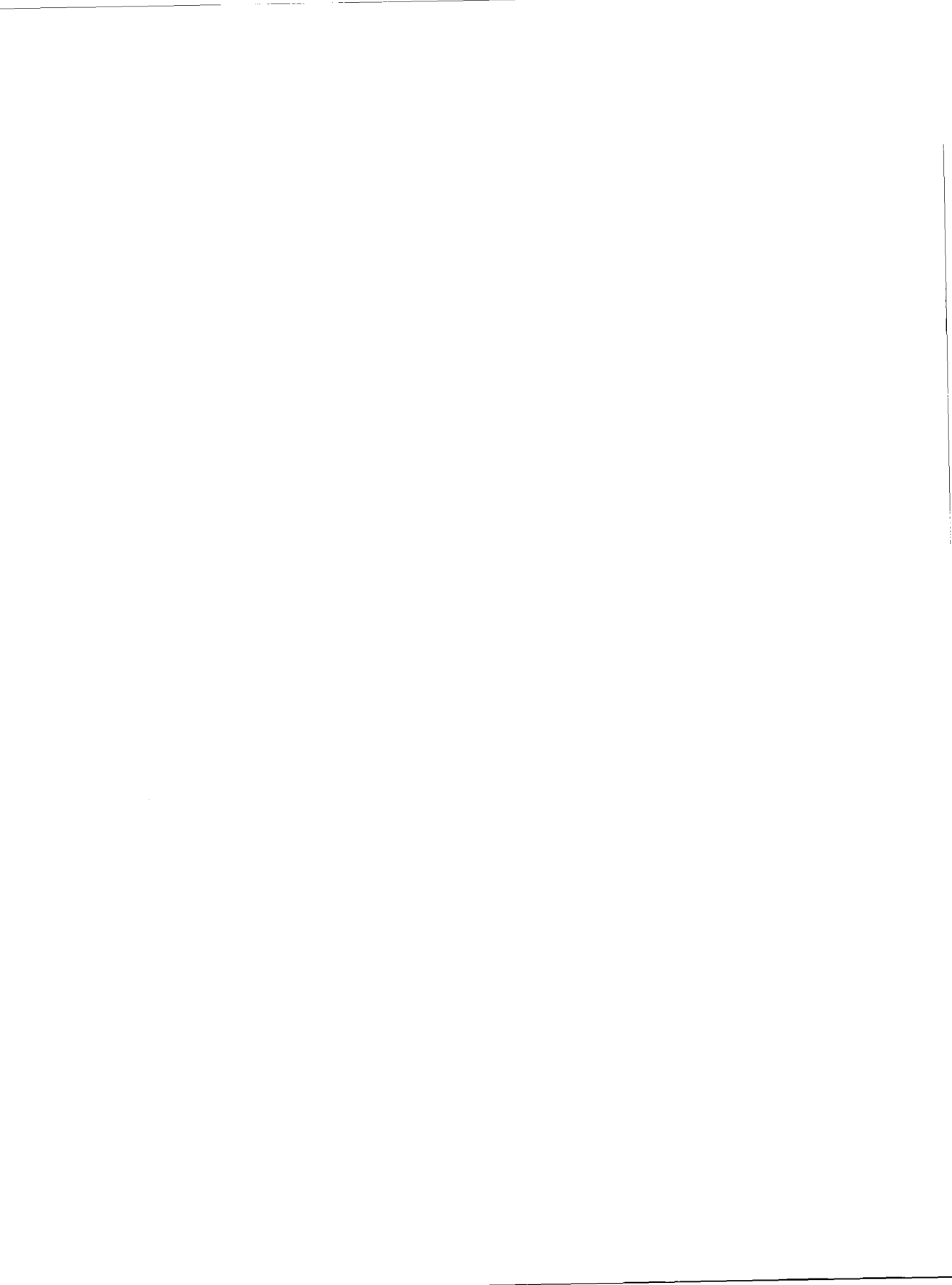


Exemplar
S-Class and
X-Class Servers

CXdb Quick Reference

Fourth Edition



Hewlett-Packard Company
Convex Division
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America

CXdb Quick Reference

Exemplar S-Class and X-Class Servers

B5639-90001

Fourth Edition

January 1997

Hewlett-Packard Company
Convex Division
Richardson, Texas
United States of America

CXdb Quick Reference

Exemplar S-Class and X-Class Servers

B5639-90001

© Copyright Hewlett-Packard Company 1997. All Rights Reserved. Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Notice

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.



This entire book is recyclable.

Printed in the United States of America

Contents

| | |
|--|-----------|
| 1 Using CXdb | 1 |
| Compiling your program for CXdb | 1 |
| Invoking CXdb | 2 |
| In GUI mode | 2 |
| In line mode (tty interface) | 2 |
| Using <code>cxdb</code> command options | 2 |
| Loading a program to debug | 3 |
| Invoking CXdb | 3 |
| Invoking CXdb with the name of an executable file | 3 |
| Invoking CXdb with the name of a core file | 3 |
| Invoking CXdb with the ID of a running process | 3 |
| Invoking CXdb with the name of an MPI application | 4 |
| Loading a program after invoking CXdb | 4 |
| User interface and windows | 5 |
| Creating CXdb windows (GUI mode) | 5 |
| CXdb windows and line mode equivalents | 6 |
| Using the Command window | 7 |
| Using the Source Code window | 9 |
| Working with multiple threads and processes | 10 |
| Setting the focus in GUI mode | 11 |
| Using the <code><focus-override></code> parameter | 12 |
| Working with breakpoints, tracepoints, and watchpoints | 13 |
| Controlling process execution | 14 |
| Stepping process execution | 15 |
| Obtaining a stack backtrace | 16 |
| Displaying and modifying program data | 17 |
| Using CXdb in line mode | 19 |
| Getting help online | 20 |
| Mouse and keyboard shortcuts | 21 |
| | |
| 2 Command syntax and descriptions | 23 |
| Parameters | 24 |
| Getting started | 27 |
| Starting CXdb — <code>cxdb</code> command | 27 |
| <code>quit</code> command | 27 |
| <code>help</code> command | 27 |
| Specifying an executable file, core file, or process to debug | 28 |

| | |
|---|----|
| Working with eventpoints | 29 |
| Setting breakpoints — break commands | 29 |
| Setting tracepoints — trace commands | 30 |
| Setting other types of eventpoints — event and watch commands | 31 |
| Displaying eventpoint information | 32 |
| Enabling, disabling, and setting ignore counts for eventpoints | 33 |
| Removing eventpoints | 33 |
| Eventpoint handler commands | 34 |
| Running your program | 35 |
| Process execution commands | 35 |
| Stepping commands | 37 |
| Viewing program source, data, and memory | 38 |
| Viewing source code | 38 |
| Viewing assembly-language code | 38 |
| Displaying and modifying data | 39 |
| Using the print command | 40 |
| Viewing and modifying memory | 41 |
| Displaying stack frame information | 42 |
| Displaying register contents | 43 |
| Searching memory | 44 |
| Searching source code | 44 |
| CXdb info commands | 45 |
| Debugging shared libraries | 49 |
| Logging and redirection | 50 |
| Logging input, output, and messages | 50 |
| Redirecting process I/O | 52 |
| CXdb redirection operators | 52 |
| For command output | 52 |
| For error and information message output | 52 |
| Configuring CXdb | 53 |
| Setting and clearing CXdb defaults | 53 |
| Setting and clearing global process defaults | 54 |
| Setting and clearing defaults for the current process | 56 |
| Working with aliases and macros | 57 |
| Path and directory commands | 58 |
| Working with signals | 58 |
| Working with threads | 59 |
| Controlling process and thread focus | 60 |
| Debugger variables | 61 |
| CXdb history commands | 61 |
| Miscellaneous commands | 62 |

Using CXdb

Compiling your program for CXdb

CXdb can debug Fortran, C, and C++ programs compiled on the Exemplar S2000, SPP1200 Series, and SPP1600 Series systems using the following compilers:

- Exemplar Fortran — `/opt/fortran/bin/f77`
- Exemplar C — `/opt/ansic/bin/C89`
- Exemplar C++ — `/opt/CC/bin/CC`
- HP Fortran 90 — `/opt/fortran90/bin/f90`

The Exemplar C, Fortran 77, and C++ compilers are based on standard Hewlett-Packard compilers, which have been modified to support the Exemplar programming model.

For your application to run properly under CXdb, you must use the Exemplar linker `ld`.

To compile an application for symbolic debugging with CXdb, use the `-g` compiler option. For example:

```
% f77 -g myprog.f
```

To compile a Fortran 90 application, you must also include the `-Wl, +tools` option. For example:

```
% f90 -g foo.f90 -Wl, +tools
```

NOTE: The `-g` option is incompatible with optimization levels above `+O0`.

If you have compiled your program *without* debugging information, you can still debug it with CXdb. However, in this case only global identifiers and function names can be referenced by their symbolic names. To access local variables and function arguments, you must specify their addresses rather than symbolic names. No source code correlation will be available.

Invoking CXdb

Use the `cxdb` command to invoke CXdb in either of two interfaces:

- **GUI mode**—Provides a window environment for X terminals that includes full use of a mouse to perform debugging functions in CXdb.
- **Line mode (tty interface)**—Allows you to use CXdb interactively without the graphical user interface on either windowless terminals (such as VT-100s) or X terminals.

In GUI mode

Assuming your `DISPLAY` environment variable is set correctly, invoke CXdb in GUI mode from the shell with the `cxdb` command. For example:

```
% /opt/cxdb/bin/cxdb a.out &
```

In the above example, `a.out` is the name of the executable file to debug. This is the most common way to invoke CXdb.

If you do not specify the name of an executable file on the shell command line, CXdb opens the Mode Selection dialog, where you can select whether you want to debug an executable file (the default), a running process, a core file, or execute the `mpirun` command.

In line mode (tty interface)

From the shell, invoke CXdb in line mode (tty interface) using the `-nw` (no windows) option of the `cxdb` command. For example:

```
% /opt/cxdb/bin/cxdb -nw a.out
```

In the above example, `a.out` is the name of the executable file to debug.

If you do not specify an executable file on the shell command line, you can use the CXdb `debug` command to select whether you want to debug an executable file, a running process, or a core file.

Using `cxdb` command options

Refer to "Starting CXdb — `cxdb` command," on page 27 for a list of the `cxdb` command options.

Loading a program to debug

With CXdb, you can debug:

- An executable file
- A core file
- A running process
- An MPI application

Normally, you specify which type of debugging you want to do when you invoke CXdb, as described in the next section.

Invoking CXdb

When you invoke CXdb, you can specify the type of debugging you want to do by using the appropriate options on the command line.

Invoking CXdb with the name of an executable file

To debug an executable file, invoke CXdb from the shell with the name of an executable. For example:

```
% /opt/cxdb/bin/cxdb <executable>
```

The above command invokes CXdb in GUI mode. CXdb loads the executable file to debug by executing the `debug exec` command on the specified file. CXdb assumes that the first file that is not preceded by an option flag is the executable file.

Invoking CXdb with the name of a core file

To invoke CXdb from the shell with the name of a core file to debug, use the `-c` option of the `cxdb` command. For example:

```
% /opt/cxdb/bin/cxdb -c <corefile> <executable>
```

The above command starts CXdb in GUI mode. CXdb loads the core file to debug by executing the `debug core` command on the specified core file.

Specifying the executable file is optional — this incorporates debugging information from the executable that generated the core file. If you do not specify an executable, you can still debug the core image using absolute addresses, but debugging information is severely limited.

Invoking CXdb with the ID of a running process

To invoke CXdb from the shell and attach to a running process to debug, use the `-a` option of the `cxdb` command.

For example:

```
% /opt/cxdb/bin/cxdb -a <process-id> <executable>
```

The above command starts CXdb in GUI mode. CXdb attaches to the process to debug by executing the `debug attach` command on the specified process ID.

Specifying the executable file is optional— this incorporates debugging information from the executable that generated the process. If you do not specify an executable, you can still debug the process image using absolute addresses, but debugging information is severely limited.

Invoking CXdb with the name of an MPI application

To invoke CXdb from the shell with the name of an MPI application to debug, use the `-mpi` option of the `cxdb` command. For example:

```
% /opt/cxdb/bin/cxdb -mpi <appFile>
```

The above command starts CXdb in GUI mode. CXdb invokes the `mpirun` command, which creates the MPI processes and attaches them to CXdb.

The `appFile` is the name of the file that specifies which executable to use, and how many processes or instances of that executable should be created. For more information about `mpirun` and how to specify the contents of the `appFile`, see the *HP MPI User's Guide*.

Loading a program after invoking CXdb

If you do not specify a file or process to debug when invoking CXdb, you must specify one later by one of the following methods:

- In GUI mode, use the Mode Selection dialog to select whether you want to debug an executable file, a core file, a running process, or an MPI application.
- In line mode, use the `debug` command to specify an executable file, a core file, or a running process to debug.

If you later want to specify a new process or file to debug, you can use the `debug` command in either line mode or GUI mode to specify a new executable file, core file, or running process to debug.

NOTE: You cannot use the `debug` command to specify an MPI application to debug. Use the `mpirun` command instead.

User interface and windows

There are two user interfaces for CXdb: GUI mode and line mode (tty interface).

This section explains how to create CXdb windows in GUI mode, then lists common debugging tasks and their associated CXdb windows, along with line mode command equivalents, where applicable.

Creating CXdb windows (GUI mode)

To create CXdb windows:

- **Command window** — Created automatically when you execute the `cxdb` command.
- **Source Code window** — By default, created automatically when you execute the `debug executable` or `debug attach` commands. Can also be created by selecting Source Code from the Windows menu in any CXdb window.
- **Process interface window** — Created automatically when you execute the `run` or `rerun` command.
- **Assembly Code window** — Select Assembly Code from the Windows menu in any CXdb window.
- **Memory Display window** — Select Memory Display from the Windows menu in any CXdb window.
- **Stack Trace window or register windows** — Select the appropriate item from the Windows menu in any CXdb window.
- **Help window** — Select the Window Overview, Using Help, or Release Notice item from the Help menu in any CXdb window, or execute the `help` command.
- **Edit window** — Execute the `edit` command.
- **Shell window** — Execute the `shell` command.

When a new window is created, it inherits the focus of the window from which it was created. That is, the new window displays the same processes and threads as the parent window. To change the focus, use the `set focus` command or the focus panel in the Command window.

CXdb windows and line mode equivalents

| If you want to... | Use this CXdb window... | Line mode (tty interface) command equivalent |
|--|--------------------------------|--|
| Enter commands; view command output and messages | Command | Invoke CXdb from the shell with the <code>-nw</code> option and enter commands at the (CXdb) prompt. |
| View source code | Source Code | <code>list</code> |
| View assembly-language code | Assembly Code | <code>disassemble</code> |
| View contents of process memory | Memory Display | <code>examine</code> |
| View process stack frames | Stack Trace | <code>backtrace</code> ; <code>info frame</code> ; <code>info args</code> ; <code>info locals</code> |
| View contents of processor status word register (PSW) | Processor Status Word | <code>info psw</code> |
| View contents of general registers | General Registers | <code>info registers</code> |
| View contents of space registers | Space Registers | <code>info space registers</code> |
| View contents of control registers | Control Registers | <code>info control registers</code> |
| View contents of floating point registers | Floating point Registers | <code>info floating point registers</code> |
| Provide interactive input to and output from the running process | Process interface window | Process input, output, and messages are sent to <code>stdin</code> , <code>stdout</code> , and <code>stderr</code> in line mode. |
| Execute shell commands | Shell | <code>shell</code> |

Using the Command window

The Command window (shown below) is the primary way to communicate with CXdb in GUI mode. It appears automatically when you start CXdb with the `cxdb` command (assuming your `DISPLAY` environment variable is set correctly and you have not invoked CXdb with the `-nw` option).

Define sets of threads that are visible in the current focus

Manipulate eventpoints

Exit CXdb

Execute CXdb info commands

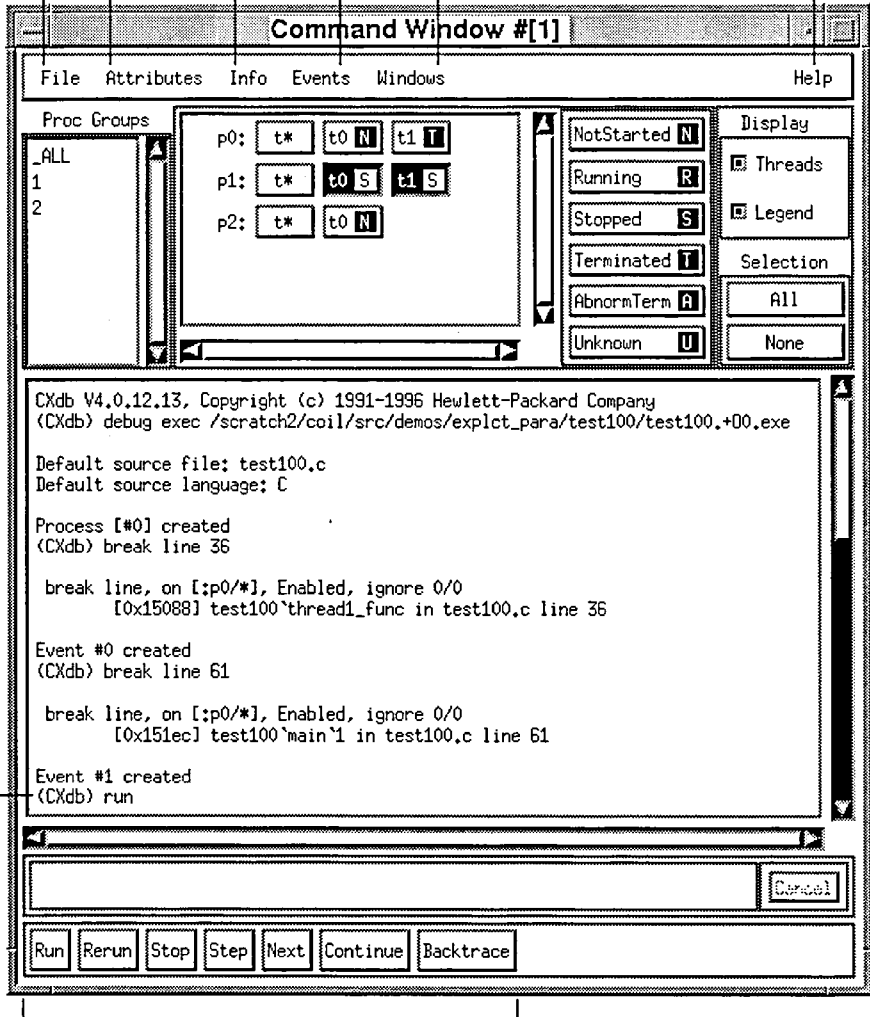
Open other CXdb windows

Access online help

Focus panel shows which processes and threads are affected by CXdb commands and window operations

Command output, error messages, and status information are displayed in Command window

Enter commands at CXdb prompt



Execute frequently used commands using buttons

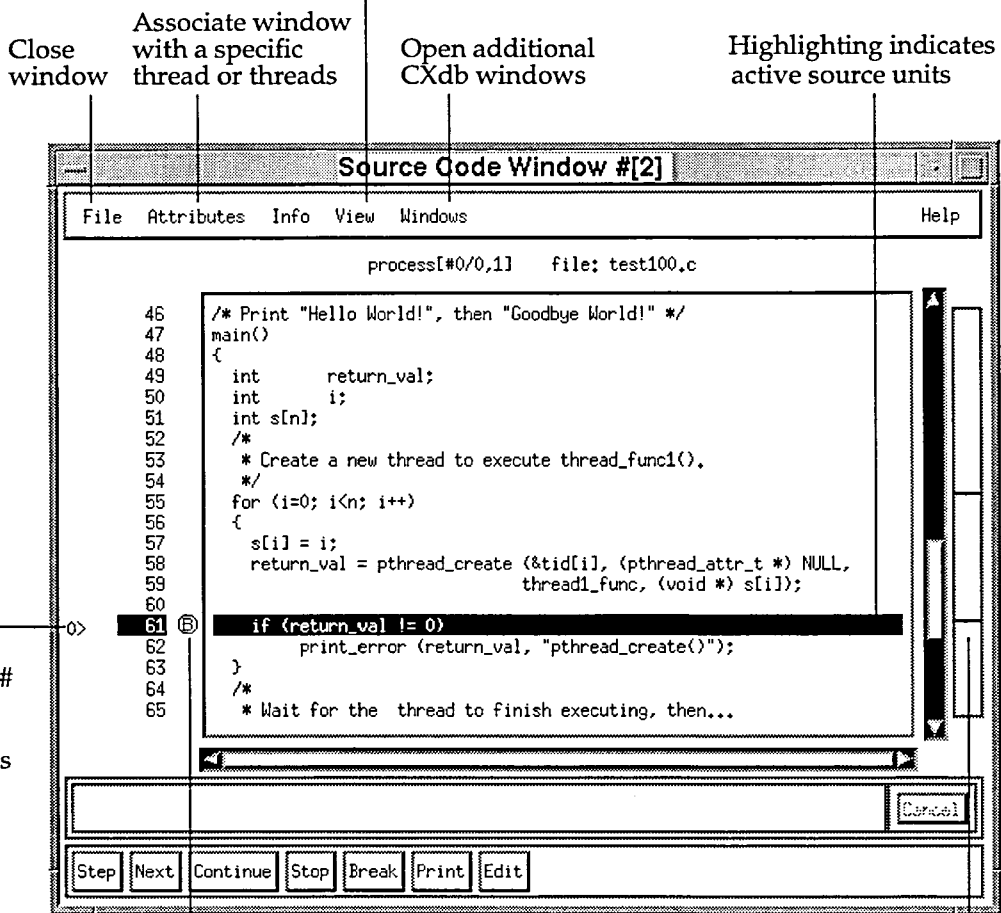
CXdb stores the last 100 commands you entered in a history buffer:

- Press **CTRL-p** to go to the previous line in the command history; use **CTRL-n** to go to the next line.
- Use the `info history` command to display the command history; use the `recall` command to retrieve and re-execute a command.

Using the Source Code window

The Source Code window (shown below) displays the source code files associated with the executable file. By default, a Source Code window is automatically created when you specify an executable file to debug. You can have multiple Source Code windows open during a debugging session.

Specify a new file or routine to display; search source code text



Click left mouse button on eventpoint markers in this column to display the Event Point dialog for enabling, disabling, and removing eventpoints

Navigation hints area shows location of thread activity in source code file

Working with multiple threads and processes

If you have an application that runs as multiple threads or multiple processes (such as a message passing application), you might want to apply CXdb commands and GUI operations to some of the threads and processes while leaving the others unaffected. You can define the focus for CXdb commands and GUI operations by using the commands listed in the following table.

| If you want to... | Use these commands... | Comments |
|--|---------------------------|--|
| Specify which threads and processes are affected by CXdb commands and GUI operations | <code>set focus</code> | The default focus is all threads of all processes. If you have changed the focus, you can return to the default state by using the <code>set focus all</code> command. |
| Display the current focus setting. | <code>info focus</code> | |
| Specify which threads are visible for processes in the current focus. | <code>set threads</code> | This command also affects which threads are visible when you use the <code><focus-override></code> parameter. |
| Display thread information. | <code>info threads</code> | This command lists the current state of each thread, ID numbers of active threads, and identifies the current thread. |
| Define a group of related processes and threads. | <code>add group</code> | CXdb automatically assigns an ID number to the group when you create it. Group 0 is the default, and it contains all threads of all processes. |
| Display current group definitions. | <code>info group</code> | |

For example:

```
(CXdb) set focus :p1,3 :t2,3,5
```

The above command sets the focus to threads 2, 3, and 5 of processes 1 and 3. Only these threads and processes are affected by subsequent CXdb commands and GUI operations.

```
(CXdb) info focus
:p1 :t2,3,5 :p2 :t2,3,5
```

The above command show the current focus setting.

```
(CXdb) set threads 3,6
```

The above command sets the focus to threads 3 and 6 for all processes in the current focus.

```
(CXdb) add group :p* :t0
group 1 created
```

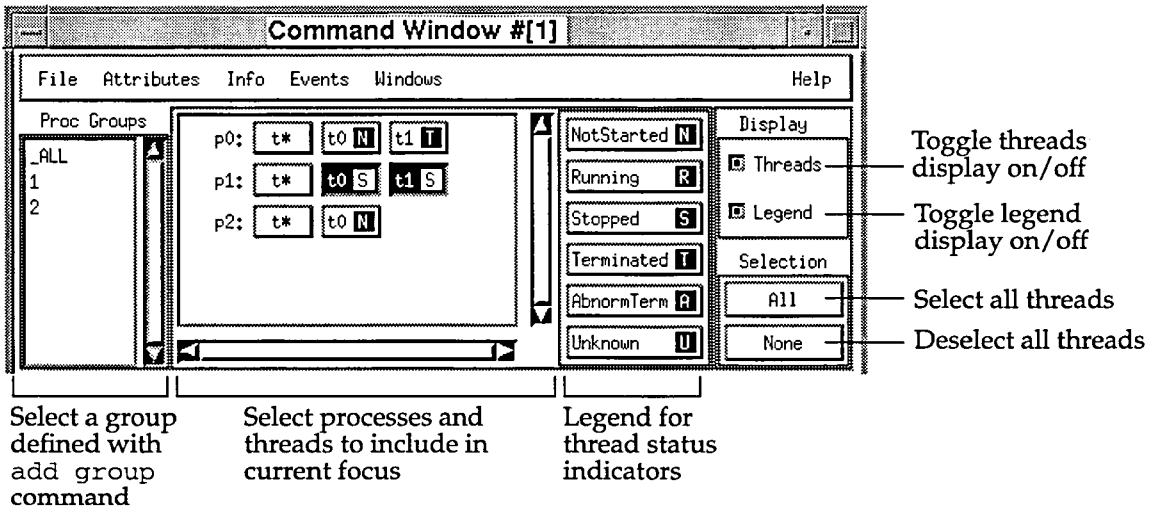
The above command creates a group that consists of thread 0 of each process. The group ID number is 1.

```
(CXdb) set focus :g1
```

The above command sets the focus to all the processes and threads in group 1.

Setting the focus in GUI mode

In GUI mode, you can display and set the current focus by using the focus panel in the Command window, as illustrated below.



Select a group defined with add group command

Select processes and threads to include in current focus

Legend for thread status indicators

Toggle threads display on/off

Toggle legend display on/off

Select all threads

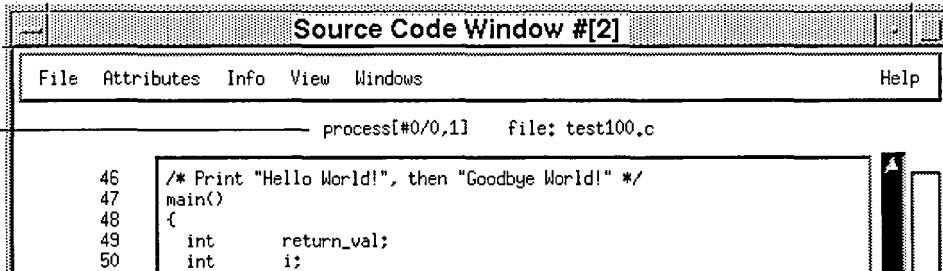
Deselect all threads

To apply a thread focus selection to another window, such as the Source Code window, perform the following drag-and-drop steps:

1. In the focus panel, place the mouse cursor over the individual process or thread toggle button you want to select. Do not select a group, a process label, or t*.

2. Hold down the middle mouse button, and drag the selected process/thread to the window where you want to apply it.
3. Release the process/thread icon in the part of the target window that lists process/thread activity, which is the bar just below the menu bar. (See figure below.)

Release
process/thread
in this area



Using the *<focus-override>* parameter

You can use the *<focus-override>* parameter with an individual CXdb command to override the current focus for that one command *only*. The *<focus-override>* parameter does not change the current focus for any concurrent or subsequent commands.

The *<focus-override>* parameter can include process lists, thread lists, and group lists. For example:

```
(CXdb) :p* :t0 step
```

The above command steps thread 0 of all processes.

Working with breakpoints, tracepoints, and watchpoints

The following table lists selected commands for working with breakpoints, tracepoints, and watchpoints. All of the commands listed are available in both GUI mode and line mode (tty interface).

| If you want to... | Use these command(s)... |
|---|--|
| Display information about breakpoints, tracepoints, watchpoints, or all eventpoints | info break info trace info watch info event |
| Set a breakpoint or tracepoint at a routine | break routine <routine-name> trace routine <routine-name> |
| Set a breakpoint or tracepoint at a line | break line <line-number> trace line <line-number> |
| Set a breakpoint or tracepoint at an instruction | break instruction <address> trace instruction <address> |
| Set a watchpoint to monitor an address range | watch <address> |
| Set a conditional breakpoint | event relation <expression> |
| Disable a breakpoint, tracepoint, or watchpoint | disable event <event-number> |
| Enable a breakpoint, tracepoint, or watchpoint | enable event <event-number> |
| Remove a breakpoint, tracepoint, or watchpoint | remove event <event-number> |

Refer to following sections for complete syntax and additional commands:

- "Setting breakpoints — break commands" on page 29
- "Setting tracepoints — trace commands" on page 30
- "Displaying eventpoint information" on page 32
- "Removing eventpoints" on page 33
- "Eventpoint handler commands" on page 34

Controlling process execution

Before using any of the CXdb commands for controlling process execution, make sure that you:

- Compile your program with the `-g` option.
- Load the executable file you want to debug.
- Set at least one breakpoint. (CXdb does not automatically set a breakpoint at the first line of your program. However, if you use the `step` or `next` command without setting a breakpoint, CXdb will set a breakpoint at the first instruction of the main program routine.)

The following table describes how to perform basic program execution tasks. Refer to "Stepping process execution" on page 15 for a description of stepping commands.

| If you want to... | Use this command or key... | Comments |
|--|--|--|
| Run your program | <code>run</code> | By default, automatically creates a process interface window in GUI mode. |
| Rerun your program using the previous argument list | <code>rerun</code> | |
| Continue execution of a stopped process | <code>continue</code> | |
| Continue process execution from within an eventpoint handler | <code>resume</code> | Can be used only within an eventpoint handler. This is the only process execution command that can be used within an eventpoint handler. |
| Stop process execution | Type CTRL-c in the Command window | Aborts the command that started process execution. |
| Terminate a running process | <code>kill process</code> | |

In GUI mode you can also control process execution by:

- Using the Run, Rerun, and Continue command buttons in the Command window
- Using the Continue command button in the Source Code or Assembly Code windows

Stepping process execution

The following table shows commands used for stepping process execution within CXdb.

The *<granularity>* (step size) you specify with stepping commands is optional, and can be:

- statement
- routine
- block

If no granularity is specified, CXdb uses the default stepping granularity.

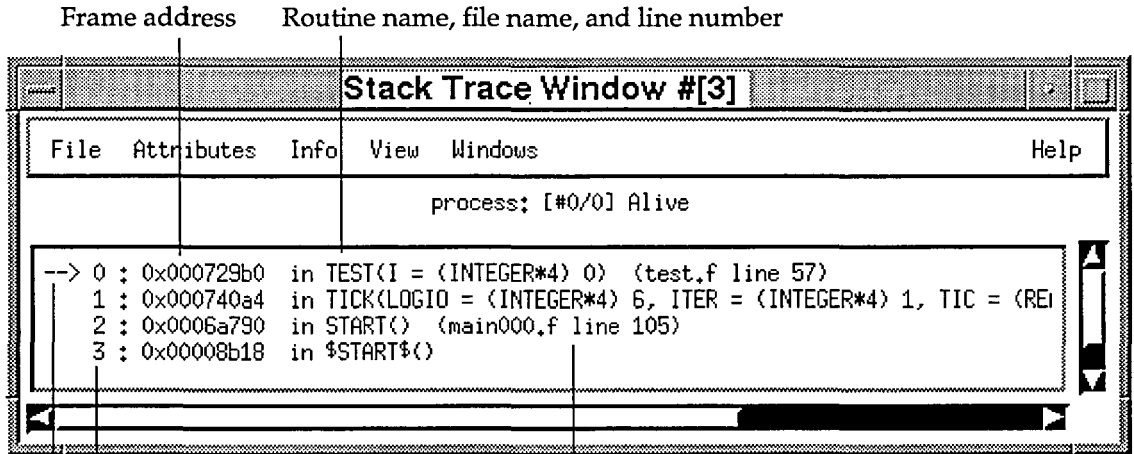
Refer to "Stepping commands" on page 37, for complete command syntax and additional commands.

| To perform this stepping task... | Use this command... |
|---|--|
| Set the default stepping granularity for CXdb. | <code>set step [<i><granularity></i>]</code> |
| Step process execution by 1 source unit of the specified granularity, stepping into subroutine calls | <code>step [<i><granularity></i>]</code> |
| Step the process by 1 instruction, stepping into subroutine calls | <code>step instruction</code> |
| Finish executing (step out of) a source unit of the specified granularity | <code>finish [<i><granularity></i>]</code> |
| Step process execution to the next source unit of specified granularity, stepping over subroutine calls | <code>next [<i><granularity></i>]</code> |
| Step to the next instruction, stepping over subroutine calls. | <code>next instruction</code> |

In GUI mode, you can also execute stepping commands by using the Step and Next command buttons in the Command, Source Code, or Assembly Code windows.

Obtaining a stack backtrace

You can obtain a stack backtrace in GUI mode by opening a Stack Trace window. You can do this by selecting the Stack Trace item from the Windows menu in any CXdb window. A sample Stack Trace window is shown below.



Frame number
Current frame indicator (-->)

Click anywhere on a line to display more detailed information about a specific stack frame

In GUI mode, you can also click on the Backtrace command button in the Command window.

At the command line (in either line mode or GUI mode), you can obtain a stack backtrace by executing the `backtrace` command at the (CXdb) prompt. For example:

(CXdb) **backtrace**

Process [#0/0]

```
cf> 0 : 0x80001658 in SUB1(MAXLEN = (INTEGER*4) 1000, A = (REAL*4(1:<TEMPO>)) ,  
B = (REAL*4(1:<TEMPO>, 1:<TEMPO>)) ) (para.f line 22)  
1 : 0x800014ce in PARA() (para.f line 11)  
2 : 0x80001d08 in _main(1, 0xffffcad8, 0xffffcae0)  
3 : 0x80001108 in ___ap$envret()
```

Displaying and modifying program data

This section describes commands for displaying or modifying the contents of the variables and memory segments associated with your program. CXdb provides commands that can access the variables either symbolically by identifier or directly by address.

Refer to the following sections for complete command syntax:

- "Using the print command" on page 40
- "Viewing and modifying memory" on page 41
- "Displaying register contents" on page 43
- "CXdb info commands" on page 45

| If you want to... | Use this command... | Comments |
|--|--|--|
| Display information about arguments. | <code>info args</code> | |
| Display the local variables of the current routine. | <code>info locals</code> | |
| Display all symbol names used in the program. | <code>info symbols</code> | |
| Display detailed information for a single variable, debugger variable, or language expression. | <code>info expression</code> | Shows the object type, storage location, size, data type, current value and <i>liveness</i> ranges (range of memory where the value of the variable is available for displaying) |
| Print a program variable or language expression. | <code>print <language-expression></code> | Use the <code>info</code> formatting command to display current settings for print options. |
| Print an array slice. | <code>print <array-name> <starting-subscript> [..<ending-subscript>]</code> | If you do not specify an <i>ending_subscript</i> , the default array slice is the single element specified by the <i>starting_subscript</i> . |

| If you want to... | Use this command... | Comments |
|--|--|--|
| Specify format for displaying units of memory with <code>print</code> command. | <pre>print/B — Binary output print/x — Hexadecimal output print/d — Signed decimal output print/e[<width>.<precision>] — Scientific notation print/f[<width>.<precision>] — Floating point notation print/i — Instruction format print/K — C++ class members</pre> | <p>Do not use a space between the word "print" and the display format specification.</p> <p>Not all print formats are available for all memory unit types.</p> |
| Set print options. | <pre>set printopts padding set printopts nopadding set printopts precision set printopts maxarray</pre> | <p>Force alignment with leading zeros (padding).</p> <p>Disable padding with leading zeros (default).</p> <p>Set precision for real (floating-point) numbers.</p> <p>Set maximum number of array elements to be printed at one time.</p> |
| Display contents of memory. | <pre>examine</pre> | <p>In GUI mode, you can also open a Memory Display window. Refer to the "examine" reference topic for formatting options.</p> |
| Display register contents for registers available on your system. | <pre>info registers info floating point registers info psw info space registers info control registers</pre> | <p>In GUI mode, you can open register windows for the registers available on your system.</p> |

Using CXdb in line mode

Line mode allows you to use CXdb interactively without the graphical user interface. Line mode is designed primarily for use on windowless terminals (for example, VT-100s), but you can also use it on X terminals.

At your shell prompt, enter the following command to invoke CXdb in line mode:

```
% cxdb -nw
```

In line mode, process output appears intermixed with CXdb output on your screen (stdout), unless you redirect the output. By default, the output is paged using the `less` utility.

NOTE: If your `PAGER` environment variable is set to anything other than `less`, paging of output will not work properly.

In line mode, echoing of input from command files and initialization files is disabled by default. To enable echoing of this input, use the `set echo` command.

CXdb line mode provides command line editing functions similar to UNIX command line editing. You can display the key bindings for these editing functions by entering `CTRL-?` on the CXdb command line.

In line mode, whenever the process is stopped or the stack frame is changed, CXdb displays 5 lines of source code surrounding the line of source code at the point of execution. For example, lines of source code context are displayed:

- When the process is stopped by a breakpoint or an eventpoint of type `spawn`, `join`, or `signal`
- When the process stops after executing commands that affect process execution, such as `step`, `next`, `finish`, or `goto`
- When you change the current stack frame by executing the `frame` command or the aliases `up` and `down`

Some CXdb windows and commands are not available in line mode. Refer to the section "CXdb windows and line mode equivalents" on page 6 for alternative or equivalent commands to use in line mode.

Getting help online

To access the CXdb Help system and bring up the CXdb Help window, select the Window Overview, CXdb Overview, Using Help, or Release Notice item from the Help menu in any CXdb window, or execute the `help` command. The CXdb Help window is shown below.

Go to hyperlinked lists of Help system contents, commands, parameters, and windows, or return to previous topic

Print help text or close window

Click on underlined text with left mouse button to view help information for that topic

Display Using Help topic or view product and version information for the Help system

The screenshot shows a window titled "CXdb Help #[31]". The window has a menu bar with "Window" and "Goto" on the left, and "Help" on the right. The main content area is titled "CXdb Online Help System" and contains the following text:

The CXdb online help system contains topics that are organized into the following categories:

- **Concepts** – Primary concepts involved in using CXdb.
- **Commands** – Descriptions, syntax rules, and examples of the CXdb commands.
- **Parameters** – Major parameters used with the CXdb commands.
- **Windows** – Descriptions of the CXdb windows and explanations of how to use them.
- **Tutorial** – An online tutorial that gives a brief introduction to the major features of CXdb. It includes examples that you can execute online.
- **Messages** – Detailed explanations of the informational and error messages issued by CXdb. (To display help on a particular message, enter the message number.)

At the bottom of the window, there is a navigation bar with the following elements:

- A "Contents" button.
- A "Go Back" button.
- A search input field with a "Titles" dropdown menu and a "Search" button.
- A "Search" button.
- Navigation buttons: "<< Browse" and "Browse >>".

Annotations with lines pointing to these elements are as follows:

- "Contents" button: Display help system contents page
- "Go Back" button: Display previous help topic
- "Titles" dropdown: Select search type (Titles or All Text)
- Search input field: Enter a character string here to search help topics
- "Search" button: Click here to activate search
- "<< Browse" button: Click to move forward (>>) or backward (<<) through current topic, one window at a time

Mouse and keyboard shortcuts

This section lists selected mouse and keyboard shortcuts for command-line editing, assuming that you have not modified the default button or key bindings.

| Key | Function |
|-------------------|--|
| CTRL-a | Beginning of line |
| CTRL-b | Back one character |
| META-b | Back one word |
| CTRL-d | Delete forward one character |
| META-d | Delete forward one word |
| CTRL-e | End of line |
| CTRL-f | Forward one character |
| META-f | Forward one word |
| CTRL-u | Beginning of line, delete line |
| CTRL-k | Delete to the end of the line |
| CTRL-v, PAGE UP | Scroll down one page (Command window only) |
| META-v, PAGE DOWN | Scroll up one page (Command window only) |
| TAB | Complete current command |
| CTRL-p | View previous entry in the command history |
| CTRL-n | View next entry in the command history |
| CTRL-l | Redraw screen |
| LEFT ARROW (←) | Left one character (Command window only) |
| RIGHT ARROW (→) | Right one character (Command window only) |
| CTRL-? | List all available key bindings (line mode only) |

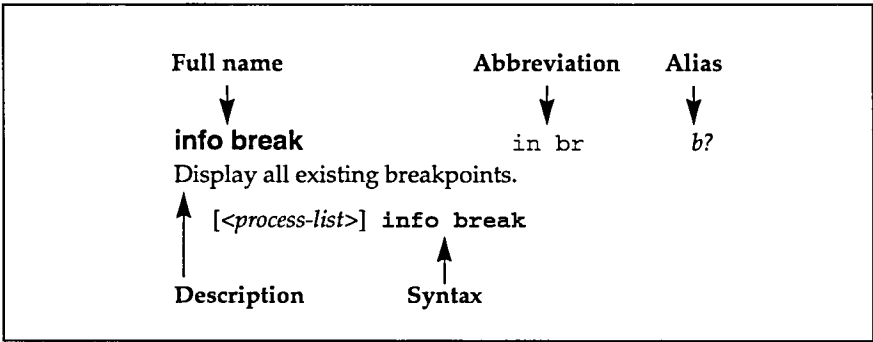
Command syntax and descriptions

2

This section lists the following information for each CXdb command:

- Full command name
- Shortest abbreviation for the command
- Default alias for the command
- Brief description of what the command does
- Syntax of the command

The information is presented in the following format:



This section also presents brief descriptions and syntax for some of the parameters used with CXdb commands.

Parameters

This section describes some of the major parameters used with CXdb commands.

<array-slice>

A subset of an array.

`<starting-subscript> [.. <ending-subscript>]`

<debugger-variable>

A variable defined and used in CXdb commands.

`[cxdbs$ | $]<variable-name>`

<directory-specifier>

A directory path name.

`<directory-specifier>`

<environment-variable>

A shell environment variable.

`<environment-variable>`

<even-handler>

A set of CXdb commands that executes when an eventpoint is triggered.

`{ <CXdb-command> ; [...] }`

<event-specifier>

An eventpoint identifier.

`{ <eventpoint-number> | <debugger-variable> | * }`

<eventtype-specifier>

An eventpoint type.

`{ break | trace | watch | exec | join | relation |
signal | spawn | * }`

<file-name>

The name of a file.

`[<directory-specifier> /] <file-name>`

<focus-override>

A list of processes, threads, or groups.

`{ <process-list> [<thread-list>] [, ...] | <group-list> }`

<frame-specifier>

A stack frame identifier.

[[+ | -]] <integer>

<granularity>

Source unit granularity.

{statement | block | routine}

<language-expression>

A valid expression in the source language.

<language-expression> [\;]

<line-specifier>

A source line identifier.

[<file-name>:] <integer>

<process-list>

A list of CXdb process numbers.

:p {<process-number>[.<process-number>][, ...] | *}

<redirection operator>

An operator that redirects CXdb command output and messages.

{> | >! | >> | >>! | >& | >&! | >>& | >>&!}

<regular-expression>

A character pattern for searches.

{<character-string> | <search-operator> [...]}

<signal-specifier>

A signal identifier.

{<signal-name> | <abbreviated-name> | <signal-number>}

<source-unit>

A source unit identifier.

[<file-name>:] <source-unit-number>

<string>

A character string.

<character-string>

<thread-list>

A list of thread numbers.

:t {<thread-number> [, ...] | *}

Starting CXdb — `cxdb` command

Use the following command to invoke CXdb. To access symbolic debugging information, you must first compile your program with the `-g` option:

cxdb cxdb

Invoke CXdb from the shell.

```
cxdb [-a <process-id>] [-csd ]
      [-D <directory-specifier>[, ...]]
      [-f <command-file>]
      [-mpi <appFile>] [-nw ] [-nx ]
      [-x <command-list>] [[-e ] <file-name>]
      [[-c ] <file-name>] [<X-Toolkit-options>]
```

| Parameter | Meaning |
|---|--|
| <code>-a <process-id></code> | Attaches CXdb to the process with the specified process ID. |
| <code>-c <file-name></code> | Specifies a core file to debug. |
| <code>-csd</code> | Invokes CXdb in line mode and incorporates aliases for <code>csd</code> debugger commands. |
| <code>-D <directory-specifier></code> | Specifies a directory to add to the default search path. |
| <code>-e <file-name></code> | Specifies an executable file to debug. |
| <code>-f <command-file></code> | Executes the specified CXdb command file. |
| <code>-mpi <appFile></code> | Specifies an MPI application to debug. |
| <code>-nw</code> | Invokes CXdb in line mode. |
| <code>-nx</code> | Prevents CXdb from executing initialization files. |
| <code>-x <command-list></code> | Specifies a list of CXdb commands to be executed at start-up. Separate multiple commands in the list with a semicolon (;). |
| <code><X-Toolkit-options></code> | Specifies an X Toolkit option. |

`quit` command

quit q

Exit from CXdb.

```
quit
```

`help` command

help h ?

Invoke the CXdb help system.

```
help [<string>]
```

The string you enter can be a CXdb command, CXdb error message identifier, or other text string.

Specifying an executable file, core file, or process to debug

debug core deb c *dbg*

Specify a core file or checkpoint file to debug.

debug core <core-file> [executable <executable-file>]

debug executable deb e *dbg*

Specify a new executable file to debug, or associate the specified executable with the existing CXdb process.

debug executable <executable-file> .

debug attach deb a *dbg*

Attach to a running process to debug.

debug attach <process-id> [executable <executable-file>]

detach det

Detach from a process.

[<process-list>] detach

Setting breakpoints — break commands

break instruction `bre i` *bi*

Set a breakpoint at an instruction.

```
[<process-list>] break instruction <language-expression>
[ {<event-handler>} ][<debugger-variable>]
```

break line `bre l` *bl*

Set a breakpoint at a source line.

```
[<process-list>] break line <line-specifier>
[ {<event-handler>} ][<debugger-variable>]
```

break routine `bre r` *br*

Set a breakpoint at the beginning of a routine.

```
[<process-list>] break routine <language-expression>
[ {<event-handler>} ][<debugger-variable>]
```

break source `bre s` *bs*

Set a breakpoint at a source unit.

```
[<process-list>] break source <source-unit>
[ {<event-handler>} ][<debugger-variable>]
```

Setting tracepoints — trace commands

trace instruction tra i ti

Set a tracepoint at an instruction.

```
[<process-list>] trace instruction <language-expression>  
[ {<event-handler>} ] [<debugger-variable>]
```

trace line tra l tl

Set a tracepoint at a source line.

```
[<process-list>] trace line <line-specifier>  
[ {<event-handler>} ] [<debugger-variable>]
```

trace routine tra r tr

Set a tracepoint at the beginning of a routine.

```
[<process-list>] trace routine <language-expression>  
[ {<event-handler>} ] [<debugger-variable>]
```

trace source tra s ts

Set a tracepoint at a source unit.

```
[<process-list>] trace source <source-unit>  
[ {<event-handler>} ] [<debugger-variable>]
```

Setting other types of eventpoints — event and watch commands

event exec eve e

Set an eventpoint to watch for an exec (2) system call by the process.

```
[<process-list>] event exec [ {<event-handler>} ] [<debugger-variable>]
```

event join eve j

Set an eventpoint to trap a thread joining.

```
[<process-list>] event exec [ {<event-handler>} ] [<debugger-variable>]
```

event relation eve rel

Set an eventpoint to watch for an expression to become true.

```
[<process-list>] [ event relation <language-expression>
  [ {<event-handler>} ] [<debugger-variable>]
```

event signal eve si

Set an eventpoint to catch a signal.

```
[<process-list>] event signal <signal-specifier> [ {<event-handler>} ]
  [<debugger-variable>]
```

event spawn eve sp

Set an eventpoint to trap the spawning of a thread.

```
[<process-list>] event spawn [ {<event-handler>} ] [<debugger-variable>]
```

watch w

Set a watchpoint to monitor an address range.

```
[<process-list>] watch <starting-address> [ { ..<ending address> | :<byte-count> } ]
  [ {<event-handler>} ] [<debugger-variable>]
```

Displaying eventpoint information

| | | |
|---|-----------|------------|
| info break | in br | <i>b?</i> |
| Display all existing breakpoints. [<process-list>] info break | | |
| info event | in event | <i>e?</i> |
| Display the specified eventpoints. info event [<event-specifier>][, ...] | | |
| info eventtype | in eventt | <i>et?</i> |
| Display all eventpoints of the specified type. [<process-list>] info eventtype {<eventtype-specifier> [, ...]} | | |
| info trace | in tr | <i>t?</i> |
| Display all tracepoints. [<process-list>] info trace | | |
| info watch | in w | <i>iw</i> |
| Display all watchpoints. [<process-list>] info watch | | |

Enabling, disabling, and setting ignore counts for eventpoints

enable event `ena event` *en*

Enable eventpoints.

```
enable event <event-specifier> [, ...]
```

enable eventtype `ena eventt`

Enable all eventpoints of the specified type.

```
[<process-list>] enable eventtype <eventtype-specifier> [, ...]
```

disable event `disab event` *dis*

Disable eventpoints.

```
disable event <event-specifier> [, ...]
```

disable eventtype `disab eventt`

Disable all eventpoints of the specified type.

```
[<process-list>] disable eventtype <eventtype_specifier> [, ...]
```

set ignore `se ig`

Set an ignore count for an eventpoint.

```
set ignore <ignore-count> <event-specifier> [, ...]
```

Removing eventpoints

remove event `rem event` *e-*

Delete the specified eventpoints.

```
remove event <event-specifier> [, ...]
```

remove eventtype `rem eventt` *et-*

Delete all eventpoints of the specified type.

```
[<process-list>] remove eventtype <eventtype-specifier> [, ...]
```

Eventpoint handler commands

clear default handler cl d h

Clear the default handler for all eventpoints.

```
clear default handler
```

clear handler cl h

Clear the handler for a specified eventpoint.

```
clear handler <event-specifier> [, ...]
```

clear typehandler cl t

Clear the handler for a specified type of eventpoint.

```
clear typehandler <eventtype-specifier> [, ...]
```

echo ec

Echo a character string.

```
echo[/n] <string> [...]
```

if if

Establish conditional execution of CXdb commands.

```
if (<relational-expression>) <command-set> [ else <command-set>]
```

resume res

Continue execution of the process from within an eventpoint handler.

```
resume
```

set default handler se de h

Set the default handler for eventpoints.

```
set default handler {<event-handler>}
```

set handler se h

Set the handler for a specified eventpoint.

```
set handler <event-specifier> [, ...] {<event-handler>}
```

set typehandler se ty

Define the default handler for all eventpoints of the specified type.

```
set typehandler <eventtype-specifier> [, ...] {<event-handler>}
```

Process execution commands

continue con c

Continue execution of the process.

[*<focus-override>*] continue [*<count>*] [&]

detach det

Detach from a process.

[*<process-list>*] detach

goto address go a

Branch to the specified address.

[*<focus-override>*] goto address *<language-expression>*

goto line go l

Branch to the specified source line.

[*<focus-override>*] goto line *<line-specifier>*

goto source go s

Branch to the specified source unit.

[*<focus-override>*] goto source *<source-unit>*

kill process k p k

Terminate a running process.

[*<process-list>*] kill process

rerun rer rr

Create and execute a new process, using the previous argument list.

[*<process-list>*] rerun [&]

resume res

Continue execution of the process from within an eventpoint handler.

resume

return ret

Return to the calling routine.

[*<focus-override>*] return *<language-expression>*

run

ru

r

Create and execute a new process.

[<process-list>] run [<argument-list>][&]

stop

sto

Stop execution of the process.

[<process-list>] stop

Stepping commands

The *<granularity>* (step size) you specify with CXdb stepping commands can be one of the following:

- statement
- block
- routine

finish fini

Finish executing (step out of) the specified source unit.

[<focus-override>] finish [*<granularity>*] [*&*]

next n

Step to the next source unit, ignoring subroutine calls.

[<focus-override>] next [*<granularity>*] [*<count>*] [*&*]

next instruction n i *ni, nexti*

Step to the next instruction, stepping over subroutine calls.

[<focus-override>] next instruction [*<count>*] [*&*]

set default step se de s

Set the default stepping granularity.

set default step *<granularity>*

set step se st

Set the stepping granularity.

[<process-list>] set step *<granularity>*

step ste s

Step to the next source unit.

[<focus-override>] step [*<granularity>*] [*<count>*] [*&*]

step instruction ste i *si, stepi*

Step to the next instruction.

[<focus-override>] step instruction [*<count>*] [*&*]

Viewing source code

list li

List lines of source code.

```
list [[<file-name>:]<starting-line>][<number-of-lines>]  
      [routine <language-expression>]
```

Viewing assembly-language code

disassemble disas

Display the assembly-language code.

```
[<focus-override>] disassemble  
  [<starting-address> [{ ..<ending-address> | :<instruction-count> }]]
```

Displaying and modifying data

evaluate

eva

Evaluate a language expression.

[<focus-override>] evaluate <language-expression>

info args

in ar

args

Display arguments of the current routine.

[<focus-override>] info args

info expression

in ex

describe, whatis

Display the characteristics of the specified language expression.

[<focus-override>] info expression <language-expression>

info line

in li

ili

Display the source units for a specified line.

[<process-list>] info line <line-specifier>

info locals

in lo

locals

Display the local variables of the current routine.

[<focus-override>] info locals

info sourceunit

in so

iso

Display the specified source unit.

[<process-list>] info sourceunit <source-unit>

info symbols

in sy

globals, symbols

Display program symbols.

[<process-list>] info symbols [<regular-expression>]

Using the `print` command

`print`

`pr`

p

Evaluate a language expression and print the result.

```
[<focus-override>] print [/ [n] <format>] <language-expression>
```

`n` Suppresses newline character at end of printed data

<format> options are as follows:

| | |
|--------------------------------------|-------------------------|
| <code>B</code> | Binary |
| <code>C</code> | Fortran complex |
| <code>L</code> | Fortran logical |
| <code>c</code> | ASCII character |
| <code>d</code> | Signed decimal |
| <code>e</code> [<width>.<precision>] | Scientific |
| <code>f</code> [<width>.<precision>] | Floating-point notation |
| <code>K</code> | C++ class members |
| <code>i</code> | Instruction |
| <code>o</code> | Octal |
| <code>s</code> | String |
| <code>u</code> | Unsigned decimal |
| <code>x</code> | Hexadecimal |

`info formatting`

`in fo`

ifo

Display settings of print options for memory display formats.

```
[<process-list>] info formatting
```

`set printopts maxarray`

`se pr m`

Set the maximum number of array elements to print.

```
set printopts maxarray <number-of-elements>
```

`set printopts nopadding`

`se pr n`

Disable padding with leading zeros when printing.

```
set printopts nopadding
```

`set printopts padding`

`se pr pa`

Enable padding with leading zeros when printing.

```
set printopts padding
```

`set printopts precision`

`se pr pr`

Set the precision used to print floating point numbers.

```
set printopts precision <width>.<precision>
```

Viewing and modifying memory

copy cop
Copy a memory region.

```
[<focus-override>] copy <source-address>
  [{ ..<ending-address> | :<byte-count> }] \; <destination-address>
```

examine exa x
Display a region of memory.

```
[<focus-override>] examine [ / {<memory-unit> <format> <fpmode>}]
  <starting-address> [{ ..<ending-address> | :<unit-count>}]
```

evaluate eva
Evaluate a language expression.

```
[<focus-override>] evaluate <language-expression>
```

fill fil
Fill a region of memory with the value of an expression.

```
[<focus-override>] fill [ / <memory-unit>] <starting-address>
  [{ ..<ending-address> | :<unit-count> }] \; <language-expression>
```

get ge
Restore the contents of memory regions from a binary file.

```
[<focus-override>] get <file-name>
  [<starting-address> [{ ..<ending-address> | :<byte-count>}]] [\; ...]
```

info formatting in fo ifo
Display the settings for memory display formats.

```
[<process-list>] info formatting
```

put pu
Save the contents of memory to a file for later retrieval.

```
[<focus-override>] put <file-name> <starting-address>
  [{ ..<ending-address> | :<byte-count>}] [\; ...]
```

set format se fo
Set the formats for displaying memory.

```
[<focus-override>] set format <memory-unit> <format>
```

set memory

se m

Set the unit size for displaying memory.

`[<focus-override>] set memory <memory-unit>`

Displaying stack frame information**backtrace**

ba

bt

Display the frames of the call stack.

`[<focus-override>] backtrace [<frame-count>]`**frame**

fr

f

Change the current stack frame.

`[<focus-override>] frame <frame-specifier>`**info frame**

in fr

ifr

Display a stack frame.

`[<focus-override>] info frame [<frame-specifier>]`**info frame at**

in fr a

ifra

Display the stack frame at the specified address.

`[<focus-override>] info frame at <language-expression>`

Displaying register contents

info control registers `in cor` `icor`

Display the control registers.

`[<focus-override>] info control registers`

info floating point registers `in fl pr` `iflpr`

Display the floating-point registers.

`[<focus-override>] info floating point registers`

info psw `in ps` `ips`

Display the processor status word (PSW) register.

`[<focus-override>] info psw`

info registers `in r` `ir`

Display the contents of general registers.

`[<focus-override>] info registers`

info space registers `in sp r` `ispr`

Display the space registers.

`[<focus-override>] info space registers`

Searching memory

find memory backward `find m b` *fmb*

Find the first occurrence of a byte pattern within a memory region.

```
[<focus-override>] find memory backward [ / <memory-unit> ]  
  <byte-pattern> <lowest-address> { .. <highest-address> | : <byte-count> }
```

find memory forward `find m f` *fmf*

Find the first occurrence of a byte pattern within a memory region.

```
[<focus-override>] find memory forward [ / <memory-unit> ]  
  <byte-pattern> <starting-address> { .. <ending-address> | : <byte-count> }
```

Searching source code

find source `find s` *fs*

Search for all occurrences of a text string in a source file.

```
find source <string> [<filename>]
```

CXdb info commands

| | | |
|---|----------|-------------|
| info alias | in al | <i>ial</i> |
| Display aliases. | | |
| info alias [<regular-expression>] | | |
| info args | in ar | <i>args</i> |
| Display arguments of the current routine. | | |
| [<focus-override>] info args | | |
| info break | in br | <i>b?</i> |
| Display all existing breakpoints. | | |
| [<process-list>] info break | | |
| info control registers | in co r | <i>icor</i> |
| Display the control registers. | | |
| [<focus-override>] info control registers | | |
| info cxdb | in cx | <i>icx</i> |
| Display the status of the current CXdb session. | | |
| info cxdb | | |
| info default environment | in d e | <i>ide</i> |
| Display all default environment variables. | | |
| info default environment [<regular-expression>] | | |
| info environment | in en | <i>env?</i> |
| Display all process environment variables. | | |
| [<process-list>] info environment [<regular-expression>] | | |
| info errno | in er | <i>ier</i> |
| Display the system error message received by the process. | | |
| [<focus-override>] info errno | | |
| info event | in event | <i>e?</i> |
| Display the specified eventpoints. | | |
| info event [<event-specifier>][, ...] | | |

| | | |
|---|-----------|------------------|
| info eventtype | in eventt | et? |
| Display all eventpoints of the specified type. | | |
| [<process-list>] info eventtype <eventtype-specifier> [, ...] | | |
| info execmode | in ex | |
| Display the current execution mode. | | |
| info execmode | | |
| info expression | in ex | describe, whatis |
| Display the characteristics of the specified language expression. | | |
| [<focus-override>] info expression <language-expression> | | |
| info floating point registers | in fl p r | i fl p r |
| Display the floating-point registers. | | |
| [<focus-override>] info floating point registers | | |
| info focus | in foc | i foc |
| Display the current CXdb focus. | | |
| info focus | | |
| info formatting | in fo | i fo |
| Display the settings for memory display formats. | | |
| [<process-list>] info formatting | | |
| info frame | in fr | i fr |
| Display a stack frame. | | |
| [<focus-override>] info frame [<frame-specifier>] | | |
| info frame at | in fr a | i fr a |
| Display the stack frame at the specified address. | | |
| [<focus-override>] info frame at <language-expression> | | |
| info group | in g | i g |
| Display currently defined groups. | | |
| info group [<group-list>] | | |
| info history | in h | i h |
| Display the CXdb command history. | | |
| info history [<command-count>] | | |

| | | |
|---|-------|---------------|
| info line | in li | <i>li</i> |
| Display the source units for a specified line. [<process-list>] info line <line-specifier> | | |
| info locals | in lo | <i>locals</i> |
| Display the local variables of the current routine. [<focus-override>] info locals | | |
| info macro | in m | <i>m</i> |
| Display macros. info macro [<regular-expression>] | | |
| info objectmap | in o | <i>o</i> |
| Display the object map. [<process-list>] info objectmap | | |
| info path | in pa | <i>p+</i> |
| Display the directories in the search path. [<process-list>] info path | | |
| info process | in pr | <i>p?</i> |
| Display the status of the process. [<process-list>] info process | | |
| info psw | in ps | <i>ps</i> |
| Display the processor status word (PSW) register. [<focus-override>] info psw | | |
| info registers | in r | <i>r</i> |
| Display the contents of the general registers. [<focus-override>] info registers | | |
| info scope | in sc | <i>where</i> |
| Display the current scope path. [<focus-override>] info scope | | |
| info shlib | in sh | <i>sh</i> |
| Display the current list of shared libraries specified for debugging. info shlib | | |

info signal in si *i si*
Display signal names, numbers, and settings for signal actions.
`[<process-list>] info signal [<signal-specifier>][, ...]`

info sourceunit in so *i so*
Display the specified source unit.
`[<process-list>] info sourceunit <source-unit>`

info space registers in sp r *i sp r*
Display the space registers.
`[<focus-override>] info space registers`

info stack in st *i st*
Display information about the process stack.
`[<focus-override>] info stack`

info symbols in sy *globals, symbols*
Display program symbols.
`[<process-list>] info symbols [<regular-expression>]`

info threads in th *i th*
Display threads of the current process.
`[<process-list>] info threads`

info trace in tr *t?*
Display all tracepoints.
`[<process-list>] info trace`

info type in ty *i ty*
Display information about type definitions for a single thread of a single process in a C program.
`[:p<process-number>][:t<thread-number>] info type [<regular-expression>]`

info watch in w *i w*
Display all watchpoints.
`[<process-list>] info watch`

Debugging shared libraries

The following commands for debugging shared libraries are enabled by default. If you do not want to debug shared libraries, you can invoke `cxdb` with the `-noshl` option to disable shared library support.

add shliblist `ad s`

Add library names to the list of shared libraries specified for debugging.

```
add shliblist <library-name> [, ...]
```

clear shliblist `cl s`

Remove all library names from the list of shared libraries specified for debugging.

```
clear shliblist
```

info shlib `in sh` `i sh`

Display the current list of shared libraries specified for debugging.

```
info shlib
```

remove shliblist `rem sh`

Delete library names from the list of shared libraries specified for debugging.

```
remove shliblist <library-name> [, ...]
```

set shliblist `se shl`

Clear and respecify the list of shared libraries to debug.

```
set shliblist [<library-name> [, ...] | *]
```

Logging input, output, and messages

You can log the following types of information to files:

- CXdb error and informational messages.
- User entries on the CXdb command line.
- Normal output generated by CXdb in response to commands.

To append to an existing log file, use redirection operators.

add cmderr ad cmde

Add file names to the list of files that log CXdb messages.

```
add cmderr <filename> [, ...]
```

add cmdlog ad cmdl

Add file names to the list of files that log CXdb input.

```
add cmdlog <filename> [, ...]
```

add cmdout ad cmdo

Add file names to the list of files that log CXdb command output.

```
add cmdout <filename> [, ...]
```

clear logging cl l

Disable logging of CXdb input to files.

```
clear logging
```

clear noclobber cl n

Allow overwriting of existing files used for logging or redirection operations.

```
clear noclobber
```

remove cmderr rem cmde

Delete file names from the list of files that log CXdb messages.

```
remove cmderr <filename> [, ...]
```

remove cmdlog rem cmdl

Delete file names from the list of files that log CXdb input.

```
remove cmdlog <filename> [, ...]
```

remove cmdout rem cmdo

Delete file names from the list of files that log CXdb command output.

```
remove cmdout <filename> [, ...]
```

set cmderr se cmde

Clear and replace the list of files that log CXdb messages.

```
set cmderr <filename> [, ...]
```

set cmdlog se cmdl

Clear and replace the list of files that log CXdb input.

```
set cmdlog <filename> [, ...]
```

set cmdout se cmde

Clear and replace the list of files that log CXdb output.

```
set cmdout <filename> [, ...]
```

set logging se l

Enable logging of CXdb input to files.

```
set logging
```

set noclobber se n

Enable the noclobber option to prevent overwriting of existing files used for logging or redirection.

```
set noclobber
```

Redirecting process I/O

To redirect process I/O, use a shell redirection operator with the `CXdb run` command. For example, the command

```
(CXdb) run < prog_input > prog_output
```

starts execution of your program, with input from the file `prog_input`. Output from the program is directed to the file `prog_output`.

CXdb redirection operators

For command output

| Operator | Action |
|----------|--|
| > | Overwrite existing command output file, unless <code>noclobber</code> is set. |
| >> | Append new output to an existing file. An error occurs if the specified file does not exist. |
| >! | Overwrite existing output file, regardless of overwrite protection. |
| >>! | Append new output to an existing file. Create the file if it does not exist. |

For error and information message output

| Operator | Action |
|----------|--|
| >& | Overwrite existing message output file, unless <code>noclobber</code> is set. |
| >>& | Append new message output to an existing file. An error occurs if the specified file does not exist. |
| >&! | Overwrite existing message output file, regardless of overwrite protection. |
| >>&! | Append new message output to an existing file. Create the file if it does not exist. |

Setting and clearing CXdb defaults

clear echo

cl ec

Disable echoing of input from initialization files and command files.

clear echo

clear noclobber

cl n

Allow overwriting of existing files used for logging or redirection operations.

clear noclobber

set echo

se ec

Enable echoing of input from command files and initialization files.

set echo

set evalopts iprecision

se ev i

Set the size of integer constants for CXdb.

set evalopts iprecision {4 | 8}

set evalopts rprecision

se ev r

Set the size of real numbers for CXdb.

set evalopts rprecision {4 | 8}

set format

se fo

Set the formats for displaying memory.

[<focus-override>] set format <memory-unit> <format>

set memory

se m

Set the unit size for displaying memory.

[<focus-override>] set memory <memory-unit>

set noclobber

se n

Enable the noclobber option to prevent overwriting of existing files used for logging or redirection.

set noclobber

set printopts maxarray

se pr m

Set the maximum number of array elements to print.

set printopts maxarray <number-of-elements>

set printopts nopadding se pr n

Disable padding with leading zeros when printing.

```
set printopts nopadding
```

set printopts padding se pr pa

Enable padding with leading zeros when printing.

```
set printopts padding
```

set printopts precision se pr pr

Set the precision used to print floating point numbers.

```
set printopts precision <width>.<precision>
```

set shell se sh

Set the type of shell invoked from within CXdb.

```
set shell {sh | csh | tcsh | ksh}
```

set typehandler se t

Define the default handler for all eventpoints of the specified type.

```
set typehandler <eventtype-specifier> [, ...] {<event-handler>}
```

Setting and clearing global process defaults

These commands affect any new process that is created by CXdb *after* the default is changed.

add default environment ad d e *denv+*

Add or modify environment variables in the default environment.

```
add default environment  
  <environment-variable> = <string> [, ...]
```

add default path ad d p *dp+*

Add directories to the default search path.

```
add default path <directory-specifier> [, ...]
```

clear default environment cl d e

Remove all environment variables from the default environment.

```
clear default environment
```

clear default handler cl d h

Clear the default handler for all eventpoints.

```
clear default handler
```

- remove default environment** `rem d e` *denv-*
Delete environment variables from the default environment.
`remove default environment <environment-variable> [, ...]`
- remove default path** `rem d p` *dp-*
Delete directories from the default search path.
`remove default path <directory-specifier> [, ...]`
- set default environment** `se de e` *denv=*
Clear and redefine the environment variables for the default environment.
`set default environment <environment-variable> = <string> [, ...]`
- set default format** `se de fo`
Set the default formats for displaying memory.
`set default format <memory-unit> <format>`
- set default handler** `se de h`
Set the default handler for eventpoints.
`set default handler {<event-handler>}`
- set default memory** `se de m`
Set the default unit size for displaying memory.
`set default memory <memory-unit>`
- set default path** `se de pa` *dp=*
Set the default search path.
`set default path <directory-specifier> [, ...]`
- set default pshell** `se de ps`
Set the default process shell.
`set default pshell {sh | csh}`
- set default step** `se de s`
Set the default stepping granularity.
`set default step <granularity>`

Setting and clearing defaults for the current process

These commands affect default settings for the *current* process only.

add environment ad e env+

Add or modify environment variables in the process environment.

[<process-list>] add environment <environment-variable> = <string> [, ...]

clear environment cl en

Remove all environment variables from the process environment.

[<process-list>] clear environment

clear handler cl h

Clear the handler for a specified eventpoint.

clear handler <event-specifier> [, ...]

clear typehandler cl t

Clear the handler for a specified type of eventpoint.

clear typehandler <eventtype-specifier> [, ...]

remove environment rem en env-

Delete environment variables from the process environment.

[<process-list>] remove environment <environment-variable> [, ...]

remove path rem p p-

Delete directories from the process search path.

[<process-list>] remove path <directory-specifier> [, ...]

set environment se en env=

Clear and redefine the environment variables for the process environment.

[<process-list>] set environment <environment-variable>=<string> [, ...]

set path se pa p=

Set the search path for the process.

[<process-list>] set path <directory-specifier> [, ...]

Working with aliases and macros

alias al

Define an alias.

```
alias <alias-name> <string>
```

csd csd

Enable compatibility with `csd` debugger commands.

```
csd
```

gdb gdb

Enable compatibility with `gdb` debugger commands.

```
gdb
```

info alias in al *i al*

Display aliases.

```
info alias [<regular-expression>]
```

info macro in m *i m*

Display macros.

```
info macro [<regular-expression>]
```

macro m

Define a macro.

```
macro <name> [( <parameter>[:<default>][, ...] ) ] <string>
```

remove alias rem a

Delete an alias.

```
remove alias <alias-name>
```

remove macro rem m

Delete a macro.

```
remove macro <macro-name>
```

Path and directory commands

add path ad p p+

Add directories to the search path.

```
[<process-list>] add path <directory-specifier> [, ...]
```

cd cd

Change the console working directory.

```
cd <directory-specifier>
```

info path in pa p+

Display the directories in the search path.

```
[<process-list>] info path
```

pwd pw

Display the name of the console working directory.

```
pwd
```

Working with signals

event signal eve si

Set an eventpoint to catch a signal.

```
[<process-list>] event signal <signal-specifier>  
[ {<event-handler>} ] [<debugger-variable>]
```

info signal in si i si

Display signal names, numbers, and settings for signal actions.

```
[<process-list>] info signal [<signal-specifier>] [, ...]
```

set signal se si

Set the actions for the specified signal.

```
[<process-list>] set signal <signal-specifier>  
[[stop | nostop],] [[pass | nopass],] [[print | noprint]]
```

signal process sig p

Send a signal to the process and continue process execution.

```
[<process-list>] signal process <signal-specifier> [ & ]
```

Working with threads

clear idlethreads `cl i`

Disable display of information about idle threads.

```
clear idlethreads
```

event join `eve j`

Set an eventpoint to trap a thread joining.

```
[<process-list>] event join [ {<event-handler> } ][<debugger-variable>]
```

event spawn `eve sp`

Set an eventpoint to trap the spawning of a thread.

```
[<process-list>] event spawn [ {<event-handler> } ][<debugger-variable>]
```

info threads `in th` `i th`

Display threads of the current process.

```
[<process-list>] info threads
```

set idlethreads `se i`

Enable display of information about idle threads.

```
set idlethreads
```

set threads `se th`

Specify a thread or set of threads to pass to CXdb commands that perform actions on or display information for specific threads.

```
set threads [<thread-number> [, ...]]
```

Controlling process and thread focus

set focus

se foc

Specify the processes and threads to which CXdb commands and operations are applied.

```
set focus {<process-list> [<thread-list>][, ...] | <group-list> | all}
```

info focus

in foc

i foc

Display the current focus setting.

```
info threads
```

add group

ad g

Create a group of processes and threads to use in setting the focus.

```
set focus <process-list> [<thread-list>][, ...]
```

info group

in g

ig

Display currently defined groups.

```
info group [<group-list>]
```

set threads

se th

Specify a thread or set of threads to pass to CXdb commands that perform actions on or display information for specific threads.

```
set threads [<thread-number> [, ...]]
```

info threads

in th

ith

Display threads of the current process.

```
[<process-list>] info threads
```

Debugger variables

A debugger variable can store

- The results of a language expression
- The contents of a register
- A signal number
- A CXdb object number, such as an eventpoint number or window number

CXdb also has predefined debugger variables for registers and other values.

Refer to the "debugger variables" reference page or online help topic for information and examples on creating and referencing debugger variables.

remove variable `rem v`

Delete a debugger variable.

```
remove variable <debugger-variable>
```

CXdb history commands

info history `in h` *ih*

Display the CXdb command history.

```
info history [<command-count>]
```

recall `rec` *!*

Reexecute a previous command.

```
recall [ ? ]<string>
```

Miscellaneous commands

echo ec

Echo a character string.

```
echo[/n] <string> [...]
```

edit ed

In GUI mode, open an editor window and edit a file.

```
edit [<file-name>]
```

if if

Establish conditional execution of CXdb commands.

```
if (<relational-expression>) <command-set> [else <command-set>]
```

shell sh

Invoke a shell.

```
shell [/<shell-specifier>] [<shell-commands>]
```

source sou

Execute a CXdb command file.

```
source <file-name>
```

Index

A

add cmderr command 50
add cmdlog command 50
add cmdout command 50
add default environment command 54
add default path command 54
add environment command 56
add group command 10, 60
add path command 58
add shliblist command 49
alias command 57
aliases, commands for working with 57
arguments, displaying 17
arguments, viewing for current routine (info args command) 39
Assembly Code window, creating 5
assembly-language code, commands for viewing 38

B

backtrace 16
backtrace command 16, 42
break instruction command 29
break line command 29
break routine command 29
break source command 29
breakpoints
 commands, listed and described 13
 deleting (remove event command) 33
 disabling (disable event command) 33
 displaying information about 32
 enabling (enable event command) 33
 setting, commands for 29

C

cd command 58
clear default environment command 54
clear default handler command 34, 54
clear echo command 53
clear environment command 56
clear handler command 34, 56
clear idlethreads command 59
clear logging command 50
clear noclobber command 50, 53
clear shliblist command 49

clear typehandler command 34, 56
cmderr 50
cmdlog 50
cmdout 50
command line editing 19
Command window
 creating 5
 description and *illus* 7
command-line editing, key bindings 21
commands
 aliases 57
 breakpoints, setting 29
 descriptions 27
 directory 58
 displaying data 39
 eventpoints
 disabling 33
 displaying information about 32
 enabling 33
 handlers 34
 removing 33
 setting 31
 explanation of syntax for 23
 focus 60
 handlers for eventpoints 34
 help 27
 history 61
 info 45
 macros 57
 miscellaneous 62
 modifying data 39
 path 58
 printing data 40
 process execution 35
 quitting CXdb 27
 registers, viewing 43
 searching memory 44
 searching source code 44
 signals 58
 specifying a file to debug 28
 specifying a process to debug 28
 stack frame information 42
 starting CXdb 27
 stepping 37
 syntax 27
 threads 59
 tracepoints, setting 30
 viewing assembly-language code 38
 viewing source code 38
compiling for CXdb 1

continue command 14, 35
copy command 41
core files, debugging 3
csd command 57
cxdb command
 described 27
 shell command-line options 2, 27

D

data
 commands for displaying 17
 displaying 17, 39
 modifying 17, 39
debug attach command 28
debug command 4
debug core command 28
debug executable command 28
debugger variables 61
defaults
 CXdb, commands for setting and clearing 53
 process, commands affecting all new processes 54
 process, commands affecting existing processes 56
detach command 28, 35
directory commands 58
disable event command 33
disable eventtype command 33
disassembly command 38

E

echo command 34, 62
edit command 62
Edit window, creating 5
editing the command line 19
enable event command 33
enable eventtype command 33
evaluate command 39, 41
event exec command 31
event join command 31, 59
event relation command 31
event signal command 31, 58
event spawn command 31, 59
eventpoints
 disabling 33
 displaying information about 32
 enabling 33
 eventpoint handler commands 34
 ignore counts, setting 33
 removing 33
 setting, commands for 31
examine command 41
executable files, loading for debugging 3
exiting CXdb 27

expressions
 evaluating (evaluate command) 39
 printing 40

F

fill command 41
find memory backward command 44
find memory forward command 44
find source command 44
finish command 15, 37
focus 60
frame command 42

G

gdb command 57
get command 41
goto address command 35
goto line command 35
goto source command 35
granularity
 defined 15
 for stepping commands 37
GUI mode 2

H

handlers, eventpoint (commands) 34
help command 27
Help system, invoking 27
Help window
 creating 5
 description and *illus* 20
history commands 61

I

if command 34, 62
info alias command 45, 57
info args command 39, 45
info break command 32, 45
info control registers command 43, 45
info cxdb command 45
info default environment command 45
info environment command 45
info errno command 45
info event command 32, 45
info eventtype command 32, 46
info execmode command 46
info expression command 39, 46
info floating point registers command 43, 46
info focus command 10, 46, 60

info formatting command 17, 40, 41, 46
info frame at command 42, 46
info frame command 42, 46
info group command 10, 46, 60
info history command 46, 61
info line command 39, 47
info locals command 39, 47
info macro command 47, 57
info objectmap command 47
info path command 47, 58
info process command 47
info psw command 43, 47
info registers command 43, 47
info scope command 47
info shlib command 47, 49
info signal command 58
info signals command 48
info sourceunit command 39, 48
info space registers command 43, 48
info stack command 48
info symbols command 39, 48
info threads command 10, 48, 59, 60
info trace command 32, 48
info type command 48
info watch command 32, 48
input, logging 50
invoking CXdb
 GUI mode 2
 line mode (tty interface) 2
 with the ID of a running process 3
 with the name of a core file 3
 with the name of an executable file 3
 with the name of an MPI application 4

K

keyboard shortcuts 21
kill process command 14, 35

L

line mode (tty interface)
 command equivalents for GUI information
 displays 6
 invoking CXdb in 2
list command 6, 38
loading a program to debug
 after invoking CXdb 4
 when invoking CXdb 3
local variables, displaying 17
logging commands 50

M

macro command 57
macros, commands for working with 57
memory
 commands for modifying 41
 commands for searching 44
 commands for viewing 41
 displaying 18
Memory Display window, creating 5
messages, logging 50
Mode Selection dialog 4
modifying program data 17, 39
mouse and keyboard shortcuts 21
MPI applications, debugging 4
multiple threads and processes 10

N

next command 15, 37
next instruction command 15, 37

O

output, logging 50

P

Parameters 24
path commands 58
print command
 described 17, 40
 formatting options 18
printing data
 aligning (padding with zeros) 18
 array elements, setting number to print 18
 array slices 17
 formatting options (for memory units) 18
 precision, setting 18
 print options 18
 program data 17
process
 defaults
 commands affecting all new processes 54
 commands affecting existing processes 56
 input and output (process interface window) 6
process execution
 commands 35
 controlling 14
 stepping 15
processes
 debugging 3
 focus 60
 multiple 10

put command 41
pwd command 58

Q

quit command 27
quitting CXdb 27

R

recall command 61
redirection
 CXdb command output 52
 CXdb error and information messages 52
 CXdb redirection operations 52
 process I/O 52
register contents, displaying 18, 43
register windows, creating 5
remove alias command 57
remove cmderr command 50
remove cmdlog command 50
remove cmdout command 50
remove default environment command 55
remove default path command 55
remove environment command 56
remove event command 33
remove eventtype command 33
remove macro command 57
remove path command 56
remove shliblist command 49
remove variable command 61
rerun command 14, 35
resume command 14, 34, 35
return command 35
run command 36
running your program 14

S

set cmderr command 51
set cmdlog command 51
set cmdout command 51
set default environment command 55
set default format command 55
set default handler command 34, 55
set default memory command 55
set default path command 55
set default pshell command 55
set default step command 37, 55
set echo command 53
set environment command 56
set evalopts iprecision command 53
set evalopts rprecision command 53
set focus command 10, 60
set format command 41, 53

set handler command 34
set idlethreads command 59
set ignore command 33
set logging command 51
set memory command 42, 53
set noclobber command 51, 53
set path command 56
set printopt padding command 54
set printopts maxarray command 40
set printopts maxarray command 53
set printopts nopadding command 54
set printopts nopadding command 40
set printopts padding command 40
set printopts precision command 40, 54
set shell command 54
set shliblist command 49
set signal command 58
set step command 15, 37
set threads command 10, 59, 60
set typehandler command 34, 54
shared libraries 49
shell command 62
shell window, creating 5
shortcuts, mouse and keyboard 21
signal process command 58
signals, commands 58
source code
 commands for searching 44
 commands for viewing 38
Source Code window
 creating 5
 description and *illus* 9
source command 62
source units, displaying for a line (info line
 command) 39
stack backtrace 16
stack frame commands 42
Stack Trace window
 creating 5
 described 16
starting CXdb 2, 27
step command 15, 37
step instruction command 15, 37
stepping commands 15, 37
symbol names, displaying 17
symbols, viewing program (info symbols
 command) 39

T

threads
 commands for manipulating 59
 focus 60
 multiple 10
trace instruction command 30
trace line command 30

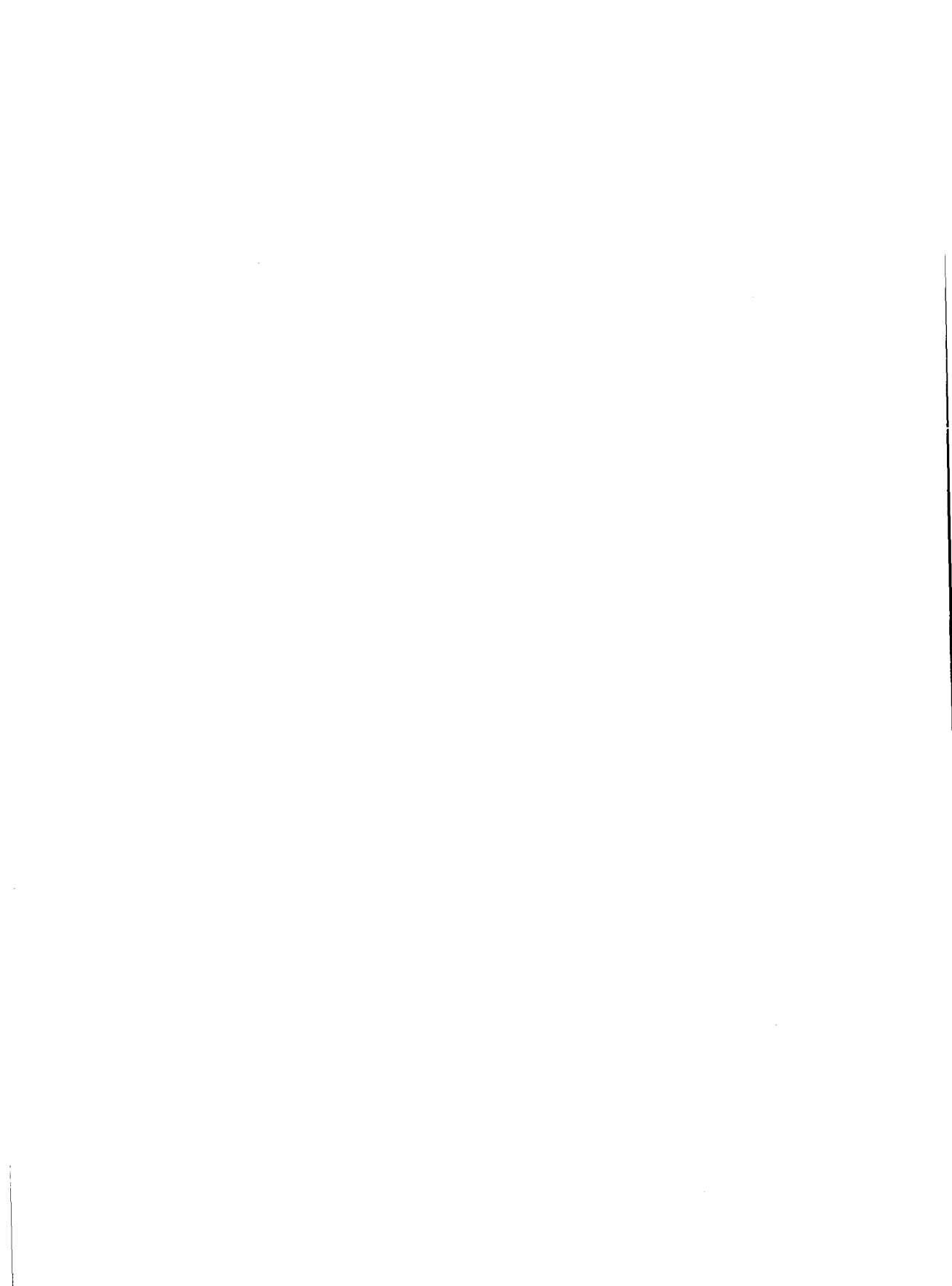
trace routine command 30
trace source command 30
tracepoints
 commands for setting 30
 commands listed 13

V

variables
 debugger 61
 program, viewing local (info locals
 command) 39

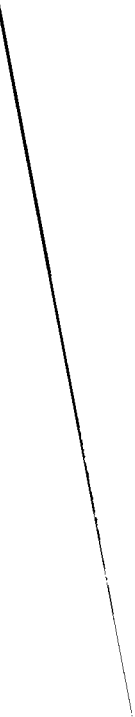
W

watch command 13, 31
watchpoints 13
windows
 Command window 7
 creating 5
 Help 20
 listed and described 6
 Source Code window 9
 Stack Trace 16











HEWLETT®
PACKARD

CONVEX
PRESS

B5639-90001

