A Problem with the SciTE Go Command on Linux

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

22nd February, 2012

1 A Problem with Input

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.

2 Resolving the Problem

The above problem with the Go command cannot be properly fixed in SciTE or Linux, but it there is a solution by defining a new 'Go in Terminal' command as follows:

```
if PLAT_GTK
 command.name.⟨number⟩.⟨filepattern⟩=Go in Terminal
 command.⟨number⟩.⟨filepattern⟩=gnome-terminal -x sh -c "\
$(command.go.⟨filepattern⟩);\
echo;\
read -p 'hit enter to close ...' x\
"
```

Where ⟨*number*⟩ and ⟨*filepattern*⟩ are replaced by appropriate values. For example:

```
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\
"
```

The new command appears in the Tool menu if a file matching ⟨*filepattern*⟩ is in the active buffer. Now instead of launching the program with the Go command, the Go in Terminal command is used instead. This opens a terminal window to run the program and the input stream works as expected.

3 Deployment

The new command can be put in local, directory or user properties files. However, it is normally best to put it in *user properties.* Separate versions of the Go in Terminal command, with distinct ⟨*filepattern*⟩, are needed for each programming language that has input stream problems. Using 0 for ⟨*number*⟩ is probably a good choice, since this is not used by most of the standard assignments, but the global language, user and relevant local and directory properties should be checked.

4 Limitations

Two techniques have to be defined before the limitations of this approach can be described:

Command indirection is using a command variable as an assignment to another command variable. For example:

```
command.go.*.java=java $(FileName)
...
command.name.4.*.java=Demo
command.4.*.java=$(command.go.*.java)
```

So in this case, invoking Demo actually executes the Go command.

LHS assignment is using a variable on the left hand side of an assignment. For example:

```
filepattern=*.java
command.go.$(filepattern)=java $(FileName)
```

The limitation of this solution can now be simple stated: *Command indirection cannot be used with LHS assignment.*

In particular, the Go in Terminal command given in §2 uses command indirection, so LHS assignment must not not be used by the Go command. Fortunately, many of SciTE's supported languages, including Java and Python, do not use LHS assignment. So they are compatible with this solution.

There are problems with some languages such as Lisp, Perl and C++. The solution in these cases is reform the given Go in Terminal command to use the standard file patterns, but to explicitly give the relevant Linux command to execute the program.