CXdb V4.1
Release Notice

B5639-90009

June 1997

Hewlett-Packard Company
Convex Division
3000 Waterview Parkway
P.O. Box 833851
Richardson TX 75083-3851 USA

CXdb V4.1
Release Notice

B5639-90009

CXdb V4.1

Release Notice

June 1997

Introduction

This document describes the visual debugger, CXdb, V4.1. It highlights new functionality and known problems in CXdb. Please review this document before using CXdb.

The remaining sections in this document describe:

- Software requirements
- Contents of this distribution
- Directories and files
- Documentation
- Obsolete features since the last release
- New features in this release
- Installation notes
- Limitations
- Known problems
- Getting assistance and reporting problems

Software requirements

This release of CXdb for Exemplar systems requires installation of the following:

- Hewlett-Packard Exemplar Fortran V1.2 or newer
- Hewlett-Packard Exemplar C V1.2 or newer
- Hewlett-Packard Exemplar C++ V10.23.04 or newer
- SPP-UX V5.2 or newer. This includes the Exemplar assembler, linker (`ld`), and system libraries.

CXdb V4.1

Distribution

The distribution package for this release of CXdb consists of:

- CXdb 4.1 release notice
- distribution media for the software (including online help)

The specific contents of the software distribution:

Part No.	Description
B5639-10006	CXdb V4.1 software, including online help.
B5639-90009	CXdb V4.1 Release Notice.
B5639-90007	CXdb Quick Reference.

Directories and files

The base directory for CXdb (/opt/cxdb) contains the following subdirectories:

- **bin**—Contains the `cxdb` executable and the `setupshdbg` utility. (`setupshdbg` is used to prepare an executable for shared library debugging.)
- **share**—Contains all man pages, online help files, and sources for the tutorial.
- **newconfig**—Contains subdirectories for the ASCII version of CXdb V4.1 Release Notice, CXdb Quick Reference, CXdb aliases, and configuration files (for example, the system-wide initialization file for CXdb,.cxdbinit, and a sample X11 app-defaults file).

Product licensing

CXdb is a licensed product. If you do not acquire a license when running CXdb, contact the system administrator at your site for assistance. Exemplar products are licensed using the FlexLM licensing system. Refer to the *FLEXlm End User Manual* for licensing information.

Documentation

CXdb is documented in the following locations:

- `cxdb(1)` man page
- CXdb V4.1 release notice
- CXdb online help system
- *CXdb Quick Reference* (B5639-90007)

Obsolete features

Shared library support is disabled by default. This support is activated under user control. To debug a shared library use the `-shl` flag on the command line or use the `set shlibdebug` command.

New features

This section describes enhancements and changes to CXdb since the V4.0 release and compiler changes that affect CXdb. Enhancements and changes were made in the following areas:

- Improved support for F90 codes at the f77 level.
- Debugging of MPI codes with shared libraries is allowed.
- Improved robustness for MPI applications.
- Improved performance when stepping or nexting applications.
- New commands for controlling shared library debugging.

Compiling applications for debugging with CXdb

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

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

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

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

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

If you have compiled your program *without* debugging information, you can still debug your program using CXdb. However, in this case only global identifiers and function names can be referenced by their symbolic names. To access local variables and function arguments, you must specify their addresses rather than symbolic names. No source code correlation is available, and shared library debugging is not supported in this case.

NOTE: If you use the shell commands `strip(1)` or `ld -s` to reduce the size of the executable file, all of the information required for symbolic debugging is deleted. In this case, no debugging is possible.

To enable CXdb assembly-level debugging at optimization levels greater than +O0, or to enable debugging of shared libraries in applications not compiled with the -g option, add the following options to your compile line:

```
-Wl,+tools /opt/langtools/lib/end.o
```

You can also link object files that have been compiled at different optimization levels. For example, if myprog.f has been compiled at optimization level +O0 but sub1.f has been compiled at level +O3, you can link the object files using a link step similar to the following:

```
% f77 -g myprog.o sub1.o -Wl,+parallel -lpthread -lcps -lpthread -lail
```

In the above example:

- The -g option generates symbolic debugging information.
- The -Wl option passes the +parallel option to the linker to allow parallelizing optimizations at level +O3.
- The -lpthread, -lcps, and -lail options link the appropriate libraries for the +O3 optimizations. To enable proper linking of the libraries, use the -lpthread option twice, in the order shown above.

NOTE: In files that are compiled at optimization levels above +O0, CXdb may be limited to debugging at the assembly-language level.

If the executable is compiled with Exemplar compilers, sbrk is not included in the a.out unless needed in the code. With statically linked executables, you may see the following error messages:

```
ERROR A222: Cannot allocate memory without "sbrk"
linked in the process.

ERROR A226: Cannot allocate memory for string literal.
```

If you encounter this problem, use the following command to extract sbrk.o from the library, then include sbrk.o on your link command line:

```
% ar -x /lib/libc.a sbrk.o
% f77 -g -o test test.f sbrk.o
```

Compiling and linking in a single step

To compile and link applications for CXdb in a single step:

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

In the above example, the `-g` option generates symbolic debugging information.

This does not apply to Fortran 90 applications. Refer to "Compiling and linking Fortran 90 applications for CXdb."

Compiling and linking in separate steps

To compile and link applications for CXdb in two separate steps:

```
% f77 -g -c myprog.f sub1.f sub2.f
% f77 -g myprog.o sub1.o sub2.o
```

In the above example, the `-g` option generates symbolic debugging information and is needed in both steps. The `-c` option suppresses the link phase. This does not apply to Fortran 90 applications.

Compiling and linking Fortran 90 applications for CXdb

To compile and link a Fortran 90 application for CXdb in a single step:

```
% f90 -g foo.c bar.o -Wl,+tools
```

In the above example, the `-g` option generates symbolic debugging information. The `-Wl,+tools` option is required for debugging with CXdb.

Invoking `ld` directly

If you are compiling and linking an application for CXdb in two separate steps, and are invoking `ld` directly, use a link line similar to the following:

```
% ld +tools <options> <files> /opt/langtools/lib/end.o
```

In the above example:

- The `+tools` is required for debugging with CXdb.
- `/opt/langtools/lib/end.o` is required for debugging with CXdb.
- The *<options>* represents user-specified linker options.
- The *<files>* represents user-specified libraries and object files.

For details about other options supported by the compilers, refer to the f90 (1), f77(1), c89(1), and CC(1) man pages on Exemplar systems.

S2000 support

The following commands and windows have been enhanced to support S2000 systems and 64-bit registers.

- `info registers` command and the General Registers window
- `info control registers` and the Control Registers window
- `info space registers` and the Space Registers window
- `info psw` and the Processor Status Word window

When modifying a register value to one greater than 32 bits, you must make sure that your code is using wide mode for register arithmetic.

MPI support

CXdb V4.1 removes the archive libraries only restriction. You can use shared libraries when debugging with CXdb. For more information about MPI, refer to the *HP MPI User's Guide* and *MPI: The Complete Reference.*

Starting MPI applications under CXdb

This section describes the three methods you can use to start MPI applications for debugging with CXdb (assuming your application is compiled for debugging with the `-g` option).

When you use one of these methods, CXdb recognizes your application as an MPI application and sets the execution mode of CXdb to MPI. Processes that are created by the MPI application are automatically attached to CXdb as they are created.

Method 1—If you are using `mpirun`, you can start CXdb from the shell command line with the `-mpi` option. You must also supply the name of an MPI appfile that contains the number of processes to be created and the name of the MPI executable (for example, `-np` *<num_processes>* *<executable_name>*). This is the preferred method.

```
% cxdb -mpi <mpiAppFile>
```

Refer to MPI documentation for information on creating an MPI appfile.

Method 2—You can invoke CXdb from the shell using the GUI, then select the "mpirun" option from the Mode Selection dialog. For example:

```
% cxdb &
```

CXdb then opens an MPI File Selection dialog where you can specify the name of the MPI appfile.

Method 3—You can start CXdb from the shell, specifying the name of the MPI executable.

```
% cxdb myMPIApp.exe &
```

The above command starts CXdb in GUI mode and specifies the name of the MPI application (in this case, `myMPIApp.exe`) to be debugged. To run the MPI executable from within CXdb, use the `run` command with the `-np` option for specifying the number of processes to create. For example:

```
(CXdb) run -np 4
```

When the above command is executed, CXdb runs the MPI executable. Processes that are created by the MPI application are automatically attached to CXdb and assigned CXdb process numbers.

NOTE: Before using the CXdb `run` command to start the MPI application, you must put a breakpoint at or before `MPI_Init()`. This also gives you a chance to set a breakpoint in the first process before the other processes are created.

The `rerun` command is not available for MPI processes. Use the `run` command.

When you use the buttons in the Source Code window to `step`, `next`, or `continue` the MPI application, these operations are applied only to the process currently being displayed in the Source Code window. The focus set in the Command window does not apply.

NOTE: When processes terminate under `mpirun`, CXdb is not notified of the terminated process. When all `mpirun` processes have terminated, CXdb can appear to hang. If this happens, type CTRL-c and quit CXdb.

Shared library support

CXdb provides the capability to debug shared library routines that you have written. Shared library support is disabled by default in CXdb. Because there is additional overhead associated with loading the symbolic debugging information for shared libraries, debugging shared libraries can slow the performance of CXdb. If you want to debug any shared libraries, invoke `cxdb` with the `-shl` option to enable this feature and increase the overhead.

There are two new commands, `set shlibdebug` and `clear shlibdebug` that allow you to change shared library debugging after invoking CXdb.

`set shlibdebug`—Enables debugging of shared libraries. For this option to become effective, you must issue a run or rerun command.

`clear shlibdebug`—Removes all shared libraries specified for symbolic debugging. For this option to become effective, you must issue a run or rerun command.

If your are attaching CXdb to a running process that uses shared libraries, you must first prepare the executable file for debugging by performing these steps:

1. Use the `setupshdbg` utility on your executable file to prepare it for debugging. For example:

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

2. Run your application.
3. Attach CXdb to the running process.

If you do not follow the above steps, CXdb does not have debugging information for any of the shared libraries used by your program.

Installation notes

Enabling unique core files on SPP Series systems

Unique core files are an optional feature allowing customer sites to send CXdb (not application) core images to the Convex Division of Hewlett-Packard, for analysis. This enables the development team to properly diagnose many of the problems that may occur. Without this option, CXdb core images are often overwritten immediately by application core images.

NOTE: To enable generation of CXdb or application core images when running CXdb, you must use the `+core` option when invoking CXdb from the shell. By default, core images are not produced.

A system tunable allows you to enable this feature on SPP Series systems.

To enable unique core file names on SPP Series systems, log in as root and perform the following operations:

1. In the file /os/tunables, modify the entry
   `Server, unique_core_names:1=0:` so that it reads:

   `Server, unique_core_names:1=1:`

   The default is 0 (OFF).

2. On the test station, in the file /spp/os/tunables, modify the entry `Server, unique_core_names:1=0:` so that it also reads:

   ```
   Server, unique_core_names:1=1:
   ```

3. Reboot the system.

CXdb activity logging

CXdb logs selected product operations through the `syslog` facility. Messages are logged to the destination(s) associated with the `user.debug` priority.

If none has been specified in the `syslogd` configuration file (the default location of this file is /etc/syslog.conf), then no activity logging will take place. Refer to the syslog(3) and syslogd(1M) man pages for information detailing the configuration and control activity logging.

The `cxdb` executable is shipped with activity logging enabled. CXdb activity logging can be permanently disabled in the executable file using the `stampEnableFlog` utility found in the directory /opt/cxdb/newconfig/etc, as shown below.

```
(CXdb) /opt/cxdb/newconfig/etc/stampEnableFlog  0
```

Specifying a value of 1 with the `stampEnableFlog` utility enables activity logging; specifying a value of 0 disables activity logging. You must have the correct permissions to modify the executable file.

Notes

The following notes apply CXdb V4.1:

- If you are debugging a core file on an SPP Series machine, you must specify an executable for the core file before using the `backtrace` command to display a stack trace. For example, you can specify the executable file `a.out` with the following command:

  ```
  (CXdb) debug executable a.out
  ```

- By default, CXdb does not generate core files if terminated abnormally. To enable generation of core files, use the `+core` option of the `cxdb` command when invoking CXdb from the shell, and enable unique core files.

- For Fortran functions with alternate entry points, Exemplar compilers create local variables with the same names as the entry points. These compiler-created local variables are displayed in the output of the `info locals` command.

Limitations

CXdb V4.1 has the following restrictions and limitations:

- You cannot step into or out of code that is contained within `#include` files, inline functions, or macros. This results in source file display and line numbering problems.

  You cannot set a breakpoint on a routine contained within a `#include` file using the `break routine` command. The message "`No source statements found.`" displays. You can, however, use the command `break instruction` *routine_name* to set a breakpoint, but no source file information displays.

- CXdb keywords cannot be used as symbolic names in the abbreviated `break` command. For example, if you have a module in your executable named module, the only correct syntax to set a breakpoint on that module is `break routine module`.

- For executables compiled with the Exemplar compilers, there is no way to update certain variables. Which variables and when updates cannot be performed, depend on how the compiler generated the code. An example of this is when a `longlong` is passed as a parameter. CXdb cannot correctly compute the address for variables allocated in this manner.

- To use the graphical interface to select a breakpoint in the Source Code window, the pencil cursor must be pointing to a line number. Selections in the source code text area are invalid.

- The `-g` and `-g1` options are incompatible with optimization levels greater than +O0. CXdb features for debugging optimized code do not apply. In the case of c89 and f77, debugging information is disabled. In the case of C++, the higher optimization level is disabled.

- No information is passed to the debugger by Exemplar compilers for the `const` keyword.

- For modules compiled with the `-g` or `g1` option, variables that are declared within a function or block are visible at the beginning of the function or block.

- CXdb reports a syntax error for pathnames that include a colon (:) Rename the file without using a colon character.

- For modules compiled with the Exemplar compiler, CXdb does not support printing or evaluation of Fortran functions that return character strings. CXdb cannot determine the length of character strings returned by Fortran functions. For example, given a Fortran function `CHARACTER *20 STRFUNC(X, Y)`, executing the following commands produces incorrect results (the return value is printed as `CHARACTER**`):

```
(CXdb) print STRFUNC
(CXdb) print STRFUNC(X,Y)
```

Executing the CXdb `return` command from within a Fortran function that returns a character string produces incorrect results.

When you execute the `info args` command within a Fortran function that returns a character string, CXdb does not display the two additional arguments (string and length) for the return value.

- Exemplar compilers do not output debugging information for function prototypes. Different local declarations for functions and variables do not override the actual function or variable definition.

  If the actual function definition is in a library or an object file without symbolic debugging information, CXdb displays a type mismatch warning when you try to invoke the function from within CXdb. Debugging can proceed; however, CXdb does not perform any type checking and assumes the arguments specified are correct.

- No debugging information is available for unused local variables, even for modules compiled without any optimizations.
- CXdb cannot debug an MPI application that spans multiple machines.
- In Fortran, CXdb cannot always determine the value of an indefinite variable such as a dynamic array boundary that is passed as an argument. In such cases, CXdb prints the value as the maximum integer value for the system on which it is running.

The following limitations and restrictions apply to debugging modules compiled with the Exemplar C++ compiler on S2000 systems:

- Only data members of C++ objects are printed within CXdb. Member functions are not printed. Attributes of objects (for example, `private`, `public`, `const`, `volatile`, and so on) are not printed.
- If an object contains another object, the contained object's inherited members are not printed when the `/K` option of the `print` command is used to print all inherited members at all levels.

  To print a contained object's inherited members, you must use the `print/K` command on the contained object directly.

- The `/K` option of the `print` command is incompatible with any other `print` command formatting options. When `/K` is used, CXdb displays the object's data members using the default formatting, and any other format specifiers are ignored.
- When printing an object's inherited data members, the names of base classes are not displayed.

- When attempting to print the object using the `this` pointer from inside a non-static member function, static data members do not print. To work around this problem, use the class name, as in

  ```
  p Foo::staticMem1
  ```

- When attempting to print from inside a static member function, an error is reported indicating the identifier is not visible. To work around this problem, use the class name.
- Inlined functions are not supported.
- CXdb does not provide any means for resolving overloaded functions (functions with the same name that have different type signatures):
  - If CXdb selects a different function than you intended when setting a breakpoint with the `break routine` command, use the `break line` command.
  - CXdb may select a different function than you intended when evaluating an overloaded function with a `print` command. This can happen regardless of the type information provided by the arguments to the function.
- Function pointer arguments are not supported.
- Local classes (classes that are declared within a function) are not supported.
- Nested classes are not supported.
- Global scope operators are not supported.
- Nested `(::)` scope operators are not supported.
- The CXdb `return` command cannot be used to return references to classes, structures, enumerated types, or unions. For example, the following are invalid:

  ```
  (CXdb) return my_struct&
  (CXdb) return my_class&
  ```

- Boolean types and `true` and `false` constants are not supported.
- You cannot pass pointers to derived objects or reference types to functions that expect base class pointers or references.
- CXdb does not support:
  - Virtual inheritance
  - Templates
  - Exceptions
- The following are not supported:
  - Use of the `template` keyword before members
  - `throw` expressions

  - `operator` member functions
  - Cast operators (`dynamic_cast`, `static_cast`, `reinter_cast`, `const_cast`)
  - `typeid(...)` expressions
- CXdb cannot evaluate or print expressions containing pointers to class members. For example:

```
a->*b
x.*y
```

Known problems

You can encounter the following problems when running CXdb:

- Double precision constants are not correctly printed in CXdb.
- Statically allocated identifiers cannot be referenced outside of their scope using scope paths.
- In some cases, CXdb cannot access or print the value of symbolic constants declared using the Fortran `PARAMETER` statement. The problem occurs when the symbolic constants are not allocated storage in memory by the compiler.
- The column number information displayed in the output of the `info line` and `info source` commands is incorrect. (The column number displayed is always 4095.)
- Using the `info symbols` command (in line mode) or selecting Symbols from the Info menu (in GUI mode) on very large executables can cause CXdb or the target process to terminate abnormally or behave slowly.
- Using **CTRL-c** while debugging a POSIX-threads application can cause CXdb to terminate abnormally.
- CXdb cannot step into some functions in large executables with linker-inserted stubs. If you encounter this problem, set a breakpoint at the problem function, then use the `continue` command to put the point of execution within the function.
- Void values are displayed as integer types with a random integer value, rather than `(void)`.
- When processes terminate under `mpirun`, CXdb is not notified. When all `mpirun` processes have terminated, CXdb can appear to hang. If this happens, type **CTRL-c** and quit CXdb.

CXdb V4.1

The following problem can be encountered when debugging modules compiled with the C++ compiler on S2000 systems:

- CXdb might not be able to make calls to functions that use `float&` values as arguments.

Getting assistance

If you have questions about CXdb, contact the Hewlett-Packard Convex Technical Assistance Center (TAC). To contact the TAC, use one of the following phone numbers:

- Within the continental U.S., call 1(800)952-0379.
- From Canada, call 1(800)345-2384
- All other locations, contact the nearest sales office.

The TAC recommends using the `contact` utility to report a hardware, software, or documentation problem. The `contact` utility is an interactive program that helps the TAC track reports and route them to the personnel most qualified to fix a problem.

After you invoke `contact`, it prompts you for information about the problem. When you finish your report, `contact` mails it to the TAC electronically. The TAC notifies you within 48 hours that your report has been received.

If possible, include the following information when submitting a problem report for CXdb:

- Version of CXdb used
- Version of Fortran, C, or C++ used
- Version of SPP-UX installed on your system
- Detailed explanation of the problem
- Copy of the source files, compilation string, and CXdb commands used to reproduce the problem

To display the version of the compilers, linker, or CXdb, use the `what` command. For example:

```
% /usr/bin/what /opt/fortran/bin/f77

SPP-UX f77 970509 (164218) B5600-10002 Exemplar V1.2.1
 Internal_Unsupported_Version libc.a_ID@@/main/r10sac//1
 /ux/core/libs/libc/archive_pa1/libc.a_ID
 Oct 24 1996 23:13:02
```

In the above example, the version of the Exemplar Fortran compiler is V1.2.1.

To display the version of the SPP-UX operating system installed on your system, use the `sysinfo` command. For example:

```
% sysinfo
SPP-UX_mk      5.0.41.3
SPP-UX_server  5.0.41.3
SPP-UX_ail     5.0.41.3
```

Using `contact` requires:

- UNIX-to-UNIX Communication Protocol (UUCP) connection to the TAC.
- Full path name of the program or utility in question.
- Version number of the program or utility in question.

Refer to the contact(1) man page for complete details.

CXdb V4.1