The
Connection Machine
System

Prism Reference Manual

Version 1.2
March 1993

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, March 1993

****

The information in this document is subject to change without notice and should not be construed as a commitment by Thinking Machines Corporation. Thinking Machines reserves the right to make changes to any product described herein.

Although the information in this document has been reviewed and is believed to be reliable, Thinking Machines Corporation assumes no liability for errors in this document. Thinking Machines does not assume any liability arising from the application or use of any information or product described herein.

****

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142-1264
(617) 234-1000

Contents

About This Manual

Objectives of This Manual

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

Intended Audience

The manual is intended for application programmers using Prism to develop programs in C, C*, Fortran, and CM Fortran on a CM-2, CM-200, or CM-5 Connection Machine system. We assume you know the basics of developing and debugging programs, as well as the basics of using a CM. Some familiarity with the UNIX debugger **dbx** is useful, but not required. Prism is based on the X and OSF/Motif standards. Familiarity with these standards is also helpful but not required.

Revision Information

This manual has been updated to include changes made in Prism Version 1.2.

Organization of This Manual

The manual presents the reference descriptions of the commands in alphabetical order.

Related Documents

Read the *Prism User's Guide* for complete information on how to use Prism.

Refer to the release notes for last-minute information on Prism. The release notes are available on-line by choosing the **Release Notes** selection from the **Help** menu,

or by issuing the **help release** command.

Notation Conventions

The table below displays the notation conventions observed in this manual.

Convention	Meaning
bold typewriter	Prism, UNIX, and CMOST commands, command options, and filenames, when they appear embedded in text. Also programming language elements, such as keywords, operators, and function names, when they appear embedded in text.
italics	Placeholders in command synopses.
% bold typewriter regular typewriter	In interactive examples, user input is shown in **bold typewriter** and system output is shown in regular typewriter font.

Customer Support

Thinking Machines Customer Support encourages customers to report errors in Connection Machine operation and to suggest improvements in our products.

When reporting an error, please provide as much information as possible to help us identify and correct the problem. A code example that failed to execute, a session transcript, the record of a backtrace, or other such information can greatly reduce the time it takes Thinking Machines to respond to the report.

If your site has an applications engineer or a local site coordinator, please contact that person directly for support. Otherwise, please contact Thinking Machines' home office customer support staff:

Internet Electronic Mail:	`customer-support@think.com`
uucp Electronic Mail:	`ames!think!customer-support`
U.S. Mail:	Thinking Machines Corporation Customer Support 245 First Street Cambridge, Massachusetts 02142-1264
Telephone:	(617) 234-4000

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.
- Type `help` *command-name* on the command line to display the reference description of the command.

In the syntax descriptions, text in `bold` is text that you type on the command line. Words in *italics* are placeholders for which you substitute the appropriate word or phrase.

Table 1 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 **on** *window*, where *window* can be:

- **command** (abbreviation **com**). This sends output to the command window; this is the default.
- **dedicated** (abbreviation **ded**). This 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** (abbreviation **sna**). This 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; the name appears in the title of the window.

The commands whose output you cannot redirect are: **cmcoldboot**, **cmfinger**, **edit**, **email**, **make**, and **sh**.

Version 1.2, March 1993

Table 1. Prism commands.

Command	Use
/string	Searches forward in the current file for *string*.
?*string*	Searches backward in the current file for *string*.
address/	Prints the contents of a location in memory.
value=*base*	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.
call	Calls a procedure or function.
catch	Tells Prism to catch the specified signal.
cd	Changes the current working directory.
cmattach*	Attaches to a CM resource.
cmcoldboot*	Cold boots a CM resource.
cmdetach*	Detaches from a CM resource.
cmfinger*	Displays information about CM users.
cmsetsafety*	Sets safety on or off for a CM resource.
collection	Turns collection of performance data on or off.
cont	Continues execution.
core	Associates a core file with an executable program.
delete	Removes an event.
detach	Detaches from a running process.
display	Displays the value of an expression.
doc	Displays on-line documentation in commands-only Prism.
down	Moves the symbol-lookup context down one level.
dump	Prints the names and values of variables.
edit	Calls up an editor.
email	Sends mail about Prism.
file	Sets the source file to the specified filename.
func	Sets the current function to the specified function name.
help	Lists currently implemented commands.
hide**	Removes a pane from a split source window.
ignore	Tells Prism to ignore the specified signal.
list	Lists source lines.
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.

*Available from CM-2 and CM-200 front ends only.
**Not available in commands-only Prism.

Table 1. Prism commands (cont'd).

Command	Use
`nexti`	Executes one or more instructions, stepping over functions.
`perf`	Displays performance data.
`perfadvice`	Displays an analysis of performance data.
`perfload`	Loads a performance data file.
`perfsave`	Saves performance data to a file.
`print`	Prints the value of an expression.
`printenv`	Displays currently set environment variables.
`pushbutton**`	Adds a Prism command to the tear-off region.
`pwd`	Prints the current working directory.
`quit`	Leaves Prism.
`reload`	Reloads the currently loaded 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.
`show**`	Splits the source window.
`show events`	Prints the event list.
`source`	Reads commands from a file.
`status`	Prints the event list.
`step`	Executes one or more source lines.
`stepi`	Executes one or more instructions.
`stepout`	Steps out to the caller of the current routine.
`stop`	Sets a breakpoint.
`stopi`	Sets a breakpoint at an instruction.
`tearoff**`	Adds a menu selection to the tear-off region.
`trace`	Traces program execution.
`tracei`	Traces instructions.
`type`	Provides type information on Paris parallel variables.
`unalias`	Removes an alias.
`unset`	Removes an abbreviation created by `set`.
`unsetenv`	Removes the setting of an environment variable.
`untearoff**`	Removes a button from the tear-off region.
`up`	Moves the symbol-lookup context up one level.
`use`	Adds a directory to the list to be searched for source files.
`whatis`	Prints the type of a variable.
`when`	Sets a breakpoint.

**Not available in commands-only Prism.

Table 1. Prism commands (cont'd).

Command	Use
`where`	Prints a stack trace.
`whereis`	Prints the list of all fully qualified names for an identifier.
`which`	Prints the fully qualified name Prism chooses for an identifier.

/
?

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

Synopsis

/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.

Version 1.2, March 1993

address/

Prints the contents of memory addresses.

Synopsis

address,*address*/[*mode*]
address | *register*/[*count*][*mode*]

Description

Use these commands to print the contents of memory or a register. If two addresses are separated by commas, Prism prints the contents of memory starting at the first address and continuing up 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 following the one printed most recently.

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 the `display` or `print` command for a list of supported registers.

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:

`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

value=base

Converts a value to the specified base.

Synopsis

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.

Synopsis

```
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.

Version 1.2, March 1993

assign

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

Synopsis

assign *lval* = *expression*

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 — for example, a variable, a Fortran array section, or a C* left-indexed parallel variable. If *lval* is a CM Fortran array or array section, the right-hand side must be a conformable expression. If *lval* is a C* parallel variable, the right-hand side must be of the same shape. Prism performs the proper type coercions if the right-hand side does not have the same type as the left-hand side.

Examples

This example assigns the value 1 to **x**:

```
assign x = 1
```

If **x** is an array or a parallel variable, 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.

This example assigns the value of element 4 of the C* parallel variable **pvar2** to element 7 of **pvar1**:

```
assign [6]pvar1 = [3]pvar2
```

attach

Attaches to a running process.

Synopsis

`attach` *pid*

Description

Use the `attach` command to attach to the running process with process ID *pid*. To work on the process within Prism, you must have previously loaded the process's executable program.

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

call

Calls a procedure or function.

Synopsis

`call` *procedure* (*parameters*)

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.

NOTE: If you call a C* function, you must pass parallel variables by reference, not by value.

catch

Tells Prism to catch the specified UNIX signal.

Synopsis

`catch` [*number* | *signal-name*]

Description

Prism can catch UNIX 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.

Version 1.2, March 1993

cd

Changes the current working directory.

Synopsis

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 UNIX counterpart. See your UNIX documentation for more information.

cmattach

Attaches to a CM-2 or CM-200 resource.

Synopsis

`cmattach` [*option* ...]

Description

Use the `cmattach` command to attach to a Connection Machine resource and cold boot. You are by default attached to the highest-numbered sequencer available on the first CM available, whether it is timeshared or exclusive.

Prism prints a message telling you the CM resource to which you are attached, or that no resource is available.

The Prism version of `cmattach` accepts the following subset of the options available in the shell-level version:

`-e`	Attach in exclusive mode only.
`-t`	Attach in timeshared mode only.
`-c` *name*	Attach to the specified CM.
`-i` *number*	Attach via the specified interface.
`-s` *seq-set*	Attach to the specified sequencer or sequencer set.

NOTE: This command is available only if you are running Prism from a front end that is connected to a CM-2 or CM-200 series Connection Machine system.

cmcoldboot

Cold boots a CM-2 or CM-200 resource.

Synopsis

`cmcoldboot` [*option* ...]

Description

Use the `cmcoldboot` command to cold boot the Connection Machine resource to which you are attached. Prism prints a message telling you that you have cold booted.

The Prism version of `cmcoldboot` accepts all options accepted by the shell-level version of `cmcoldboot`. Namely:

`-h` Print a help message.

`-g` *axis-length, axis-length ...*
Configure the CM to have the specified geometry.

`-b` *fraction* Enter back-compatibility mode with the specified fraction of memory allocated for back-compatibility mode.

`-u` *ucode* Load the specified version of the microcode.

`-v` *nprocs* Enter back-compatibility mode and configure the CM to have the specified number of virtual processors.

`-x` *xdim* `-y` *ydim*
Enter back-compatibility mode and configure the CM to have the specified numbers of processors arranged in a NEWS grid.

You cannot redirect the output of this command.

NOTE: This command is available only if you are running Prism from a front end that is connected to a CM-2 or CM-200 series Connection Machine system.

cmdetach

Detaches from a CM-2 or CM-200 resource.

Synopsis

```
cmdetach
```

Description

Use the `cmdetach` command to detach from the Connection Machine resource to which you are attached. You can detach from the CM whether or not you attached from within Prism. Prism prints a message telling you that you have been detached.

NOTE: This command is available only if you are running Prism from a front end that is connected to a CM-2 or CM-200 series Connection Machine system.

cmfinger

Displays information about CM-2 or CM-200 users.

Synopsis

```
cmfinger
```

Description

Use the `cmfinger` command to display information about the CMs to which your front end is connected and who is using them.

You cannot redirect the output of this command.

NOTE: This command is available only if you are running Prism from a front end that is connected to a CM-2 or CM-200 series Connection Machine system.

cmsetsafety

Turns safety on or off for a CM-2 or CM-200 resource.

Synopsis

```
cmsetsafety on|off
```

Description

Use the `cmsetsafety on` command to turn safety on for the Connection Machine resource to which you are attached; `cmsetsafety off` turns safety off. Issuing `cmsetsafety` also affects the setting of the toggle box in the **CMsetsafety** selection of the **CM** menu.

Turn off safety when you are collecting performance data; otherwise, your results will be inaccurate. If you attempt to run a program with collection and safety both on, Prism turns safety off and prints a message informing you of this.

NOTE: This command is available only if you are running Prism from a front end that is connected to a CM-2 or CM-200 series Connection Machine system.

collection

Turns collection of performance statistics on or off.

Synopsis

```
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.

cont

Continues execution.

Synopsis

`cont` [*number* | *signal-name*]

Description

Use the `cont` command to continue execution of the process from the point where it stopped. If you specify a UNIX 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.

Version 1.2, March 1993

core

Associates a core file with the loaded program.

Synopsis

`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; and, if you were running the program on a CM (for a CM-2 or CM-200) or on the nodes (for a CM-5), you cannot display the values of parallel variables or CM-resident arrays.

delete

Removes one or more events from the event list.

Synopsis

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.

detach

Detaches a process running within Prism.

Synopsis

```
detach
```

Description

Use the `detach` command to detach the process that is currently running within Prism. The process 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.

display

Displays the values of one or more variables or expressions.

Synopsis

[`where` (*expression*)] `display` *expression* [, *expression* ...]

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.

In CM Fortran, the expression can operate on a conformable array. In C*, the expression can operate on a parallel variable that is of the same shape as the parallel variable being displayed.

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; each window is 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.

The SPARC registers are:

`$pc`	program counter
`$npc`	next program counter
`$fsr`	floating status register
`$fq`	floating queue
`$wim`	window invalid mask
`$tbr`	trap base register
`$g0-$g7`	global registers
`$i0-$i7`	input registers
`$l0-$l7`	local registers
`$o0-$o7`	output registers
`$sp`	stack pointer (synonym for `$o6`)
`$fp`	frame pointer (synonym for `$i6`)
`$f0-$f31`	floating-point registers
`$y`	Y register

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 greater than 3.

doc

Displays on-line documentation in commands-only Prism.

Synopsis

```
doc
```

Description

Use the `doc` command to display on-line documentation in commands-only Prism. Issuing `doc` initially displays a menu of available documents. Choose the number associated with the document you want to see and press **Return**. If the document is made up of other documents (for example, chapters in a manual), another menu is displayed; once again, you can choose the number of the document you want to see. Otherwise, you see the first screenful of text in the document.

Use the syntax

number `@` *filename*

to redirect the text of the document associated with *number* to the file you specify. Use

number `@@` *filename*

to add the text to the end of an existing file.

When the first screenful of a document or a long menu is displayed, you will see a **more?** prompt at the bottom of your screen. Type **y** or simply press **Return** to see the next screen of text. Type **n** or **q** to return to the menu from which you came.

Some documents list on-line manual pages that you may want to see. Enter **m** from a menu, followed by the name of any CMOST or UNIX manual page, to display the manual page. If you don't enter the name of a manual page, you will be prompted for it.

Enter **p** from a menu to return to the previous menu.

Enter **q** from a menu to leave the **doc** command and return to the `(prism)` prompt.

NOTE: The **doc** command is not available in graphical Prism. Use the **Online Doc** selection from the **Doc** menu instead.

down

Moves down the call stack.

Synopsis

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

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 *procedure*, 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.

Synopsis

`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.

You cannot redirect the output of this command.

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

Version 1.2, March 1993

email

Sends electronic mail about Prism.

Synopsis

```
email
```

Description

Use the `email` command to send electronic mail about Prism to Thinking Machines Corporation. The address to which the mail is sent is set when Prism is installed; it may be sent to your local applications engineer, or directly to Thinking Machines' customer support.

When you issue the command, Prism invokes an editor. In the edit buffer are the last few Prism error messages you received (as a convenience in case you are reporting a bug), along with information about how to send the mail. The editor that Prism invokes depends on the setting of the Prism resource **Prism.editor**. If there is no setting, Prism uses the setting of your `EDITOR` environment variable.

You cannot redirect the output of this command.

file

Changes the current source file.

Synopsis

`file` [*filename*]

Description

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

Version 1.2, March 1993

func

Changes the current procedure or function.

Synopsis

`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.

Synopsis

help [**commands** | **release** | *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.

Use the **release** option to display the release notes for the version of Prism that you are running.

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

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

Version 1.2, March 1993

hide

Removes a pane from a split source window.

Synopsis

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 UNIX signal.

Synopsis

`ignore` [*number* | *signal-name*]

Description

Prism can catch UNIX 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.

Version 1.2, March 1993

list

Lists lines in the current source file.

Synopsis

`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.

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.

Synopsis

`load` *filename*

Description

The `load` command loads the file specified by *pathname* into Prism. The file must be an executable program compiled with the appropriate debugging switch and/or performance analysis switch (if you want to collect performance statistics).

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.

Synopsis

`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.

Synopsis

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 UNIX version of **make**.

By default, Prism uses the standard UNIX **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.

Version 1.2, March 1993

next

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

Synopsis

next [*n*]

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.

nexti

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

Synopsis

nexti [*n*]

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.

Version 1.2, March 1993

perf

Displays performance data.

Synopsis

```
perf
```

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.

`perf` is actually an alias for `performance statistics`.

perfadvice

Displays an analysis of performance data.

Synopsis

```
perfadvice
```

Description

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

perfload

Loads a performance data file.

Synopsis

`perfload` *filename*

Description

Use the `perfload` 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.

Synopsis

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 **perfload** command.

print

Prints the values of one or more variables or expressions.

Synopsis

[`where` (*expression*)] `print` *expression* [`,` *expression* ...]

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.

In CM Fortran, the expression can operate on a conformable array. In C*, the expression can operate on a parallel variable that is of the same shape as the parallel variable being displayed.

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; each is 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.

The SPARC registers are:

`$pc`	program counter
`$npc`	next program counter
`$fsr`	floating status register
`$fq`	floating queue
`$wim`	window invalid mask
`$tbr`	trap base register
`$g0-$g7`	global registers
`$i0-$i7`	input registers
`$l0-$l7`	local registers
`$o0-$o7`	output registers
`$sp`	stack pointer (synonym for `$o6`)
`$fp`	frame pointer (synonym for `$i6`)
`$f0-$f31`	floating-point registers
`$y`	Y register

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

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.

Synopsis

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 UNIX C shell counterpart. See your UNIX documentation for more information.

pushbutton

Adds a Prism command to the tear-off region.

Synopsis

`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.

Synopsis

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 UNIX counterpart. See your UNIX documentation for more information.

quit

Leaves Prism.

Synopsis

```
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.

Synopsis

```
reload
```

Description

Use the `reload` command to reload the program currently loaded in Prism. Events you have associated with the program are lost.

rerun

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

Synopsis

`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 UNIX manner.

Version 1.2, March 1993

return

Steps out to the caller of the current function.

Synopsis

return [*count*]

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**.

run

Executes the currently loaded program.

Synopsis

`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 UNIX manner.

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

select

Chooses the master pane in a split source window.

Synopsis

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 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.

Synopsis

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
```

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 the 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,

Version 1.2, March 1993

```
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 specified explicitly. The default is 10.

$fortran_adjust_limit — Prism uses this value as the limit of a front-end or partition-manager 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.

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

Issue the **unset** command to remove a setting.

setenv

Displays or sets environment variables.

Synopsis

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 UNIX C shell counterpart. See your UNIX documentation for more information.

sh

Passes a command line to the shell for execution.

Synopsis

`sh` [*command-line*]

Description

Use the `sh` command to execute a UNIX 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.

You cannot redirect the output of this command.

show

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

Synopsis

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.

Examples

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

```
show .s
```

If you have used CMAX to translate a Fortran 77 program into CM Fortran, and you have loaded the CM Fortran executable program, issue this command to display the Fortran 77 source code:

```
show .f
```

This assumes that a mapping file (with the name *source-code*`.ttab`) is available to provide the mapping between the CM Fortran and Fortran 77 sources.

show events

Prints out the event list.

Synopsis

```
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.

source

Reads commands from a file.

Synopsis

`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

Prints out the event list.

Synopsis

```
status
```

Description

Use the `status` 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.

`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.

Synopsis

`step` [*n*]

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.

stepi

Executes one or more machine instructions.

Synopsis

`stepi` [*n*]

Description

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

stepout

Steps out to the caller of the current function.

Synopsis

`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.

Synopsis

stop [*var* | **at** *line* | **in** *func*] [**if** *expression*] [{*cmd; cmd ...*}] [**after** *n*]

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** *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 *line* stops execution when the specified line is reached. If the line is not in the current file, use the form *"filename":line-number*.

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.

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.

Synopsis

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

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.

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.

Synopsis

`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.

trace

Traces program execution.

Synopsis

`trace` [*var* | `at` *line* | `in` *func*] [`if` *expression*] [{*cmd*; *cmd* ...}] [`after` *n*]

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-number* 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*. 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.

Version 1.2, March 1993

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 "next"s over them if they do not have source (for example, if they are run-time routines or CM Fortran procedures in files compiled with **-nodebug**).

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.

Synopsis

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

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.

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.

type

Provides type information about Paris parallel variables, arrays, pointers, and structures.

Synopsis

type *type variable*

Description

Use the **type** command to tell Prism the type of a parallel variable, array, pointer, or structure in a Paris program. For example, if you define a variable of length 32 and use it in Paris routines dealing with **floats**, you need to tell Prism that the variable represents a **float**.You must do this before you print the variable, array, pointer, or structure or use it in an expression. You only need to do it for variables, arrays, or structures that are allocated on the CM, and you only need to do it once per Prism session.

Examples

This example defines **car** to be a parallel **unsigned int**:

```
type unsigned int car
```

This example defines **bus** to be a parallel **struct** of type **vehicle**:

```
type struct vehicle bus
```

This example defines **train** to be a parallel array of 20 **ints**:

```
type int train[20]
```

This example defines **boat** to be a pointer to a parallel **int**:

```
type int *boat
```

unalias

Removes an alias.

Synopsis

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.

Synopsis

`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
```

You can unset it as follows:

```
unset fred
```

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

unsetenv

Unsets an environment variable.

Synopsis

unsetenv *variable*

Description

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

Prism's **unsetenv** command is identical to its UNIX C shell counterpart. See your UNIX documentation for more information.

untearoff

Removes a button from the tear-off region.

Synopsis

`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.

Version 1.2, March 1993

up

Moves up the call stack.

Synopsis

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.

Synopsis

use [*directory*]

Description

Issue the **use** command to add *directory* to 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.

whatis

Prints the declaration of a name.

Synopsis

whatis *name*

Description

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

For CM arrays, the information printed includes **[CM array]**. For example:

```
integer foo(1:65536) [CM array]
```

when

Sets a breakpoint. The **when** command is comparable to the **stop** command.

Synopsis

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** *addr* | **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-number* stops execution when the specified line is reached. If the line is not in the current file, use the form *"filename":line-number*.

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. This form of when slows execution considerably. The condition can be any expression that evaluates to true or false. Unless combined with the **at** *line* syntax, this form of **where** 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.

Version 1.2, March 1993

`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

Shows the call stack.

Synopsis

where [*count*]

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 **t** for this command.

whereis

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

Synopsis

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

Prints the fully qualified name of an identifier.

Synopsis

`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*.

Version 1.2, March 1993