Chapter 2

Using Prism

This chapter describes general aspects of using Prism. Succeeding chapters describe how to perform specific functions within Prism. To learn:

- **What to do before entering the Prism programming environment,** see Section 2.1.
- **How to enter Prism,** see Section 2.2.
- **How to perform actions within Prism,** see Section 2.3.
- **How to use the menu bar**, see Section 2.4.
- **How to use windows, dialog boxes, and lists,** see Section 2.5.
- **How to use the source window and line-number region,** see Section 2.6.
- **How to use the command window,** see Section 2.7.
- **How to use Prism with Paris programs,** see Section 2.8. Read this section only if you are going to be running Paris programs on a CM-2 or CM-200 series Connection Machine system.
- **How to write expressions in Prism,** see Section 2.9.
- **How to issue UNIX commands**, see Section 2.10.
- **How to leave Prism,** see Section 2.11.

The best way to learn how to use Prism is to try it out for yourself. We encourage you to do this as you read this chapter. A quick way to try out some of Prism's features is to take its on-line tutorial. To do this, left-click

Copyright © 1991–1994 Thinking Machines Corporation

with your mouse on **Help** in the menu bar at the top of the main Prism window. Then left-click on **Tutorial** in the menu. A window will appear that will let you choose among several brief tutorials covering Prism's major features.

2.1 Before Entering Prism

2.1.1 Supported Languages and Compilers

You can work on FORTRAN 77, CM Fortran, C, and C* programs within Prism. Specifically, Prism supports these compilers:

- **`cmf`**
- **`cs`**
- Sun **`f77`**
- Sun **`cc`**
- Sun **`acc`**
- **`gcc`**

Note these points:

- Prism does not support **`gcc`**'s extensions to Standard C. If you make use of these extensions, or if you use another compiler, you may receive error messages about bad symbol table entries.
- Performance data is available only for C* and CM Fortran programs.
- Prism can be used with global/local CM Fortran programs; see Section 10.4.6 for more information.

2.1.2 Compiling and Linking Your Program

Note these points in compiling and linking your program:

- To use Prism's debugging features, compile and link each program module with the **`-g`** compiler option to produce the necessary debugging information.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

- To obtain performance data about a C* or CM Fortran program, compile it with the **-cmprofile** option. For CM-2/200 C* programs, you must not turn off optimization by compiling with the **-O0** option.
- You can combine the **-g** and **-cmprofile** options. For slicewise and CM-5 CM Fortran programs, however, this slows execution and distorts performance data. In CM Fortran programs (both slicewise and Paris) and CM-5 C* programs, you can do some debugging by specifying **-cmprofile** alone. This can cause problems in setting breakpoints and stepping through a program, however.
- To use Prism for a Paris program on a CM-2 or CM-200, you must link with the library **libprism2.a**.
- To use Prism for a C/Paris program, you must include this text on the **cc** command line:

```
-Zcc "-u _CMC_dbx"
```

2.1.3 Setting Up Your Environment

To enter the Prism programming environment, you must be logged in to a terminal or workstation running the X Window System (unless you want to run Prism in commands-only mode; see below).

- To use a CM-2 or CM-200 series Connection Machine system, you must be logged in to a front end that is connected to a CM.
- To use a CM-5, you must be logged in to a CM-5 partition manager, unless you are using the remote execution facility for Node Prism (see Section 10.2.3).

Prism works under these X servers:

- MIT X11R4
- NCD X11R4
- Silicon Graphics X11R4

Prism works under these window managers:

- **twm**
- **tvtwm**

Copyright © 1991–1994 Thinking Machines Corporation

- **mwm**
- **gwm**
- **olwm**
- **uwm**

Make sure your **DISPLAY** environment variable is set for the terminal or workstation from which you are running X. For example, if your workstation is named Valhalla, you can issue this command (if you are running the C shell):

```
% setenv DISPLAY valhalla:0
```

Note for CM-2 and CM-200 users: You may want to attach to the CM before entering Prism. Prism provides a utility for attaching to a CM, but you may need more flexibility than this utility provides. For information on the Prism utility, see Section 3.4. For general information on attaching to a CM, see the *CM User's Guide*.

2.2 Entering Prism

2.2.1 Invoking Prism

To enter Prism, issue the **prism** command at your UNIX prompt. When you enter Prism, you see the main window shown in Figure 1 in Chapter 1.

2.2.2 Command-Line Options

Loading a Program

If you specify the name of an executable program on the command line, that program is automatically loaded into Prism. For example:

```
% prism primes1.x
```

You can also load a process that is currently running. Add its process ID after the name of the program. For example:

```
% prism primes1.x 2256
```


Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

You can obtain the process's process ID by issuing the **ps** or **cmps** command.

You can also associate a core file with a program. Add the name of the core file after the name of the program.

See Chapter 3 for more information on loading a program.

Specifying Commands-Only Prism

Use the **-C** option to bring up Prism in commands-only mode. This allows you to run Prism on a terminal with no graphics capability.

Use the **-CX** option from an xterm to bring up a commands-only Prism that lets you redirect the output of certain Prism commands to X windows.

See Appendix A for information on commands-only Prism.

Specifying Node Prism (CM-5 Only)

Use the **-node** option to bring up Node Prism. Use Node Prism to debug, visualize data in, and analyze the performance of message-passing programs on the CM-5. See Chapter 10 for information on Node Prism.

You can combine **-node** with other options. For example, this command brings up a commands-only version of Node Prism:

```
% prism -C -node
```

Specifying Remote Execution (Node Prism Only)

In Node Prism, you can log in to a computer that is linked via a network to the CM-5 on which your message-passing program is to be run; Prism does not have to be executed on a partition manager. This is known as *remote execution*. (Note that remote execution is currently available only for Node Prism.) Use the **-connect** option to specify the name of the partition manager on which you want your program to run. To run your program on the partition manager cm5a, for example, issue this command:

```
% prism -node -connect cm5a
```

See Section 10.2.3 for complete information on remote execution.

Copyright © 1991–1994 Thinking Machines Corporation

Specifying X Toolkit Options

You can include most standard X toolkit command-line options when you issue the **prism** command; for example, you can use the **-geometry** option to change the size of the main Prism window. See your X documentation for information on these options. Also, note these limitations:

- The **-font**, **-title**, and **-rv** options have no effect.
- The **-bg** option is overridden in part by the setting of the **Prism.textBgColor** resource, which specifies the background color for text in Prism; see Section 9.4.5.

X toolkit options are meaningless, of course, if you use **-C** to run Prism in commands-only mode.

Specifying Input and Output Files

You can use the form

```
% prism < input-file
```

to specify a file from which Prism is to read and execute commands upon startup. Similarly, use the form

```
% prism > logfile
```

to specify a file to which Prism commands and their output are to be logged.

If you have created a **.prisminit** initialization file, Prism automatically executes the commands in the file when it starts up. See Section 9.5 for information on **.prisminit**.

2.3 Within Prism

Within Prism, you can perform most actions in one of three ways:

- by using a mouse; see Section 2.3.1
- by using keyboard alternatives to the mouse; see Section 2.3.2
- by issuing commands from the keyboard; see Section 2.3.3

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

2.3.1 Using the Mouse

You can point and click with a mouse in Prism to choose menu items and to perform actions within windows and dialog boxes. Prism assumes that you have a standard three-button mouse.

This manual uses these terms when discussing the mouse:

- The *mouse pointer* is the graphical image that appears on the screen and represents the current location of the mouse. Mouse actions are generally related to the location of the mouse pointer.
- *Left-clicking* refers to pressing the left mouse button, *right-clicking* refers to pressing the right button, etc. If we don't mention which button to click, the left button is assumed.
- To *double-click* the mouse is to press the left mouse button twice in rapid succession. This is a shortcut for selecting items in lists.
- To *drag* the mouse pointer is to move it while holding down a mouse button; you do this, for example, to select text in the source window. To *slide* the mouse pointer is to move it without holding down a mouse button.

In any window where you see this mouse icon:

you can left-click on the icon to obtain information about using the mouse in the window.

2.3.2 Using Keyboard Alternatives to the Mouse

You can use the keyboard to perform many of the same functions you can perform with a mouse. This section lists these keyboard alternatives.

In general, to use a keyboard alternative, the *focus* must be in the screen region where you want the action to take place. The focus is generally indicated by the *location cursor*, which is a heavy line around the region.

Copyright © 1991–1994 Thinking Machines Corporation

General keyboard alternatives are:

Esc	Use the **Esc** key instead of the **Close** or **Cancel** button to close the window or dialog box in which the mouse pointer is currently located.
Tab	Use the **Tab** key to move the location cursor from field to field within a window or dialog box. The buttons in a window or box constitute one field. The location cursor highlights one of the buttons when you tab to this field.
Shift-Tab	**Shift-Tab** performs the same function as **Tab**, but moves through the fields in the opposite direction.
Return	Use the **Return** key to choose a highlighted choice in a menu, or to perform the action associated with a highlighted button in a window or dialog box.
arrow keys	Use the up, down, left, and right arrow keys to move within a field. For example, when the location cursor highlights a list, you can use the up and down arrow keys to move through the choices in the list. In some windows that contain text, pressing the **Control** key along with an up or down arrow key scrolls the text one-half page.
F1	Use the **F1** key instead of the **Help** button to obtain help about a window or dialog box.
F10	Use the **F10** key to move the location cursor to the menu bar.
Meta	Use the **Meta** key along with a mnemonic (see below) to display a menu. The **Meta** key has different names on different keyboards; on some it is the **Left** or **Right** key.
Ctrl-c	Use the **Ctrl-c** key combination to interrupt command execution.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

The following keys and key combinations work on the command line and in *text-entry boxes* — that is, fields in a dialog box or window where you can enter or edit text:

Back Space	Deletes the character to the left of the I-beam cursor.
Delete	Same as **Back Space**.
Ctrl-a	Moves to the beginning of the line.
Ctrl-b	Moves back one character.
Ctrl-d	Deletes the character to the right of the I-beam cursor.
Ctrl-e	Moves to the end of the line.
Ctrl-f	Moves forward one character.
Ctrl-k	Deletes to the end of the line.
Ctrl-u	Deletes to the beginning of the line.

In addition, you can use *mnemonics* and *keyboard accelerators* to perform actions from the menu bar; see Section 2.4.

2.3.3 Issuing Commands

You can issue commands in Prism from the command line in the command window. Most commands duplicate functions you can perform from the menu bar; it's up to you whether you use the command or the corresponding menu selection. Some functions are only available via commands. See the *Prism Reference Manual* for complete information on Prism commands. Section 2.7 below describes how to use the command window.

Many commands have the same syntax and perform the same action in both Prism and the UNIX debugger **`dbx`**. There are differences, however; we recommend that you check the reference description of a command before using it.

Copyright © 1991–1994 Thinking Machines Corporation

2.4 Using the Menu Bar

The menu bar is the line of titles across the top of the main window of Prism; see Figure 2.

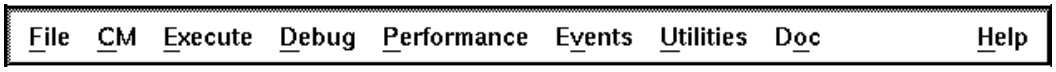


Figure 2. The menu bar.

Each title is associated with a pull-down menu, from which you can perform actions within Prism; see Figure 3 for an example. (The **CM** title appears only if you are running Prism from a CM-2 or CM-200 front end.)

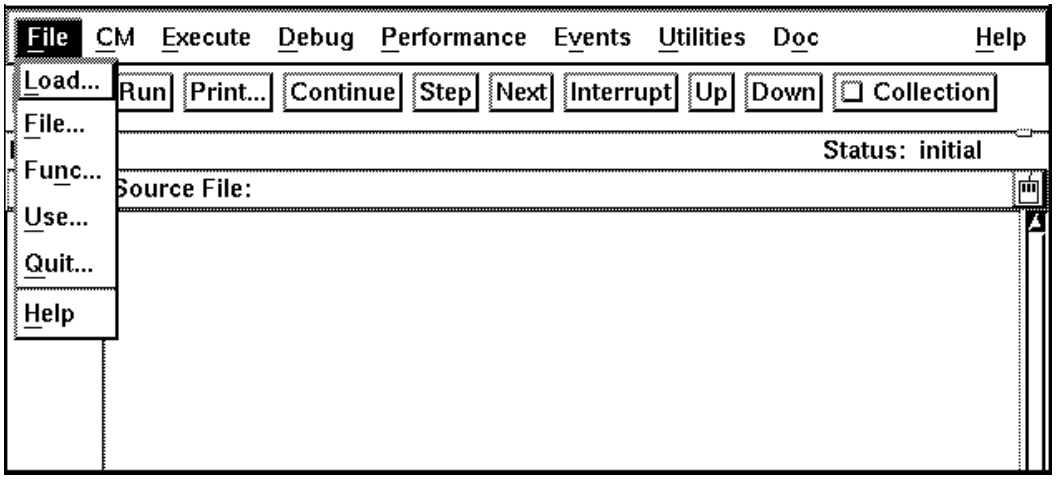


Figure 3. A Prism menu.

2.4.1 With a Mouse

To display the pulldown menu associated with a title in the menu bar, left-click on the title with your mouse. The menu appears beneath the title. To browse through the menus, drag the mouse pointer over the menu-bar titles; each menu appears in turn.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

There are two ways to choose an item from a menu:

- Drag the mouse pointer until it is on the selection you want, and then let go.
- Slide the pointer onto the selection you want, and then left-click the mouse.

To make a menu disappear, either drag the mouse pointer off the menu and then release the mouse button, or slide the mouse pointer off the menu and then left-click the mouse.

2.4.2 From the Keyboard

To go to the menu bar, press the **F10** key.

When the location cursor is in the menu bar, you can use the left and right arrow keys to display the menus under each title. To select an item from a menu, use the up and down arrow keys to move through the items until the one you want is highlighted. Then press **Return** or the spacebar to choose it.

Another way to display menus from the keyboard is to type the mnemonic associated with the menu, as described below.

Mnemonics

Mnemonics are a quick way of displaying menus, or of choosing items from a menu, simply by typing a letter.

Each title on the menu bar, and each item in their pulldown menus, has a letter underlined. For example:

File

The underlined letter is the mnemonic for the item. Typing the mnemonic is equivalent to clicking on the item with the mouse. For example, typing **F** from the menu bar displays the **File** pulldown menu. The menu stays on the screen until you make a choice from it or press the **Esc** key to cancel it.

Copyright © 1991–1994 Thinking Machines Corporation

Note these points in using mnemonics:

- To use a mnemonic for a menu bar, the location cursor must be in the menu-bar region. If the cursor is outside this region, press the **F10** key to move the location cursor to the menu bar. Or press the **Meta** key along with the mnemonic.
- The mnemonics for items in a pulldown menu work only when the menu is displayed. Thus, to display the dialog box for loading a file, type **F** (in the menu bar region) to display the **File** menu, and then type **L** to display the **Load** dialog box.

To perform a function directly, without displaying the pulldown menu, you can use keyboard accelerators instead.

Keyboard Accelerators

A keyboard accelerator is a shortcut that lets you choose a frequently used menu item without displaying its pulldown menu. Keyboard accelerators consist of the **Control** key plus a function key; you press both at the same time to perform the action. The keyboard accelerator for a menu selection is displayed next to the name of the selection; if nothing is displayed, there is no accelerator for the selection.

The keyboard accelerators are:

Ctrl-F1	**Run**
Ctrl-F2	**Continue**
Ctrl-F3	**Interrupt**
Ctrl-F4	**Step**
Ctrl-F5	**Next**
Ctrl-F6	**Where**
Ctrl-F7	**Up**
Ctrl-F8	**Down**
Ctrl-F9	**Collection**

2.4.3 What Happens When You Choose an Item

When you choose an item, Prism performs the action associated with the item. If it needs more information, it displays a window or dialog box, which you use to provide Prism with the information. If a menu selection has "..."

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

after it, a window or dialog box is displayed when you choose it. Later chapters in this guide describe the actions associated with specific titles in the menu bar.

2.5 Using Windows, Dialog Boxes, and Lists

This section discusses general aspects of using windows, dialog boxes, and scrollable lists in Prism. If you are familiar with other X applications, you can skip this section.

2.5.1 Windows

Windows are used to display information that you may want to keep on your screen; they also sometimes request information from you. The main Prism display is a window. The source window and the command window are *panes* within this main window. Other windows are displayed as the result of choosing menu items.

Prism does not determine how window operations are performed (except for panes — like the source window — that appear within other Prism windows); this is under the control of the window manager for your X Window System. Different window managers let you operate windows in different ways; check the documentation for your window manager, or ask your system administrator for more information. Generally, you will be able to perform these actions:

- **Delete the window.** It is possible to delete windows. It is also possible to "destroy" a window. *Do not destroy any Prism windows.* This will kill Prism.
- **Turn the window into an icon.** An *icon* is a small graphical image representing a window. Often you can close a window by clicking on a box in the upper left corner; this creates an icon. You can then click on the icon if you want to open it again.
- **Move the window.** Often you can do this by clicking on a horizontal bar along the top of the window, then dragging the window to its new location.

Copyright © 1991–1994 Thinking Machines Corporation

- **Change the size of the window.** Often you can do this by clicking on the small *resize box* in a corner of the window, then dragging the window until it is the size you want.

- **Scroll through the window**, if the text is longer than the height of the window. Typically there is a *scroll bar* at one side of the window; The scroll bar has arrows at the top and bottom; you can click on the arrows to move the text in the window up or down. The scroll bar also has an *elevator* that shows where the text is located in terms of the total text that can be displayed. For example, the elevator is at the top when the beginning text is displayed. You can drag this elevator to scroll quickly through the text. You can also click above or below the elevator to move up or down a page.

  There can also be a *horizontal* scroll bar that lets you scroll through text that is too wide to be displayed in the window.

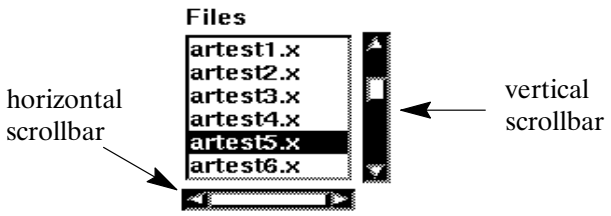


Figure 4. Scroll bars.

2.5.2 Dialog Boxes

Prism uses dialog boxes to obtain information from you. Once you have provided the information, the dialog box goes away. For example, the **Run (args)** selection from the **Execute** menu displays a dialog box. You can close it without providing information by clicking on **Cancel.**

Your window manager may also let you perform actions such as moving a dialog box or turning it into an icon.

If a dialog box gets buried beneath windows, reissue the command that displayed the dialog box; the box will reappear at the front of your screen.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

2.5.3 Lists

Many windows and dialog boxes in Prism contain lists that you can scroll through (for example, in choosing a program to load). If the list is too long for the space provided, there is a scroll bar to the right, which operates as described in Section 2.5.1. You can also use the up and down arrow keys to move through the list.

If names in the list are too long for the space provided (for example, because files have long pathnames), there is a scroll bar beneath the list; you can use this scroll bar to scroll horizontally. You can also use the left and right arrow keys to scroll across the line.

Typically, you will be selecting an item from the list. To do this, first highlight it by clicking on it or moving to it with the up or down arrow key; then click on the **Select** button (or its equivalent), or press the **Return** key. Or you can simply double-click, rapidly, on the item.

2.6 Using the Source Window and Line-Number Region

2.6.1 The Source Window

The source window displays the source code for the executable program loaded into Prism. (Chapter 3 describes how to load a program into Prism, and how to display the different source files that make up the program.) When you execute the program, and execution then stops for any reason, the source window updates to show the code being executed at the stopping place. The **Source File:** field at the top of the source window lists the file-name of the file displayed in the window.

The source window is a separate pane within the main Prism window. You can resize it by dragging the small resize box at the upper right of the window. If you change its size, the new size is saved when you leave Prism.

You cannot edit the source code displayed in the source window. To edit source code within Prism, you must call up an editor; see Chapter 7.

Copyright © 1991–1994 Thinking Machines Corporation

Moving through the Source Code

As mentioned above, you can move through a source file displayed in the source window by using the scroll bar on the right side of the window. You can also use the up and down arrow keys to scroll a line at a time, or press the **Control** key along with the arrow key to move half a page at a time. To return to the current execution point, type **Ctrl-x** in the source window.

To search for a text string in the current source file, issue the **/***string* or **?***string* command in the command window. The **/***string* command searches forward in the file for the string you specify, and repositions the file at the first occurrence it finds. The **?***string* command searches backward in the file for the string you specify.

You can display different files by choosing the **File** or **Func** selection from the **File** menu; see Section 3.6. You can also move *between* files. Prism keeps a list of the files you have displayed. With the mouse pointer in the source window, do this to move through the list:

- To display the previous file in the list, click the middle mouse button while pressing the left button. You are returned to the location at which you left the file.
- To display the next file in the list, click the right mouse button while pressing the left button.

Selecting Text

You can select text in the source window by dragging over it with the mouse; the text is then highlighted. Or double-click with the mouse pointer pointing to a word to select just that word. Left-click anywhere in the source window to "deselect" selected text.

Right-click in the source window to display a menu that includes actions to perform on the selected text; see Figure 5. For example, select **Print** to display a visualizer containing the value(s) of the selected variable or expression at the current point of execution. (See Chapter 5 for a discussion of visualizers and printing.) To close the popup menu, right-click anywhere else in the main Prism window.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

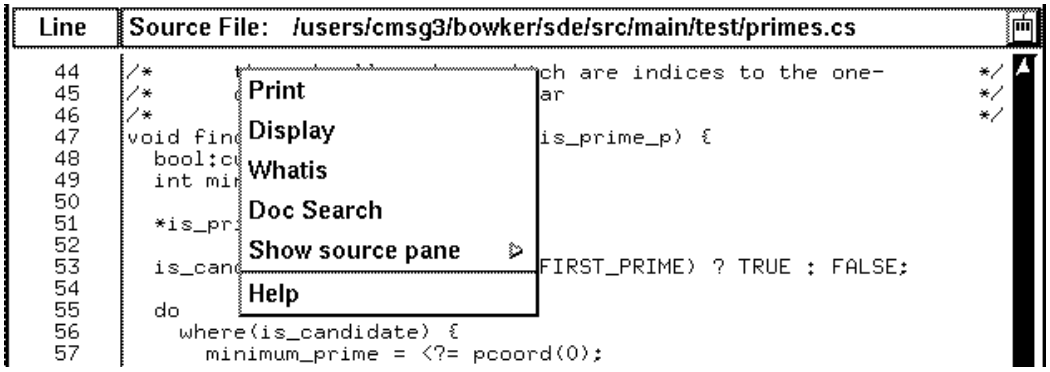


Figure 5. The popup menu in the source window.

You can print the value of a variable or expression directly from the source window, without displaying the menu. To do this, press the **Shift** key while selecting the variable or expression. A visualizer is displayed, showing the value(s) of the variable or expression.

You can display the definition of a function by pressing the **Shift** key while selecting the name of the function in the source window. This is equivalent to choosing the **Func** selection from the **File** menu and selecting the name of the function from the list; see Chapter 3. Do not include the arguments to the function, just the function name.

Splitting the Source Window

You can split the source window to simultaneously display the source code and assembly code of the loaded program. Follow these steps to split the source window:

1. First load a program, as described in Chapter 3.
2. Right-click in the source window to display the popup menu, as described above.
3. Click on **Show source pane** in the popup menu.
4. This displays another menu. Choose **Show .s source** from it.

This causes the assembly code for your program to be displayed in the bottom pane of the window, as shown in Figure 6.

Copyright © 1991–1994 Thinking Machines Corporation

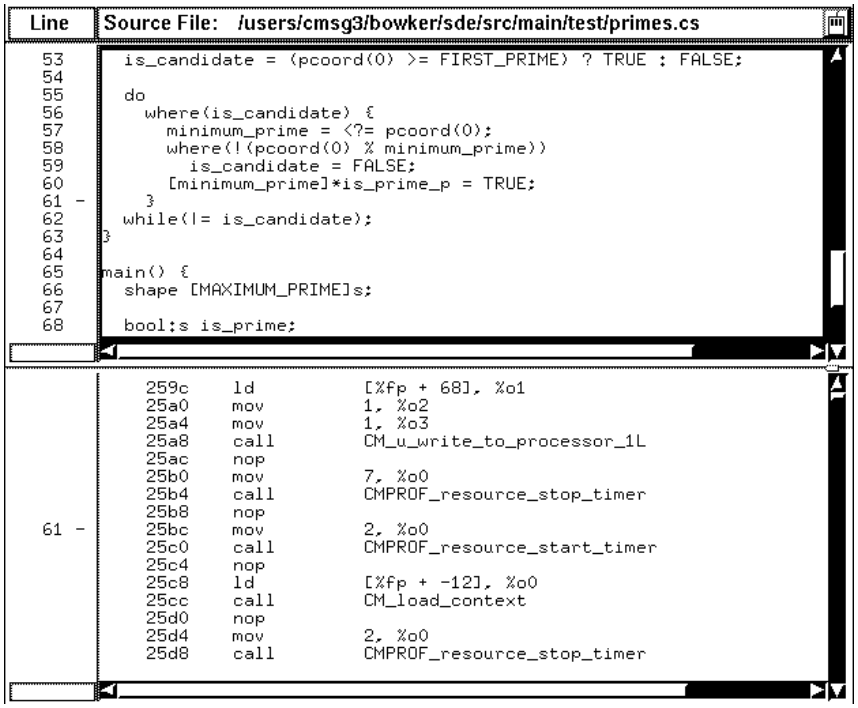


Figure 6. A split source window.

When you split the source window, the top pane is highlighted; it is the *master pane*. Left-click in the slave pane to make it the master. If you scroll through the master, the slave pane scrolls to the corresponding place as well. Scrolling through the slave does not cause the master to scroll.

To return to a single source window, right-click in the pane you want to get rid of, and choose **Hide this source pane** from the popup menu.

NOTE: If you have used the CMAX Converter to translate a FORTRAN 77 program into CM Fortran, you can use Prism's split-screen technique to view the FORTRAN 77 version of a CM Fortran program, as well as its assembly code. The use of Prism with CMAX is described in Appendix B.

2.6.2 The Line-Number Region

The line-number region shows the line numbers associated with the source code displayed in the source window. Figure 7 shows a portion of a line-number region, with a breakpoint set.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

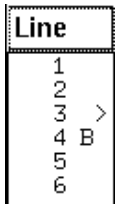


Figure 7. The line-number region.

The **>** symbol in the line-number region in Figure 7 is the *execution pointer*. When the program is being executed, the execution pointer points to the next line to be executed. If you move elsewhere in the source code, typing **Ctrl-x** returns to the current execution point.

A **B** appears in the line-number region next to every line at which execution is to stop. You can set simple breakpoints directly in the line-number region; all methods for setting breakpoints are described in Section 4.3.

A **T** appears in the line-number region next to a line for which Prism is tracing execution. See Section 4.4 to learn how to trace program execution.

Shift-click on **B** or **T** in the line-number region to display the *event* associated with the breakpoint or tracepoint. See Section 4.1 for a discussion of events.

The display of breakpoints and tracepoints in the line-number region is slightly more complicated in Node Prism; see Section 10.5.1.

There are two other symbols you will see in the line-number region:

- The **–** symbol is the *scope pointer*; it indicates the current source position (that is, the scope). Prism uses the current source position to interpret names of variables. When you scroll through source code, the scope pointer moves to the middle line of the code that is displayed. Various Prism commands also change the position of the scope pointer.

- The ***** symbol is used when the current source position is the same as the current execution point; this happens whenever execution stops.

If you right-click in the line-number window, you display the source-window popup menu discussed in the previous section. Right-click anywhere in the main Prism window to close this menu.

Copyright © 1991–1994 Thinking Machines Corporation

2.7 Using the Command Window

The command window is the area at the bottom of the main Prism window in which you type commands and receive Prism output.

The command window consists of two boxes: the command line, at the bottom, and the history region, above it. Figure 8 shows a command window, with a command on the command line and messages in the history region.

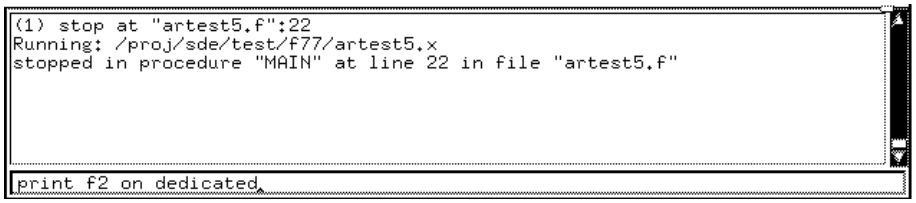


Figure 8. The command window.

The command window is a separate pane within the main Prism window. You can resize this window (using the resize box at the top right of the window) and scroll through it. If you don't intend to issue commands in the command window, you may want to make this window smaller, so that you can display more code in the source window. If you use the command window frequently, you may want to make it bigger. If you change the size of the window, the new size is saved when you leave Prism.

2.7.1 Using the Command Line

You type commands on the command line at the bottom of the command window. You can type in this box whenever it is highlighted and an I-shaped cursor, called an *I-beam*, appears in it. See Section 2.3.2 for a list of keystrokes you can use in editing the command line. Press **Return** to issue the command. Type **Ctrl-c** to interrupt execution of a command (or choose the **Interrupt** selection from the **Execute** menu).

You can issue multiple commands on the Prism command line; separate them with a semicolon (**;**). One exception: If a command takes a filename as an argument, you cannot follow it with a semicolon, because Prism can't tell if the semicolon is part of the filename.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

Prism keeps the commands that you issue in a buffer. Type **Ctrl-p** to display the previous command in this buffer. Type **Ctrl-n** to display the next command in the buffer. You can then edit the command and issue it in the usual way.

During long-running commands (for example, when you have issued the **`run`** command to start a program executing), you may still be able to execute other commands. If you issue a command that requires that the current command complete execution, you receive a warning message and Prism waits for the command to complete.

2.7.2 Using the History Region

Commands that you issue on the command line are echoed in the history region, above the command line. Prism's response appears beneath the echoed command. Prism also displays other messages in this area, as well as command output that you specify is to go to the command window. Use the scroll bar at the right of this box to move through the display.

You can select text in the history region, using one of these methods:

- Double-click to select the word to which the mouse pointer is pointing.
- Triple-click to select the line on which the mouse pointer is located.
- Press the left mouse button and drag the mouse over the text to select it.

You can then paste the selected text into other text areas within Prism by clicking the middle mouse button.

To re-execute a command, triple-click on a line in the history region to select it, then click the middle mouse button with the mouse pointer still in the history region. If you middle-click with the mouse pointer on the command line, the selected text appears on the command line but is not executed. This gives you a way to edit the text before executing it.

Copyright © 1991–1994 Thinking Machines Corporation

2.7.3 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 (a stack trace) 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** (abbreviated **com**). This sends output to the command window; this is the default.
- **dedicated** (abbreviated **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 are) and specify that output is to be sent to the dedicated window, this window will be updated. For example,

  ```
  list on ded
  ```

  displays the output of the **list** command in a dedicated window. (Some commands that have equivalent menu selections display their output in the standard window for the menu selection.)
- **snapshot** (abbreviated **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. This is useful if you want a particular label for a window. For example, if you were doing a stack trace at line 22, you could issue this command:

```
where on line22
```

to label the window with the location of the stack trace.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

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

2.7.4 Logging Commands and Output

As we mentioned in Section 2.2.2, you can specify on the Prism command line the name of a file to which commands and output are to be logged. You can also do this from within Prism, by issuing the **log** command.

Use the **log** command to log Prism commands and output to a file. This can be helpful in saving a record of a Prism session. For example,

```
log @ prism.log
```

logs output to the file **prism.log**. Use **@@** instead of **@** to append the log to an already existing file. Issue the command

```
log off
```

to turn off logging.

You can use the **log** command along with the **source** command to replay a session in Prism; see the next section. If you want to do this, you must edit the log file to remove Prism output.

2.7.5 Executing Commands from a File

As we mentioned in Section 2.2.2, you can specify on the Prism command line the name of a file from which commands are to be read in and executed. You can also do this from within Prism, by issuing the **source** command.

Using the **source** command lets you rerun a session you saved via the **log** command. You might also use **source** if, for example, your program has a long argument list that you don't want to retype constantly.

For example,

```
source prism.cmds
```

reads in the commands in the file **prism.cmds**. They are executed as if you had actually typed them in the command window. When reading the file, Prism interprets lines beginning with a pound sign (#) as comments.

Copyright © 1991–1994 Thinking Machines Corporation

The **.prisminit** file is a special file of commands; if it exists, Prism executes this file automatically when it starts up. See Section 9.5 for more information.

2.8 Using Prism with Paris Programs

NOTE: Read this section only if you are going to run Paris programs on a CM-2 or CM-200 series Connection Machine system.

You can work on Paris programs in the Prism programming environment. However, before printing a Paris parallel variable, array, pointer, or structure, or using it in an expression, you must tell Prism its type. For example, if you define a variable of length 32 and use it in Paris routines dealing with **float**s, you need to tell Prism that the variable represents a **float**.

There are two ways of defining types:

From the source window: (This method does not work for structures, pointers, or arrays.) Select the variable in the source window, as described in Section 2.6.1. Then right-click to display a popup menu. Choose **Define Type** from this menu. (NOTE: **Define Type** appears in this menu only if you are logged in to a CM-2 or CM-200 front end.) Choosing **Define Type** displays another menu, listing C types. Click on the type of the variable. You can then operate on the variable within Prism.

From the command window: Issue the **type** command, specifying the type of the variable, structure, pointer, or array. For example,

```
type unsigned int a
```

specifies that **a** is a parallel **unsigned int**.

```
type struct foo bar
```

specifies that **bar** is a parallel **struct** of type **foo**.

```
type float array_a[10][20]
```

specifies that **array_a** is a parallel 10-by-20 array of **float**s.

```
type int *ptr
```

specifies that **ptr** is a pointer to a parallel **int**.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

As mentioned above, you have to define the types only once during a Prism session. To avoid having to do this during every session, put the appropriate **`type`** commands in your **`.prisminit`** file, or in a special initialization file for each Paris program you are working on; see Section 2.7.5 to learn how to use such initialization files.

2.9 Writing Expressions in Prism

While working in Prism, there are circumstances in which you may want to write expressions that Prism will evaluate. For example, you can print or display expressions, and you can specify an expression as a condition under which an action is to take place. You can write these expressions in the language of the program you are working on. This section discusses additional aspects of writing expressions.

2.9.1 How Prism Chooses the Correct Variable or Procedure

Multiple variables and procedures can have the same name in a program. This can be a problem when you specify a variable or procedure in an expression. To determine which variable or procedure you mean, Prism tries to *resolve its name* by using these rules:

1. It first tries to resolve the name using the scope of the current function. For example, if you use the name **`x`** and there is a variable named **`x`** in the current function, Prism uses that **`x`**. The current function is ordinarily the function at the program's current stopping point, but you can change this. See Section 3.6.

2. If this fails to resolve the name, Prism goes up the call stack and tries to find the name in the caller of the current function, then its caller, and so on.

3. If the name is not found in the call stack, Prism arbitrarily chooses one of the variables or procedures with the name in the source code. When Prism prints out the information, it adds a message of the form "[using *qualified name*]". Qualified names are discussed below.

Copyright © 1991–1994 Thinking Machines Corporation

Issue the **`which`** command to find out which variable or procedure Prism would choose; the command displays the fully qualified name, as described below.

Using Qualified Names

You can override Prism's procedure for resolving names by *qualifying* the name.

A fully qualified name starts with a back-quotation mark (`` ` ``). The leftmost symbol in the name is the file, followed optionally by the procedure, followed by the variable name. Each is preceded by a `` ` ``. Thus,

```
`foo`a
```

specifies the variable **`a`** in file **`foo`**. (Note that you drop the extension in the filename.) And

```
`foo`foo`a
```

specifies the **`a`** in the procedure **`foo`** in the file **`foo`**.

Partially qualified names do not begin with `` ` ``, but have a `` ` `` in them. For example,

```
foo`a
```

In this case, Prism looks up the leftmost name first and picks the innermost symbol with that name that is visible from your current location.

Use the **`whereis`** command to display a list of all the fully qualified names that match the identifier you specify.

Prism assigns its own names (for example, **`$b1`**) to local blocks of C code. This disambiguates variable names, in case you reuse a variable name in more than one of these local blocks.

Prism attempts to be case-insensitive in interpreting names, but will use case to resolve ambiguities.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

2.9.2 Using Fortran Intrinsic Functions in Expressions

Prism supports the use of a subset of Fortran intrinsic functions in writing expressions; the intrinsics work for all languages that Prism supports, except as noted below. For complete information on the intrinsics, see the *CM Fortran Language Reference Manual*.

The intrinsics, along with the supported arguments, are:

- **AIMAG(***complex number***)** — Returns the imaginary part of a complex number. Works for Fortran and CM Fortran only.
- **ALL(***logical array***)** — Determines whether all elements are true in a logical array. Works for Fortran and CM Fortran only.
- **ANY(***logical array***)** — Determines whether any elements are true in a logical array. Works for Fortran and CM Fortran only.
- **CMPLX(***numeric-arg***,** *numeric-arg***)** — Converts the arguments to a complex number. If the intrinsic is applied to Fortran variables, the second argument must not be of type complex or double-precision complex.
- **COUNT(***logical array***)** — Counts the number of true elements in a logical array. Works for Fortran and CM Fortran only.
- **DSIZE(***array***)** — Counts the total number of elements in the array.
- **MAXVAL(***array***)** — Computes the maximum value of all elements of a numeric array.
- **MINVAL(***array***)** — Computes the minimum value of all elements of a numeric array.
- **PRODUCT(***array***)** — Computes the product of all elements of a numeric array.
- **RANK(***scalar* or *array***)** — Returns the rank of the array or scalar.
- **REAL(***numeric argument*) — Converts an argument to real type. Works for Fortran and CM Fortran only.
- **SUM(***array***)** — Computes the sum of all elements of a numeric array.

The intrinsics can be either upper- or lowercase.

Copyright © 1991–1994 Thinking Machines Corporation

2.9.3 Writing C* Expressions

Prism currently does not parse many parts of the C* language. Here are some of the features of C* that Prism doesn't recognize:

- The functions **`pcoord`**, **`shapeof`**, **`rankof`**, and **`dimof`** are not supported.
- Dot notation is not understood.
- Shapes are not known objects.
- Current context isn't respected or understood.
- Reduction operators are not supported, except as listed below.
- The communication intrinsic functions are not supported.
- The auto-increment and auto-decrement operators are not understood.

Using C* Reduction Operators

Prism allows the use of these C* reduction operators in writing expressions:

`>?=`	Computes the maximum value of an array or parallel variable.
`<?=`	Computes the minimum value of an array or parallel variable.
`*=`	Computes the product of all elements of an array or parallel variable.
`+=`	Computes the sum of all elements of an array or parallel variable.

You can also use certain Fortran intrinsics, as described in Section 2.9.2.

Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

Passing Parallel Variables to Functions

If you call a C* function in Prism, you must pass parallel variables to it by reference, not by value. For example, if you have a parallel integer **p1**, you would pass it to function **f** as follows:

```
f(&p1)
```

Using Array-Section Syntax in C* Left Indexes

In Prism, you can use left indexes as you normally would to specify an element of a C* parallel variable. For example:

```
[112][147]pvar1
```

In addition, you can use array-section syntax from Fortran 90 to specify a range of elements. This syntax is useful, for example, if you want to print the values of only a subset of the elements of a parallel variable. The syntax is:

lower-bound`:` *upper-bound*`:` *stride*

where:

lower-bound	is the lowest-numbered coordinate you choose along the axis; it defaults to 0.
upper-bound	is the highest-numbered coordinate you choose along the axis; it defaults to the highest-numbered coordinate for the axis.
stride	is the increment by which elements are chosen between the lower bound and upper bound; it defaults to 1.

For example,

```
[5:25:5]pvar1
```

specifies elements 5, 10, 15, 20, and 25 of the parallel variable **pvar1**.

```
[0:10][100:110:2]pvar2
```

specifies the elements of the parallel variable **pvar2** that have coordinates 0 through 10 along axis 0 and coordinates 100, 102, 104, 106, 108, and 110 along axis 1.

Copyright © 1991–1994 Thinking Machines Corporation

For more information about array sections, see the *CM Fortran Programming Guide*.

2.9.4 Using C and C* Arrays in Expressions

Prism handles arrays slightly differently from the way C and C* handle them.

In a C or C* program, if you have the declaration

```
int a[10];
```

and you use **a** in an expression, the type of **a** converts from "array of **ints**" to "pointer to **int**". Following the rules of C and C*, therefore, a Prism command like

```
print a + 2
```

should print a hexadecimal pointer value. Instead, it prints two more than each element of **a** (that is, **a[0] + 2**, **a[1] + 2**, etc.). This allows you to do array operations and use visualizers on C and C* arrays in Prism. (The **print** command and visualizers are discussed in Chapter 5.)

To get the C/C* behavior, issue the command as follows:

```
print &a + 2
```

2.9.5 Hints for Detecting NaNs and Infinities

Prism provides expressions you can use to detect NaNs (values that are "not a number") and infinities in your data. These expressions derive from the way NaNs and infinities are defined in the IEEE standard for floating-point arithmetic.

To find out if **x** is a NaN, use the expression:

```
(x .ne. x)
```

For example, if **x** is an array, issue the command

```
where (x .ne. x) print x
```


Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

to print only the elements of **x** that are NaNs. (The **print** command is discussed in Chapter 5.)

Also, note that if there are NaNs in an array, the mean of the values in the array will be a NaN. (The mean is available via the **Statistics** selection in the **Options** menu of a visualizer — see Chapter 5.)

To find out if **x** is an infinity, use the expression:

```
(x * 0.0 .ne. 0.0)
```

2.10 Issuing UNIX Commands

You can issue UNIX and CMOST commands from within Prism.

From the menu bar: Choose the **Shell** selection from the **Utilities** menu. Prism creates a UNIX shell. The shell is independent of Prism; you can issue UNIX commands from it just as you would from any UNIX shell. The type of shell that is created depends on the setting of your **SHELL** environment variable.

From the command window: Issue the **sh** command on the command line. With no arguments, it creates a UNIX shell. If you include a UNIX command line as an argument, the command is executed, and the results are displayed in the history region.

Some UNIX commands have Prism equivalents, as described below.

2.10.1 Changing the Current Working Directory

By default your current working directory within Prism is the directory from which you started Prism. To change this working directory, use the **cd** command, just as you would in UNIX. For example,

```
cd /allen/bin
```

changes your working directory to **/allen/bin**.

```
cd ..
```


Copyright © 1991–1994 Thinking Machines Corporation

changes your working directory to the parent of the current working directory. Issue **cd** with no arguments to change the current working directory to your login directory.

Prism interprets all relative filenames with respect to the current working directory. Prism also uses the current working directory to determine which files to show in file-selection dialog boxes.

To find out what your current working directory is, issue the **pwd** command, just as you would in UNIX.

2.10.2 Setting and Displaying Environment Variables

You can set, unset, and display the settings of environment variables from within Prism, just as you do in UNIX.

Use the **setenv** command to set an environment variable. For example,

```
setenv EDITOR emacs
```

sets your **EDITOR** environment variable to **emacs**.

Use the **unsetenv** command to remove the setting of an environment variable. For example,

```
unsetenv EDITOR
```

removes the setting of the **EDITOR** environment variable.

Use the **printenv** command to print the setting of an individual environment variable. For example,

```
printenv EDITOR
```

prints the current setting of the **EDITOR** environment variable. Or, issue **printenv** or **setenv** with no arguments to print the settings of all your environment variables.

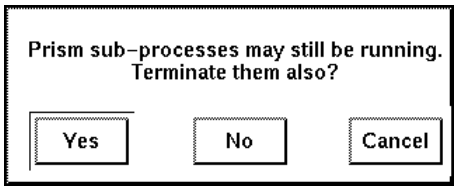
Version 2.0, April 1994
Copyright © 1991–1994 Thinking Machines Corporation

2.11 Leaving Prism

To leave Prism:

- **From the menu bar:** Choose the **Quit** selection from the **File** menu. You are asked if you are sure you want to quit. Click on **OK** if you're sure; otherwise, click on **Cancel** or press the **Esc** key to stay in Prism.
- **From the command window:** Issue the **`quit`** command on the command line. (You aren't asked if you're sure you want to quit.)

If you have created subprocesses while in Prism (for example, a UNIX shell), Prism displays this message before exiting:



Choose **Yes** (the default) to leave Prism and terminate the subprocesses. Choose **No** to leave Prism without terminating the subprocesses. Choose **Cancel** to stay in Prism.

Copyright © 1991–1994 Thinking Machines Corporation