Release Notice
CONVEX CXdb V2.0

Document No. 710-014830-009

December 1992

CONVEX Computer Corporation
Richardson, Texas USA

Release Notice Notice, CONVEX CXdb V2.0

Printed in the United States of America

1

Release Notice

Introduction

This document describes the V2.0 production release of the CONVEX Visual Debugger, *CXdb*. It highlights new functionality and known problems in CXdb. Always refer to this release notice before reporting questions or problems with CXdb; your questions may be answered here. All outstanding bugs are listed at the end of this document. Please review this list to avoid rediscovering known problems.

The remaining sections in this chapter describe the contents of this release.

- Contents of this distribution.
- Enhancements to CXdb.
- Notes and warnings about the product.
- Use of CXwindows.
- Source line numbers in C programs.
- Known hardware problems.
- Known software problems.

For instructions on installing CXdb, see *Installation Procedure, CONVEX CXdb V2.0.*

Contents of This Distribution

The distribution package for this release of CONVEX CXdb consists of:

- This Release Notice
- CXdb Installation Procedure
- CXdb Reference Manual (two-volume set)
- CXdb User's Guide
- CXdb Quick Reference
- Distribution media for the software

The specific contents of the software distribution is described in the following tables:

Table 1-1: Software for USA and International Distribution

Qty	Part Number	Description	
1	710-008215-008	CONVEX CXdb, V2.0	

Table 1-2: Documentation Release Package

Qty	Order Number	Part Number	Description
1	N/A	710-014930-009	Installation Procedure, CONVEX CXdb V2.0
1	N/A	710-014830-009	Release Notice, CONVEX CXdb V2.0
0	DSW-473	710-015530-002	CONVEX CXdb User's Guide
5	DSW-474	710-015630-001	CONVEX CXdb Quick Reference
1	DSW-477	710-015430-003	CONVEX CXdb Reference, Command and Parameters
1	DSW-478	710-024130-000	CONVEX CXdb Reference, Concepts and Messages

Enhancements

CXdb V2.0 is the first widely distributed debugger release since CXdb V1.1. CXdb V1.2 has also been released, but on a restricted basis. Listed below are the enhancements since CXdb V1.1. After that, the enhancements since V1.2 are listed.

This release adds enhancements beyond CXdb V1.1 in the following areas:

- **GUI enhancements:** The management of windows in the user interface has been changed to better accommodate the degree of parallelism possible in ConvexRTS/rtk systems and in newer C-series systems. Window management has changed from a window-per-thread paradigm to a methodology that allows multiple threads to be associated with a window (when feasible). A thread selection capability has been added to each window to allow user selection of which threads to associate with a window. See the "threads" menu selection item in the window's pull-down menu in the source, disassembly, examine, and register windows. Commands that have been added or modified include:

clear autocreate	display disassembly	display examine	display routine
display source	display stack	set autocreate	set threads

  See the online documentation or reference manual for more information on these commands.

- **Remote debugging:** CXdb has been enhanced to support remote debugging. Remote CXdb servers have been added for ConvexOS and ConvexRTS/rtk systems. The ConvexOS CXdb server is distributed with CXdb and the ConvexRTS/rtk server is available from the ConvexRTS Realtime Group.

  Note: User authentication is left up to the remote server. The CXdb server for ConvexOS uses a trusted host method much like *rsh*, and it relies on an appropriate host entry in */etc/host.equiv* and an appropriate host entry in the user's *~/.rhosts*.

  Commands that have been added or modified include:

attach	clear default remotewd	core	debug core
debug exec	debug proc	executable	set default remotewd

set remotewd

See the online documentation or reference manual for more information on these commands.

- **ConvexRTS/rtk support:** For debugging ConvexRTS/rtk applications, a suite of commands has been added to control and examine attributes of ConvexRTS/rtk applications and tasks. Also, a few existing commands have been modified for ConvexRTS/rtk applications. These commands are:

info process	info task	info threads	set affinity
set app_name	set app_priority	set filesys	set kernel_mode
set max_size_mem	set max_tasks	set priority	set stack_size
set tpd_size			

  See *New Features in CONVEX CXdb V1.2* (DSW-476) for more information on these commands.

- **Saving program data:** The **put** and **get** commands have been added to allow you to save program data into user-specified files and then reload such data back into a running process. See the online documentation or reference manual for more information on these commands.

- **Miscellaneous:** In addition, a few other commands have been added. They include the **list** and **info path** commands. See the online documentation or reference manual for more information on these commands.

In addition to these enhancements, CXdb V2.0 includes a significant number of bug fixes and performance enhancements that make CXdb a more robust and effective tool. In particular:

- The quantity of informational messages related to the debugger data files has been significantly reduced.
- The **run** command has been changed to allow arguments without quoting.

This release adds enhancements beyond CXdb V1.1 in the following areas:

- **Line mode & csd compatibility:** To facilitate usage on CRTs, a CRT line mode has been added to CXdb V2.0. To use CXdb in line mode, pass the *-l* flag to CXdb. This mode reads and executes one command at a time using the standard UNIX tty line editing capabilities.

  To help users familiar with csd, a csd compatiblity macro package has been added to CXdb. The csd compatibility macros can be instantiated by invoking the *csd* command in CXdb or by passing the *-csd* flag to CXdb. The *-csd* flag also starts CXdb in line mode, giving CXdb a *csd* look and feel. See the reference manual page on *csd* for additional details.

- **Hardcoded Paths:** Hardcoded paths eliminate the need (in most cases) to set up directory search paths in CXdb. The FORTRAN and C compilers have been enhanced to encode the absolute pathname of each compiled source file in the object module for the compiled file. Embedding hardcoded paths into the user's object module allows CXdb to determine the location of any CDI data files or source code files associated with your executable. Using hardcoded paths is much more efficient than searching through directory search paths.

  If for any reason the data or source files are moved after the compilation, CXdb provides a directory path translation mechanism to allow the hardcoded directory path to be translated from the orignal directory in which the application was compiled to the new directory where the data files currently reside. See the online documentation or reference manual for more information on the following commands:

dirpath info dirpath remove dirpath

- **Efficiency:** In addition to functionality increases, performance tuning has enhanced the initial loading of CDI data files. Data file loading of the namespace tables, which are needed to resolve any external symbols, typically occurs early in the debugging process. This was a significant bottleneck that has been sped up by a factor of more than four.

Notes and Warnings

This subsection contains general useful information and words of caution about the product.

- Unique core files are an optional feature that allows you to keep any core files generated by CXdb separate from core files generated by your application. Without this option enabled, the CONVEX development team will be unable to properly diagnose many of the problems that may occur.

  To enable unique core file names, perform the following operations:

  — Add the entry **tune cpu enable_unique_core = 1** to the file */mnt/os/bootcmd.local* on the SPU.
  — Reboot the system.

- The debugger is stamped with a unique serial number. At runtime, the debugger checks the serial number of the machine on which it is running. If the serial number does not match the expected one, a user error message is given and execution is aborted.

- This release of CXdb comes with an online tutorial. This tutorial runs only under the X window system. The tutorial can be accessed from the hypertext links within the help window when running under the X window system.

- For the Domestic distribution, this release requires installation of the following:

  — CONVEX FORTRAN V8.0, or CONVEX C V4.3 or V5.0. These compilers generate instrumented code for use with CXdb.
  — ConvexOS V10.0, V10.1, or V10.2.
  — To use the X window system features of CXdb, you do not need CONVEX CXwindows. See the section *Use of CXwindows* for further details.

Use of CXwindows

While it is preferable that CXwindows be in place for use with CXdb, CXwindows is no longer required. The CXdb installation installs a version of *xterm* on those systems without CXwindows. The *xterm* and the app-defaults file for CXdb allow you to run CXdb in X mode on any X server. You need to obtain a version of the XKeysymDB, which is available from most any X server.

Source Line Numbers in C Programs

CXdb does not support full symbolic debugging of the source code in #include files. In some cases, the line numbers reported by CXdb for such include files will not be accurate.

In addition, the #line preprocessor directive changes the sequence of line numbers used by the compiler. Therefore, if you use this directive in any of your C source code, it will also change the line numbers reported by CXdb.

Known Hardware Problems

There is a problem on C-3800 systems involving trace traps. When taking a trace trap across a return or call instruction, the trace trap is not delivered to the process being debugged until two instructions have been executed. Normally only one instruction should be executed before the trace trape is delivered. Trace traps occur when single-stepping or on return to a function that has the trace trap bit set in its saved PSW. This problem not only affects the behavior of the command **step instruction** across calls and returns, but also affects stepping by expresssion and statement. Stepping by expressions or statements is implemented via instruction stepping, and this hardware problem will affect stepping or nexting into or out of functions.

On C-3400 systems, differences in instruction timing can occasionally cause unexpected behavior if you try to single-step a process across a routine boundary. To avoid this problem, set a breakpoint at the target routine and use the **continue** command to continue execution to that point.

On C100 Series systems, do not use the **signal process** command if your process is stopped exactly at a breakpoint. Either remove the breakpoint or step the process past the breakpoint, then you can use the **signal process** command.

Known Software Problems

This section contains the known problems with CONVEX CXdb software and compiler instrumentation as of this release notice. Any problems reported after this date may not be reflected in this document. Please refer to this list before reporting a problem to ensure that it has not been reported previously.

Multi-threaded Applications

In some cases, a multi-threaded application can cause a kernel panic in the operating system. This is a known bug that will be fixed in a future release of ConvexOS.

B Priority Bugs

- X-19921

When a program segmentation faults in a library routine, the source window gets confused. Even though it knows where the source for the library routine is, it stays in the file that main() is in. When I create a new Source Window, everthing is OK.

Example:

Ok, here is what I do:

- startup cxdb with a program
- type 'run' in the Command Window
- the program segmentation faults in a library routine
  (The library is debuggable and its path is in CXdb's path)
- The Source window puts me on the correct line number, but in the wrong file. It is still in the main() .c file.

- When I create a new Source Window, it puts me in the right place.

- X-20406

CXdb cannot properly step over a call to longjmp. The stack manipulations that CXdb performs are undone when longjmp reinstates its saved stack frame.

Currently, when stepping into a longjmp, the program will continue until the next breakpoint. The correct solution should place the user at the location of the setjmp.

Example in comments

Example:

See code in X20406 for example

- X-26686

After creating a disassembly window, if one modifies its shape (X/motif interface), does a stepi, the remodifies its shape, then the pc pointer disappears.

Rpt-by:
Compile code from the 'Example' section and then follow the CXdb session as it shows in the comment section.

Example:

```
      program humma
      real*4  a(100), aa(100)
        do i=1,100
        aa(i) = 1
        enddo
      call foobar(a,aa,100)
      end
c
      subroutine foobar(x,y,n)
      real*4 x(*), y(*)
      integer n
        do j=1,n
        do i=1,n
        x(i) = 2 + y(i-1)
        enddo
        y(j) = 5
      enddo
      return
      end
```

- X-26702

When printing a string (char *) in C and the string is longer than the current MAXARRY setting, the string is truncated, but no elipsès is printed to inform the user that more data is available.

Example:

Compile the following program:

```
#include <stdio.h>

main() {
   char *s1 = "this is a long string, long enough to go over MAXARRAY";

   printf( "s(%d) = '%s'

n", strlen( s1 ), s1 );
   exit(0);
}
```

call up cxdb, break on line 6 (the printf), and the print s1. The value will be truncated and no elipses is printed.

- X-26706

CXdb reports the wrong value for a variable that has two active liveness ranges: home location as a local and a register. The value from the home location is being reported instead of the register.

Example:

Bring up CXdb on itself. Use the 'list' command to get into the cmdList function. Then get to the line after:

```
    if (csw) curSource = csw->getSourceFile();
```

Check on the value of __1curSource, it's the wrong one.

- X-26801

If an examine window is pulled up, the address to be examined set to something meaningful (like the address of some variable in your program), a breakpoint set, a "run" issued, a "rerun" will blank out the examine window.

Example:

```
main()
{
        float a[100];

        foobar(a,100);
```

```
}
void foobar(float *a, int n)
{
        int i;
        while( n > 0 )
                        a[i] = 0.;
        return;
}
```

- X-26839

When using subroutines written in assembly or executing code that is generated on the fly, the watch breakpoint fails. Although CXdb can follow the instruction flow (as shown with disassembly window), values modified in a subroutine written in assembly or generated on the fly (as is the case with SAS Institute) are not detected by cxdb *until* control returns to the caller. This is a royal pain when attempting to isolate memory corruption problems.

Example:

Makefile:

```
CFLAGS=         -cxdb -no
a.out:  main.o foo.o
        cc $(CFLAGS) main.o foo.o
foo.o:  foo.s
        as -cxdb foo.s
```

Debug session:

```
bi main
run
watch &a[3]
continue
continue
continue
```

(Please see the 'Comment section' for a complete session and source files).

- X-26986

User gets:'Fatal internal error' when debugging wiyh cxdb.The message is:q

Dynamic memory has been exhausted, terminating!

cxdb: FATAL INTERNAL ERROR!
A core dump will be produced. Please save this for submission with
Contact report to the Convex Technical Assistance Center.

Example:

The compressed cxdb core file is on mozart in /mnt/pieptea/pr28922/cxdbcore.Z.

Please, see the 'Comment' section.

C Priority Bugs

- X-17944

When I set an exec eventpoint with an eventpoint handler, my process receives a segmentation fault. The cxdb prompt then disappears and I get an error message saying that there is an invalid command in the eventpoint handler (even though the handler is valid). Even though there is no longer a prompt, I can still enter commands.

Example:

```
(CXdb) event exec {print "hello";}

#3: exec on [#0], Enabled, ignore 0/0
   {
   print "hello";
   }
Eventpoint 3 created
(CXdb) run
Starting process [#0]: a.out

INFO: 89
Process [#0] exec'ed.

Process [#0/0] received signal 11 (Segmentation fault) at [0x80032d28]
___ap$sbrk
(CXdb) cont

ERROR: 108
Invalid command in eventpoint handler.  Handler aborted.
```

- X-19047

Maryland:
Reshaping the Xterm window smaller than the command window size get strange behavior. The screen is not updated properly.

Example:

1. Run cxdb -nw in an Xterm.
2. Fill the command window with output (info bind is good).
3. Make the Xterm smaller than the size of the command window
4. type ^L.

5. Hit return a few times.

- X-19791

Stepping using loop granularity doesn't work correctly.

Example:

```
debug exec cxdb04.e
set step loop
break line 32
run
remove event *
step over loop 2

Stepping process [#0/*] by 2 loops outside current loop
Process [#0/0] stopped stepping at [0x800013f2] CXDB04 in cxdb04.f line 32
```

If the process is really stepping by loops, it shouldn't step at line 32, but at the entrypoint to a loop.

- X-21856

When running CXdb in CXwindows, CXdb does not respond to a breakpoint that has an eventpoint handler until the mouse cursor is placed in the Command Window. It appears that the Command Window is waiting for an X event, and not immediately responding to signals from the debugged process. This results in a delay between when an eventpoint is reached and when it is reported by CXdb.

Example:

```
% cxdb

(cxdb) debug exec a.out
(cxdb) break routine myproc {print "got event"; resume;}
(cxdb) run
```

Move mouse cursor anywhere outside of the Command Window. Even though the breakpoint is hit in the program, it won't be acknowledged by CXdb until the mouse cursor is put inside the Command Window where CXdb will respond.

- X-23531

When cxdb encounters a FPE signal, somehow an illegal instruction signal (SIGILL) is generated. Subsequently execution of the process being debugged cannot be continued. set signal SIGILL nostop, nopass, doesn't seem to help. Compile the following example with -fi.

Example:

```
#include <signal.h>
#include <math.h>
#include <stdio.h>
#include <machine/psw.h>
main()
{
        int foo();
        double temp, a=1.0, b=0.0;
        signal(SIGFPE, foo);
        temp = a/b;
        printf("temp = %f
n",temp);
}
foo(sgm, code, sgct)
int sgm, code;
struct sigcontext *sgct;
{
        printf ("Signal rcvd: %d
n",sgm);
        printf ("Code rcvd: %d
n", code);
        sgct->sc_ps &=
                        ~(PSW_AIV|PSW_ADZ|PSW_SIV|PSW_SDZ|PSW_UN|
                          PSW_OV|PSW_RO|PSW_FDZ|PSW_FIN);
      *((int*) ((char *) (sgct) +sizeof(*sgct)+ 4)) = sgct->sc_ps;
        return;
}
```

- X-23655

  After I add a path, I bring up a source window in the new directory I just specified. I try setting a break point and I get a profuse # of errors. Then if I set the exact same break point again, it works. Just seems kinda silly. There is a script in /tmp/cxdbbugs/double.scr that can be run on an R5 sun to show what happens using 'xtmexecute double.scr'.

  Example:

  On an R5 sun (or R5 server), add /swenv/twork/xspecter to your path. Then in /tmp/cxdbbugs run 'xtmexecut double.scr' for a replay of the problem. All

- X-25080

  When working with a multi-threaded application with a "spawn" event in place CXdb can sometimes report the process stopping twice with an exiting thread and then show both threads as exiting (a condition which really can't exist). Attempts to step the threads result in the full speed execution until the next eventpoint occurs.

  Example:

  Start CXdb, turn on fixed scheduling, create a spwan event with "event spawn"

and then repeatedly run and continue the process until you see this output:

```
Process [#0/0] stopped by exiting thread at [0x8000153a] INIT in mmunge.f line 23
Process [#0/1] stopped at [0x8000153a] INIT in mmunge.f line 23
(CXdb) Process [#0/0] stopped at [0x8000153a] INIT in mmunge.f line 23
Process [#0/1] stopped by exiting thread at [0x8000153a] INIT in mmunge.f line 23
```

Both threads are now marked as exiting.

- X-26190

  Application doesn't work under batch mode but works under GUI mode with same command file.

  Example:

  See comments.

- X-26415

  The installation of cxdb 2.0 fails on 9.1 systems due to the version of perl. The exact error message output is 'syntax error in file at line 23, next 2 tokens "= scalar"'.

  Example:

  Install a newer version of perl.

- X-26430

  The installation of cxdb will fail on C1's because the execuable cannot be run.

  Example:

```
#installsw -i -d /dev/rmt20
[Installing CXdb v2.0.0.1]
Directory in which to install cxdb subdirectory [/usr/lib/cxdb]?
CXdb requires an activation key.
What are your activation keys? 35207-88735-6249
Done Reading tape
Activating executable /usr/lib/cxdb/bin/cxdb, please wait...Done.
Now testing activation of cxdb, please wait...sh: /usr/lib/cxdb/bin/cxdb: cannot execute
#file /usr/lib/cxdb/bin/cxdb
/usr/lib/cxdb/bin/cxdb:  Convex SOFF c2mp demand paged POSIX executable not stripped native fpmode
```

- X-26585

Using the Maryland windows mode, in the Process Window if line editing keys such as backspace and space bar are used to change input when providing it, the input editing keys are ignored and the original input used. Also, sometimes the editing keys are included in the input.

Example:

c sample program requiring input from stdin

```
program main
read(5,*) i,j
write(6,*) i,j
end
```

- X-26656

When in tcsh, if you enter nothing (just hit enter), then subsequent ^p 's will move you "up" the history stack. In CXdb, it appears that this is not the case. The "history pointer" isn't updated by a "no-op" return as it is with tcsh.

Example:

C220/OS 10.0.134

Enter following (invalid) commands: (note ^p = control p)
(CXdb) foo1
(CXdb) foo2
(CXdb) foo3
(CXdb) ^p
*Note you will see foo3 appear
(CXdb) ^p
*Note you will see foo2 appear
(CXdb) ^u<carriage return>
*Note this clears the line and gives another CXdb prompt
(CXdb) ^p
*Note this shows foo1 !

Do the above in tcsh and you will see foo3 appear at last step.

- X-26703

With online help in Maryland Windows mode, the help text scrolls to the bottom of the page for every help topic you invoke. In addition, the help buffer is not cleared with each new topic. So if you scroll to the top of the buffer, you see part of the previous topic as well as the one just invoked.

Similar behavior occurs in line mode as well.

Example:

Do the following:

```
cxdb -nw
help alias
META-<                  (to scroll to top of buffer)
META-n                  (to move back to command window)
help break line
META-p                  (to move to help window)
META-<                  (to scroll to top of buffer)
```

This should show part of the alias help page and the break line help page. Invoking more help topics causes further corruption to the buffer.

D Priority Bugs

- X-19392

The output of the various "info" commands dealing with events is inconsistent when there are no events defined.

info break, info trace: both emit a blank line
info watch: no line
info event: states "No events currently defined"
info eventtype: states "Status of eventpoints of type...", even when there aren't any.

If there aren't any events of the type specified, the output for all of these commands should be:

(CXdb) info <whatever>

No events currently defined

(CXdb)

Example:

(CXdb) info eventtype break

Status of eventpoints of type Breakpoint:

(CXdb) info break

(CXdb) info event

No events currently defined

(CXdb) info trace

(CXdb) info watch
(CXdb)