User's Manual for Psd

for version 1.1

Pertti Kellomäki, `pk@cs.tut.fi`
Tampere University of Technology
Software Systems Lab
Finland

August 19, 1997

1 Introduction

This is the user's manual for psd, the Portable Scheme Debugger. Psd is a source level debugger for the Scheme language, based on instrumenting the original source code. Psd is placed under the GNU General Public License ("copyleft"), so you are free to use, distribute and modify it as long as you let other people do it, too. See the file `COPYING` in the psd distribution for more details.

This manual describes version 1.1 of psd. To see what has changed from version 1.0, see end of this manual.

2 What You Need

In order to use psd you need GNU Emacs (either version 18 or 19 will do, I have tried psd with with 18.58 and 19.19), the cmuscheme package for Emacs, a preferably R4RS [1] compliant Scheme interpreter and the psd package. GNU Emacs can be found at many major ftp sites, for example nic.funet.fi. These ftp sites also carry Scheme interpreters. A good place to look for Scheme related material is the Scheme Repository at nexus.yorku.ca maintained by Ozan Yigit, or its mirror sites, at least ftp.inria.fr and faui80.informatik.uni-erlangen.de. Psd is available at the repository as well as at cs.tut.fi using anonymous ftp (`/pub/src/languages/schemes`).

3 Supported Systems

Psd is known to work with Aubrey Jaffer's scm, Oliver Laumann's elk, and sci (the interpreter in the Scheme to C compiler system).

By default psd assumes that you are using slib, the utility library maintained by Aubrey Jaffer. Psd tries to find the file `psd-slib.scm`, and complains if it can not find it. If you are not using slib, set the Emacs variable `psd-using-slib` to nil, for example in your `.emacs`:

```
(setq psd-using-slib nil)
```

If `psd-using-slib` is nil, psd first tries to find the file `psd-`*foo*`.scm`, where *foo* is the value of the Emacs variable `scheme-program-name`, and if it is not found, it loads the file `psd.scm`.

4 Preparing Programs for Debugging

The easiest — many would probably argue the only — way to use psd is to use the GNU Emacs interface. For now, we'll assume you have all set up for using psd. See section "Installing Psd" at the end of this manual, if your friendly system administrator has not already installed psd for you.

To use the interface, start up a Scheme session with `M-x run-scheme`. If the mode line of the `*scheme*` buffer says "Inferior Scheme Psd: run", then psd is already ready for use. If not, you will need to give the command `M-x psd-mode` in the Scheme buffer. The necessary Scheme code is now loaded into your Scheme session. If you want psd to be loaded automagically when you start up a Scheme session, you can do that by using a "hook". In Emacs 18 this is done by putting the line

```
(setq inferior-scheme-mode-hook
      (cons '(lambda () (psd-mode 1))
            inferior-scheme-mode-hook))
```

in your `.emacs` file. With Emacs 19, the kosher way to do is

```
(add-hook 'inferior-scheme-mode-hook
          (function (lambda () (psd-mode 1))))
```

Psd can only debug procedures that reside in files. Thus, it is not possible to type procedures directly in the Scheme process buffer and debug them with psd.

To debug all procedures in a Scheme source file, give the command `C-c d` or `M-x psd-debug-file` either in the inferior Scheme buffer, or in a buffer containing Scheme source. If everything goes well, psd will first produce an instrumented version of your file, and then load it. You can also pick a single procedure to be debugged by going to a Scheme buffer and using the commands `ESC C-x`, `C-c e` and `C-c C-e`. If given a prefix argument, they will instrument the definition the cursor is on and load it into Scheme. For example, the command `C-u C-c C-e` will instrument a definition and send it to the Scheme process.

5 Running the Debugged Program

After the instrumentation is complete, you have in your Scheme environment top level definitions equivalent to those that you would have gotten if you had loaded the original source file. However, all the procedures have been instrumented so that you can step thru the evaluation process, and see in another Emacs window where in the source code you are.

The debugged procedures can be invoked either from top level or from other procedures. In order to stop the execution to a specific line, go to the line and give the command `M-x psd-set-breakpoint` or `C-x SPC`. This will set a breakpoint to the specified line. In Emacs 19, you can also use the menu bar to set and clear breakpoints.

You can also specify that the debugger should be invoked when a specified procedure is entered. Use the command `M-x psd-break` or `C-c b`.

6 Debugger Commands

The user interface is not very sophisticated, but undoubtedly it will have more commands in the future. If you have suggestions of what would be useful, feel free to contact me. The commands are:

val *sym* gives the value of *sym* in the current scope.

set! *sym val* sets the value of *sym* to *val* in the current scope.

g continues evaluation until the next breakpoint

c clear any breakpoints on the current line. Also clears break on entry specified with psd-break.

w shows the current context as file name and a list of procedure names. For example, `"/tmp/killme.scm:(encode encode-symbol)"` means, that the you are in file `/tmp/killme.scm`, inside a procedure called `encode-symbol`, which is inside the procedure `encode`. I am considering removing this feature, because I don't remember using it at all myself.

s steps one step in the evaluation process. Each time an expression is about to be evaluated, psd displays it and waits for a command. When an expression has been evaluated, psd displays the result and waits for a command.

n continues evaluation until evaluation reaches a different line

r *expr* evaluates expr and returns its value as the return value of the current expression

A list is taken to be a procedure call that is to be evaluated. All the essential procedures in R4RS are visible to the evaluator. Any other command displays a list of available commands.

If the debugger does not seem to be doing the right things, try the Emacs command `M-x psd-reset`, which will clear all the breakpoints and reset the runtime system.

7 Catching Run Time Errors

Psd tries to detect any run time errors before they are encountered. Before every call to an essential procedure in R4RS, it checks the number of arguments and their types. If a run time error would occur, psd stops execution and lets the user inspect and modify the environment. In order to continue, use the debugger command `r` and give a value that is to be returned as the result of the call. This works of course only if the code where the call is made from is being debugged.

8 Limitations of the Current Implementation

The current version handles all syntactic forms except `=>`, `delay` and unquoting. Unquoting is supported in the sense that procedures containing quasiquote and unquotations can de debugged, but it is not possible to step thru an unquotation, or set a breakpoint within a quasiquotation.

Macros are not supported at all. It would require that the instrumentation code would have to keep track of all macro definitions and be able to expand macros in their full glory.

The reader understands symbols, boolean values, strings, vector, characters, integers, simple floats and lists. Fancier numbers like complex numbers etc. are not supported. They are not very hard to implement, they are just not on top of the priority list for me. Hex, octal and binary numbers do work, though, thanks to Edward Briggs.

One thing that psd is not guaranteed to preserve is the order of evaluation. Because of the additional code that psd adds to the program, it is possible that the instrumented version of a procedure call is evaluated in a different order than the original. If the Scheme implementation used evaluates all arguments from left to right or right to left, there is no problem. If, however, the order of evaluation is something more exotic, the order of evaluation may change. In practice this is probably not a problem.

Because the instrumented programs and the runtime support for the debugger live in the same name space, there are some names that can not be used in the debugged programs. In psd, all the globally visible procedures start with the prefix `psd-`, and variables with the prefix `*psd-`. Do not use these prefixes in your programs.

9 Command Summary

This is a short list of the available commands. The Emacs commands are:

`C-c b`	psd-break
`C-c d`	psd-debug-file
`C-c e`	scheme-or-psd-send-definition
`C-c C-e`	scheme-or-psd-send-definition-and-go
`C-x SPC`	psd-set-breakpoint
`ESC C-x`	scheme-or-psd-send-definition
`M-x psd-reset`	clear all breakpoints and reset the psd runtime

The debugger commands are:

`val` *sym*	give the value of *sym*
`set!` *sym val*	set the value of *sym* to *val*
`g`	run until the next breakpoint
`c`	clear breakpoints on current line
`w`	give the current context
`s`	step one step in the evaluation process
`n`	run until evaluation reaches another line
`r` *expr*	return expr as the value of current expression

A list is taken to be a procedure call to be evaluated. It can also be a `set!` form.

10 Customizing the Emacs Interface

If you want to customize the Emacs interface, eg. to change key bindings, you can do it by using a hook called `psd-mode-hook`. If, for example, you want to bind the key `C-b` to run `psd-break` in Scheme buffers (not a very bright idea!), you can do it by putting the following in your `.emacs` (this is for Emacs 19):

```
(add-hook 'psd-mode-hook
         (function (lambda () (define-key scheme-mode-map
                                           "\C-b"
                                           'psd-break))))
```

11 Installing Psd

As mentioned, you need the cmuscheme package for GNU Emacs in order to use psd. There is a version of cmuscheme.el along with comint.el in the psd distribution, but be warned that they may be a bit dated. For Emacs version 19 you only need `cmuscheme.el`, the comint package has been integrated into Emacs. If you do not have cmuscheme, Emacs will give an error message saying it can not find the file cmuscheme when you load psd.el. The following files need special care:

***.scm** are the Scheme files that psd needs. Copy them to some suitable place.

psd.el is the Emacs interface. Copy it to a directory that is mentioned in the Emacs variable `load-path` You can check the value of `load-path` by typing `load-path LFD` (subsitute control-J for `LFD` if you do not have a line feed key) in the Emacs `*scratch*` buffer. To make psd autoloadable, put the the following in your `.emacs` or the global `default.el`:

```
(autoload 'psd-mode
          "psd"
          "Minor mode for running psd (the Portable Scheme Debugger)
in a cmuscheme buffer."
          t)
```

If you want to add a directory in your load path, put something like the following in your `.emacs`:

```
(setq load-path (cons "~/elisp" load-path))
```

cmuscheme.el is needed, if Emacs complains it can not find it. Copy it to the same place you copied `psd.el`.

comint.el is needed only with GNU Emacs version 18. Copy it to the same place as `cmuscheme.el`, if needed.

Note: if you already have psd 1.0 installed, make sure that you remove at least the file `psd-elk.scm` from `/usr/local/lib/psd`, or wherever you have psd. It is generally a good idea to wipe out the old distribution before installing a new one, so you might just as well remove all the old psd files.

12 Changes from version 1.0

Version 1.1 corrects a number of bugs. In addition to that, the user interface is improved a bit. The ability to remove breakpoints is new, as well as the menu stuff for Emacs 19. Breakpoints can be specified by giving the name of a procedure. Elk does not need any special treatment, so there is no file `psd-elk.scm` in the distribution.

13 Acknowledgements

Thanks go to Edward Briggs and Aubrey Jaffer for their suggestions and modifications to psd. Also, special thanks go to Tatu Männistö for lively discussions about psd and for keeping up the Scheme spirit in the department. My sympathies go to the students in Introduction to Programming and Principles of Programming in Tampere University of Technology, who have struggled with psd.

References

[1] Jonathan A. Rees and William Clinger, editors. The revised^4 report on the algorithmic language Scheme. *LISP Pointers*, IV(3):1–55, July–September 1992.