for Each Source Package Folder** option. This tells NetBeans to use a different classpath depending on which `src` directory your working in. Each plug-in may need different resources, (e.g. from it's own `lib` directory) so it is best to define a separate classpath for each plug-in when using NetBeans.
14. From **Source Package Folder** drop-down list, select your plug-in's source folder (the one you set on the previous screen).

15. Press the **Add JAR/Folder...** button. A file browser will appear.

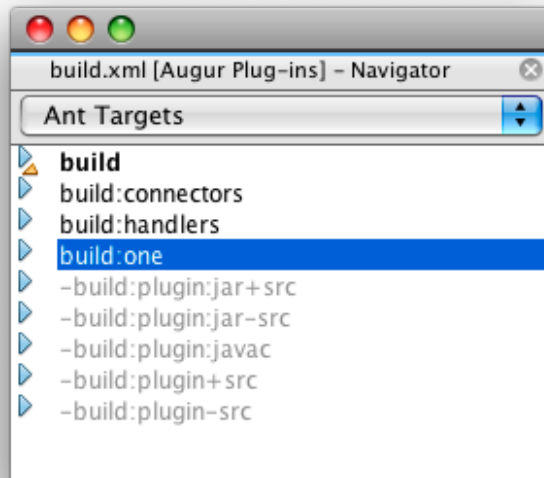


16. Navigate to Augur's `~/lib/augur/` directory, then select all the JAR files in there. If your plug-in has its own JARs (in its own lib directory), add those too.

17. Press the **OK** button. In the background, NetBeans will start processing your new classpath and source settings, so the error messages related to unknown classes should start disappearing in the editor.

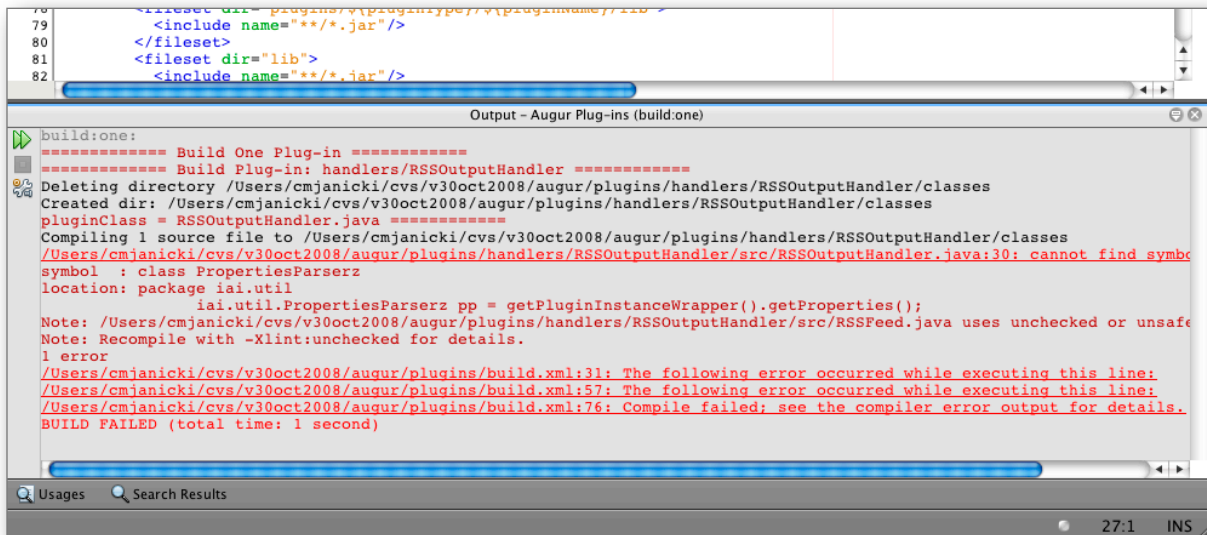
Running the Ant Script

From within NetBeans, select the `build.xml` file from either the Files window (**Window > Files**) or Projects window (**Window > Projects**). Then, from the Navigator window (**Window > Navigator**), you can right-click on the `build` target and select **Run Target** from the pop-up context menu.



The build output will appear in a new panel. If there were errors, you can click on any underlined error message to open the IDE to the associated line of your source code.

Also note that following any code errors, you will also see error lines showing the last location of the build script. You can usually ignore these lines.



If all goes well, the build output will end with “BUILD SUCCESSFUL”.

```
build:one:
==== Build One Plug-in =====
==== Build Plug-in: handlers/RSSOutputHandler =====
Deleting directory /Users/cmjanicki/cvs/v30oct2008/augur/plugins/handlers/RSSOutputHandler/classes
Created dir: /Users/cmjanicki/cvs/v30oct2008/augur/plugins/handlers/RSSOutputHandler/classes
pluginClass = RSSOutputHandler.java =====
Compiling 1 source file to /Users/cmjanicki/cvs/v30oct2008/augur/plugins/handlers/RSSOutputHandler/classes
Note: /Users/cmjanicki/cvs/v30oct2008/augur/plugins/handlers/RSSOutputHandler/src/RSSFeed.java uses unchecked or unsafe
Note: Recompile with -Xlint:unchecked for details.
Updated version date at /Users/cmjanicki/cvs/v30oct2008/augur/plugins/handlers/RSSOutputHandler/versions/1.0.01.properties
Building jar: /Users/cmjanicki/cvs/v30oct2008/augur/plugins/handlers/RSSOutputHandler.jar
BUILD SUCCESSFUL (total time: 0 seconds)
```

Notes

If you change or update any of the ZIP/JAR files in your plug-in's classpath, NetBeans may not notice. However, you will notice if the editor warns of code errors that you know should be fine, per your latest libraries. If this happens, you just have to stop/start NetBeans.