

Prism 5.0 Reference Manual



THE NETWORK IS THE COMPUTER™

Sun Microsystems Computer Company

A Sun Microsystems, Inc. Business
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
650 960-1300 fax 650 969-9131

Part No.: 805-1553-10
Revision A, November 1997

Copyright 1997 Sun Microsystems, Inc., 901 San Antonio Road • Palo Alto, CA 94303 USA. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, AnswerBook, SunDocs, Solaris, OpenWindows, Sun HPC Software, Ultra HPC, Ultra HPC Cluster, UltraSPARC, Sun Performance WorkShop Fortran, and Sun Performance Library are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 1997 Sun Microsystems, Inc., 901 San Antonio Road • Palo Alto, CA 94303 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, AnswerBook, SunDocs, Solaris, OpenWindows, Sun HPC Software, Ultra HPC, Ultra HPC Cluster, UltraSPARC, Sun Performance WorkShop Fortran, et Sun Performance Library sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REPRENDRE A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Contents

Preface vii

1. Commands Reference 1-1

 Redirecting Output 1-1

 Using Pset Qualifiers in MP Prism 1-2

 These commands can take a pset qualifier: 1-3

 Prism Commands 1-3

 /, ? 1-7

address/ 1-8

value=base 1-11

 alias 1-12

 assign 1-13

 attach 1-14

 call 1-15

 catch 1-16

 cd 1-17

 collection 1-18

 cont 1-19

 contw 1-20

 core 1-21

 cycle 1-22

define pset 1-23
delete 1-25
delete pset 1-26
detach 1-27
disable 1-28
display 1-29
down 1-32
dump 1-33
edit 1-34
enable 1-35
eval pset 1-36
fg 1-37
file 1-38
func 1-39
help 1-40
hide 1-41
ignore 1-42
interrupt 1-43
kill 1-44
list 1-45
load 1-46
log 1-47
make 1-48
next 1-49
nexti 1-50
perf 1-51
perflload 1-52
perfsave 1-53
perftip 1-54
print 1-55

printenv 1-58
process 1-59
pset 1-60
pstatus 1-61
pushbutton 1-62
pwd 1-63
quit 1-64
reload 1-65
rerun 1-66
return 1-67
run 1-68
select 1-69
set 1-70
setenv 1-72
sh 1-73
show 1-74
show events 1-75
show pset 1-76
show psets 1-77
source 1-79
status 1-80
step 1-81
stepi 1-82
stepout 1-83
stop 1-84
stopi 1-86
tearoff 1-88
tmrunargs 1-89
trace 1-90
tracei 1-92

unalias 1-94
unset 1-95
unsetenv 1-96
untearoff 1-97
up 1-98
use 1-99
varsave 1-100
wait 1-101
whatis 1-102
when 1-103
where 1-105
whereis 1-106
which 1-107

A. Prism man page A-1

Index Index-1

Preface

This manual provides reference descriptions of commands available in the Prism programming environment.

The manual is intended for application programmers developing serial or parallel programs that are to run on a Sun™ HPC System. We assume you know the basics of developing and debugging programs, as well as the basics of the system on which you will be using Prism. Some familiarity with the UNIX® debugger dbx is helpful but not required. The Prism interface is based on the X and OSF/Motif standards. Familiarity with these standards is also helpful but not required.

Using UNIX Commands

This document may not contain information on basic UNIX® commands and procedures such as shutting down the system, booting the system, and configuring devices.

See one or more of the following for this information:

- AnswerBook™ online documentation for the Solaris™ 2.x software environment
- Other software documentation that you received with your system

Typographic Conventions

TABLE P-1 Typographic Conventions

Typeface or Symbol	Meaning	Examples
AaBbCc123	The names of commands, files, and directories; on-screen computer output.	Edit your <code>.login</code> file. Use <code>ls -a</code> to list all files. % You have mail.
AaBbCc123	What you type, when contrasted with on-screen computer output.	% su Password:
<i>AaBbCc123</i>	Book titles, new words or terms, words to be emphasized. Command-line variable; replace with a real name or value.	Read Chapter 6 in the <i>User's Guide</i> . These are called <i>class</i> options. You <i>must</i> be <i>root</i> to do this. To delete a file, type <code>rm filename</code> .

Shell Prompts

TABLE P-2 Shell Prompts

Shell	Prompt
C shell	<i>machine_name%</i>
C shell superuser	<i>machine_name#</i>
Bourne shell and Korn shell	\$
Bourne shell and Korn shell superuser	#

Related Documentation

TABLE P-3 Related Documentation

Application	Title	Part Number
All	<i>Sun HPC Software 2.0 System Administrator's Guide</i>	805-1554-10
All	<i>Sun HPC Software 2.0 Release Notes</i>	805-2191-10
Sun MPI Programming	<i>Sun MPI 3.0 Guide</i>	805-1556-10
Prism	<i>Prism 5.0 User's Guide</i>	805-1552-10
Prism	<i>Prism 5.0 Reference Manual</i>	805-1553-10
Sun HPF Programming	<i>Sun HPF 1.0 Guide</i>	805-1558-10
S3L	<i>S3L 2.0 Guide</i>	805-1557-10

Ordering Sun Documents

SunDocsSM is a distribution program for Sun Microsystems technical documentation. Contact SunExpress for easy ordering and quick delivery. You can find a listing of available Sun documentation on the World Wide Web.

TABLE P-4 SunExpress Contact Information

Country	Telephone	Fax
Belgium	02-720-09-09	02-725-88-50
Canada	1-800-873-7869	1-800-944-0661
France	0800-90-61-57	0800-90-61-58
Germany	01-30-81-61-91	01-30-81-61-92
Holland	06-022-34-45	06-022-34-46
Japan	0120-33-9096	0120-33-9097
Luxembourg	32-2-720-09-09	32-2-725-88-50
Sweden	020-79-57-26	020-79-57-27

TABLE P-4 SunExpress Contact Information

Switzerland	0800-55-19-26	0800-55-19-27
United Kingdom	0800-89-88-88	0800-89-88-87
United States	1-800-873-7869	1-800-944-0661
World Wide Web: http://www.sun.com/sunexpress/		

Sun Documentation on the Web

The `docs.sun.com` web site enables you to access Sun technical documentation on the World Wide Web. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com`

Sun Welcomes Your Comments

We are interested in improving our documentation and welcome your comments and suggestions. You can email or fax your comments to us. Please include the part number of your document in the subject line of your email or fax message.

- Email: `smcc-docs@sun.com`
- Fax: SMCC Document Feedback
1-650-786-6443

LSF Technical Support

LSF 3.0, a product of Platform Computing Corporation, is part of the Sun HPC Software 2.0 Foundation Package. As such, it is supported by Sun as part of Sun HPC Software 2.0.

Sun HPC Software includes LSF Base and LSF Batch. However, LSF JobScheduler and LSF MultiCluster are not included and, therefore, not supported by Sun.

Information Sources for PVM and PETSc

TABLE P-5 lists organizations and resources for information about the publicly available libraries PVM and PETSc. This information is subject to change.

TABLE P-5 Information Sources for PVM and PETSc

Product	Contact
PVM	Copyright holders: University of Tennessee, Oak Ridge National Laboratory, Emory University Electronic mail: pvm@msr.epm.ornl.gov Newsgroup: comp.parallel.pvm Web site: http://www.epm.ornl.gov/pvm/pvm_home.html
PETSc	Developed and supported by the Mathematics and Computer Science Division of the Argonne National Laboratory.

Commands Reference

This reference manual gives, in alphabetical order, the syntax and reference description of every Prism command. This information is also available online:

- Choose the Commands Reference selection from the Help menu to obtain reference information about all Prism commands.
- Type `help commands` on the Prism command line to obtain summary information about Prism commands.
- Issue a command of the form `help commandname` on the command line to display the reference description of the command.

TABLE 2 lists the commands discussed in this manual.

Redirecting Output

You can redirect the output of most Prism commands to a file by including an *at* sign (@) followed by the name of the file on the command line. For example,

`where @ where.output`

puts the output of a `where` command into the file `where.output` in your current working directory within Prism.

You can also redirect output of a command to a window by using the syntax *commandname* on *window*, where *window* can be

- `command` (abbreviated `com`). *commandname* on `command` sends output to the command window; it is the default.
- `dedicated` (abbreviated `ded`). *commandname* on `ded` sends output to a window dedicated to output for this command. If you subsequently issue the same command (no matter what its arguments) and specify that output is to be sent to the dedicated window, this window will be updated. `snapshot` (abbreviated `sna`). *commandname* on `snapshot` creates a window that provides a snapshot of

the output. If you subsequently issue the same command and specify that output is to be sent to the snapshot window, Prism creates a separate window for the new output. The time each window was created is shown in its title. Snapshot windows let you save and compare outputs.

- You can also make up your own name for the window; *commandname* on *myname*, the name *myname* appears in the title of the window.

Note – You cannot redirect the output of the commands `edit`, `make`, and `sh`.

Using Pset Qualifiers in MP Prism

In MP Prism, certain commands can take a set of processors (pset) as a qualifier; they are listed at the end of this section. The format is

command pset *set_name* | *set_definition*

where *set_name* is the name of a pset and *set_definition* is the definition of an unnamed pset; see the `define pset` command for a discussion of how to define a pset.

Place this qualifier after any arguments to the command, but before the optional `on window` syntax that specifies the window to which output is directed (see the previous section). A command with a pset qualifier applies only to the processes in the set. If you omit the qualifier, the command applies to the processes in the current set.

These commands can take a pset qualifier:

TABLE 1 Commands Taking a pset Qualifier

address/	pstatus
assign	print
call	return, stepout
catch	step, stepi
cont, contw	stop, stopi
display	trace, tracei
ignore	wait
interrupt	whatis
next, nexti	where

Prism Commands

TABLE 2 lists all the Prism commands and brief descriptions in alphabetical order. It is followed by the complete command reference, also in alphabetical order.

TABLE 2 Prism Commands

Command	Use
<i>/string</i>	Searches forward in the current file for <i>string</i>
<i>?string</i>	Searches backward in the current file for <i>string</i> .
address/	Prints the contents of memory addresses.
<i>value=base</i>	Converts a value to a different base.
alias	Defines an alias.
assign	Assigns the value of an expression to a variable or array.
attach	Attaches to a running process or task.
call	Calls a procedure or function.
catch	Tells Prism to catch the signal you specify.

TABLE 2 Prism Commands

Command	Use
<code>cd</code>	Changes the current working directory.
<code>collection</code>	Turns collection of performance data on or off.
<code>cont</code>	Continues execution.
<code>contw</code>	Continues execution and then waits for members of the current pset to finish execution. (MP Prism only.)
<code>core</code>	Associates a core file with an executable program. (Not available in MP Prism.)
<code>cycle</code>	Makes the next member of the <code>cycle</code> pset the current set. (MP Prism only.)
<code>define pset</code>	Creates a named pset. (MP Prism only.)
<code>delete</code>	Removes one or more events from the event list.
<code>delete pset</code>	Deletes a user-defined pset. (MP Prism only.)
<code>detach</code>	Detaches from a running process or task.
<code>disable</code>	Disables an event.
<code>display</code>	Displays the values of one or more expressions or variables.
<code>down</code>	Moves the symbol-lookup context down one level.
<code>dump</code>	Displays the names and values of variables.
<code>edit</code>	Calls up an editor.
<code>enable</code>	Enables a previously disabled event.
<code>eval pset</code>	Updates the membership of a variable pset. (MP Prism only.)
<code>fg</code>	Runs the executable program in the foreground. (MP Prism only.)
<code>file</code>	Sets the source file to the specified file name.
<code>func</code>	Sets the current function to the specified function name.
<code>help</code>	Lists currently implemented commands.
<code>hide</code>	Hides a pane of a split source window. (Not available in commands-only Prism.)
<code>ignore</code>	Tells Prism to ignore the specified signal.
<code>interrupt</code>	Interrupts execution of processes. (MP Prism only.)
<code>kill</code>	Kills a process or task running within Prism.
<code>list</code>	Lists lines in the current source file.

TABLE 2 Prism Commands

Command	Use
load	Loads a program.
log	Creates a log file of your commands and Prism's responses.
make	Executes the make utility.
next	Executes one or more source lines, stepping over functions.
nexti	Executes one or more instructions, stepping over functions.
perf	Displays performance data.
perflload	Loads a performance data file.
perfsave	Saves a performance data file.
perftip	Displays an analysis of performance data.
print	Displays the values of one or more expressions or variables.
printenv	Displays currently set environment variables.
process	Sets or displays the current process of the current pset. (MP Prism only.)
pset	Sets or displays the current pset. (MP Prism only.)
pstatus	Displays the execution status of processes. (MP Prism only.)
pushbutton	Adds a Prism command to the tear-off region. (Not available in commands-only Prism.)
pwd	Displays the current working directory.
quit	Leaves Prism.
reload	Reloads the currently loaded program.
rerun	Reruns the currently loaded program, using arguments previously passed to the program.
return	Steps out to the caller of the current routine.
run	Starts execution of a program.
select	Chooses the master pane in a split source window.
set	Defines an abbreviation for a variable or expression.
setenv	Displays or sets environment variables.
sh	Passes a command line to the shell for execution.
show	Splits the source window. (Not available in commands-only Prism.)
show events	Displays the event list.

TABLE 2 Prism Commands

Command	Use
<code>show pset</code>	Displays the contents of a pset. (MP Prism only.)
<code>show psets</code>	Displays information about all psets. (MP Prism only.)
<code>source</code>	Reads commands from a file.
<code>status</code>	Displays the event list.
<code>step</code>	Executes one or more source lines.
<code>stepi</code>	Executes one or more instructions.
<code>stepout</code>	Steps out to the caller of the current routine.
<code>stop</code>	Sets a breakpoint.
<code>stopi</code>	Sets a breakpoint at an instruction.
<code>tearoff</code>	Adds a menu selection to the tear-off region. (Not available in commands-only Prism.)
<code>tmrnargs</code>	Specifies <code>tmrun</code> or <code>tmsub</code> options to use in executing multiprocess programs.
<code>trace</code>	Traces program execution.
<code>tracei</code>	Traces instructions.
<code>unalias</code>	Removes an alias.
<code>unset</code>	Removes an abbreviation created by <code>set</code> .
<code>unsetenv</code>	Removes the setting of an environment variable.
<code>untearoff</code>	Removes a button from the tear-off region. (Not available in commands-only Prism.)
<code>up</code>	Moves the symbol-lookup context up one level.
<code>use</code>	Adds a directory to the list to be searched for source files.
<code>varsave</code>	Saves values of a variable or expression to a file.
<code>wait</code>	Waits for a process or processes to stop execution. (MP Prism only.)
<code>whatis</code>	Displays the type of a variable.
<code>when</code>	Sets a breakpoint.
<code>where</code>	Displays a stack trace.
<code>whereis</code>	Displays the list of all fully qualified names for an identifier.
<code>which</code>	Displays the fully qualified name Prism chooses for an identifier.

/, ?

Searches forward or backward for a text string in the current source file.

SYNTAX

/string

?string

DESCRIPTION

Use the `/` command to search forward in the current source file for the text string you specify. If the string is found, the source pointer moves to the line that contains the string, and the line is echoed in the history region of the command window.

The `?` command works in the same way, except that it searches backward in the source file.

If the text string is not found, Prism displays the message

`No match`

in the history region.

address/

Prints the contents of the specified memory address.

SYNTAX

```
address , address / [mode] [pset set_name | set_definition]  
address | register / [count] [mode]
```

DESCRIPTION

Use these commands to print the contents of memory or of a register. If two addresses are separated by commas, Prism prints the contents of memory starting at the first address and continuing to the second address. If you specify a *count*, Prism prints *count* locations, starting from the address you specify.

If the address is . (period), Prism prints the address that follows the most recently printed address.

Specify a symbolic address by preceding the name with an &. For example,

```
&x/
```

prints the contents of memory for variable *x*.

The address you specify can be an expression made up of other addresses and the operators +, -, and indirection (unary *). For example,

```
0x1000+100/
```

prints the contents of the location 100 addresses above address 0x1000.

Specify a register by preceding its name with a dollar sign. For example,

```
$f0/
```

prints the contents of the f0 register. See below for a list of supported registers. If you specify *count* with a register, that number of registers is printed, starting with the specified register.

The *mode* argument specifies how memory is to be printed; if it is omitted, Prism uses the previous mode that you specified. The initial mode is *x*. These modes are supported:

TABLE 3 Mode Arguments

Mode	Description
d	Print a short word in decimal.
D	Print a long word in decimal.
o	Print a short word in octal.
O	Print a long word in octal.
x	Print a short word in hexadecimal.
X	Print a long word in hexadecimal.
b	Print a byte in octal.
c	Print a byte as a character.
s	Print a string of characters terminated by a null byte.
f	Print a single-precision real number.
F	Print a double-precision real number.
i	Print the machine instruction.

Supported UltraSPARC™ registers are listed below.

TABLE 4 Sun UltraSPARC Registers

Name	Register
\$g0-\$g7	Global registers (64 bits).
\$o0-\$o7	Output registers (64 bits).
\$l0-\$l7	Local registers.
\$i0-\$i7	Input registers.
\$psr	Processor state register.
\$pc	Program counter.
\$npc	Next program counter.
\$y	Y register.
\$wim	Window invalid mask.
\$tbr	Trap base register.

TABLE 4 Sun UltraSPARC Registers

Name	Register
\$f0-\$f31	Floating-point registers.
\$fsr	Floating status register (64 bits).
\$f0f1-\$f62f63	Floating-point registers.
\$xg0-\$xg7	Upper 32 bits of \$g0-\$g7.
\$xo0-\$xo7	Upper 32 bits of \$o0-\$o7.
\$xfsr	Upper 32 bits of \$fsr.
\$fprs	Floating-point registers state.
\$tstate	Trap state register.
\$fp	Frame pointer (synonym for \$i6).
\$sp	Stack pointer (synonym for \$o6).

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

value=base

Converts a value to the specified base.

SYNTAX

value=base

DESCRIPTION

Use the *value=base* command to convert the value you specify to the base you specify. The value can be a decimal, hexadecimal, or octal number. Precede hexadecimal numbers with 0x; precede octal numbers with 0 (zero). The base can be D (decimal), X (hexadecimal), or O (octal). Prism prints the converted value in the command window.

EXAMPLES

0x100=D
256

256=X
0x100

0x100=O
0400

0400=X
0x100

alias

Sets up an alias for a command or a string.

SYNTAX

```
alias  
alias new-name command  
alias new-name [ (parameters) ] "string"
```

DESCRIPTION

Use the `alias` command to set up an alias for a command or a string. When commands are processed, Prism first checks to see if the word is an alias for either a command or a string. If it is an alias, Prism treats the input as though the corresponding string (with values substituted for any parameters) had been entered.

For example, to define an alias `rr` for the command `rerun`, issue the command

```
alias rr rerun
```

To define an alias called `b` that sets a breakpoint at a particular line, you can issue the command

```
alias b(x) "stop at x"
```

You could then issue the command `b(12)`, which Prism expands to

```
stop at 12
```

Prism sets up some aliases for you automatically. Issue `alias` with no parameters to list the current set of aliases.

Issue the `unalias` command to remove an alias.

assign

Assigns the value of an expression to a variable or array.

SYNTAX

```
assign lval = expression [pset set_name | set_definition]
```

DESCRIPTION

Use the `assign` command to assign the value of *expression* to *lval*. *lval* can be any value that can go on the left-hand side of a statement in the language you are using, such as a variable or a Fortran array section. Prism performs the proper type coercions if the right-hand side does not have the same type as the left-hand side.

When issued in MP Prism, this command can take a `pset` qualifier. If used with a qualifier, it applies to the `pset` you specify. If used without a qualifier, it applies to the current `pset`. See page 2 for more information on `pset` qualifiers.

EXAMPLES

This example assigns the value 1 to `x`:

```
assign x = 1
```

If `x` is an array, 1 is assigned to each element.

This example adds 2 to each element of `array2` and assigns these values to `array1`:

```
assign array1 = array2 + 2
```

Note that `array2` and `array1` must be conformable.

attach

Attaches to a running process or task.

SYNTAX

```
attach pid | tid
```

DESCRIPTION

Use the `attach` command to attach to the running process with process ID *pid*, or to the running task with task ID *tid*. To work on a process or task within Prism, you must have previously loaded its executable program.

Use the `detach` command to detach a process running within Prism.

call

Calls a procedure or function.

SYNTAX

```
call procedure (parameters) [pset set_name | set_definition]
```

DESCRIPTION

Use the `call` command to call the specified procedure or function at the current stopping point in the program. Prism executes the procedure as if the call to it had occurred from the current stopping point. Breakpoints within the procedure are ignored, however.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

catch

Tells Prism to catch the specified Solaris™ signal.

SYNTAX

```
catch [number | signal_name] [pset set_name | set_definition]
```

DESCRIPTION

Prism can catch Solaris signals before they are sent to the program. Use the `catch` command to tell Prism to catch the signal you specify. When Prism receives the signal, execution stops, and Prism prints a message. A subsequent `cont` from a naturally occurring signal that is caught causes the signal to be propagated to signal handlers in the program (if any); if there is no handler for the signal, the program terminates—in other words, the program proceeds as if Prism were not present.

By default, Prism catches all signals except `SIGHUP`, `SIGEMT`, `SIGKILL`, `SIGALRM`, `SIGTSTP`, `SIGCONT`, `SIGCHLD`, and `SIGWINCH`; you can use the `ignore` command to add other signals to this list.

Specify the signal by number or by name. Signal names are case-insensitive, and the `SIG` prefix is optional.

Issue `catch` without an argument to list the signals that Prism is to catch.

When issued in MP Prism, this command can take a `pset` qualifier. If used with a qualifier, it applies to the `pset` you specify. If used without a qualifier, it applies to the current `pset`. See page 2 for more information on `pset` qualifiers.

cd

Changes the current working directory.

SYNTAX

`cd` [*directory*]

DESCRIPTION

Use the `cd` command to change your current working directory in Prism to *directory*; with no arguments, `cd` makes your login directory the current working directory.

The `cd` command is identical to its Solaris counterpart. See your Solaris documentation for more information.

collection

Turns collection of performance statistics on or off.

SYNTAX

```
collection [on|off]
```

DESCRIPTION

Use the `collection on` command to turn on collection of performance statistics. Use `collection off` to turn collection off. Issuing the command with the `on` or `off` argument also affects the setting of the toggle box in the Collection menu selection.

Issue `collection` with no arguments to display the current status of collection. This is primarily useful in commands-only Prism.

Note – Performance statistics collection works only for Sun HPF programs.

cont

Continues execution.

SYNTAX

```
cont [number | signal_name] [pset set_name | set_definition]
```

DESCRIPTION

Use the `cont` command to continue execution of the process from the point at which it stopped. If you specify a Solaris signal, by name or number, the process continues as though it received the signal. Otherwise, the process continues as though it had not been stopped.

You can use the default alias `c` for this command.

When issued in MP Prism, this command can take a `pset` qualifier. If used with a qualifier, it applies to the `pset` you specify. If used without a qualifier, it applies to the current `pset`. See page 2 for more information on `pset` qualifiers.

contw

In MP Prism, continues execution and then waits for the members of the current pset to finish execution.

SYNTAX

```
contw [number | signal_name] [pset set_name | set_definition]
```

DESCRIPTION

The `contw` command is available only in MP Prism. It is an alias for `cont; wait`

Issuing the command continues execution of the process from the point at which it stopped, then waits for the members of the current pset to finish execution. Most Prism commands are unavailable during this time.

If you specify a Solaris signal, by name or number, the process continues as though it received the signal. Otherwise, the process continues as though it had not been stopped.

This command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

core

Associates a core file with the loaded program.

SYNTAX

`core corefile`

DESCRIPTION

Use the `core` command to associate the specified core file with the program currently loaded in Prism. Prism reports the error that caused the core dump and sets the current line to the location at which the error occurred. You can then work with the program within Prism—for example, you can print the values of variables. You cannot continue execution from the current line, however.

The `core` command is not available in MP Prism. Instead, you must specify the name of the process core file you are interested in on the Prism command line. See the *Prism User's Guide* for more information.

cycle

In MP Prism, makes the next member of the `cycle` pset the current set.

SYNTAX

`cycle`

DESCRIPTION

The `cycle` command is available only in MP Prism.

Use the `cycle` command in MP Prism to cycle through the members of the `cycle` pset. The `cycle` pset is by default equivalent to the current set; you can set it to some other set via the `define pset` command. Issuing the `cycle` command advances the current process in the `cycle` pset to be the next member in the set and makes the current pset consist of only this process. This provides a convenient way of looking at each individual process within a pset.

EXAMPLE

This example defines a pset, makes it current, then cycles through its members, making each one the current set in turn:

```
(prism all) define pset foo 0:3
(prism all) pset foo
(prism foo) cycle
(prism 1) cycle
(prism 2) cycle
(prism 3) cycle
(prism 0)
```

define pset

In MP Prism, creates a named pset.

SYNTAX

```
define pset name definition
```

DESCRIPTION

The `define pset` command is available only in MP Prism.

Use the `define pset` command to create a pset with the name and definition you specify.

For the *name* argument, you can use any name except the predefined names `all`, `running`, `error`, `interrupted`, `break`, `stopped`, `done`, `current`, and `cycle`. The name must begin with a letter; it can contain any alphanumeric character, plus the dollar sign and underscore.

For the definition, you can specify any of the following:

- *An individual process number.*
- *The name of a pset.* The new pset will have the same definition as the existing set.
- *A list of process numbers.* Separate the numbers with commas. Use a colon between two process numbers to indicate a range. Use a second colon to indicate the stride to be used within this range.
- *A union, difference, or intersection of psets.* To specify the union, use the symbol `+`, `|`, or `||`. To specify the difference, use the minus sign `-`. To specify the intersection, use the symbol `&`, `&&`, or `*`.

Prism evaluates these expressions from left to right. For a union, if a process returns `true` for the first part of the expression, it is not evaluated further. For an intersection, if a process returns `false` for the first part of the expression, it is not evaluated further.

- *A condition to be met.* Put braces around an expression that evaluates to `true` or `false` on each process. Processes in which the expression is `true` are part of the set. This is referred to as a *variable* pset, since membership in it can vary depending on the current state of your program. Use the command `eval pset` to update the membership of a variable pset.

If a variable isn't active on a process, Prism prints an error message and does not execute the command. To ensure that the command is executed, use the intrinsic `isactive` in the pset definition. The expression `isactive(variable)` returns `true` if *variable* is on the stack for a process or is a global.

If a process that Prism tries to evaluate is running, the evaluation fails and the command is not executed. To avoid this, you can use the intersection of the predefined set `stopped` and the expression you want to evaluate. For example,

```
define pset xon stopped && { isactive(x) \
&& (x .NE. 0) }
```

This command defines a pset `xon` consisting of processes that are stopped and on which `x` is active and not equal to 0.

You cannot use this command in an event action.

Use the command `delete pset` to delete a pset that you have created using `define pset`.

EXAMPLES

This command creates a pset `foo` containing the processes 0, 4, and 7:

```
define pset foo 0, 4, 7
```

This command defines a pset `odd` containing the odd-numbered processes between 1 and 31:

```
define pset odd 1:31:2
```

This command defines a pset `quux` that contains processes that are members of either pset `foo` or pset `bar`:

```
define pset quux foo | bar
```

This command defines a pset `noty` that consists of all processes that are stopped except those in which `y` is equal to 1:

```
define pset noty stopped - { y == 1 }
```

delete

Removes one or more events from the event list.

SYNTAX

```
delete all | ID [ID...]
```

DESCRIPTION

Use the `delete` command to remove the events corresponding to the specified ID numbers. You obtain these numbers by issuing the `show events` command. Use the `all` argument to delete all existing events. Deleting the events also removes them from the event list in the event table.

You can use the default alias `d` for this command.

delete pset

In MP Prism, deletes a user-defined pset.

SYNTAX

```
delete pset set_name
```

DESCRIPTION

The `delete pset` command is available only in MP Prism.

Use the `delete pset` command to delete the pset *set_name*. If you have created events that apply to this pset, the events continue to exist. Their printed representation, however, is changed so that it shows the processes that were members of the pset at the time you deleted the set.

You cannot include the `delete pset` command in an event action.

Use the command `define pset` to create a pset.

detach

Detaches a process or task running within Prism.

SYNTAX

```
detach
```

DESCRIPTION

Use the `detach` command to detach the process or task that is currently running within Prism. The process or task must be stopped before it can be detached. Once detached, the process or task continues to run in the background, but it is no longer under the control of Prism.

Use the `attach` command to attach to a running process or task.

disable

Disables one or more events

SYNTAX

```
disable event_ID [event_ID ...]
```

DESCRIPTION

Use the `disable` command to disable the events with the corresponding ID numbers. (Use the `show events` command to list events along with their IDs.) Disabled events are kept in the event list, but they no longer affect execution. You can subsequently enable events by issuing the `enable` command. This can be more convenient than deleting events and then redefining them.

display

Displays the values of one or more variables or expressions.

SYNTAX

```
[where (expression)] display[/radix] expression [, expression ...]  
[pset set_name | set_definition]
```

DESCRIPTION

Use the `display` command to display the value(s) of the specified variable(s) or expression(s). The `display` command prints the values immediately and creates a display event, so that the values are updated automatically each time the program stops execution.

The optional `where` expression provides a mask for the elements of the parallel variable or array being displayed. The mask can be any expression that evaluates to true or false for each element of the variable or array. Elements whose values evaluate to true are considered *active*; elements whose values evaluate to false are considered *inactive*. If values are displayed in the command window, values of inactive elements are not printed. If values are displayed graphically, the treatment of inactive elements depends on the type of representation you choose.

The optional `/radix` syntax specifies the radix to be used in displaying the value(s). Possible settings of `/radix` are described in TABLE 5

TABLE 5 Radix Settings

Symbol	Radix
/b	Binary
/d	Decimal
/x	Hexadecimal
/o	Octal

The default radix setting is decimal, unless you have overridden the default via the `set $radix` command.

Redirection of output to a window via the `on window` syntax works slightly differently for `display` (and `print`) from the way it works for other commands. If you don't send output to the command window (the default), separate windows are created for each variable or expression that you display. Thus, the commands

display x on dedicated
display y on dedicated

create two dedicated windows, one for each variable; the two windows are updated separately.

Displaying to a window other than the command window creates a visualizer for the data.

To display the contents of a register, precede the name of the register with a dollar sign. For example,

display \$pc on dedicated

displays the contents of the program counter register.

Supported UltraSPARC registers are listed below.

TABLE 6 UltraSPARC Registers

Name	Register
\$g0-\$g7	Global registers (64 bits)
\$o0-\$o7	Output registers (64 bits)
\$l0-\$l7	Local registers
\$i0-\$i7	Input registers
\$psr	Processor state register
\$pc	Program counter
\$npc	Next program counter
\$y	Y register
\$wim	Window invalid mask
\$tbr	Trap base register
\$f0-\$f31	Floating-point registers, printable only as floats
\$fsr	Floating status register (64 bits)
\$f0f1-\$f62f63	Floating-point registers, printable only as doubles
\$xg0-\$xg7	Upper 32 bits of \$g0-\$g7
\$xo0-\$xo7	Upper 32 bits of \$o0-\$o7
\$xfsr	Upper 32 bits of \$fsr
\$fprs	Floating-point registers state

TABLE 6 UltraSPARC Registers

Name	Register
\$tstate	Trap state register
\$fp	Frame pointer (synonym for \$i6)
\$sp	Stack pointer (synonym for \$o6)

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

EXAMPLES

```
display sum(foo)
```

displays the sum of the elements of the array `foo`.

```
where (foo .ne. 0) display foo on dedicated
```

displays (in a dedicated window) the values of `foo` that are not equal to 0.

down

Moves the symbol lookup context down one level in the call stack.

SYNTAX

`down` [*count*]

DESCRIPTION

Use the `down` command to move the current function down the call stack (that is, toward the current stopping point in the program) *count* levels. If you omit *count*, the default is one level.

Issuing `down` repositions the source window at the new current function.

dump

Prints the names and values of local variables.

SYNOPSIS

```
dump [ function | . ]
```

DESCRIPTION

Note: The `dump` command is not available in MP Prism, or for data parallel programs.

Use the `dump` command to print the names and values of all the local variables in the function or procedure you specify. If you omit *function*, Prism uses the current function. If you specify a period (`.`), `dump` does a stack trace (like `where`), and prints the names and values of all local variables in the functions in the stack.

edit

Invokes an editor.

SYNTAX

```
edit [filename | procedure]
```

DESCRIPTION

Use the `edit` command to invoke an editor. With no arguments, the editor is invoked on the current file. If you specify *filename*, it is invoked on that file. If you specify *procedure*, it is invoked on the file that contains that procedure or function; Prism positions you at the start of the procedure.

The editor that is invoked depends on the setting of the Prism resource `Prism.editor`. If this resource is not set, Prism uses the setting of your `EDITOR` environment variable. If neither is set, the default editor is `vi`.

You cannot redirect the output of this command.

You can use the default alias `e` for this command.

enable

Enables previously disabled events.

SYNTAX

```
enable event_ID [ event_ID ... ]
```

DESCRIPTION

Use the `enable` command to enable the event with corresponding ID numbers. Use the `show events` command to list events along with their IDs and to determine which events are disabled. Use the `disable` command to disable events. Disabled events are kept in the event list, but they no longer affect execution. Enabling events via the `enable` command allows them to affect execution once again.

eval pset

In MP Prism, updates the membership of a variable pset.

SYNTAX

```
eval pset set_name
```

DESCRIPTION

The eval pset command is available only in MP Prism.

Use the eval pset command to update the membership of the variable pset *set_name*. You create a variable pset by issuing the define pset command and specifying a condition to be met. For example,

```
define pset foo stopped && {isactive(x) && (x>0)}
```

defines a pset *foo* that consists of all stopped processes in which *x* is active and is greater than zero. The membership of such a set can change as a program executes. To update its membership, issue the command

```
eval pset foo
```

If the evaluation fails (for example, because a process that was previously stopped is now running, and you didn't include the *stopped &&* syntax in your pset definition), the membership of the pset remains what it was before you issued the eval pset command.

fg

In MP Prism, runs the executable program in the foreground.

SYNTAX

fg

DESCRIPTION

The `fg` command is available only in the commands-only version of MP Prism, or if you are using graphical Prism without an Xterm for I/O.

Use the `fg` command to bring your executable program into the foreground. When you execute a message-passing program in the commands-only version of MP Prism, the program starts up in the background. You would need to bring it into the foreground if it needs to read terminal input. You cannot execute Prism commands while the program is executing in the foreground.

To have the program run in the background again and regain the `(prism)` prompt, type `Ctrl-z`.

file

Changes the current source file.

SYNTAX

```
file [filename]
```

DESCRIPTION

Use the `file` command to set the current source file to *filename*. If you do not specify a file name, `file` prints the name of the current source file.

Changing the current file causes the new file to be displayed in the source window. The scope pointer (-) in the line-number region moves to the current file to indicate the beginning of the new scope that Prism uses in identifying variables.

func

Changes the current procedure or function.

SYNTAX

`func` [*function*]

DESCRIPTION

Use the `func` command to set the current procedure or function to *function*. If you do not specify a procedure or function, `func` prints the name of the current function.

Changing the current function causes the file containing it to be displayed in the source window; this file becomes the current file. The scope pointer (–) in the line-number region moves to the current function to indicate the beginning of the new scope that Prism uses in identifying variables.

help

Gets help.

SYNTAX

`help [commands | command_name]`

DESCRIPTION

Use the `help` command to get help about Prism commands.

Use the `commands` option to display a list of Prism commands. Specify a command name to display reference information about that command.

Issuing `help` with no arguments displays a brief help message.

You can use the default alias `h` for this command.

hide

Removes a pane from a split source window.

SYNTAX

```
hide file_extension
```

DESCRIPTION

Use the `hide` command to remove one of the panes in a split source window. The pane that is removed contains the code specified by the file extension you supply as the argument to the command.

Use the `show` command to create a split source window.

The `hide` command is not meaningful in commands-only Prism.

EXAMPLES

To remove the pane containing the assembly code for the loaded program, issue this command:

```
hide .s
```

To remove the pane containing Fortran 77 source code, issue this command:

```
hide .f
```

ignore

Tells Prism to ignore the specified Solaris signal.

SYNTAX

```
ignore [number | signal_name] [pset set_name | set_definition]
```

DESCRIPTION

Prism can catch Solaris signals before they are sent to the program. Use the `ignore` command to tell Prism to ignore the specified signal. If the signal is ignored, Prism sends it to the program and allows the program to continue running without interruption; the program can then do what it wants with the signal. By default, Prism catches all signals except `SIGHUP`, `SIGEMT`, `SIGKILL`, `SIGALRM`, `SIGTSTP`, `SIGCONT`, `SIGCHLD`, and `SIGWINCH`; you can use the `catch` command to catch these signals as well.

Specify the signal by number or by name. Signal names are case-insensitive, and the `SIG` prefix is optional.

Issue `ignore` with no arguments to list the signals that Prism ignores.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

interrupt

In MP Prism, interrupts execution on processes.

SYNTAX

```
interrupt [pset set_name | set_definition]
```

DESCRIPTION

The `interrupt` command is available only in MP Prism.

Use the `interrupt` command to interrupt execution on processes. Without a pset qualifier, `interrupt` interrupts execution on the processes in the current pset. With a pset qualifier, `interrupt` interrupts execution on the processes in the set you specify.

The interrupted processes become members of the predefined pset `interrupted`.

This command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

EXAMPLES

This command interrupts the execution of the members of the predefined pset `running`:

```
interrupt pset running
```

This command interrupts the execution of process 5:

```
interrupt pset 5
```

kill

Kills a process or task running within Prism.

SYNOPSIS

```
kill
```

DESCRIPTION

Use the `kill` command to terminate the process or task that is currently running within Prism.

list

Lists lines in the current source file.

SYNTAX

```
list [source_line_number [ , source_line_number] ]  
list procedure
```

DESCRIPTION

Use the `list` command to list lines in the current file. The source window is repositioned. The command also affects the scope that Prism uses for resolving names. By default, the lines are displayed in the command window.

With no arguments, `list` lists the next 10 lines from the current line.

If you specify line numbers, the lines are listed from the first line number through the second.

If you specify a procedure or function, `list` lists 10 lines starting with the first statement in the procedure or function.

You can use the default alias `l` for this command.

load

Loads an executable program into Prism.

SYNTAX

`load filename`

DESCRIPTION

The `load` command loads the file specified by *filename* into Prism. The file must be an executable program compiled with the appropriate debugging switch.

When you execute `load`, the name of the program appears in the `Program` field of the main Prism window, and the source code that contains the main function of the program is displayed in the source window.

Use the `reload` command to reload the program currently loaded in Prism.

log

Creates a log file.

SYNTAX

log @ *filename*

log @@ *filename*

log off

DESCRIPTION

Use the log command to create a log file, *filename*, of your commands and Prism's responses.

Use the @@ form of the command to append the log to an already existing file.

Use log off to turn off logging.

make

Executes the `make` utility.

SYNTAX

`make` [*option*...]

DESCRIPTION

Use the `make` command to execute the `make` utility to update and regenerate one or more programs. You can specify any arguments that are valid in the Solaris version of `make`.

By default, Prism uses the standard Solaris `make`, `/bin/make`. You can change this by using the `Customize` utility or by changing the setting of the Prism resource `Prism.make`.

You cannot redirect the output of this command.

next

Executes one or more source lines, counting functions or procedures as single statements.

SYNTAX

```
next [n] [pset set_name | set_definition]
```

DESCRIPTION

Use the `next` command to execute the next *n* source lines, stepping over procedures and functions. If you do not specify a number, `next` executes the next source line.

You can use the default alias `n` for this command.

When issued in MP Prism, this command can take a `pset` qualifier. If used with a qualifier, it applies to the `pset` you specify. If used without a qualifier, it applies to the current `pset`. See page 2 for more information on `pset` qualifiers.

nexti

Executes one or more machine instructions, stepping over procedure and function calls.

SYNTAX

```
nexti [n] [pset set_name | set_definition]
```

DESCRIPTION

Use the `nexti` command to execute the next *n* machine instructions, stepping over procedures and functions. If you do not specify a number, `nexti` executes the next machine instruction.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

perf

Displays performance data.

SYNTAX

```
perf [util]
```

DESCRIPTION

Use the `perf` command to display an ASCII version of the performance data for a program. This typically produces a lot of output. However, you can use the `@ filename` syntax to redirect the output to a file in the usual way.

By default, `perf` displays absolute times in seconds for performance data. Use the `util` argument to display usage as percentages of elapsed time.

perflload

Loads a performance data file.

SYNTAX

`perflload filename`

DESCRIPTION

Use the `perflload` command to load the specified file containing performance data. If no program is loaded, Prism also loads the program for which the data has been saved. If another program is loaded, Prism asks if you want to load the program associated with the data; it loads the performance data in any case. If other performance data is already loaded, the new data overwrites it; you can load only one set of performance data at a time.

Use the `perfsave` command to save performance data to a file.

perfsave

Saves performance data to a file.

SYNTAX

```
perfsave filename
```

DESCRIPTION

Use the `perfsave` command to save the current performance data to the file you specify. You can then reload the data into Prism via the `perflload` command.

perftip

Displays an analysis of performance data.

SYNTAX

```
perftip
```

DESCRIPTION

Use the `perftip` command to display the output of Prism's performance advisor, which provides an analysis of the performance data for a program.

print

Prints the values of one or more variables or expressions.

SYNTAX

```
[where ( expression )] print[/radix] expression [ , expression ... ]  
                                [pset set_name | set_definition]
```

DESCRIPTION

Use the `print` command to print the values of the specified variable(s) or expression(s).

The optional `where` expression provides a mask for the elements of the parallel variable or array being printed. The mask can be any expression that evaluates to true or false for each element of the variable or array. Elements whose values evaluate to true are considered *active*; elements whose values evaluate to false are considered *inactive*. If values are printed in the command window, values of inactive elements are not printed. If values are printed graphically, the treatment of inactive elements depends on the type of representation you choose.

The optional `/radix` syntax specifies the radix to be used in printing the value(s). Possible settings of `/radix` are described in TABLE 7

TABLE 7 Radix Settings

Symbol	Radix
/b	Binary
/d	Decimal
/x	Hexadecimal
/o	Octal

The default radix is decimal, unless you have overridden the default via the `set $radix` command.

Redirection of output to a window via the `on window` syntax works slightly differently for `print` (and `display`) from the way it works for other commands. If you don't send output to the command window (the default), separate windows are created for each variable or expression that you print. Thus, the commands

```
print x on dedicated  
print y on dedicated
```

create two dedicated windows, one for each variable; the two windows are updated separately.

Printing to a window other than the command window creates a visualizer for the data.

To print the contents of a register, precede the name of the register with a dollar sign. For example,

```
print $pc on dedicated
```

prints the contents of the program counter register.

Supported UltraSPARC registers are listed below.

TABLE 8 UltraSPARC Registers

Name	Register
\$g0-\$g7	Global registers (64 bits)
\$o0-\$o7	Output registers (64 bits)
\$l0-\$l7	Local registers
\$i0-\$i7	Input registers
\$psr	Processor state register
\$pc	Program counter
\$npc	Next program counter
\$y	Y register
\$wim	Window invalid mask
\$tbr	Trap base register
\$f0-\$f31	Floating-point registers, printable only as floats
\$fsr	Floating status register (64 bits)
\$f0f1-\$f62f63	Floating-point registers, printable only as doubles
\$xg0-\$xg7	Upper 32 bits of \$g0-\$g7
\$xo0-\$xo7	Upper 32 bits of \$o0-\$o7
\$xfsr	Upper 32 bits of \$fsr
\$fprs	Floating-point registers state
\$tstate	Trap state register
\$fp	Frame pointer (synonym for \$i6)
\$sp	Stack pointer (synonym for \$o6)

You can use the default alias `p` for the `print` command.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

EXAMPLES

```
print maxval(a)
```

prints the maximum value of the array `a`.

```
where (a > 3) print a on dedicated
```

prints (in a dedicated window) the values of `a` that are greater than 3.

printenv

Displays currently set environment variables.

SYNTAX

```
printenv [variable]
```

DESCRIPTION

Use the `printenv` command to display the value of the specified environment variable. If you omit *variable*, the command prints the values of all environment variables that are currently set.

Prism's `printenv` command is identical to its Solaris C-shell counterpart. See your Solaris documentation for more information.

process

In MP Prism, sets or displays the current process of the current pset.

SYNTAX

```
process [process_number]
```

DESCRIPTION

The `process` command is available only in MP Prism.

Use the `process` command to change the current process of the current pset to *process_number*. If you omit the argument, `process` displays the current process of the current pset.

You cannot include this command in event actions.

pset

In MP Prism, sets or displays the current pset.

SYNTAX

```
pset [set_name | set_definition]
```

DESCRIPTION

The `pset` command is available only in MP Prism.

Use the `pset` command to change the current pset. You can either specify the name of a pset or the definition of a pset. See `define pset` for an explanation of how to define a pset.

The `(prism)` prompt changes to reflect the new current set.

With no arguments, `pset` displays the membership of the current process set.

You cannot include the `pset` command in an event action.

EXAMPLES

This example changes the current pset a couple of times and displays its membership:

```
(prism all) pset 0:12:2
(prism 0:12:2) pset
The current set is defined as '0:12:2'.
The set contains processes: 0,2,4,6,8,10,12.
(prism 0:12:2) pset foo
(prism foo) pset
The current set is defined as 'foo'.
The set contains processes: 0:3.
```


pstatus

In MP Prism, displays the execution status of processes.

SYNTAX

```
pstatus [set_name | set_definition] [pset set_name | set_definition]
```

DESCRIPTION

The `pstatus` command is available only in MP Prism.

Use the `pstatus` command to display the execution status of the processes in pset *set_name* or who fit the pset definition *set_definition*. See the `define pset` command for a discussion of how to define a pset. If you issue `pstatus` with no arguments, it displays the execution status of the processes in the current pset. In its output, `pstatus` aggregates processes that have the same status.

EXAMPLE

```
(prism foo) pstatus
process 0: interrupted in procedure "make_move" at "chess.c":1261
process 1: running
processes 2,3: interrupted in procedure "bishop_moves" at
"chess.c":478
processes 4,5: interrupted in procedure "knight_moves" at
"chess.c":383
processes 6,7: interrupted in procedure "generate_moves" at
"chess.c":883
```

pushbutton

Adds a Prism command to the tear-off region.

SYNTAX

```
pushbutton label command
```

DESCRIPTION

Use the `pushbutton` command to create a customized button in the tear-off region. The button will have the label you specify; clicking on it will execute the command you specify. The label must be a single word. The command can be any valid Prism command, along with its arguments.

To remove a button created via the `pushbutton` command, either enter tear-off mode and click on the button, or issue the `untearoff` command, using *label* as its argument.

Changes you make to the tear-off region are saved when you leave Prism.

This command is not available in commands-only Prism.

EXAMPLE

This command creates a button labeled `printfoo` that executes the command `print foo on dedicated`:

```
pushbutton printfoo print foo on dedicated
```

pwd

Displays the pathname of the current working directory.

SYNTAX

```
pwd
```

DESCRIPTION

Use the `pwd` command to display the pathname of the current working directory in Prism.

Prism's `pwd` command is identical to its Solaris counterpart. See your Solaris documentation for more information.

quit

Leaves Prism.

SYNTAX

```
quit
```

DESCRIPTION

Issue the `quit` command to leave Prism. Note that, unlike its menu equivalent, `quit` does not ask you if you are sure you want to quit.

reload

Reloads the currently loaded program.

SYNTAX

```
reload
```

DESCRIPTION

Use the `reload` command to reload the program currently loaded in Prism.

rerun

Reruns the currently loaded program, using arguments previously passed to the program.

SYNTAX

```
rerun [args] [< filename] [> filename]
```

DESCRIPTION

Use the `rerun` command to execute the program currently loaded in Prism. If you do not specify *args*, `rerun` uses the argument list previously passed to the program. Otherwise, `rerun` is identical to the `run` command. You can specify any command-line arguments as *args*, and you can redirect input or output using `<` or `>` in the standard Solaris manner.

return

Steps out to the caller of the current function.

SYNTAX

```
return [count] [pset set_name | set_definition]
```

DESCRIPTION

Use the `return` command to execute the current function, then return to its caller. If you specify an integer as an argument, `return` steps out the specified number of levels in the call stack.

`return` is a synonym for `stepout`.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

run

Executes the currently loaded program.

SYNTAX

```
run [args] [< filename] [> filename]
```

DESCRIPTION

Use the `run` command to execute the program currently loaded in Prism. Specify any command-line arguments as *args*. You can also redirect input or output using `<` or `>` in the standard Solaris manner.

You can use the default alias `r` for this command.

select

Chooses the master pane in a split source window.

SYNTAX

```
select file_extension
```

DESCRIPTION

Use the `select` command to choose the “master pane” when the source window is split into more than one pane. The master pane will contain the code with the file extension you specify as the argument to `select`.

Prism interprets unqualified line numbers in commands in terms of the source code in the master pane. It also uses the master pane to determine the source code and language to use in displaying messages, events, performance data, the call stack, and so on.

Scrolling through the master pane causes the slave pane to scroll to the corresponding location. You can scroll the slave pane independently, but this does not cause the master pane to scroll.

In commands-only Prism, `select` determines the programming language that Prism uses in displaying messages, events, and so on.

EXAMPLES

```
select .s
```

makes the pane containing the loaded program’s assembly code the master pane.

```
select .f
```

selects the pane containing the Fortran 77 source code to be the master pane.

set

Defines abbreviations and sets values for variables.

SYNTAX

```
set variable = expression
```

DESCRIPTION

Use the `set` command to define other names (typically abbreviations) for variables and expressions. The names you choose cannot conflict with names in the program loaded in Prism; they are expanded to the corresponding variable or expression within other commands. For example, if you issue this command:

```
set x = variable_with_a_long_name
```

then

```
print x
```

is equivalent to

```
print variable_with_a_long_name
```

In addition to `print` and `display`, the `whatis`, `whereis`, and `which` commands know about variables you set via the `set` command. For example, issuing the command `whatis x` after issuing the `set` command above produces this response:

```
user-set variable, x = variable_with_a_long_name
```

In addition, you can use the `set` command to set the value of certain internal variables used by Prism. These variables begin with a `$` so that they will not conflict with the names of user-set variables. You can change the settings of these internal variables:

- `$d_precision`, `$f_precision` — Use these variables to specify the default number of significant digits Prism prints for doubles and floating-point variables, respectively. Prism's defaults are 16 for doubles and 7 for floating-point variables; this is the maximum precision for these variables. The value you set applies to printing in both the command window and text visualizers. For example,

```
set $f_precision = 5
```

causes Prism to print five significant digits for floating-point values.

- `$history` — Prism stores the maximum number of lines in the history region in this variable. When the history region reaches the maximum, Prism starts throwing away the earliest lines in the history. The default number of lines in the history region is 10000. To specify an infinite length for the history region, use any negative number. For example,

```
set $history = -1
```

Prism uses up memory in maintaining a large history region. A smaller history region, therefore, may improve performance and prevent Prism from running out of memory.

- `$fortran_string_length` — Prism uses this value as the length of a character string when the length is not explicitly specified. The default is 10.
- `$fortran_adjust_limit` — Prism uses this value as the limit of an adjustable array. The default is 10.
- `$page_size` — This value is used only in commands-only Prism. It specifies the number of output lines Prism displays before stopping and prompting with a `more?` message. Prism obtains its default from the size of your screen. If you specify 0, Prism never displays a `more?` message.
- `$print_width` — This value is used only in commands-only Prism. It specifies the number of items to be printed on a line. The default is 1.
- `$prompt_length` — This value is used only in MP Prism. It specifies the maximum number of characters to appear in the `pset` part of the `(prism)` prompt. The default is 25.
- `$radix` — This value specifies the radix to be used for printing the values of variables. Possible settings are 2 (binary), 8 (octal), 10 (decimal), and 16 (hexadecimal). The default is 10.

Issue the `set` command with no arguments to display your current settings.

Issue the `unset` command to remove a setting (except a predefined one).

setenv

Displays or sets environment variables.

SYNTAX

```
setenv [ VARIABLE [ setting ] ]
```

DESCRIPTION

Use the `setenv` command to set an environment variable within Prism. With no arguments, `setenv` displays all current settings.

Prism's `setenv` command is identical to its Solaris C-shell counterpart. See your Solaris documentation for more information.

sh

Passes a command line to the shell for execution.

SYNTAX

`sh [command_line]`

DESCRIPTION

Use the `sh` command to execute a Solaris command line from a shell; the response is displayed in the history region. If you don't specify a command line, Prism invokes an interactive shell in a separate window. The setting of your `SHELL` environment variable determines which shell is used; if it isn't set, the C shell is used.

You cannot redirect the output of this command.

show

Splits the source window to display the file with the specified extension.

SYNTAX

`show file_extension`

DESCRIPTION

Use the `show` command to split the source window and display the assembly code, or the version of the source code with the specified extension, in the new pane.

The `show` command is not meaningful in commands-only Prism.

Use the `hide` command to cancel the display of the assembly code or source-code version, and return to a single source window.

EXAMPLE

To display the assembly code for the loaded program, issue this command:

```
show .s
```

show events

Displays the event list.

SYNTAX

```
show events
```

DESCRIPTION

Use the `show events` command to print the event list. The list includes an ID for each command; you use this ID when issuing the `delete` command to delete an event.

You can use the default alias `j` for this command.

show pset

In MP Prism, displays the contents of a pset.

SYNTAX

```
show pset [set_name | set_definition]
```

DESCRIPTION

This command is available only in MP Prism.

Use the `show pset` command to display the contents of the pset with the name *set_name* or that is defined by *set_definition*. (See the `define pset` command for a discussion of how to define a pset.) With no arguments, `show pset` displays the contents of the current pset.

EXAMPLE

Issue this command to display the contents of the pset `stopped`:

```
show pset stopped
```

```
The set contains the following processes: 0:3.
```


show psets

In MP Prism, displays information about all psets.

SYNTAX

```
show psets
```

DESCRIPTION

This command is available only in MP Prism.

Use the `show psets` command to display information about all currently defined psets. The output includes each set's definition, members, and current process. The sets listed include user-named sets, predefined sets, and sets that the user has defined but not named.

In graphical MP Prism, or in commands-only MP Prism started with the `-CX` option, issuing the command `show psets` on dedicated displays the Psets window.

EXAMPLE

Here is sample output from a `show psets` command:

```
(prism foo) show psets
foo:
  definition = 0:7
  members = 0:7
  current process = 0
break:
  definition = break
  members = nil
  current process = (none)
done:
  definition = done
  members = nil
  current process = (none)
interrupted:
  definition = interrupted
  members = 0:31
  current process = 0
error:
  definition = error
  members = nil
  current process = (none)
```

```
running:
    definition = running
    members = nil
    current process = (none)
stopped:
    definition = stopped
    members = 0:31
    current process = 0
current:
    definition = foo
    members = 0:7
    current process = 0
cycle:
    definition = foo
    members = 0:7
    current process = 0
all:
    definition = all
    members = 0:31
    current process = 0
```

source

Reads commands from a file.

SYNTAX

`source filename`

DESCRIPTION

Use the `source` command to read in and execute Prism commands from *filename*. This is useful if, for example, you have redirected the output of a `show events` command to a file, thereby saving all events from a previous session.

In the file, Prism interprets lines beginning with `#` as comments. If `\` is the final character on a line, Prism interprets it as a continuation character.

status

Displays the event list.

SYNTAX

`status`

DESCRIPTION

Use the `status` command to display the event list. The list includes an ID for each command; you use this ID when issuing the `delete` command to delete an event.

`status` is an alias for the `show events` command.

You can use the default alias `j` for this command.

step

Executes one or more source lines.

SYNTAX

```
step [n] [pset set_name | set_definition]
```

DESCRIPTION

Use the `step` command to execute the next *n* source lines, stepping into procedures and functions. If you do not specify a number, `step` executes the next source line.

You can use the default alias `s` for this command.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

stepi

Executes one or more machine instructions.

SYNTAX

```
stepi [n] [pset set_name | set_definition]
```

DESCRIPTION

Use the `stepi` command to execute the next *n* machine instructions, stepping into procedures and functions. If you do not specify a number, `stepi` executes the next machine instruction.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

stepout

Steps out to the caller of the current function.

SYNTAX

`stepout` [*count*]

DESCRIPTION

Use the `stepout` command to execute the current function, then return to its caller. If you specify an integer as an argument, `stepout` steps out the specified number of levels in the call stack.

`return` is a synonym for `stepout`.

stop

Sets a breakpoint.

SYNTAX

```
stop [var | at line | in func] [if expression] [{cmd; cmd ...}] [after n]
      [pset set_name | set_definition]
```

DESCRIPTION

Use the `stop` command to set a breakpoint at which the program is to stop execution. You can abbreviate this command to `st`.

The first option listed in the synopsis (`var` | `at line` | `in func`) must come first on the command line; you can specify the other options, if you include them, in any order.

`var` is the name of a variable. Execution stops whenever the value of the variable changes. If the variable is an array or a parallel variable, execution stops when the value of any element changes. This form of the command slows execution considerably. You cannot specify both a variable and a location.

`at line` stops execution when the specified line is reached. If the line is not in the current file, use the form "`filename`" : `line_number`, using quotation marks around the file name.

`in func` stops execution when the specified procedure or function is reached.

`if expression` specifies the logical condition, if any, under which execution is to stop. The logical condition can be any expression that evaluates to true or false. Unless combined with the `at line` syntax, this form of `stop` slows execution considerably.

`{cmd; cmd ...}` specifies the actions, if any, that are to accompany the breakpoint. The actions can be Prism commands; if you include multiple commands, separate them with semicolons.

`after n` specifies how many times a trigger condition (for example, reaching a program location) is to occur before the breakpoint occurs. The default is 1. If you specify both a condition and an after count, Prism checks the condition first.

When issued in MP Prism, this command can take a `pset` qualifier. If used with a qualifier, it applies to the `pset` you specify. If used without a qualifier, it applies to the current `pset`. See page 2 for more information on `pset` qualifiers.

EXAMPLES

`stop in foo {print a; where} after 10`

stops execution the tenth time in the function `foo`, prints `a`, and executes the `where` command.

`stop at "bar":17 if a == 0`

stops execution at line 17 of file `bar` if `a` is equal to 0.

`stop a`

stops execution whenever the value of `a` changes.

`stop if a .eq. 5 after 3`

stops execution the third time `a` equals 5.

stopi

Sets a breakpoint at a machine instruction.

SYNTAX

```
stopi [var | at addr | in func] [if expression] [{cmd; cmd ...}] [after n]  
[pset set_name | set_definition]
```

DESCRIPTION

Use the `stopi` command to set a breakpoint at a machine instruction.

The first option listed in the synopsis (*var* | at *addr* | in *func*) must come first on the command line; you can specify the other options, if you include them, in any order.

var is the name of a variable. Execution stops whenever the value of the variable changes. If the variable is an array or a parallel variable, execution stops when the value of any element changes. This form of the command slows execution considerably. You cannot specify both a variable and a location.

at *addr* stops execution when the specified address is reached.

in *func* stops execution when the specified procedure or function is reached.

if *expression* specifies the logical condition, if any, under which execution is to stop. The logical condition can be any expression that evaluates to true or false. Unless combined with the at *addr* syntax, this form of `stopi` slows execution considerably.

{*cmd*; *cmd* ...} specifies the actions, if any, that are to accompany the breakpoint. The actions can be Prism commands; if you include multiple commands, separate them with semicolons.

after *n* specifies how many times a trigger condition (for example, reaching a program location) is to occur before the breakpoint occurs. The default is 1. If you specify both a condition and an after count, Prism checks the condition first.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

EXAMPLES

```
stopi at 0x1000
```

stops execution at address 1000 (hex).

`stopi at 0x500 if a == 0`

stops execution at address 500 (hex) if `a` is equal to 0.

tearoff

Places a menu selection in the tear-off region.

SYNTAX

```
tearoff "selection"
```

DESCRIPTION

Use the `tearoff` command to add a menu selection to the tear-off region of the main Prism window. Put the selection name in quotation marks. Case and blank spaces don't matter, and you can omit the three dots that indicate that choosing the selection displays a dialog box. If the selection name is available in more than one menu, put the name of the menu you want in parentheses after the selection name.

Use the `untearoff` command to remove a menu selection from the tear-off region.

Changes you make to the tear-off region are saved when you leave Prism.

This command is not available in commands-only Prism.

EXAMPLES

```
tearoff "file"
```

puts the `File` selection in the tear-off region.

```
tearoff "print (events)"
```

puts the `Print` selection from the `Events` menu in the tear-off region.

tmsubargs

Specifies `tmsub` or `tmsub` options to use when executing multiprocess programs.

SYNTAX

```
tmsubargs [option | off]
```

DESCRIPTION

Use the `tmsubargs` command to specify `tmsub` or `tmsub` options that are to be used in subsequently executing multiprocess programs within Prism. Options you specify via `tmsubargs` supersede the corresponding options set via the Prism command line or the `TMSUB_FLAGS` environment variable.

Use the `off` option to remove existing `tmsub` or `tmsub` options (except for those set via `TMSUB_FLAGS`).

Issue `tmsubargs` with no options to display the current `tmsub` or `tmsub` options.

trace

Traces program execution.

SYNTAX

```
trace [var | at line | in func] [if expression] [{cmd; cmd ...}] [after n] \  
      [pset set_name | set_definition]
```

DESCRIPTION

Use the `trace` command to print tracing information when the program is executed. In a trace, Prism prints a message in the command window when a program location is reached, a value changes, or a condition becomes true; it then continues execution.

The first option listed in the synopsis (*var* | at *line* | in *func*) must come first on the command line; you can specify the other options, if you include them, in any order.

var is the name of a variable. The value of the variable is displayed whenever it changes. If the variable is an array or a parallel variable, values are displayed if the value of any element changes. This form of the command slows execution considerably. You cannot specify both a variable and a location.

at *line* specifies that the specified line is to be printed immediately prior to its execution. If the line is not in the current file, use the form "*filename*" : *line_number*, placing the file name between quotation marks. You can also specify a line number without the at; Prism will interpret it as a line number rather than a variable.

in *func* causes tracing information to be printed only while executing inside the specified procedure or function.

if *expression* specifies the logical condition, if any, under which tracing is to occur. The logical condition can be any expression that evaluates to true or false. Unless combined with the at *line* syntax, this form of `trace` slows execution considerably.

{*cmd*; *cmd* ...} specifies the actions, if any, that are to accompany the trace. Put the actions in braces. The actions can be Prism commands; if you include multiple commands, separate them with semicolons.

after *n* specifies how many times a trigger condition (for example, reaching a program location) is to occur before the trace occurs. The default is 1. If you specify both a condition and an after count, Prism checks the condition first.

When tracing source lines, Prism steps into procedure calls if they have source associated with them. It "nexts" over them if they do not have source.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

EXAMPLES

```
trace {print a; where}
```

does a trace, prints the value of `a`, and executes the `where` command at every source line.

```
trace at 17 if a .gt. 10
```

traces line 17 if `a` is greater than 10.

```
trace "bar":20
```

traces line 20 of file `bar`.

tracei

Traces machine instructions.

SYNTAX

```
tracei [var | at addr | in func] [if expression] [{cmd; cmd ...}] [after n]  
[pset set_name | set_definition]
```

DESCRIPTION

Use the `tracei` command to trace machine instructions when the program is executed.

The first option listed in the synopsis (*var* | at *addr* | in *func*) must come first on the command line; you can specify the other options, if you include them, in any order.

var is the name of a variable. The value of the variable is displayed whenever it changes. If the variable is an array or a parallel variable, values are displayed if the value of any element changes. This form of the command slows execution considerably. You cannot specify both a variable and a location.

at *addr* specifies that a message is to be printed immediately prior to the execution of the specified address.

in *func* causes tracing information to be printed only while executing inside the specified procedure or function.

if *expression* specifies the logical condition, if any, under which tracing is to occur. The logical condition can be any expression that evaluates to true or false. Unless combined with the at *addr* syntax, this form of `tracei` slows execution considerably.

{*cmd*; *cmd* ...} specifies the actions, if any, that are to accompany the trace. Put the actions in braces. The actions can be Prism commands; if you include multiple commands, separate them with semicolons.

after *n* specifies how many times a trigger condition (for example, reaching a program location) is to occur before the trace occurs. The default is 1. If you specify both a condition and an after count, Prism checks the condition first.

When tracing instructions, Prism follows all procedure calls down.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

EXAMPLES

`tracei 0x1000 after 3`

traces the instruction at address 1000 (hex) the third time it is reached.

`tracei 0x500 if a == 0`

traces the instruction at address 500 (hex) if `a` is equal to 0.

unalias

Removes an alias.

SYNTAX

`unalias name`

DESCRIPTION

Use the `unalias` command to remove the alias with the specified name. Issue the `alias` command with no arguments to obtain a list of your current aliases.

unset

Deletes a user-set name.

SYNTAX

`unset name`

DESCRIPTION

Use the `unset` command to delete the setting associated with *name*. See the `set` command for a discussion of setting names for variables and expressions.

Do not use the `unset` command to unset any of the Prism internal variables (beginning with \$).

EXAMPLE

If you use the `set` command to set this abbreviation for a variable name:

```
set fred = frederick_bartholomew
```

then you can unset it as follows:

```
unset fred
```

Note: After issuing the `unset` command, you can no longer use `fred` as an abbreviation for `frederick_bartholomew`.

unsetenv

Unsets an environment variable.

SYNTAX

`unsetenv variable`

DESCRIPTION

Use the `unsetenv` command to remove the specified variable from the environment.

Prism's `unsetenv` command is identical to its Solaris C-shell counterpart. See your Solaris documentation for more information.

untearoff

Removes a button from the tear-off region.

SYNTAX

```
untearoff "label"
```

DESCRIPTION

Use the `untearoff` command to remove a button from the tear-off region of the main Prism window. Put the button's label in quotation marks. Case and blank spaces don't matter, and you can omit the three dots that indicate that choosing the button displays a dialog box. If the tear-off region includes more than one button with the same label, include the name of the selection's menu in parentheses after the label.

Changes you make to the tear-off region are saved when you leave Prism.

This command is not available in commands-only Prism.

EXAMPLES

```
untearoff "load"
```

removes the Load button from the tear-off region.

```
untearoff "print (events)"
```

removes the button that executes the Print selection in the Events menu.

up

Moves the symbol lookup context up one level in the call stack.

SYNTAX

up [*count*]

DESCRIPTION

Use the `up` command to move the current function up the call stack (that is, away from the current stopping point in the program toward the main procedure) *count* levels. If you omit *count*, the default is one level.

Issuing `up` repositions the source window at the new current function.

use

Adds a directory to the list of directories to be searched when looking for source files.

SYNTAX

`use [directory]`

DESCRIPTION

Issue the `use` command to add *directory* to the front of the list of directories Prism is to search when looking for source files. This is useful if you have moved a source file since compiling the program, or if for some other reason Prism can't find the file. If you do not specify a directory, `use` prints the current list.

No matter what the contents of the directory list is, Prism always searches first in the directory in which the program was compiled.

varsave

Saves the values of a variable or expression to a file.

SYNTAX

```
varsave "filename" expression
```

DESCRIPTION

Use the `varsave` command to save the values of the variable or expression specified by *expression* to the file *filename*. You can subsequently restore the values in *filename* via the `varfile` intrinsic (except in MP Prism) and compare them with another version of the variable or expression via the Diff or Diff With selection from a visualizer's Options menu.

EXAMPLES

```
varsave "alpha.data" alpha
```

saves the values of the variable `alpha` in the file `alpha.data` (in your current working directory within Prism).

```
varsave "/u/kathy/alpha2.data" alpha*2
```

saves the results of the expression `alpha*2` in the file with the pathname `/u/kathy/alpha2.data`.

wait

In MP Prism, waits for a process or processes to stop execution.

SYNTAX

```
wait [every | any] [pset set_name | set_definition]
```

DESCRIPTION

The `wait` command is available only in MP Prism.

Use the `wait` command to tell Prism to wait for the specified process or processes to stop execution before accepting commands that affect other processes (for example, commands that start or stop execution). A process is considered to have stopped if it has entered the `done`, `break`, `interrupted`, or `error` state.

This command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

Use the form `wait` or `wait every` to wait for every process in the pset to stop execution. `wait every` is the default.

Use the form `wait any` to wait for any process in the pset to stop execution.

You can end the wait by doing one of the following

- Typing Ctrl-c; this does not affect processes that are running.
- Choosing (in graphical Prism) the Interrupt selection from the Execute menu; this stops processes that are running, as well as ending the wait.

whatis

Displays the declaration of a name.

SYNTAX

```
whatis name [pset set_name | set_definition]
```

DESCRIPTION

Use the `whatis` command to display information known by Prism about a specified name in the program.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

when

Sets a breakpoint. The `when` command is similar to the `stop` command.

SYNTAX

```
when [var | at line | in func | stopped] [if expr] [{cmd; cmd ...}] [after n]
```

DESCRIPTION

Use the `when` command to set a breakpoint at which the program is to stop execution.

The first option listed in the synopsis (*var* | at *line* | in *func* | stopped) must come first on the command line; you can specify the other options, if you include them, in any order.

var is the name of a variable. Execution stops whenever the value of the variable changes. If the variable is an array or a parallel variable, execution stops when the value of any element changes. This form of the command slows execution considerably. You cannot specify both a variable and a location, as described below.

at *line* stops execution when the specified line is reached. If the line is not in the current file, use the form "*filename*" : *line_number*, placing the file name between quotes.

in *func* stops execution when the specified procedure or function is reached.

stopped specifies that the actions associated with the command occur every time the program stops execution.

if *expr* specifies the condition, if any, under which execution is to stop. The condition can be any expression that evaluates to true or false. Unless combined with the at *line* syntax, this form of `when` slows execution considerably.

{*cmd*; *cmd* ...} specifies the actions, if any, that are to accompany the breakpoint. The actions can be any valid commands; if you include multiple commands, separate them with semicolons.

after *n* specifies how many times a location is to be reached before the breakpoint occurs. The default is 1. If you specify both a condition and an after count, Prism checks the condition first.

EXAMPLE

```
when stopped {print a on dedicated}
```

prints the value of `a` in a dedicated window whenever execution stops.

where

Displays the call stack.

SYNTAX

```
where [count] [pset set_name | set_definition]
```

DESCRIPTION

Use the `where` command to print out a list of the active procedures and functions on the call stack. With no argument, `where` displays the entire list. If you specify *count*, `where` displays the specified number of functions.

You can use the default alias `τ` for this command.

In graphical MP Prism, the command `where on dedicated` displays a `Where` graph, a dynamic call graph of the program.

When issued in MP Prism, this command can take a pset qualifier. If used with a qualifier, it applies to the pset you specify. If used without a qualifier, it applies to the current pset. See page 2 for more information on pset qualifiers.

whereis

Displays the full qualification of all the symbols matching a given identifier.

SYNTAX

```
whereis identifier
```

DESCRIPTION

Use the `whereis` command to display a list of the fully qualified names of all symbols whose name matches *identifier*. The symbol class (for example, `procedure`, `variable`) is also listed.

EXAMPLE

Issuing this command:

```
whereis x
```

might produce this response:

```
variable foo`x  
procedure fum`x
```

which

Displays the fully qualified name of an identifier.

SYNTAX

`which identifier`

DESCRIPTION

Use the `which` command to display the fully qualified name of *identifier*. This indicates which (of possibly several) variables or procedures by the name *identifier* Prism would use at this point in the program (for example, in an expression). The fully qualified name includes the filename or function name with which the identifier is associated. For more information on fully qualified names, see the *Prism User's Guide*.

Prism man page

prism

Enter the Prism programming environment

SYNTAX

```
prism [program_name] [-C | -CX] [-np nprocs]  
      [-p partition] [-s system] [-tmrun "option [option..."]]]  
      [-pvm] [Xoption ...] [core_file | pid | tid] [< infile]  
      [> outfile]
```

DESCRIPTION

`prism` enters Prism, an X-based graphical programming environment within which you can develop, execute, debug, and visualize data in serial and parallel programs.

You must execute `prism` from a terminal or workstation running the X Window System (unless you specify the `-C` option).

If issued without the `-np` option `prism` starts a version of Prism for use on serial programs.

If issued with the `-np` option, `prism` starts MP Prism, a version of Prism that can be used for debugging message-passing or other multiprocess programs. These programs must be written in the SPMD style (that is, each process must run the same executable program).

If you specify `-np`, you can also include other options to specify where the processes are to run. You can in addition specify `tmrun` or `tmsub` options via the `TMRUN_FLAGS` environment variable. These control where the processes are run

unless overridden by a Prism option. If this environment variable is not set, and you don't specify `-np`, `-p`, `-s`, or `-tmrun`, one copy of the program runs on your login node.

Once you have entered Prism, you can issue the `tmrunargs` command to specify any `tmrun` or `tmsub` options that you want to apply to your message-passing program. Prism stores these options, then applies them when you start up a multiprocess program. Specifying the setting of an option via the `tmrunargs` command overrides the setting of the same option you have established via `TMRUN_FLAGS` or the `prism` command line. If it is an option that has otherwise not been specified, it is added to the existing settings.

You can issue `tmrunargs` in DP Prism. If you use the `-np` option to specify multiple processes, MP Prism starts up.

To remove any existing `tmrun` or `tmsub` options you have specified, issue the command `tmrunargs off`. This removes options you have set via the `prism` command line and the `tmrunargs` command, but not via `TMRUN_FLAGS`. Issuing the `tmrunargs` command with no options shows the current `tmrun` and `tmsub` options.

If issued without the name of an executable program, `prism` displays the main window of the Prism environment, with no program loaded. If you issue it with the name of an executable program, `prism` loads that program into Prism upon startup.

If you specify *core_file*, `prism` associates that core file with the program you load. Within Prism, you can then examine the stack and display the values of variables at the point at which core was dumped.

If you specify *pid*, `prism` loads the running process with that process ID into Prism. The process is interrupted, and you can then work with the program in Prism as you normally would. When attaching to a running serial process in this manner, Prism must be started on the same node on which the process is running. When attaching to a running PVM process, Prism must be started on a node on which a PVM daemon is running. In MP Prism, you can specify *tid*, the task ID for a multiprocess program running in a Sun HPC Cluster.

If you specify *infile*, `prism` reads and executes commands from the specified file upon startup. If you specify *outfile*, `prism` logs all its input and output to this file. This includes commands from *infile* and commands typed on the command line within Prism.

If there is a `.prisminit` file in your current working directory, `prism` executes the commands in it upon startup. If `.prisminit` isn't in your current working directory, `prism` looks for it in your home directory. If it isn't in either place, `prism` starts up without executing a `.prisminit` file.

Note – To use Prism’s debugging features, compile and link each program module with the `-g` compiler option to produce the necessary debugging information. If you use the C or Fortran SPARCompiler, you must also use the `-xs` option when compiling and linking; this causes the debugging information to be placed in the executable program rather than in the object files. To collect performance data on Sun HPF programs, you must compile and link your program with the `-tmprofile` option.

OPTIONS

`-C` – *Commands-only execution.* prism displays a prompt from which you can issue any Prism commands. If you use this option, you don't need an X terminal or workstation.

`-CX` – *Commands-only execution with output.* Start a version of prism that uses commands-only execution (like `-C`), but in which the output of certain Prism commands can be sent to X windows.

`-np nprocs` – *Start nprocs processes of the executable program.* Without this argument, prism starts a single process (if the program loaded at startup is an HPF program). Specify 0 to indicate that you want to start one process on each available node.

`-p partition` – *Start the executable program in the specified Sun HPC partition.* The default is to start the program in the partition to which you are logged in. Using the `-p` option implies MP Prism.

`-pvm` – *Start MP Prism on a PVM message-passing program.* If you specify `-pvm`, you must also specify the pid of a running PVM process to attach to, and Prism must start on a node on which a PVM daemon is running.

`-s server` – *Start the executable program on the specified Sun HPC System.* The default is to start it on the System to which you are logged in. Using the `-s` option implies MP Prism.

`-tmrun "option [option]..."` – *Start the executable program using the specified tmrun or tmsub options.* Using the `-tmrun` option implies `-np` and starts MP Prism.

prism also accepts all standard X toolkit options. However, the `-font`, `-title`, and `-rv` options have no effect, and the `-bg` option is overridden in part by the setting of the `Prism.textBgColor` resource. X toolkit options are meaningless, of course, if you use `-C` to run Prism in commands-only mode.

FILES

`.prisminit` – Prism initialization file.

`.prism_defaults` – Prism defaults file.

IDENTIFICATION

Prism Version 5.0.

SEE ALSO

`tmr`(1), `tmsub`(1),
Prism User's Guide, *Prism Reference Manual*

Index

SYMBOLS

/ command, 1-7
? command, 1-7
@, 1-1

A

address/ command, 1-8
alias command, 1-12
assign command, 1-13
attach command, 1-14

C

call command, 1-15
catch command, 1-16
cd command, 1-17
collection command, 1-18
cont command, 1-19
contw command, 1-20
core command, 1-21
cycle command, 1-22

D

dedicated window, 1-1
define pset command, 1-23
delete command, 1-25
delete pset command, 1-26
detach command, 1-27
disable command, 1-28

display command, 1-29
down command, 1-32

E

edit command, 1-34
enable command, 1-35
eval pset command, 1-36

F

fg command, 1-37
file command, 1-38
func command, 1-39

H

help command, 1-40
hide command, 1-41

I

ignore command, 1-42
interrupt command, 1-43

K

kill, 1-44

L

list command, 1-45
load command, 1-46
log command, 1-47

M

make command, 1-48

N

next command, 1-49
nexti command, 1-50

O

output
 redirecting, 1-1

P

perf command, 1-51
perflod command, 1-52
perfsave command, 1-53
perftip command, 1-54
print command, 1-55
printenv command, 1-58
process command, 1-59
pset command, 1-60
pset qualifiers, 1-2
pstatus command, 1-61
pushbutton command, 1-62
pwd command, 1-63

Q

quit command, 1-64

R

reload command, 1-65
rerun command, 1-66
return command, 1-67
run command, 1-68

S

select command, 1-69
set command, 1-70
setenv command, 1-72
sh command, 1-73
show command, 1-74
show events command, 1-75
show pset command, 1-76
show psets command, 1-77
snapshot window, 1-1
source command, 1-79
status command, 1-80
step command, 1-81
stepi command, 1-82
stepout command, 1-83
stop command, 1-84
stopi command, 1-86

T

tearoff command, 1-88
tmrunargs command, 1-89
trace command, 1-90
tracei command, 1-92

U

unset command, 1-95
untearoff command, 1-62, 1-96, 1-97
up command, 1-98
use command, 1-99

V

value=base command, 1-11
varfile command, 1-100

W

wait command, 1-101
whatis command, 1-102
when command, 1-103
where command, 1-105
whereis command, 1-106
which command, 1-107