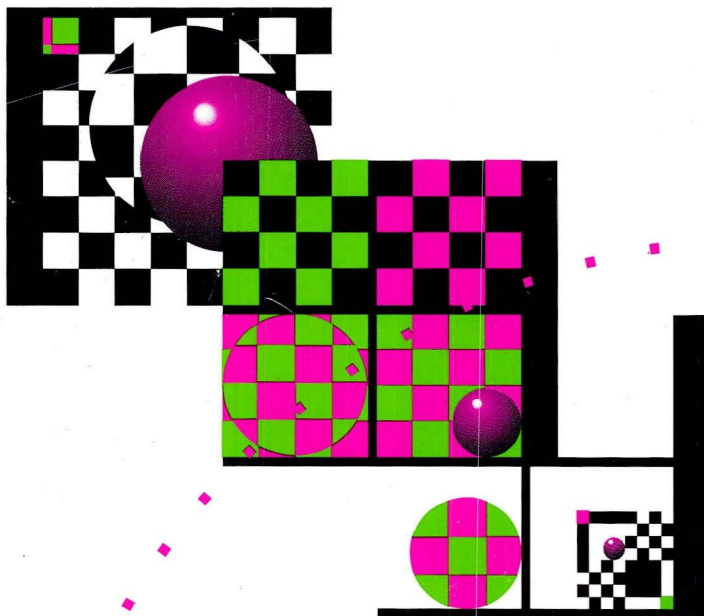


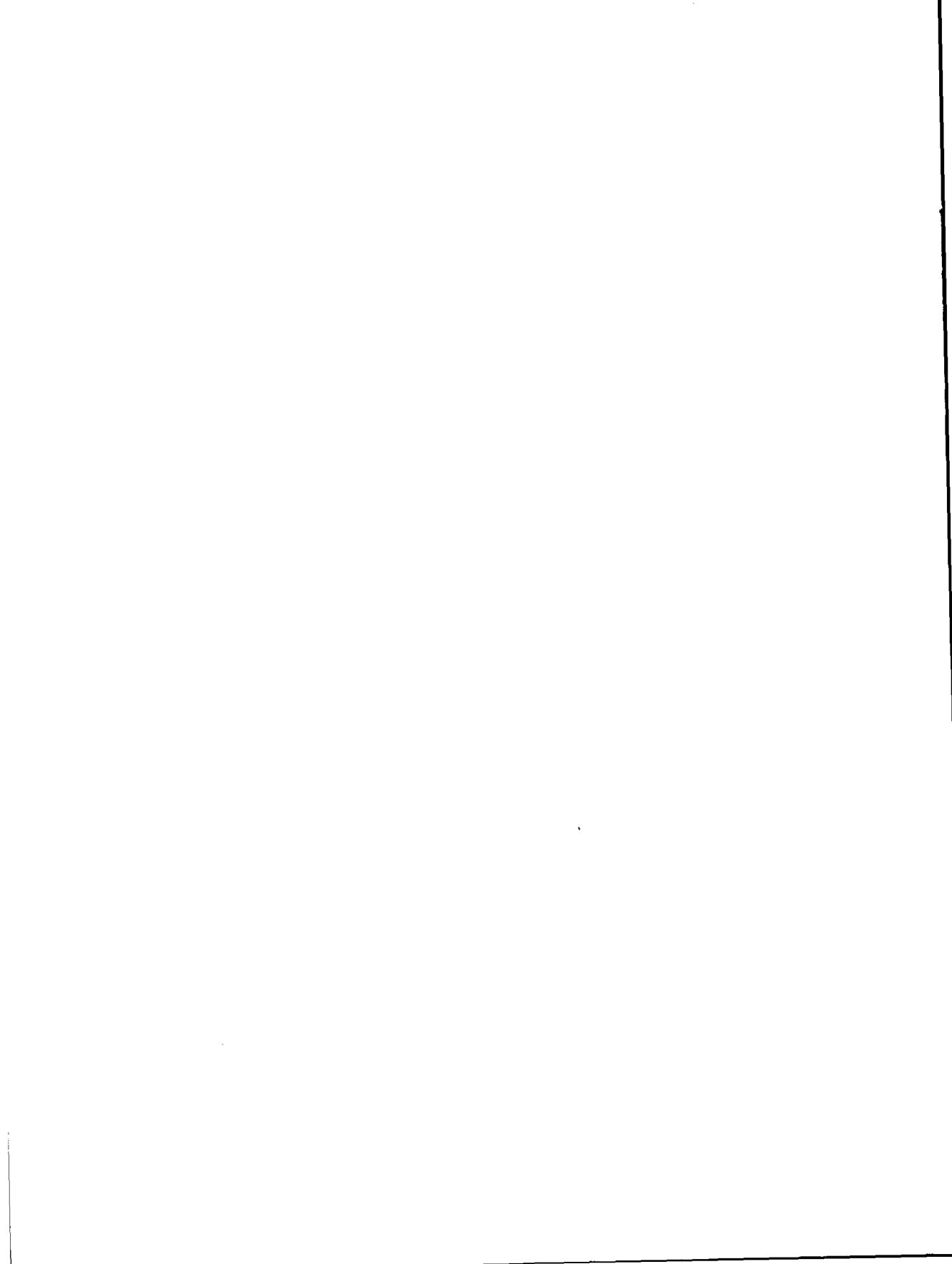


CONVEX

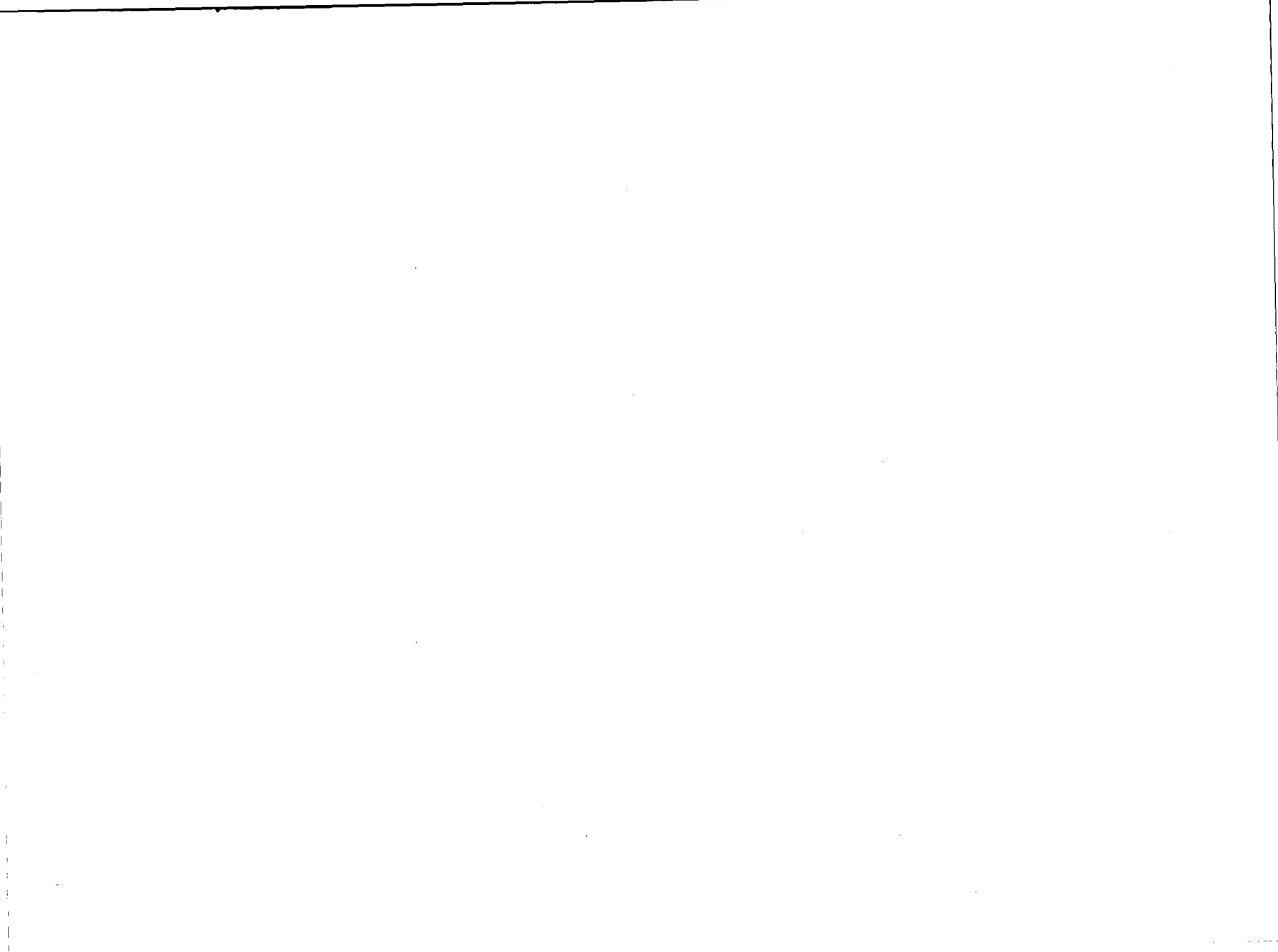
CXdb
Quick Reference

Third Edition





CONVEX Computer Corporation
3000 Waterview Parkway
P.O. Box 833851
Richardson, TX 75083-3851
United States of America
(214) 497-4000



CXdb Quick Reference



Order No. DSW-474

Third Edition
November 1994

CONVEX Press
Richardson, Texas
United States of America

CXdb Quick Reference

Document No. 710-015630-002

Order No. DSW-474

Released with V3.1 of the CXdb software.

Copyright © 1994 CONVEX Computer Corporation
All rights reserved.

This document is copyrighted. This document may not, in whole or part, be copied, duplicated, reproduced, translated, electronically stored, or reduced to machine readable form without prior written consent from CONVEX Computer Corporation.

Although the material contained herein has been carefully reviewed, CONVEX Computer Corporation does not warrant it to be free of errors or omissions. CONVEX reserves the right to make corrections, updates, revisions or changes to the information contained herein. CONVEX does not warrant the material described herein to be free of patent infringement.

UNLESS PROVIDED OTHERWISE IN WRITING WITH CONVEX COMPUTER CORPORATION (CONVEX), THE PROGRAM DESCRIBED HEREIN IS PROVIDED AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. SOME STATES DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES. THE ABOVE EXCLUSION MAY NOT BE APPLICABLE TO ALL PURCHASERS BECAUSE WARRANTY RIGHTS CAN VARY FROM STATE TO STATE. IN NO EVENT WILL CONVEX BE LIABLE TO ANYONE FOR SPECIAL, COLLATERAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, INCLUDING ANY LOST PROFITS OR LOST SAVINGS, ARISING OUT OF THE USE OR INABILITY TO USE THIS PROGRAM. CONVEX WILL NOT BE LIABLE EVEN IF IT HAS BEEN NOTIFIED OF THE POSSIBILITY OF SUCH DAMAGE BY THE PURCHASER OR ANY THIRD PARTY.

CONVEX and the CONVEX logo ("C") are registered trademarks of CONVEX Computer Corporation.

UNIX is a registered trademark of UNIX System Laboratories, Inc., a wholly owned subsidiary of Novell, Inc.

X Window System is a trademark of M.I.T.



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 X Windows mode	2
	In line mode (tty interface)	2
	Using <code>cxdb</code> command options	2
	Loading a program to debug	3
	Invoking CXdb with the name of an executable	3
	Invoking CXdb with the name of a core file	3
	Specifying a different executable, a core file, or a process to debug	4
	User interface and windows	5
	Creating CXdb windows (X Windows mode)	5
	CXdb windows and line mode equivalents	6
	Using the Command window	7
	Using the Source Code window	8
	Working with breakpoints, tracepoints, and watchpoints	9
	Using commands	9
	Using the mouse in the Source Code window	10
	Controlling process execution	11
	Stepping process execution	12
	Obtaining a stack backtrace	13
	Displaying information about stack frames	14
	Displaying and modifying program data	15
	Viewing thread information	17
	Commands	17
	Thread Activity window	18
	Using CXdb in line mode	19
	Getting help online	20
	Mouse and keyboard shortcuts	21
	Command-line editing	21
	Source Code window	22
	Assembly Code window	22
	Stack Trace window	23
	Thread Activity window	23
2	Command syntax and descriptions	25
	Getting started	26
	Starting CXdb— <code>cxdb</code> command	26
	quit command	26

help command	26
Specifying a file or process to debug	27
Working with eventpoints	28
Setting breakpoints—break commands	28
Setting tracepoints—trace commands	29
Setting other types of eventpoints—event and watch commands	30
Displaying eventpoint information	31
Enabling, disabling, and setting ignore counts for eventpoints	32
Removing eventpoints	32
Eventpoint handler commands	33
Running your program	34
Process execution commands	34
Stepping commands	35
Viewing program source, data, and memory	37
Viewing source code	37
Viewing assembly-language code	37
Displaying and modifying data	38
Using the print command	39
Viewing and modifying memory	40
Displaying stack frame information	41
Displaying register contents	42
Searching memory	43
Searching the Source Code window	43
CXdb info commands	44
Logging and redirection	48
Logging input, output, and messages	48
Redirecting process I/O	50
CXdb redirection operators	50
For command output	50
For error and information message output	50
Configuring CXdb	51
Setting and clearing CXdb defaults	51
Setting and clearing global process defaults	53
Setting and clearing defaults for the current process	55
Working with aliases and macros	57
Path and directory commands	58
Working with signals	59
Working with threads	60
Debugger variables	61
CXdb history commands	61
Miscellaneous commands	62

Using CXdb

1

Compiling your program for CXdb

CXdb can debug both C and Fortran programs. To enable full symbolic debugging with CXdb, compile your source code with the `-cxdb` option. For example:

```
% fc -cxdb myprog.f
```

When you compile with the `-cxdb` option, the compiler produces a set of data files that contain all the symbolic debugging information for your program and places them in a subdirectory called `.CTI`.

The `.CTI` subdirectory is created in the same directory where the compiler places the object (`.o`) files for your program.

If you move any of your source files, executable files, or CTI data files after compiling or if your directory structure changes, then you must give CXdb the new locations of those files, as shown in the following table:

If you move	Specify new location with
Source files	add path or set path command
Executable files	add default path or set default path command
CTI data files	add path or set path command

Invoking CXdb

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

- **X Windows 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 X Windows mode

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

```
% cxdb &
```

In line mode (tty interface)

From the shell, invoke CXdb in line mode (tty interface) using the `-nw` option of the `cxdb` command. For example:

```
% cxdb -nw
```

Using `cxdb` command options

The following table lists and describes selected `cxdb` command options. Refer to "Starting CXdb—`cxdb` command," on page 26 for complete syntax and options.

Option	Purpose
<code><executable></code> or <code>-e <executable></code>	Performs the <code>debug exec</code> command and loads the executable file to debug.
<code>-c <corefile></code>	Performs the <code>debug core</code> command and loads the core file to debug.
<code>-a <process_id></code>	Attaches to the specified process.
<code>-F</code>	Enables fixed scheduling (C Series only).
<code>-nw</code>	Invokes CXdb in line mode (tty interface).
<code>-csd</code>	Automatically incorporates aliases for <code>csd</code> debugger commands. Using this option also invokes CXdb in line mode.

Loading a program to debug

When you begin a debugging session, or if a process object does not exist, use the `debug exec` command on the CXdb command line to load your program. (The executable file should have been compiled with the `-cxdb` option.) For example,

```
(CXdb) debug exec <executable_file>
```

The `debug exec` command:

- Gives CXdb the name of the executable file you wish to debug.
- Creates a process object. A process object contains the image of the process, core file, or executable; debugging information; and process settings.
- Incorporates debugging information associated with the executable file into the process object so that symbolic debugging is possible.
- In X Windows mode, brings up a Source Code window by default that displays source files associated with your executable.

Invoking CXdb with the name of an executable

You can invoke CXdb from the shell with the name of an executable. For example:

```
% cxdb <executable>
```

The above command invokes CXdb in X Windows 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:

```
% cxdb -c <corefile> <executable>
```

The above command starts CXdb in X Windows 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.

Specifying a different executable, a core file, or a process to debug

The following table describes CXdb commands for specifying a different executable file, a core or checkpoint file, or a process to debug. Any executable files specified should have been compiled with the `-cxd` option.

If you want to...	Use these commands...	Explanation
Specify a different executable file	<code>executable <executable_file></code>	<p>A process object must already exist (a process object is created when you invoke CXdb with the name of an executable file or when you execute the <code>debug exec</code>, <code>debug proc</code>, or <code>debug core</code> commands).</p> <p>This command replaces the old debugging information with new information associated with the specified executable file.</p>
Debug an existing process	<code>debug proc <process-id></code> <code>executable <executable_file></code>	<p>This involves two commands. The <code>debug proc</code> command stops the process, then creates a process image that is incorporated into the process object.</p> <p>The <code>executable</code> command incorporates the debugging information from the executable file that generated the process, so that symbolic debugging is possible.</p>
Debug a core file	<code>debug core <core_file></code> <code>executable <executable_file></code>	<p>This involves two commands. The <code>debug core</code> command creates a process object, creates an image based on the specified core file, and incorporates that image into the process object.</p> <p>The <code>executable</code> command associates the appropriate debugging information from the executable file that generated the core image, so that symbolic debugging is possible.</p>

User interface and windows

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

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

Creating CXdb windows (X Windows mode)

To create CXdb windows:

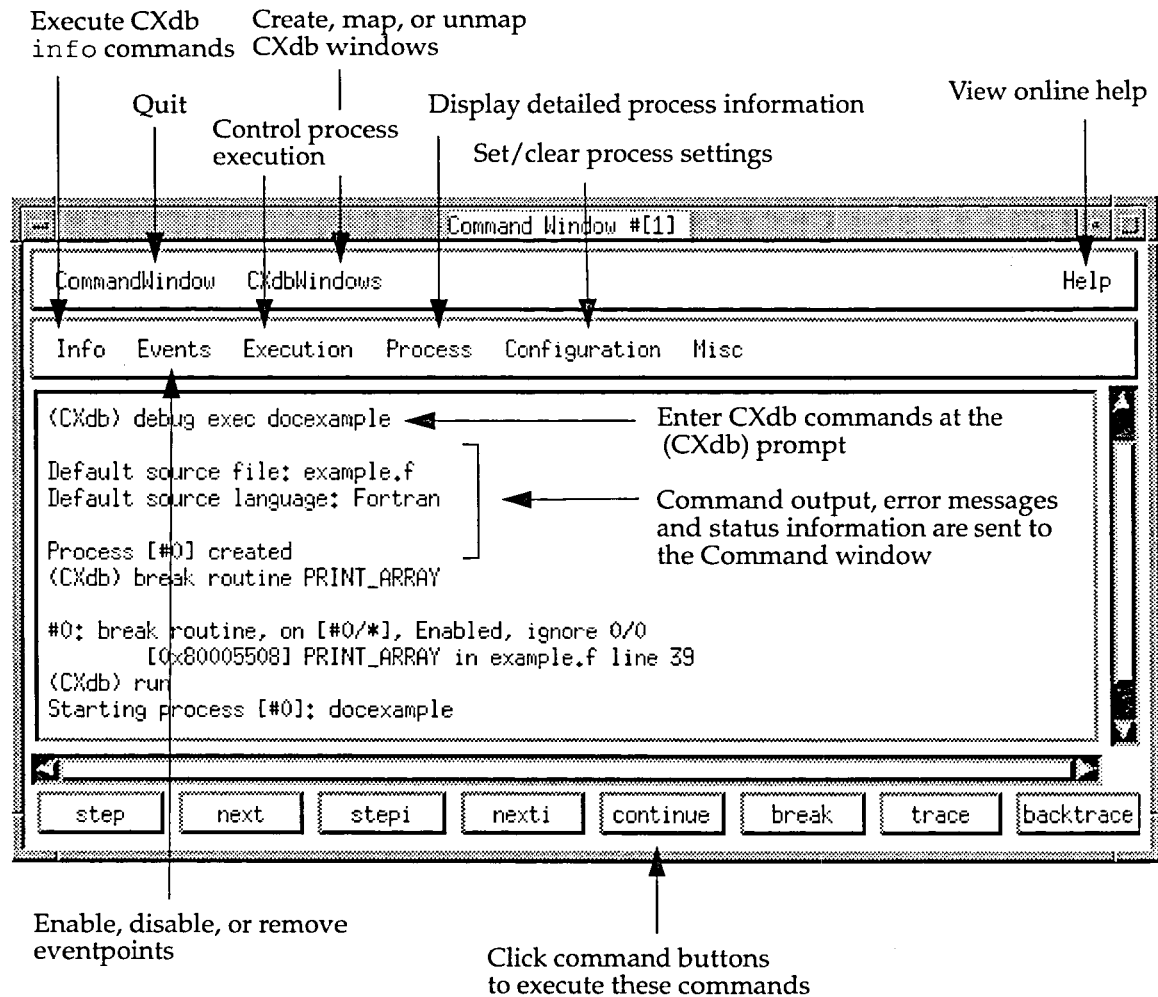
- **Command window**—Created automatically when you execute the `cxdb` command.
- **Source Code window**—By default, created automatically when you execute the `debug exec`, `debug proc`, `executable`, `display source`, or `display routine` commands. Can also be created by selecting `Create window`, then `Source Code`, from the `CXdbWindows` menu in the `Command window`.
- **Process interface window**—Created automatically when you execute the `run` or `rerun` command.
- **Assembly Code window**—Select `Create window`, then `Assembly Code`, from the `CXdbWindows` menu in the `Command window`, or use the `display disassembly` command.
- **Memory Display window**—Select `Create window`, then `Memory Display`, from the `CXdbWindows` menu, or execute the `display examine` command.
- **Stack Trace window, Thread Activity window or register windows**—Select `Create window`, then the appropriate item from the `CXdbWindows` menu in the `Command window`.
- **Help window**—Select the `Window Overview`, `Using Help` or `Tutorial` item from the `Help` menu in any CXdb window, or execute the `help` command.
- **Edit window**—Select the `edit` item from the `Misc` menu in the `Command window`, or execute the `edit` command.
- **Shell window**—Select the `shell` item from the `Misc` menu in the `Command window`, or execute the `shell` command.

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 thread activity relative to source code	Thread Activity	None
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 vector registers	Vector Registers	<code>info vregisters</code>
View contents of communication registers	Communication Registers	<code>info cregisters</code>
View contents of scalar and address registers	Scalar Registers	<code>info registers</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. 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).



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.

The screenshot shows the Source Code window titled "Source Code Window #[2]". The window contains a menu bar with "SourceCodeWindow", "FileView", "CXdbWindows", and "Help". Below the menu bar, it displays "process[#0/0,1]" and "file: mtrx_mult.f". The main area shows Fortran source code with line numbers 44 to 61. A right-click context menu is open over the code, listing actions like "print", "print *", "breakpoint", "tracepoint", "info expression", "info source", "info line", and "run to". A vertical column on the left contains eventpoint markers (star, square, inverted triangle, circle) corresponding to lines 47, 48, 51, and 58. A vertical scrollbar is on the right.

Close window, associate window with a specific thread or threads, or enable auto update

Create, map, or unmap CXdb windows

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

Click right mouse button to bring up Actions popup menu

Highlighting indicates active source units

Thread markers show location of active threads

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 breakpoints, tracepoints, and watchpoints

This section lists commands for manipulating breakpoints, tracepoints, and watchpoints and also describes how to use the mouse in the Source Code window to manipulate them.

Using commands

The following table contains a list of selected commands for working with breakpoints, tracepoints, and watchpoints. All of the commands listed are available in both X Windows 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 trace routine
Set a breakpoint or tracepoint at a line	break line trace line
Set a breakpoint or tracepoint at an instruction	break instruction trace instruction
Set a watchpoint to monitor an address range	watch
Set a conditional breakpoint	event relation
Disable a breakpoint, tracepoint, or watchpoint	disable event
Enable a breakpoint, tracepoint, or watchpoint	enable event
Remove a breakpoint, tracepoint, or watchpoint	remove event

Refer to following sections for complete syntax and additional commands:

- "Setting breakpoints—break commands" on page 28
- "Setting tracepoints—trace commands" on page 29
- "Displaying eventpoint information" on page 31
- "Removing eventpoints" on page 32
- "Eventpoint handler commands" on page 33

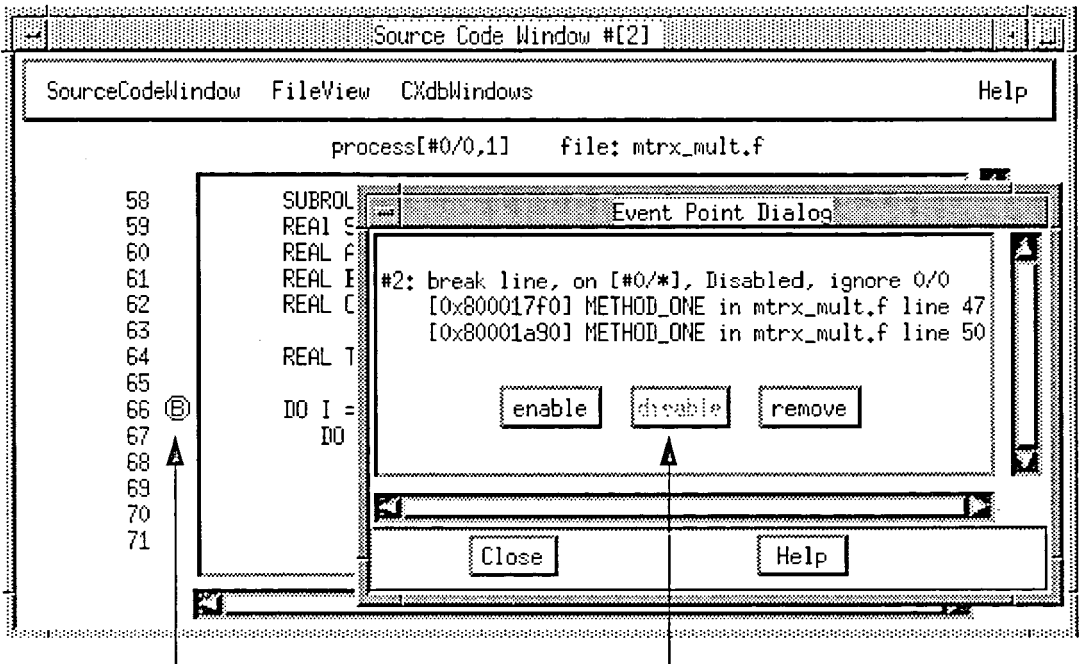
Using the mouse in the Source Code window

In the Source Code window, you can use the mouse to set a breakpoint or tracepoint using the following method:

1. Position the mouse pointer over the line of source code at which you wish to set the breakpoint or tracepoint.
2. Click the right mouse button to display the popup action menu.
3. Select the breakpoint or tracepoint item.

To enable, disable, or remove a breakpoint or tracepoint in the Source Code window:

1. Left click on the eventpoint marker to bring up the Event Point dialog.
2. Locate the eventpoint you wish to work with in the Event Point dialog.
3. Select the action you wish to perform on the eventpoint (enable, disable, or remove).
4. Click the Close button.



Left click on eventpoint marker with mouse to open dialog

Dialog window appears for controlling eventpoints

Controlling process execution

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

- Compiled your program with the `-cxdb` option.
- Loaded 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.

The following table describes how to perform basic program execution tasks. Refer to "Stepping process execution" on page 12 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 Source Code window in X Windows mode.
Rerun your program using the previous argument list	<code>rerun</code>	
Continue execution of a stopped process	<code>continue</code>	A process object must have already been created with the <code>run</code> or <code>rerun</code> command.
Continue process execution from within an eventpoint handler	<code>resume</code>	Can only be used 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 X Windows mode you can also control process execution by:

- Using the `run` and `continue` command buttons in the Command window
- Selecting process execution commands from the Execution menu in the Command window
- Selecting "run to" from the Actions popup menu in the Source Code window to run to a specific location

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 expression, statement, loop, routine, or block. If no granularity is specified, CXdb uses the default stepping granularity.

Refer to "Stepping commands" on page 35, for complete command syntax and additional commands.

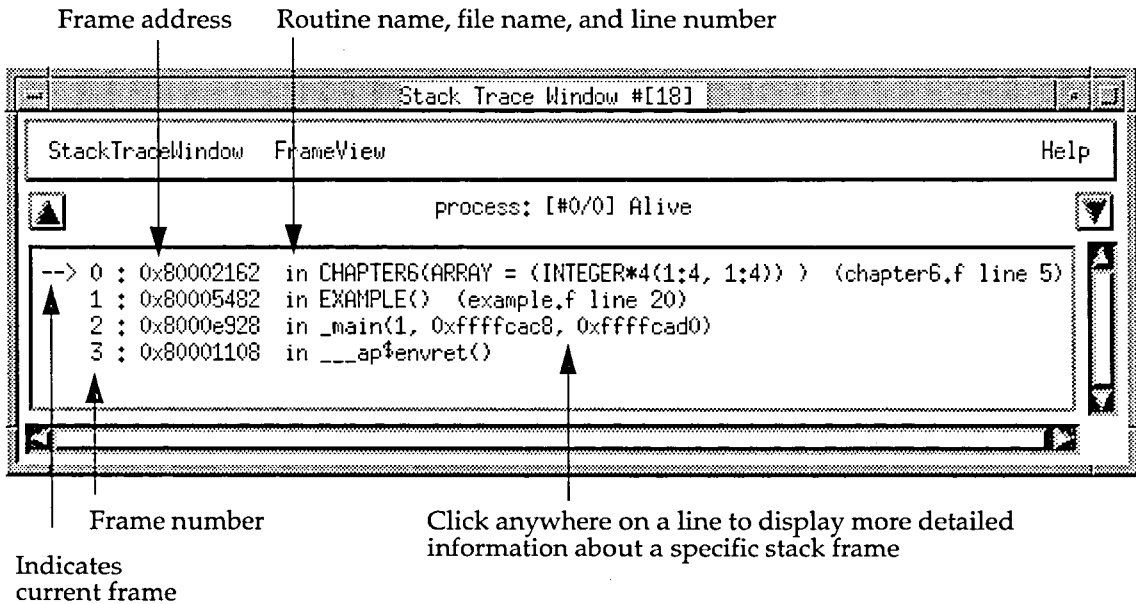
To perform this stepping task...	Use this command...
Set the default stepping granularity for CXdb.	set step [<i><granularity></i>]
Step process execution by 1 source unit of the specified granularity, stepping into subroutine calls	step [<i><granularity></i>]
Step the process by 1 instruction, stepping into subroutine calls	step instruction
Finish executing (step out of) a source unit of the specified granularity	finish [<i><granularity></i>]
Step process execution to the next source unit of specified granularity, stepping over subroutine calls	next [<i><granularity></i>]
Step from the current source of the specified granularity to the next source unit of default granularity, not counting subroutine calls.	next over <i><granularity></i>
Step to the next instruction, stepping over subroutine calls.	next instruction

In X Windows mode, you can also execute stepping commands by:

- Using the step, stepi, next, nexti command buttons in the Command window
- Selecting stepping commands from the Execution menu in the Command window
- Pressing the following keys while in the Source Code window (assuming you have not modified the default key bindings):
 - s—Step
 - n—Next
 - c—Continue

Obtaining a stack backtrace

You can obtain a stack backtrace in X Windows mode by opening a Stack Trace window. You can do this by selecting the Stack Trace item from the CXdbWindows menu in either the Command window or the Source Code window. A sample Stack Trace window is shown below.



In X Windows mode, you can also click on the backtrace command button in the Command window.

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

(CXdb) **backtrace**

Process [#0/0]

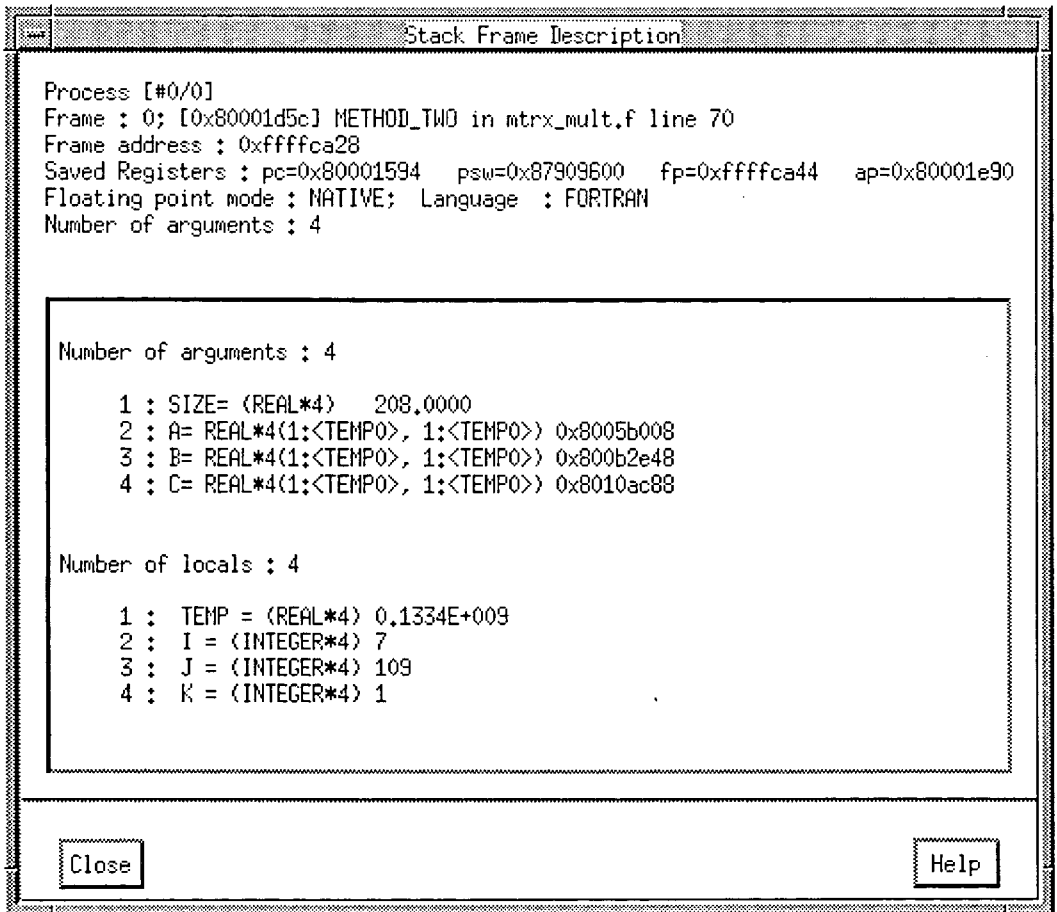
```

cf> 0 : 0x80001658 in SUB1(MAXLEN = (INTEGER*4) 1000, A = (REAL*4(1:<TEMP0>)) ,
    B = (REAL*4(1:<TEMP0>, 1:<TEMP0>)) ) (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 information about stack frames

To obtain detailed information about a specific stack frame in X Windows mode:

1. Open a Stack Trace window by selecting Create window, then Stack Trace from the CXdbWindows menu in either the Command window or the Source Code window.
2. In the Stack Trace window, click with the left mouse button anywhere on the line describing the stack frame you want information for. This will bring up the Stack Frame Description dialog:



In line mode, you can use the following sequence of commands to obtain this information:

```
(CXdb) info frame; info args; info locals
```

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 39
- "Viewing and modifying memory" on page 40
- "Displaying register contents" on page 42
- "CXdb info commands" on page 44

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</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</pre>	Note that there should not be a space between the word "print" and the display format specification. Not all print formats are available for all memory unit types.
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>	In X Windows mode, you can also open a Memory Display window. Refer to the "examine" reference topic for formatting options.
Display register contents for registers available on your system.	<pre>info registers info vregisters info cregisters info floating point registers info psw info space registers info control registers</pre>	In X Windows mode, you can open register windows for the registers available on your system.
Set floating-point mode for evaluating language expressions.	<pre>set evalopts.fpmode {ieee native dual}</pre>	Not available on SPP Series systems (on SPP Series systems, only IEEE floating-point mode is available.)

Viewing thread information

A *thread* is a sequence of machine instructions that executes on a single processor (CPU). If your machine has multiple CPUs, you can create multiple threads in your program by compiling your source code at optimization level `-O3`, including compiler directives (such as `FORCE_PARALLEL`) in your source code, or by using assembly-language instructions such as `spawn` or `pthread` to create threads at specific locations.

CXdb provides windows and commands that enable you to view information about threads in your program. In addition, you can associate various CXdb windows with specific threads, view navigation hints in the Source Code or Assembly Code windows, or use CXdb commands on individual threads. Refer to the "threads" reference topic page for more information.

Commands

The following table lists tasks associated with multithreaded debugging and describes corresponding CXdb commands.

If you want to...	Use these commands...	Comments
Enable fixed scheduling.	<code>set fixed sched</code>	C Series systems only. Fixed scheduling reserves all the CPUs of the machine for each timeslice that your process executes. Fixed scheduling does not guarantee that your process will generate multiple threads, but it does provide the most stable environment for multiple threads by minimizing nondeterminism.
Enable default fixed scheduling.	<code>set default fixed sched</code>	
Disable fixed scheduling.	<code>clear fixed sched</code>	
Disable default fixed scheduling.	<code>clear default fixed sched</code>	
Set an eventpoint to detect the creation of a new thread.	<code>event spawn</code>	When this eventpoint is triggered, it stops all threads of the process.
Set an eventpoint to detect when a thread joins.	<code>event join</code>	When this eventpoint is triggered, it stops all threads of the process.
Display thread information.	<code>info threads</code>	Lists the current state of each thread, ID numbers of active threads, and identifies the current thread.

Thread Activity window

The Thread Activity window (shown below) displays a 2-D graph of thread activity as it relates to the source code of your program. Using this window, you can identify and display source files or routines associated with active threads in your program. This is especially useful when debugging programs with routine-level and asymmetric parallelism. To create this window, select Create window, then Thread Activity from the CXdbWindows menu in the Command window.

Contains items for closing the window and saving the graph to a file

Contains items for specifying scaling and sorting options for X- and Y-axis data

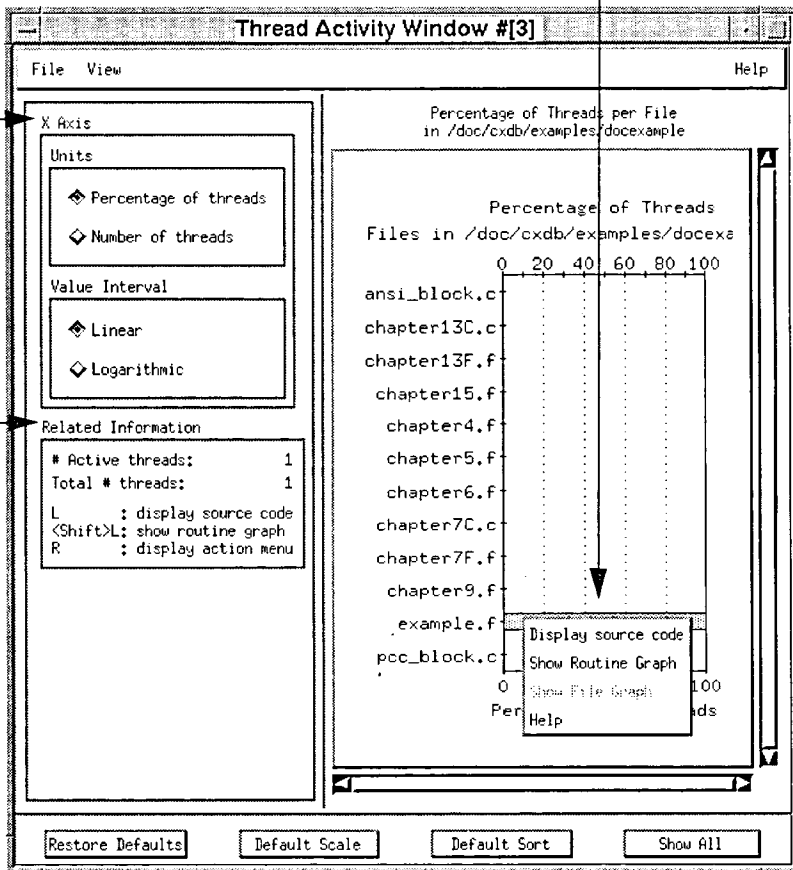
Click left mouse button on bar to display source code for routine or file

Click **SHIFT**-left mouse button on bar to toggle display between Threads per File and Threads per Routine

Click right mouse button on bar to display pop-up menu

Configure display units and value interval for X axis

Displays total number of threads and number of active threads



Scrollbars indicate additional information is available

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
```

NOTE: The `-nw` option overrides your `DISPLAY` environment variable.

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.

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.

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

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`, `signal`, or `reached`
- When the process stops after executing commands affect process execution such as `step`, `stepi`, `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.

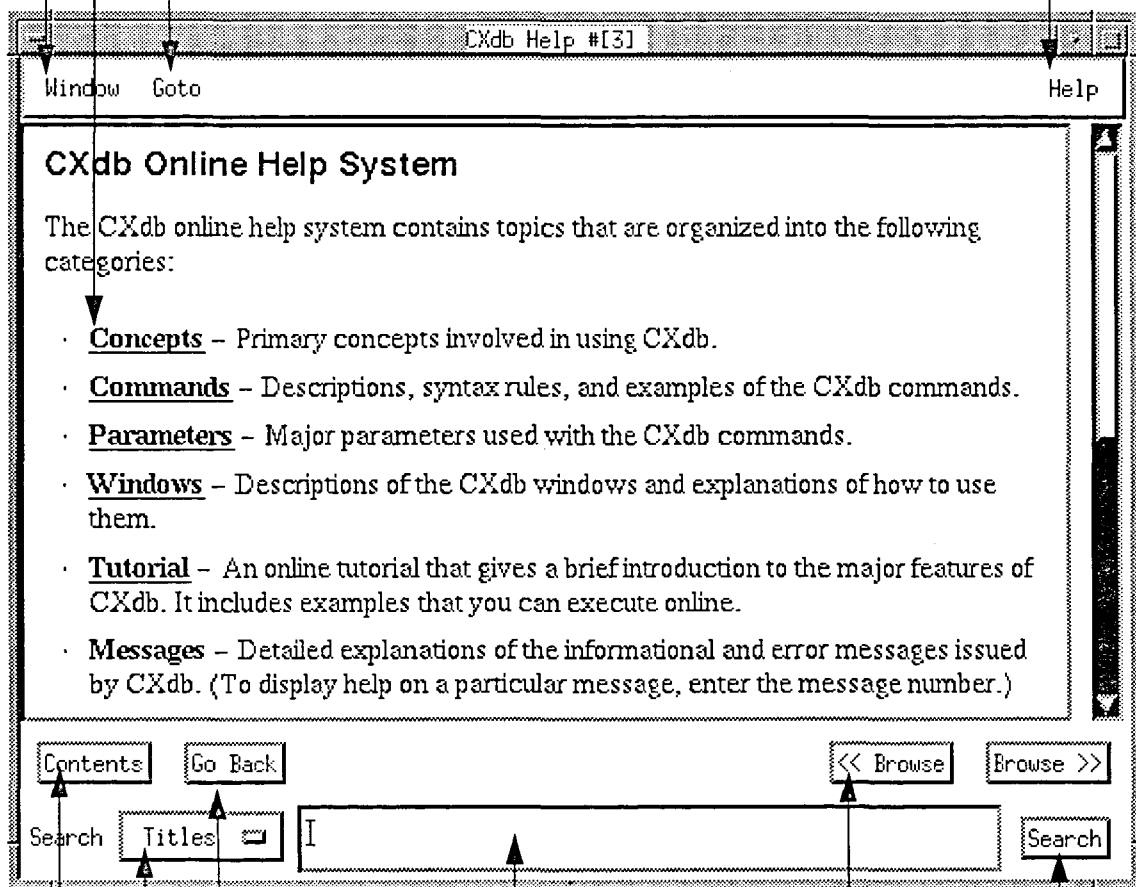
Getting help online

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

Print help text or close window Go to hyperlinked lists of Help system contents, commands, parameters, and windows, or return to previous topic

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

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



Display help system contents page

Select search type (Titles or All Text)

Enter a character string here to search help topics

Click to move forward (>>) or backward (<<) through current topic, one window at a time

Click here to activate search

Mouse and keyboard shortcuts

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

Command-line editing

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-?	Lists all available key bindings (line mode only)

Source Code window

Key/button	Function
r	Executes the <code>run</code> command.
s	Executes the <code>step</code> command using the default granularity.
c	Executes the <code>continue</code> command.
Left button in text area	Selects (highlights) text characters in the text scrolling area.
Left button on eventpoint marker	Brings up an Event Point dialog where you can disable, enable, or remove eventpoints.
Right button	Brings up an actions popup menu and highlights the source unit at the location of the mouse pointer. When you release the mouse button, the command (action) you selected uses the highlighted source unit as its argument. The actions menu contains the following items: print print * (print indirect) breakpoint tracepoint info expression info source info line run to
CTRL-Left button	Highlights (selects) the source unit at the location of the mouse pointer.

Assembly Code window

Key	Function
c	Executes the <code>continue</code> command.
n	Executes the <code>next instruction</code> command.
s	Executes the <code>step instruction</code> command.
Left button	Click the left mouse button on any eventpoint marker to bring up an Event Point dialog where you can enable, disable, or remove eventpoints.

Stack Trace window

Key	Function
0 through 9	Changes the current frame to the number pressed.
d	Moves up the stack one frame at a time (equivalent to the <code>frame +1</code> command).
t	Makes the top frame the current frame (equivalent to the <code>frame 0</code> command).
u	Moves down the stack one frame at a time (equivalent to the <code>frame -1</code> command).
Left button	Click anywhere on the line describing a stack frame in the Stack Trace window to open a Stack Frame Description dialog displaying detailed information about that frame.

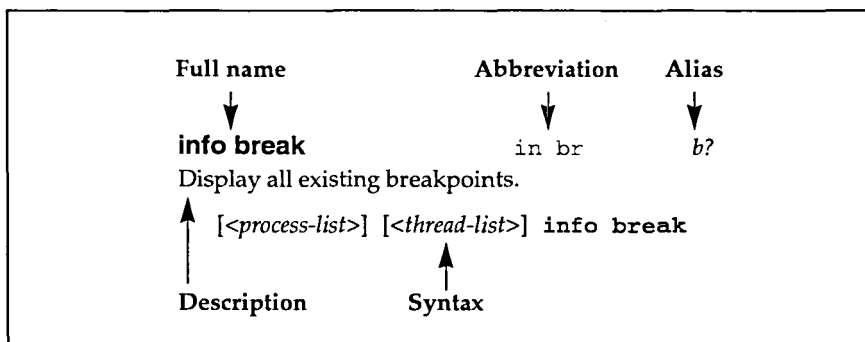
Thread Activity window

Button	Function
Left button	Click on bar in thread activity graph to display source code for the corresponding routine or file.
SHIFT-Left button	Click on bar in thread activity graph to toggle between displaying Threads per File and Threads per Routine in file.
Right button	Click on bar in thread activity graph to pop up an actions menu.

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:



Starting CXdb—`cxdb` command

Use the following command to invoke CXdb. To access symbolic debugging information, you must have compiled your program with the `-cxdb` option:

cxdb cxdb

Invoke CXdb from the shell.

```
cxdb [-a <process-id>] [-csd ]
      [-D <directory-specifier>[, ...]]
      [-f <command-file>] [-F ] [-ns ] [-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>-F</code>	Enables fixed scheduling.
<code>-ns</code>	Disables dynamic creation of Source Code windows.
<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 a file or process to debug

attach at

Debug the image of a running process.

[<process-list>] attach [<remote-host>:] <process-id>

core cor

Debug the image of a core file or a checkpoint file.

[<process-list>] core [<remote-host>:] <file-name>

debug core deb c dbgc

Create a process object and debug the image of a core or checkpoint file.

debug core <core-file> [executable [<remote-host>:] <executable-file>]
[<debugger-variable>]

debug exec deb e dbg

Create a process object and debug the image of an executable file.

debug exec [<remote-host>:] <executable-file> [<debugger-variable>]

debug proc deb p dbgp

Create a process object and debug the image from a running process.

debug proc [<remote-host>:] <process-id> [executable [<remote-host>:]
<executable-file>] [<debugger-variable>]

detach det

Detach from a process.

[<process-list>] detach

executable exe

Specify an executable file to provide CTI data and the executable image.

[<process-list>] executable [<remote-host>:] <file-name>

Setting breakpoints—break commands

- break instruction** bre i *bi*
Set a breakpoint at an instruction.
[<process-list>] [<thread-list>] break instruction <language-expression>
[{<event-handler>}] [<debugger-variable>]
- break line** bre l *bl*
Set a breakpoint at a source line.
[<process-list>] [<thread-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>] [<thread-list>] break routine <language-expression>
[{<event-handler>}] [<debugger-variable>]
- break source** bre s *bs*
Set a breakpoint at a source unit.
[<process-list>] [<thread-list>] break source <source-unit>
[{<event-handler>}] [<debugger-variable>]

Setting tracepoints—trace commands

trace instruction t i ti

Set a tracepoint at an instruction.

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

trace line t l tl

Set a tracepoint at a source line.

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

trace routine t r tr

Set a tracepoint at the beginning of a routine.

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

trace source t s ts

Set a tracepoint at a source unit.

```
[<process-list>] [<thread-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 modify

eve m

Set an eventpoint to watch for a value change within an address range.

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

event reached instruction

eve rea i

Set an eventpoint at an instruction.

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

event reached line

eve rea l

Set an eventpoint at a source line.

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

event reached routine

eve rea r

Set an eventpoint at the beginning of a routine.

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

event reached source

eve rea s

Set an eventpoint at a source unit.

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

event relation

eve rel

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

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

Displaying eventpoint information

info break in br b?

Display all existing breakpoints.

```
[<process-list>] [<thread-list>] info break
```

info event in event e?

Display the specified eventpoints.

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

info eventtype in eventt et?

Display all eventpoints of the specified type.

```
[<process-list>] info eventtype {<event-specifier>} [, ...]
```

info trace in tr t?

Display all tracepoints.

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

info watch in w i w

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 <eventpoint_specifier> [, ...]
```

disable eventtype disab eventt

Disable all eventpoints of the specified type.

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

set ignore se i

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 t

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.

```
[<process-list>] [<thread-list>] continue [<count>] [&]
```

detach

det

Detach from a process.

```
[<process-list>] detach
```

goto address

g a

Branch to the specified address.

```
[<process-list>] [<thread-list>] goto address <language-expression>
```

goto line

g l

Branch to the specified source line.

```
[<process-list>] [<thread-list>] goto line <line-specifier>
```

goto source

g s

Branch to the specified source unit.

```
[<process-list>] [<thread-list>] goto source <source-unit>
```

kill process

k p

k

Terminate a running process and remove the image being debugged from the process object.

```
[<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.

```
[<process-list>] [<thread-list>] 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:

- expression
- statement
- block
- loop
- routine

clear step cl st
 Reset the stepping granularity to the default setting.
 [*<process-list>*] clear step

finish fini
 Finish executing (step out of) the specified source unit.
 [*<process-list>*][*<thread-list>*] finish [*<granularity>*][&]

next n
 Step to the next source unit, ignoring subroutine calls.
 [*<process-list>*][*<thread-list>*] next [*<granularity>*][*<count>*][&]

next instruction n i *ni, nexti*
 Step to the next instruction, stepping over subroutine calls.
 [*<process-list>*][*<thread-list>*] next instruction [*<count>*][&]

next over n o *no*
 Step from the current source unit of specified granularity to the next source unit of default granularity, stepping over subroutine calls.
 [*<process-list>*][*<thread-list>*] next over [*<granularity>*][&][*<count>*]

set default step

se de s

Set the default stepping granularity.

set default step <granularity>

set handler

se h

Set the handler for a specified eventpoint.

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

set ignore

se i

Set an ignore count for an eventpoint.

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

set step

se st

Set the stepping granularity.

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

step

ste

s

Step to the next source unit.

[<process-list>] [<thread-list>] step [<granularity>] [<count>] [&]

step instruction

ste i

si, stepi

Step to the next instruction.

[<process-list>] [<thread-list>] step instruction [<count>] [&]

step over

ste o

so

Step from the current source unit of specified granularity to the next source unit of default granularity.

[<process-list>] [<thread-list>] step over [<granularity>] [<count>] [&]

Viewing source code

display routine disp r

In X Windows mode, open a new Source Code window displaying source code for a routine. Not available in line mode.

```
[<process-list>][<thread-list>] display routine <language-expression> [\;
<thread-number> [, ...]]
```

display source disp so

In X Windows mode, create a new Source Code window to display a source file. Not available in line mode.

```
[<process-list>] display source <file-name> [<thread-number> [, ...]]
```

list l

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.

```
[<process-list>][<thread-list>] disassemble
[<starting-address> [{ ..<ending-address> | :<instruction-count> }]]
```

display disassembly disp d

In X Windows mode, create an Assembly Code window displaying assembly-language code. Not available in line mode.

```
[<process-list>][<thread-list>] display disassembly [<address-expression>]
[\; <thread-number> [, ...]]
```

Displaying and modifying data

evaluate

eva

Evaluate a language expression.

[<process-list>] [<thread-list>] evaluate <language-expression>

info args

in ar

args

Display arguments of the current routine.

[<process-list>] [<thread-list>] info args

info expression

in ex

describe, whatis

Display the characteristics of the specified language expression.

[<process-list>] [<thread-list>] info expression <language-expression>

info line

in li

i li

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.

[<process-list>] [<thread-list>] info locals

info sourceunit

in so

i so

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.

```
[<process-list>] [<thread-list>] print [/ [n] <format> <fpmode>] <language-expression>
```

<format> options are as follows:

B	Binary
C	Fortran complex
L	Fortran logical
c	ASCII character
d	Signed decimal
e[<width>.<precision>]	Scientific
f[<width>.<precision>]	Floating-point notation
i	Instruction
n	Suppresses newline character at end of printed data
o	Octal
s	String
u	Unsigned decimal
x	Hexadecimal

info formatting in fo ifo

Display settings for print options/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.

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

display examine disp e

Create a Memory Display window in X Windows mode.

```
[<process-list>] [<thread-list>] display examine <address-expression>
  [ \; <thread-number> ]
```

examine exa x

Display a region of memory.

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

evaluate eva

Evaluate a language expression.

```
[<process-list>] [<thread-list>] evaluate <language-expression>
```

fill fil

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

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

get ge

Restore the contents of memory regions from a binary file.

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

info formatting in fo i fo

Display the settings for memory display formats.

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

put pu

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

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

set format se fo
 Set the formats for displaying memory.
 [*<process-list>*] [*<thread-list>*] set format *<memory-unit>* *<format>*

set memory se m
 Set the unit size for displaying memory.
 [*<process-list>*] [*<thread-list>*] set memory *<memory-unit>*

Displaying stack frame information

backtrace ba bt
 Display the frames of the call stack.
 [*<process-list>*] [*<thread-list>*] backtrace [*<frame-count>*]

display stack disp st
 Create a Stack Trace window in X Windows mode. Not available in line mode.
 [*<process-list>*] display stack [*<frame-specifier>*] [:*<thread-number>*]

frame fr f
 Change the current stack frame.
 [*<process-list>*] [*<thread-list>*] frame *<frame-specifier>*

info frame in fr i fr
 Display a stack frame.
 [*<process-list>*] [*<thread-list>*] info frame [*<frame-specifier>*]

info frame at in fr a i fr a
 Display the stack frame at the specified address.
 [*<process-list>*] [*<thread-list>*] info frame at *<language-expression>*

Displaying register contents

info control registers in co r i co r

Display the control registers (Exemplar systems only).

[<process-list>] [<thread-list>] info control registers

info cregisters in cr i cr

Display the communication registers (C Series only).

[<process-list>] info cregisters

info floating point registers in fl p r i fl pr

Display the floating-point registers (Exemplar systems only).

[<process-list>] [<thread-list>] info floating point registers

info psw in ps i ps

Display the processor status word (PSW) register.

[<process-list>] [<thread-list>] info psw

info registers in r i r

Display the contents of scalar and address registers (on C Series systems) or general registers (Exemplar systems).

[<process-list>] [<thread-list>] info registers

info space registers in sp r i sp r

Display the space registers (Exemplar systems only).

[<process-list>] [<thread-list>] info space registers

info vregisters in v i vr

Display the vector registers (C Series only).

[<process-list>] [<thread-list>] info vregisters

Searching memory

find memory backward `find m b` *fmb*

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

```
[<process-list>][<thread-list>] 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.

```
[<process-list>][<thread-list>] 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 [<i><regular-expression></i>]		
info args	in ar	<i>args</i>
Display arguments of the current routine. [<i><process-list></i>][<i><thread-list></i>] info args		
info break	in br	<i>b?</i>
Display all existing breakpoints. [<i><process-list></i>][<i><thread-list></i>] info break		
info cregisters	in cr	<i>icr</i>
Display the communication registers (C Series only). [<i><process-list></i>] info cregisters		
info cxdb	in cx	<i>icx</i>
Display the status of the current CXdb session. info cxdb		
info default environment	in de	<i>ide</i>
Display all default environment variables. info default environment [<i><regular-expression></i>]		
info dirpath	i di	
List alternate CTI directory paths or the names of object files that match a regular expression. info dirpath [<i><regular-expression></i>]		
info dynamicobject	in dy	<i>idy</i>
Display memory segments of dynamically loaded objects (C Series only). [<i><process-list></i>] info dynamicobject		
info environment	in en	<i>env?</i>
Display all process environment variables. [<i><process-list></i>] info environment [<i><regular-expression></i>]		

- info errno** in er *ier*
 Display the system error message received by the process.
 [*<process-list>*][*<thread-list>*] info errno
- info event** in event *e?*
 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 expression** in ex *describe, whatis*
 Display the characteristics of the specified language expression.
 [*<process-list>*][*<thread-list>*] info expression *<language-expression>*
- info formatting** in fo *ifo*
 Display the settings for memory display formats.
 [*<process-list>*] info formatting
- info frame** in fr *ifr*
 Display a stack frame.
 [*<process-list>*][*<thread-list>*] info frame [*<frame-specifier>*]
- info frame at** in fr a *ifra*
 Display the stack frame at the specified address.
 [*<process-list>*][*<thread-list>*] info frame at *<language-expression>*
- info history** in h *ih*
 Display the CXdb command history.
 info history [*<command-count>*]
- 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.
 [*<process-list>*][*<thread-list>*] info locals

info macro	in m	<i>im</i>
Display macros.		
info macro [<i><regular-expression></i>]		
info objectmap	in o	<i>io</i>
Display the object map.		
[<i><process-list></i>] info objectmap		
info path	in pa	<i>p+</i>
Display the directories in the search path.		
[<i><process-list></i>] info path		
info process	in pr	<i>p?</i>
Display the status of the process.		
[<i><process-list></i>] info process		
info psw	in ps	<i>ips</i>
Display the processor status word (PSW) register.		
[<i><process-list></i>][<i><thread-list></i>] info psw		
info registers	in r	<i>ir</i>
Display the contents of scalar and address registers (on C Series systems) or general registers (Exemplar systems).		
[<i><process-list></i>][<i><thread-list></i>] info registers		
info scope	in sc	<i>where</i>
Display the current scope path.		
[<i><process-list></i>][<i><thread-list></i>] info scope		
info signal	in si	<i>isi</i>
Display signal names, numbers, and settings for signal actions.		
[<i><process-list></i>] info signal [<i><signal-specifier></i>][, ...]		
info sourceunit	in so	<i>iso</i>
Display the specified source unit.		
[<i><process-list></i>] info sourceunit <i><source-unit></i>		
info stack	in st	<i>ist</i>
Display information about the process stack.		
[<i><process-list></i>][<i><thread-list></i>] 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 in C programs.

`[<process-list>] [<thread>] info type [<regular-expression>]`

info vregisters in v *i vr*

Display the vector register (C Series only).

`[<process-list>] [<thread-list>] info vregisters`

info watch in w *i w*

Display all watchpoints.

`[<process-list>] info watch`

Logging input, output, and messages

You can log the following types of information:

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

To append to an existing log file, use redirection operators. A viewport can be a file, the CXdb Command window (X Windows mode only), or `stderr`, `stdin`, and `stdout` in line mode.

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 for `cmdlog`.

```
clear logging
```

clear noclobber cl n

Allow overwriting of existing log files or creation of new files for appending for all viewports.

```
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 redefine the viewport list for cmderr.

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

set cmdlog se cmdl

Clear and redefine the viewport list for cmdlog.

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

set logging se l

Enable logging for cmdlog.

```
set logging
```

set noclobber se n

When noclobber is set, CXdb responds with an error message if you try to overwrite an existing log file.

```
set noclobber
```

Redirecting process I/O

To redirect process I/O, use a shell redirection operator with the `CXdb run` command. Use a backslash (\) before the redirection operator to distinguish it from a `CXdb` redirection operator. 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
<code>></code>	Overwrite existing command output file, unless <code>noclobber</code> is set.
<code>>></code>	Append new output to an existing file. An error occurs if the specified file does not exist.
<code>>!</code>	Overwrite existing output file, regardless of overwrite protection.
<code>>>!</code>	Append new output to an existing file. Create the file if it does not exist.

For error and information message output

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

Setting and clearing CXdb defaults

clear autocreate

cl a

In X Windows mode, disable the dynamic creation of Source Code windows.
Not available in line mode.

clear autocreate

clear echo

cl ec

Disable echoing of input from initialization files and command files.

clear echo

clear noclobber

cl n

Allow overwriting of existing log files or creation of new files for appending for all viewports.

clear noclobber

clear seq

cl se

Clear the sequential mode (SEQ) bit (C Series only).

[<process-list>] [<thread-list>] clear seq

clear sqs

cl sq

Clear the sequential store enable (SQS) bit (C Series only).

[<process-list>] [<thread-list>] clear sqs

set autocreate

se a

In X Windows mode, enable dynamic creation of Source Code windows.

set autocreate

set echo

se ec

Enable echoing of input from command files and initialization files.

set echo

set evalopts fpmode

se ev f

Set the floating point mode for evaluating expressions (C Series only).

set evalopts fpmode { ieee | native | dual }

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.

```
[<process-list>] [<thread-list>] set format <memory-unit> <format>
```

set memory se m

Set the unit size for displaying memory.

```
[<process-list>] [<thread-list>] set memory <memory-unit>
```

set noclobber se n

When noclobber is set, CXdb responds with an error message if you try to overwrite an existing log file.

```
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 remotewd se r

Set the remote working directory.

```
[<process-list>] set remotewd <directory-specifier>
```

set seq se se

Set the sequential mode (SEQ) bit (C Series only).

```
[<process-list>] [<thread-list>] set seq
```

set shell se sh

Set the type of shell invoked from within CXdb.

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

set sqs se sq

Set the sequential store enable (SQS) bit (C Series only).

```
[<process-list>] [<thread-list>] set sqs
```

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 fixed sched cl d f s

Disable fixed scheduling in the default settings (C Series only).

```
clear default fixed sched
```

clear default handler cl d h

Clear the default handler for all eventpoints.

```
clear default handler
```

clear default remotewd cl de re

Clear the default remote working directory (C Series only).

```
clear default remotewd
```

clear step cl st
 Reset the stepping granularity to the default setting.
`[<process-list>] clear step`

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 fixed sched se de fi s
 Enable fixed scheduling in the default settings (C Series only).
`set default fixed sched`

set default format se de fo
 Set the default formats for displaying memory.
`set default format <memory-unit> <format>`

set default fpmode se de fp
 Set the default floating point mode for new processes (C Series only).
 On Exemplar systems, the floating-point mode is IEEE.
`set default fpmode { ieee | native | dual }`

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 remotewd se de re

Set the default remote working directory.

```
set default remotewd <directory-specifier>
```

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 fixed sched cl f s cfs

Disable fixed scheduling in the process settings.

```
[<process-list>] clear fixed sched
```

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 fixed sched	se fi s	sfs
Enable fixed scheduling in the process settings (C Series only).		
[<process-list>] set fixed sched		
set fpmode	se fp	
Set the floating point mode for the process (C Series only).		
[<process-list>] set fpmode { ieee native }		
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>
```

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>
```

dirpath dir

Specify an alternate directory path for the CTI data files.

```
dirpath <original-directory> <alternate-directory>
```

info dirpath i di

List alternate CTI directory paths or the names of object files that match a regular expression.

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

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
```

remove dirpath rem di

Remove CTI directory paths created with the `dirpath` command.

```
remove dirpath [<original-directory>]
```

set directory se di

Set the process working directory.

```
[<process-list>] set directory <directory-specifier>
```

set remotewd se r

Set the remote working directory.

```
[<process-list>] set remotewd <directory-specifier>
```

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>
  [ & ]
```

signal thread

sig t

Send a signal to a specific thread of the process, and continue execution of that thread.

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

Working with threads

clear default fixed sched cl d f s
Disable fixed scheduling in the default settings (C Series only).

```
clear default fixed sched
```

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 default fixed sched se de fi s
Enable fixed scheduling in the default settings (C Series only).

```
set default fixed sched
```

set fixed sched se fi s sfs
Enable fixed scheduling in the process settings (C Series only).

```
[<process-list>] set fixed sched
```

set threads se th
In X Windows mode, associate windows with particular threads.

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

signal thread sig t
Send a signal to a specific thread of the process, and continue execution of that thread.

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

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 X Windows 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>]
```

load object l o

Load CTI data for an object file that has been dynamically loaded (C Series only).

```
[<process-list>] load object <file-name> <text-address>  
<data-address> <tdata-address> <bss-address> <tbss-address>
```

shell sh

Invoke a shell.

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

source sou

Execute a CXdb command file.

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

Index

Symbols

.CTI subdirectory 1

A

add cmderr command 48
add cmdlog command 48
add cmdout command 48
add default environment command 53
add default path command 1, 53
add environment command 55
add path command 1, 58
alias command 57
aliases, commands for working with 57
arguments, displaying 15
arguments, viewing for current routine (`info args` command) 38
Assembly Code window
 creating 5
 mouse and keyboard shortcuts 22
assembly-language code, commands for viewing 37
attach command 27

B

backtrace 13
backtrace command 13, 41
break instruction command 28
break line command 28
break routine command 28
break source command 28
breakpoints
 commands, listed and described 9
 deleting (`remove event` command) 32
 disabling (`disable event` command) 32
 displaying information about 31
 enabling (`enable event` command) 32
 setting, commands for 28

C

cd command 58
clear autcreate command 51
clear default environment command 53
clear default fixed sched command 53, 60
clear default handler command 33, 53

clear default remotewd command 53
clear echo command 51
clear environment command 55
clear fixed sched command 55
clear handler command 33, 55
clear logging command 48
clear noclobber command 48, 51
clear seq command 51
clear sqs command 51
clear step command 35, 54
clear typehandler command 33, 55
cmderr 48
cmdlog 48
cmdout 48
command line editing 19
Command window
 creating 5
 description and *illus* 7
command-line editing key bindings 21
commands 26
 aliases 57
 breakpoints, setting 28
 directory 58
 displaying data 38
 eventpoints
 disabling 32
 displaying information about 31
 enabling 32
 handlers 33
 removing 32
 setting 30
 explanation of syntax for 25
 handlers for eventpoints 33
 help 26
 history 61
 info 44
 macros 57
 miscellaneous 62
 modifying data 38
 path 58
 printing data 39
 process execution 34
 quitting CXdb 26
 registers, viewing 42
 searching memory 43
 searching source code 43
 signals 59
 specifying a file to debug 27
 specifying a process to debug 27
 stack frame information 41

- starting CXdb 26
- stepping 35
- threads 60
- tracepoints, setting 29
- viewing assembly-language code 37
- viewing source code 37
- compiling for CXdb 1
- continue command 11, 34
- copy command 40
- core command 27
- core files, debugging 3, 4
- csd command 57
- cxdb command 26
 - shell command-line options 2, 26

D

- data
 - commands for displaying 15
 - displaying 15, 38
 - modifying 15, 38
- debug core command 4, 27
- debug exec command 3, 27
- debug proc command 4, 27
- debugger variables 61
- defaults
 - CXdb, commands for setting and clearing 51
 - process, commands affecting all new processes 53
 - process, commands affecting existing processes 55
- detach command 27, 34
- directory commands 58
- dirpath command 58
- disable event command 32
- disable eventtype command 32
- disassembly command 37
- display disassembly command 37
- display examine command 40
- display routine command 37
- display source command 37
- display stack command 41

E

- echo command 33, 62
- edit command 62
- Edit window, creating 5
- editing the command line 19
- enable event command 32
- enable eventtype command 32
- evaluate command 38, 40
- event exec command 30
- event join command 17, 30, 60
- event modify command 30
- Event Point dialog 10
- event reached instruction command 30
- event reached line command 30

- event reached routine command 30
- event reached source command 30
- event relation command 30
- event signal command 31, 59
- event spawn command 17, 31, 60
- eventpoints
 - disabling 32
 - displaying information about 31
 - enabling 32
 - eventpoint handler commands 33
 - ignore counts, setting 32
 - removing 32
 - setting, commands for 30
- examine command 40
- executable command 4, 27
- executable files, loading for debugging 3
- exiting CXdb 26
- expressions
 - evaluating (evaluate command) 38
 - printing 39

F

- fill command 40
- find memory backward command 43
- find memory forward command 43
- find source command 43
- finish command 12, 35
- fixed scheduling
 - commands 60
 - discussed 17
 - enabling with -F option 26
- floating-point mode, setting 16
- frame command 41

G

- gdb command 57
- get command 40
- goto address command 34
- goto line command 34
- goto source command 34
- granularity
 - defined 12
 - for stepping commands 35

H

- handlers, eventpoint (commands) 33
- help command 26
- Help system, invoking 26
- Help window
 - creating 5
 - description and *illus* 20
- history commands 61

I

- if command 33, 62
- info alias command 44, 57
- info args command 38, 44
- info break command 31, 44
- info control registers command 42
- info cregisters command 42, 44
- info cxdb command 44
- info default environment command 44
- info dirpath command 44, 58
- info dynamicobject command 44
- info environment command 44
- info errno command 45
- info event command 31, 45
- info eventtype command 31, 45
- info expression command 38, 45
- info floating point registers command 42
- info formatting command 15, 39, 40, 45
- info frame at command 41, 45
- info frame command 41, 45
- info history command 45, 61
- info line command 38, 45
- info locals command 38, 45
- info macro command 46, 57
- info objectmap command 46
- info path command 46, 58
- info process command 46
- info psw command 42, 46
- info registers command 42, 46
- info scope command 46
- info signal command 59
- info signals command 46
- info sourceunit command 38, 46
- info space registers command 42
- info stack command 46
- info symbols command 38, 47
- info threads command 17, 47, 60
- info trace command 31, 47
- info type command 47
- info vregisters command 42, 47
- info watch command 31, 47
- input, logging 48
- invoking CXdb
 - line mode (tty interface) 2
 - X Windows mode 2

K

- keyboard shortcuts 21
- kill process command 11, 34

L

- line mode (tty interface) 2

- commands equivalents for X Windows information display 6
- list command 6, 37
- load object command 62
- loading a program to debug 3
- local variables, displaying 15
- logging
 - commands 48

M

- macro command 57
- macros, commands for working with 57
- memory
 - commands for modifying 40
 - commands for searching 43
 - commands for viewing 40
 - displaying 16
- Memory Display window, creating 5
- messages, logging 48
- modifying program data 15, 38
- mouse and keyboard shortcuts 21

N

- next command 12, 35
- next instruction command 12, 35
- next over command 35

O

- output, logging 48

P

- path commands 58
- print command 15, 39
 - formatting options 16
- printing data
 - aligning (padding with zeros) 16
 - array elements, setting number to print 16
 - array slices 15
 - formatting options (for memory units) 16
 - precision, setting 16
 - print options 16
 - program data 15
- process
 - defaults
 - commands affecting all new processes 53
 - commands affecting existing processes 55
 - input and output (process interface window) 6
 - specifying a process to debug 4
- process execution
 - commands 34

- controlling 11
- stepping 12
- process object, defined 3
- put command 40
- pwd command 58

Q

- quit command 26
- quitting CXdb 26

R

- recall command 61
- redirection
 - CXdb command output 50
 - CXdb error and information messages 50
 - CXdb redirection operations 50
 - process I/O 50
- register contents, displaying 16, 42
- register windows, creating 5
- remove alias command 57
- remove cmderr command 48
- remove cmdlog command 48
- remove cmdgout command 48
- remove default environment command 54
- remove default path command 54
- remove dirpath command 58
- remove environment command 55
- remove event command 32
- remove eventtype command 32
- remove path command 56
- remove variable command 61
- rerun command 11, 34
- resume command 11, 33, 34
- return command 34
- run command 35
- running your program 11

S

- set autocreate command 51
- set cmderr command 49
- set cmdlog command 49
- set default environment command 54
- set default fixed sched command 54, 60
- set default format command 54
- set default fpmode command 54
- set default handler command 33, 54
- set default memory command 54
- set default path command 1, 54
- set default pshell command 55
- set default remotewd command 55
- set default step command 36, 55
- set directory command 58

- set echo command 51
- set environment command 56
- set evalopts fpmode command 51
- set evalopts iprecision command 51
- set evalopts rprecision command 52
- set fixed sched command 56, 60
- set format command 41, 52
- set fpmode command 56
- set handler command 33, 36
- set ignore command 32, 36
- set logging command 49
- set memory command 41, 52
- set noclobber command 49, 52
- set path command 1, 56
- set printopt padding command 52
- set printopts maxarray command 39
- set printopts maxarray command 52
- set printopts nopadding command 52
- set printopts nopadding command 39
- set printopts padding command 39
- set printopts precision command 39, 52
- set remotewd command 52, 58
- set seq command 52
- set shell command 53
- set signal command 59
- set sqs command 53
- set step command 12, 36
- set threads command 60
- set typehandler command 33, 53
- shell command 62
- shell window, creating 5
- shortcuts, mouse and keyboard 21
- signal process command 59
- signal thread command 59, 60
- signals, commands 59
- Source Code window
 - commands for searching 43
 - creating 5
 - description and *illus* 8
 - manipulating eventpoints with the mouse 10
 - mouse and keyboard shortcuts 22
- source code, commands for viewing 37
- source command 62
- source units, displaying for a line (info line command) 38
- stack backtrace 13
- Stack Frame Description dialog 14
- stack frames 14
 - commands 41
- Stack Trace window
 - creating 5
 - described 13
 - mouse and keyboard shortcuts 23
- starting CXdb 2, 26
- step command 12, 36
- step instruction command 12, 36
- step over command 36

stepping commands 12, 35
symbol names, displaying 15
symbols, viewing program (info symbols
command) 38

T

Thread Activity window
 creating 5
 description and *illus* 18
 mouse and keyboard shortcuts 23
threads
 commands 17
 commands for manipulating 60
 displaying information about 17
trace instruction command 29
trace line command 29
trace routine command 29
trace source command 29
tracepoints
 commands for setting 29
 commands listed 9

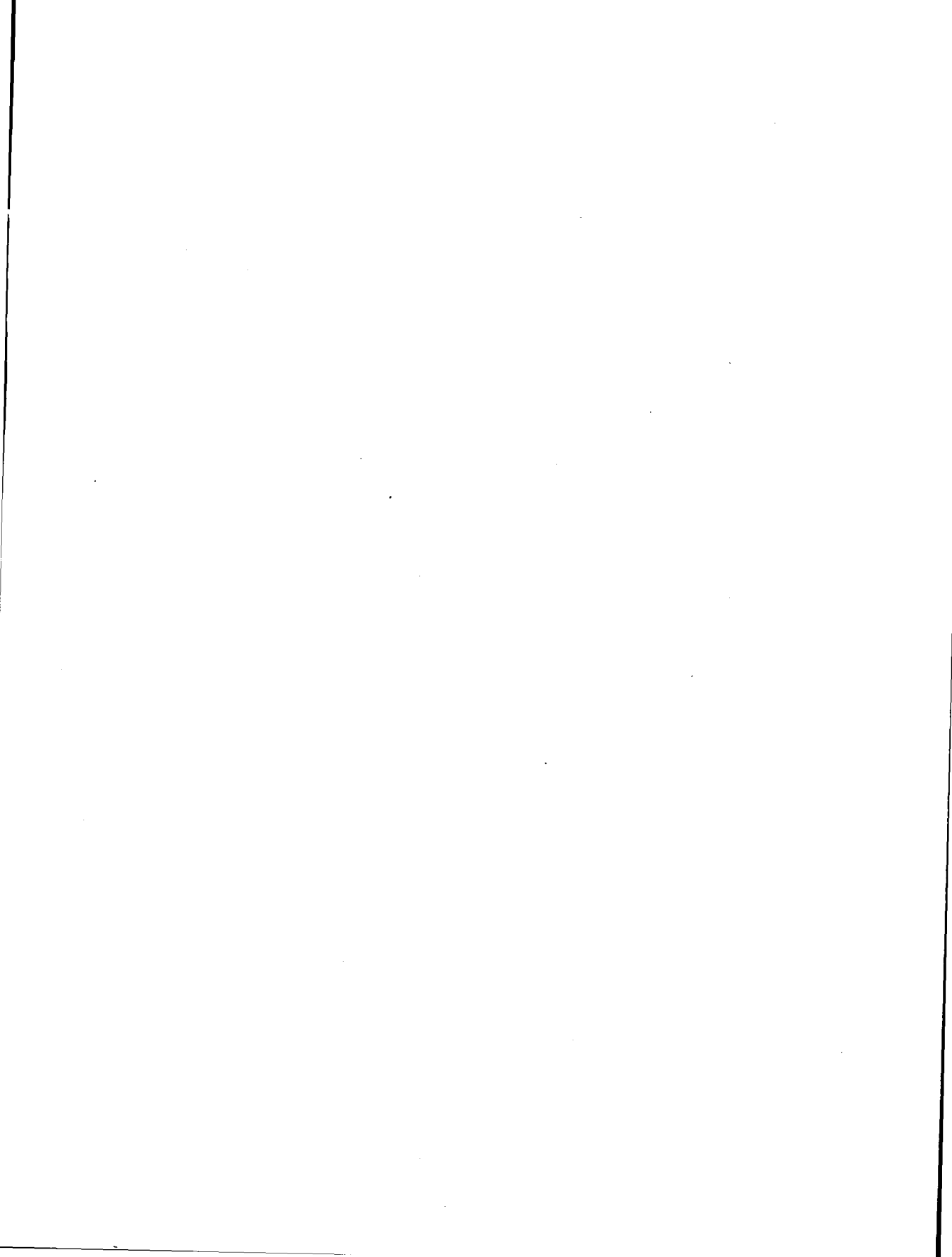
V

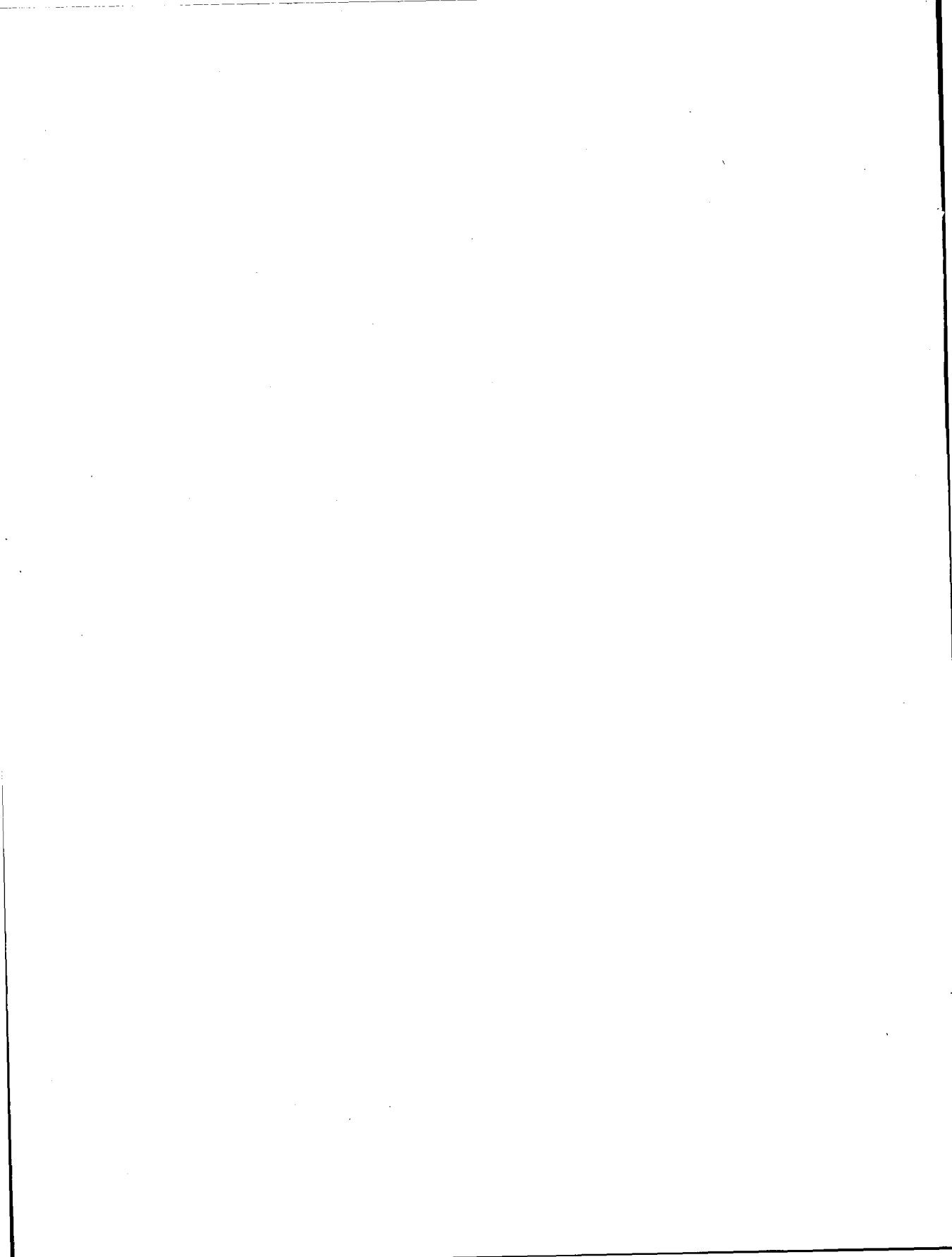
variables
 debugger 61
 program
 viewing local (info locals command) 38

W

watch command 9, 31
watchpoints 9
windows
 Command window 7
 creating 5
 Help 20
 listed and described 6
 Source Code window 8
 Stack Trace 13
 Thread Activity window 17

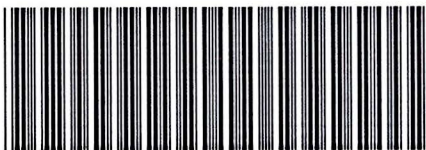






ORDER NUMBER
DSW-474

DOCUMENT NUMBER
710-015630-002



CONVEX
PRESS