# Java Programming with SciTE on Linux and Windows

Adrian P. Robson
adrian.robson@nepsweb.co.uk

23rd February, 2012

## 1 Introduction

The SciTE editor is an excellent tool for Java programming projects, and it is multi-platform. It can be obtained from

`www.scintilla.org/SciTE.html`

SciTE is a little unusual, in that most of its configuration is done using manually edited *properties files*. Here we provide directory and user properties files that change SciTE's defaults to make Java programming easier. Multiple source folders with projects, a class folder and JAR generation are supported without using construction tools like Make, Ant or SCons. Tab and indent settings, output panel orientation and non-proportional fonts are also set to improve the programming environment.

Furthermore, the properties files provide project mobility. Correctly set up projects can be moved between any Windows or Unix computers that have SciTE and Java SDK installed.

There is a known, but poorly documented, problem with SciTe's go command on Linux. Standard command line input does not work in launched programs. This is resolved by introducing an additional go command.

## 2 Setting Up a Project

### 2.1 Creating the Project Folders and Files

To set up the folders and files for a project do the following:

1. Create a root folder for the project
2. Create a `SciTEDirectory.properties` file from §7.2, or obtain the file from `nepsweb.co.uk`, and put it in the project's root folder.

   in addition, add the assignments in §7.1 to the SciTE user properties file. See §4 for more discussion about the features that these set up. But the `properties.directory.enable` property must be set to `1`.

3. Edit `SciTEDirectory.properties` as discussed in §2.2 to specify the configuration of the project's class and source folders, and some other options.

   The default is for all the source files to be in the project's root folder; for the class folder to be called `classes`; and for the JAR option to be disabled.

4. Create source and class sub-folders as specified in 3 above.

5. Put any existing source files in the correct folder.

## 2.2 Configuring the Project

The behaviour of the directory properties file is configured by a number of variables. These must be set to be compatible with the project's folder structure, and other requirements. For convenience, default values are defined, or a typical assignment is give but commented out.

If the default values are left unchanged, then all the project's source files must be in its root folder, classes are in in a folder called `classes`, and there is no JAR command.

Most of the following options need *path separation* or *list separation* characters. These are specified in variable definitions as `${ps}` and `${ls}` respectively. Path separators are converted to `/` or `\`, and list separators to `:` or `;` depending on the operating system. So the assignment

```
1 $(root)$(ps)utility$(ls)\
2 $(root)$(ps)sourceMain
```

on a Unix platform, could become this:

```
/home/apr/java/proj1/utility:/home/apr/java/proj1/sourceMain
```

The `\` on line 1 is the SciTE next line continuation symbol, which is used to split long lines.

### 2.2.1 Source Folders

Uncomment the `sfolds` assignment, and change it to list all of the project's source folders, separated by `${ls}`. Use `$(root)` to make paths relative to the project's root folder, and use `${ps}`. For example:

```
sfolds=$(root)$(ps)utility$(ls)$(root)$(ps)sourceMain
```

Source files can also be put in the project's root folder, and if `sfolds` is not declared, then *all* the project's source files must be in the root.

### 2.2.2 Main Class Folder

Change the `main` assignment to be the folder that contains the classes with a `main` method. For example one of the following might be used

```
1 main=$(root)
2 main=$(root)$(ps)sourceMain
```

Line 1 is the default, which is project's root folder.

### 2.2.3 Classes Folder

The folder where all the project's class files will be stored is given by the `cfold` variable. By default it is `classes` in the project's root folder:

```
cfold=$(root)$(ps)classes
```

This should not normally be changed. But take care if it is, because *all of the files* in this folder are deleted by the clean command.

### 2.2.4 More Classes Folders

Additional class paths can be added using the `moreclasses` variable. These paths are typically used to access classes from other Java projects. They are not affected by the clean command.

If there are more than one path, they should be separated by `$(ls)`, like this

```
moreclasses=$(root)$(ps)..$(ps)proj3$(ls)\
$(root)$(ps)..$(ps)proj4
```

In the above example, the paths are defined relative to the project's root folder, using '`..`' to go up the folder hierarchy. Doing this means that the project can be moved between computers with different operating systems providing all the paths are moved without rearrangement.

### 2.2.5 JAR Build

The JAR build tool is enabled with the `JAR` variable like this

```
JAR=1
```

Once this is done, then `jarname` and `jarmain` must also be given values. The variable `jarmain` is the name of the JAR file; and `jarmain` is the name of a class with a main method that is the entry point to the program. For example:

```
JAR=1
jarname=myprog
jarmain=MainClass
```

In addition, the variable `jarmore` can be used to identify more resources for inclusion in the JAR. Its value must have the following format if it is used:

$$
\begin{aligned}
\mathit{jarmore} &\models \langle \mathit{resource} \rangle \mid \langle \mathit{resource} \rangle \, \langle \mathit{jarmore} \rangle \\
\mathit{resource} &\models -C \, \langle \mathit{folder} \rangle \, \langle \mathit{inputfiles} \rangle \\
\mathit{folder} &\models \text{folder containing resource} \\
\mathit{inputfiles} &\models \langle \mathit{input} \rangle \mid \langle \mathit{input} \rangle \, \langle \mathit{inputfiles} \rangle \\
\mathit{input} &\models \text{file or folder for inclusion}
\end{aligned}
$$

For example

```
jarmore=-C $(root)$(ps)class2 . -C home$(ps)class3 More.class
```

### 2.2.6 Compiler Options

There a number of options that can be selected by uncommenting the relevant `opt`$n$ variable:

| | |
|---|---|
| `opt1` | Verbose output |
| `opt2` | Disable warning messages |
| `opt3` | Generate debugging information |
| `opt4` | Enable recommended warnings |

## 2.3 Packages

If a custom package is used by one of the project's classes:

```
import daybreak.*;
```

Then the package's source files must be stored in an eponymous sub-folder of one of the project's source folders (see §2.2.1). So this package's source code could be in

```
$(root)$(ps)packages$(ps)daybreak
```

which on Unix might be

```
/home/apr/java/proj1/packages/daybreak
```

All of the package's source files must, of course, have a suitable package statement:

```
package daybreak;
```

# 3 Using SciTE for Programming

Once the project folders and directory properties file are created, the SciTE editor can used to edit the Java source code files that are in the project's root and source folders.

To compile and execute the program, the following tools menu commands can be used

**Compile:** Compiles the current Java file.

**Build:** Compiles all the Java files in the project.

**Go:** Execute the program provided the current file has a main method. Use this unless there are is a problem with standard stream input (see §3.1).

Compilation errors are displayed the output pane, and a double mouse click will take you to the offending line of code.

As well as the standard commands given above, there are some additional commands in the tools menu:

**Clean:** Delete all the files and sub-folders in the project's class folder. The name of this folder is specified in the project's property file. By default it is called `classes`.

**Make JAR:** Builds a JAR file for the project, and put it in the project's root folder. This command is only available if has been enabled and configured in the project's property file.

**Go in Terminal:** *Only available on Linux.* Execute the program in a terminal window, provided the current file has a main method. Use this command if the program uses the `System.in` stream. This is functionally equivalent to the Go command above, but avoids the problem discussed in §3.1.

### 3.1 Problem with Go on Linux

*Programs launched with SciTE's Go command command on Linux cannot use the standard input stream ...*

The standard input and output streams of a launched program are directed to SciTE's output pane, where output can be printed, and input can be typed in. Unfortunately on Linux, the input stream just returns End-of-File and all typed input is ignored.

This appears to be a problem only with Linux. Program input works when SciTE is running on Windows. SciTE has no documentation relating to this problem.

*The problem is resolved by the Go in Terminal command described above.*

## 4 General Properties

The project set up procedure given in §2 suggests that all of the assignments in §7.1 are put into the user properties file. Doing this applies them to all file types in all locations.

However, not all of these have to be in user properties. Some can be directory properties, and thus be applied to only the files in the project, or they can not be used at all. The following features are in the set up:

**Directory Properties:** Enables directory properties files. *This must be in the user properties file.*

**Output Pane:** A horizontal pane displays compiler errors better than the default vertical pane, but it can be temporarily changed back to vertical from the options menu. *If this feature is wanted, it must be set in the user properties file.*

**Save Buffers:** Save all buffers before executing the compile, build and go tools menu commands. *This does not have to be in the user properties file.*

**Tabbing:** The editors automatic indentation and tab width are set to three spaces, and the tabs converted to spaces. This is a personal preference, but converting tabs to spaces gives consistent display and printing. *This does not have to be in the user properties file.*

**Fonts:** The editor's font are be made permanently non-proportional. Non-proportional fonts are preferred for programming. However this feature applies to all file types, and cannot be disabled because the option menu's 'use monospaced font' toggle stops working. *This does not have to be in the user properties file.*

# 5 How it Works

The successful operation of this approach relies on SciTE's properties files. In particular the directory and user properties files.

## 5.1 SciTE Properties Files

SciTE has four types of properties files:

**Local properties file** called `SciTE.properties` that can be in the same directory as the file being edited. If present, it applies to all the file in that folder.

**Directory properties file** called `SciTEDirectory.properties` that can in the same or in a parent directory as the file being edited. If present, it is applied to all the files in the folder and all of its sub-folders.

**User properties file** called `.SciTEUser.properties` on Unix, and on Windows it is `SciTEUser.properties`. It is normally stored in the user's root folder, and it applies to all of the user's files.

**Global properties file** called `SciTEGlobal.properties`. This is provided as part of SciTE's installation, and it is applied to all files. *It is normally best if the global properties are not changed.*

The settings in the various properties files have priority. A local setting overrides a directory setting, which overrides a user setting, which finally overrides the setting in the global properties file.

## 5.2 The Sample Properties Files

The directory priorities file provides Tools menu commands for Java compilation and JAR generation. The user properties file provides some general settings, and the fix for the 'no standard input stream on Linux' problem.

### 5.2.1 Directory Properties

In the directory properties, separate source file folders and a class file folder are achieved by using suitable parameters for `javac` and `java` commands. The variables `cfold` and `moreclasses` configure their class path, and `root` and `sfolds` configure their source path.

The `root` variable is of particular importance. It is automatically set to the absolute path of the folder that the directory properties file is in with

```
root=$(SciteDirectoryHome)
```

This is the project's root folder, because that is where the properties file has been placed. So `root` can be used to specify all of the project's folders in a way that is position and platform independent. Move the project folder and the root changes appropriately. Even across platforms, so `$(root)` will become `/home/adrian/java/proj1` or `C\:Users\Adrian\java\proj1` as required.

Using `root` as part of a full path in a way that is platform independent is more of a problem. It is achieved at the cost of a clear notation by using

variables for the slash in path names, and the list separator character. Unix needs `/` and `:` characters, and Windows wants `\` and `;` characters.

So the variables `${ps}` and `$(ls)` are conditionally assigned depending on the current operating system like this:

```
1 if PLAT_WIN
2  ls=;
3  ps=\\
4
5 if PLAT_GTK
6  ls=:
7  ps=/
```

The variables `PLAT_WIN` and `PLAT_GTK` are set by SciTE to indicate what the operation system is. So when `${ps}` and `$(ls)` are used, they will be replaced by the correct character for the current platform. The `\\` in line 3 is an 'escape sequence' that is needed because `\` on its own is interpreted as next line continuation. Line 4 must be completely empty.

Conditional logic is also used for the make JAR and clean tools menu commands. The delete operation used for cleaning the class folder is different in Windows and Unix, so `PLAT_WIN` and `PLAT_GTK` are used again to select the correct version. The make JAR command is optional, so the variable `JAR` and an if statement are used to manage this.

SciTE commands that need multiple shell commands are implemented with command lists, where shell commands are separated by `;` characters in Unix and `&` characters in Windows.

### 5.2.2 User Properties

These are mostly simple assignments to SciTE variables to set up the general properties described in §4. Selecting a monospaced font as the default is rather more verbose, but there is unfortunately no simpler way to to this.

The solution to the 'no standard input stream on Linux' problem is also here. It is a 'Go in Terminal' command that executes the Go command in a separate terminal window. It uses conditional logic so that it is only available on Linux.

# 6 Limitations

Using SciTE in this way works very well for most Java projects, but it has some identified limitations:

- Launching programs with command line arguments is not supported.
- Debugging is not supported.

# 7 Sample Properties Files

This are sample properties files that are used to configure SciTE Java projects. The user properties in §7.1 is added to `SciTEUser.properties`, and the direc-

tory properties in §7.2 becomes a `SciTEDirectory.properties` file thst goes in the root folder of every project.

The latest versions of these files can be obtained from the `nepsweb.co.uk` web site.

## 7.1 User Properties

```
############################################
# Must be in user properties if wanted
############################################

# Directory properties enabled
#
properties.directory.enable=1

# Output Pane is horizontal
#
split.vertical=0
clear.before.execute=1

############################################
#  Do not have to be in user properties
############################################

# Save all buffers for build
#
save.all.for.build=1

# Tabbing is 3 spaces
#
tabsize=3
indent.size=3
use.tabs=0

# Always use monospace font
#
font.base=$(font.monospace)
font.small=$(font.monospace)
font.comment=$(font.monospace)
font.text=$(font.monospace)
font.text.comment=$(font.monospace)
font.embedded.base=$(font.monospace)
font.embedded.comment=$(font.monospace)
font.vbs=$(font.monospace)
```

```
# For Java Go command input stream problem
#
if PLAT_GTK
 command.name.0.*.java=Go in Terminal
 command.0.*.java=gnome-terminal -x sh -c "\
$(command.go.*.java);\
echo;\
read -p 'hit enter to close ...' x\
"
```

## 7.2 Directory Properties

```
#################################
# Separators - do not change
#################################
# ls = list separator
# pc = path separator

if PLAT_WIN
 ls=;
 ps=\\

if PLAT_GTK
 ls=:
 ps=/

#################################
# Project root - do not change
#################################
# This is the project's root directory path.
# Use this variable to define other paths as relative to
# project folder.
root=$(SciteDirectoryHome)

###############################################
# Configuration variables - change as required
###############################################

# Various compiler options.
#    opt1 -> Verbose output
#    opt2 -> Disable warning messages
#    opt3 -> Generate debugging information
#    opt4 -> Enable recommended warnings
#    Uncomment the following arrequired...
#opt1= -verbose
#opt2= -nowarn
#opt3= -g
#opt4= -Xlint
opts=$(opt1)$(opt2)$(opt3)$(opt4)
```

```
# Main class
#    Set this to reach the folder where the project's classes
#    with maim methods are stored.
#    Where a '/' or a '\' is needed use $(ps)
main=$(root)
#main=$(root)$(ps)sourceMain

# Source folders
#    Uncomment and change as required.
#    Separate list items with $(ls).
#    Use $(root) for a path relative to project's root folder.
#    Where a '/' or a '\' is needed use $(ps)
#sfolds=$(root)$(ps)utility$(ls)$(root)$(ps)sourceMain

# classes folder - must exist in project's root folder
# It will be emptied by the clean command (!)
cfold=$(root)$(ps)classes

# More class paths
#    Uncomment and change as required for more class paths
#    Separate list items with $(ls).
#    Where a '/' or a '\' is needed use $(ps)
#moreclasses=classpath1$(ls)classpath2

# JAR build information command
#   JAR=1 enables the JAR creation tool
#   jarname is the name of the JAR file
#   jarmain is the name of the class with the main method
#   jarmore are more resources for inclusion in the JAR
JAR=0
jarname=helloworld
jarmain=Hello
#jarmore=-C $(root)$(ps)class2 MyClass.class


##############################################
# Standard commands
##############################################

command.compile.*.java=javac$(opts)\
 -d $(cfold)\
 -classpath $(cfold)$(ls)$(moreclasses)\
 -sourcepath $(root)$(ls)$(sfolds)\
 $(FileNameExt)

command.build.*.java=javac$(opts)\
 -d $(cfold)\
 -classpath $(cfold)$(ls)$(moreclasses)\
 -sourcepath $(root)$(ls)$(sfolds)\
 $(main)$(ps)*.java
```

```
command.go.*.java=java\
 -classpath $(cfold)$(ls)$(moreclasses)\
 $(FileName)

##########################################
# Custom commands
##########################################

#-----------------------------------------
# Java clean
#    Delete all files in class folder
#-----------------------------------------

command.name.1.*.java=Clean

if PLAT_WIN
 command.1.*.java=\
@echo deleting class folder &\
rd /s/q $(cfold) &\
@echo recreating class folder &\
md "$(cfold)" &\
@echo done

if PLAT_GTK
 command.1.*.java=rm -r $(cfold)/*

#-----------------------------------------
# Java create JAR
#-----------------------------------------

if JAR
 command.name.2.*.java=Make JAR

if PLAT_GTK
 command.2.*.java=\
jar cfe $(root)$(ps)$(jarname).jar $(jarmain)\
 -C $(cfold) . $(jarmore);\
echo JAR made ;\
chmod a+x $(jarname).jar

if PLAT_WIN
 command.2.*.java=\
jar cfe $(root)$(ps)$(jarname).jar $(jarmain)\
 -C $(cfold) . $(jarmore)
```