HP WDB 4.5 Release Notes

Edition 8

Manufacturing Part Number: 5187-8590
June 2004

United States

Legal Notices

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

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

Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

U.S. Government License

Proprietary computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

UNIX® is a registered trademark in the United States and other countries, licensed exclusively through The Open Group.

1 HP WDB 4.5 Release Notes

The information in this document applies to the release of HP WDB 4.5 software product (5187-4553).

Announcement

The HP WDB debugger is an HP-supported implementation of the GDB debugger. It supports source-level debugging of the object files written in HP C, HP aC++, and Fortran on HP-UX releases 11.00 and 11.11.

HP WDB 4.5, based on GDB 5.0, is available for any PA-RISC 1.1 or PA-RISC 2.0 HP 9000 system running HP-UX release 11.00 and 11.11.

HP WDB 4.5 is not supported on any of the HP-UX operating system releases prior to 11.00.

What's in This Version

This version of the HP WDB debugger bundle consists of the following components:

- HP WDB debugger, based on GDB 5.0.
- Visual Interface for HP WDB, an open-source-based terminal mode and graphical mode interface to the debugger.
- HP WDB GUI, included if you are using the WDB-GUI depot file.

New features in HP WDB 4.5

HP WDB 4.5 includes the following new functionality and defect fixes:

- **Support for Batch mode RTC:** HP WDB supports batch mode memory leak detection, also called batch Run Time Checking (RTC). You can specify the batch mode configuration through a configuration file called rtcconfig. The configuration file supports these variables:

Table 1-1

Variable	Description
check_heap=on	To enable heap info
check_leaks=on	To enable leak detection
scramble_blocks=on|off	To enable block scrambling
min_leak_size=block_size	Minimum block size to use for leak detection
frame_count=no_frames	Number of frames to be printed for leak context
output_dir=output_data_dir	Name of the output data directory
files=file1:file2:....|fileN>	Executables for which memory leak detection is enabled.

Batch mode leak detection stops the application at the end, when libraries are being unloaded, and invokes HP WDB to print the leak or heap data.

To use batch memory leak detection:

1. Debug Enable the application using any of the following:
   - `chatr +dbg enable <application>`
   - `pxdb -s on <application>`
2. Create an rtcconfig file in the current directory.
3. Define an environment variable `BATCH_RTC`. If you are using the Korn or Posix shell, type:

   ```
   export BATCH_RTC=on
   ```
4. For 64-bit applications, set the GDB_SERVER to pick the gdb64 as follows:

   ```
   export GDB_SERVER=/opt/langtools/bin/gdb64
   ```
5. Start the target application by defining `LD_PRELOAD` of `librtc`. For example,

   ```
   LD_PRELOAD=/opt/langtools/lib/librtc.sl application_name
   ```

   ```
   LD_PRELOAD=/opt/langtools/lib/pa20_64/librtc.sl application_name
   ```
6. At the end of the run output data file is created in the `output_data_dir`, if defined in `rtcconfig`, or the current directory.

   HP WDB creates output data file for each run. It creates a separate file for leak detection and heap information.

   The naming convention for output files is:

   ```
   file_name.pid.suffix
   ```

   The value for suffix could be either `leaks` or `heap`.

Batch memory leak detection uses these environment variables:

Table 1-2

Variables	Description
`GDBRTC_CONFIG`	Specifies the location of `rtcconfig`.
`BATCH_RTC`	Enables or disable batch memory leak detection.
`GDB_SERVER`	By default, memory leak detection uses `/opt/langtools/bin/gdb` to print output. Override this with `GDB_SERVER`.

- **Preliminary support for GDB-MI:** GDB-MI is a line-based, machine-oriented text interface to GDB. It is specifically intended to support the development of systems which use the debugger as just one small component of a larger system.

GDB-MI can be invoked by giving the command line option `-i=mi` with gdb.

For example:

```
$gdb -i=mi
```

NOTE

- Refer to the "Debugging with GDB" document for details on the specification of the GDB/MI interface.
- GDB-MI is still under construction. Some of the features are incomplete and subject to change. The following GDB/MI commands do not work:
  - Setting breakpoints using regular expressions (`-break-insert -r`)
  - Setting access watchpoints and read watchpoints (`-break-watch -a|-r`)

- **Support for Deferred Watchpoints:** When you try to set a watchpoint in an expression, HP WDB places a deferred watchpoint if HP WDB cannot evaluate the expression. The watchpoint is automatically enabled whenever the expression can be evaluated during the program's execution. This is especially useful when placing the watchpoints on unallocated addresses.
- **Support for Alternate Root Functionality:** HP WDB supports alternate root functionality, which helps you use HP WDB and its components installed on an alternate root rather than the system with HP WDB or its components installed.

  The environment variable, `WDB_ROOT`, specifies the alternate root for HP WDB GUI. You must specify a structure similar to the default, `/opt/langtools`. You can use the environment variable, `GDB_ROOT`, to specify an alternate root for WDB.

  For example, `$ export GDB_ROOT=$HOME/opt/langtools.`

  If you specify both `WDB_ROOT` and `GDB_ROOT`, the value for `GDB_ROOT` is ignored. HP WDB supports these environment variables to override the location of different component of HP WDB. The following table specify these environment variables in the increasing order of precedence:

Table 1-3

Defined Variable	WDB GUI Location	WDB Location	librtc.sl location
NONE	`/opt/langtools/bin`	`/opt/langtools/bin`	`/opt/langtools/lib`

Table 1-3 (Continued)

Defined Variable	WDB GUI Location	WDB Location	librtc.sl location
WDB_ROOT	$ WDB_ROOT/bin	$ WDB_ROOT/bin	$ WDB_ROOT/lib
GDB_ROOT	N/A	$ WDB_ROOT/bin	$ WDB_ROOT/lib
GDB_SERVER	N/A	$ GDB_SERVER	N/A
LIBRTC_SERVER	N/A	N/A	$ LIBRTC_SERVER

NOTE

— If you define WDB_ROOT or GDB_ROOT but do not create the correct directory structure below it, the debugger may fail.

— If alternate root is defined and the application is not processed by pxdb, then gdb runs [$WDB_ROOT | $GDB_ROOT]/bin/pxdb on the executable in non-existence of /opt/langtools/bin/pxdb.

- **New `pathmap` Command:** The new command, pathmap, allows you to define a list of substitution rules to be applied to path names to identify object files and the corresponding source files. The pathmap command, however, may not find source files if the object files are not available.

  This minimizes or eliminates the need to specify multiple objectdir commands when object files are moved from the compilation directories or when compilation directories are mounted over NFS.

  To use this feature, your program must be compiled with the +objdebug option. For information on how pathmap works type help pathmap at the HP WDB prompt.

- **Core File Validation:** HP WDB automatically detects mismatches between a library in a core file and the library used by HP WDB when it is invoked on the core file. Core library mismatch detection requires dld.sl version B.11.38 or above on both the system used for compiling and the system used for debugging.

  The debugger shows a list of the mismatched libraries. You must then provide the location of the correct libraries. For example, you can set the GDB_SHLIB_PATH environment variable to point to the correct directories.

  A core library mismatch can occur when a core file is produced on one computer and debugged on another.

The libraries used by the executable when it produced the core file on one computer may not be the same on another. Libraries that might cause this problem include `libc, libm`, etc. They may also include libraries specific to your executable. In these cases, debugging core files may produce wrong or inconsistent behavior such as truncated or missing stack trace, incorrect symbols, or incorrect data values.

- **New `packcore` command:** The `packcore` command takes a user-specified core file, identifies the shared libraries used by the core and creates a tar file called `core.tar` with all those libraries along with the executable. You can view usage information by executing the `help packcore` command at the gdb prompt.
- **Setting Object File Paths:** The Object File Paths dialog box lets you enter paths so that WDB GUI can locate object files for the current program. In this dialog box, you can add, delete, and change the priority order of object file paths. You can access the Object File Paths dialog box on the Edit menu by clicking Object File Paths. Alternately, you may also set the object file paths from WDB GUI's command view.

New features in HP-WDB 3.3

HP WDB 3.3 includes the following new functionality and defect fixes:

- Support for the new environment variable, LIBRTC_SERVER, which you can use to override the default location of librtc.sl.
- Implementation of `ask` mode for `set follow-fork-mode`

  This command prompts user to select between parent and child as the debugger response to a program call of `fork/vfork`. Based on the user selection, the parent or the child process is debugged.

  For example:

```
(gdb) set follow-fork-mode ask
(gdb) show follow-fork-mode
Debugger response to a program call of fork or vfork is "ask".

(gdb) run
Starting program: sample
[New process 4941]
warning: reading `r3' register: No data
Select follow-fork-mode :
[0] parent
[1] child
>
```

- HP WDB 3.3 has fixes for debugging q3p and q4p enabled binaries.
- HP WDB 3.3 has fixes for defects in performing command line calls.

Summary of Changes in HP WDB 3.2

HP WDB 3.2 includes the following new functionality:

- Support for a new command, `dumpcore`.
- Support for enabling and disabling threads.
- Support for the `steplast` command for both C and C++.
- Additional support for command line calls in a stripped executable.

Support for a New Command, `dumpcore`

HP WDB 3.2 provides a new command, `dumpcore`, to generate a core image file for a process running under the debugger in the middle of execution.

The `dumpcore` command does not require any argument. It saves the core image for the current process being debugged in the file named `core.<pid>`, where `<pid>` is the process ID number.

To generate and analyze the core file with HP WDB, refer to the following sample code:

To dump the core for a live process, you must pass the following commands:

```
(gdb) run
Starting program: sample
Breakpoint 3, main () at sample.c:10
10              b = foo(a) ;
(gdb) dumpcore
Dumping core to the core file core.24091
(gdb)


When starting from HP WDB command line:
(gdb) set live_core 1
(gdb) core-file core.<pid>
(gdb) file sample
Reading symbols from sample...done.
(gdb) set live-core 1
(gdb) core core.24091
Core was generated by ‘sample’.
```

```
#0  main () at sample.c:10

10              b = foo(a) ;

(gdb) backtrace

#0  main () at sample.c:10

(gdb)


When starting from shell prompt:

% gdb --lcore a.out core.<pid>

%> ./gdb  --lcore sample core.24091

HP gdb 3.2 for PA-RISC 1.1 or 2.0 (narrow), HP-UX 11.00.

Copyright 1986 - 2001 Free Software Foundation, Inc.

Hewlett-Packard Wildebeest 3.2 (based on GDB) is covered by the GNU General Public License.
Type "show copying" to see the conditions to change it and/or distribute copies. Type "show
warranty" for warranty/support.

...

Core was generated by `sample'.

#0  main () at sample.c:10

(gdb)
```

Support for Enabling and Disabling Threads

While debugging a multi-threaded application, if you suspect that a specific thread is causing a problem, suspend other threads in the debugger and debug the doubtful thread. HP WDB 3.2 provides the following commands to disable and enable specific threads:

- **`thread disable` -** This command prevents specified threads from running until they are enabled again using the `thread enable` command.
- **`thread enable` -** This command enables the specified thread to run when you issue the `continue` or `step` command. By default, all threads are in the enabled state. You can use the `thread enable` command to reactivate a disabled thread.

Consider the following example:

- To disable a thread, execute the following commands:

```
(gdb) info thread

    3 system thread 4189  0x7f666da8 in __pthread_create_system+0x3d8 () from
/usr/lib/libpthread.1

    2 system thread 4188  worker (wptr=0x40004640 "@") at quicksort.c:135
```

```
    1 system thread 4184  0x7f66f728 in _lwp_create+0x10 () from /usr/lib/libpthread.1
(gdb) thread disable 1
warning: ATTENTION!! Disabling threads may result in deadlocks in your program.
Disabling thread 1
```

- To enable a thread, execute the following command:

```
(gdb) thread enable 1
Enabling thread 1
```

Support for the `steplast` Command for Both C and C++

Typically, if a function call has arguments that make further function calls, executing a simple `step` command in GDB steps into the argument evaluation call. HP WDB 3.2 includes the `steplast` command, which helps to step into a function, and not into the calls for evaluating the arguments.

The following example illustrates how GDB behaves when you execute the `steplast` command:

```
(gdb)
16        foo (bar ());      ---> bar() will return 10
(gdb) steplast
foo (x=10) at foo.c:4
4         int k = 10;
```

If the `steplast` command is not meaningful for the current line, GDB displays the following error message:

```
"Steplast is not meaningful for the current line."
```

For example,

```
(gdb)
4         int k = 10;
(gdb) sl                      ---> alias to "steplast" command
error: Steplast is not meaningful for the current line.
```

To execute the `steplast` command in C++ compiled applications, you must compile the application using the HP aC++ version A.03.50 or later with `-g0` option.

In C++, the `steplast` command is helpful while debugging heavy templated functions, because it directly steps into the call, thus skipping the constructor calls, if any. This behavior is unlike the `step` command that steps into the constructor itself.

Consider the following example:

```
void call_me ( string s ) { ... }
(gdb)
10            call_me ( "hello" ) ;
(gdb) steplast
call_me (s={static npos = 4294967295, static nullref = {ref_hdr = {mutex_ = {
      dummy1 = 0x7f4f79e0, dummy2 = 2136325568}, refs_ = 2136327612,
      capacity_ = 2136327468, nchars_ = 2136327464}, eos_char = 64 '@'},
      alloc_ = {<No data fields>}, value_allocator = {alloc_ = 0x7f7f133c},
      data_ = 0x40003a64 "hello"}) at str.C:5
5     printf ("Will just print the value of \n");
```

If there are multiple top-level calls, the `steplast` command enables you to step into each top-level call.

For example, for the following line, the `steplast` command takes you to the first top-level call, `(foo())`:

```
foo(bar()) + bar(foo());
```

Debug `foo()`, use the `finish` command to exit from the first top-level call, `(foo())`, execute the `steplast` command to step into the next top-level call, `(bar())`. Refer to the following sample code:

```
(gdb)
10       foo( bar() ) + bar( foo() )
(gdb) sl      ======> Use the steplast (sl) command to step into the first top-level call
foo (x=0) at steptop2.c:12
3        printf ("in foo()\n");
(gdb) step    ======> Debug foo()
5        return 1;
(gdb) finish  ======> Use the finish command to exit from the first top-level function call
Run till you exit from #0  foo (x=0) at steptop2.c:12
0x5778 in main () at steptop2.c:22
22         val = foo ( bar (10) ) +  bar ( foo (10) ) ;
Value returned is $1 = 0
(gdb) sl      ======> Now, the sl command takes you to the next top-level call
```

```
bar (x=100) at steptop2.c:16
16        return x-10;
(gdb)
```

Additional Support for Command Line Calls in a Stripped Executable

HP WDB 3.2 enables you to perform command line calls in a stripped executable. The various scenarios in which you can make command line calls in a stripped executable are as follows:

❑ **For 32-bit applications:**

- To perform command line calls in a shared library, without the help of dynamic linker (using `end.o`), you must perform the following operations:
  - — Execute the `chatr -B immediate <executable>` command.
  - — Modify all the calls to `shl_load()` to specify `BIND_IMMEDIATE`, if any.
- To perform command line calls after attaching GDB to a running process, without the help of dynamic linker (using `end.o`), you must do the following for the program:
  - — Execute the `chatr -B immediate <executable>` command
  - — Modify all the calls to `shl_load()` to specify `BIND_IMMEDIATE`, if any.
  - — Execute the `/opt/langtools/bin/pxdb -s on <executable>` or `chatr +dbg enable <executable>` command.
- To avoid changing of the run-time binding behavior of a program to BIND_IMMEDIATE, in order to perform command line call, do the following:
  - — Use the new linker option, `+ea`, to export symbols from an object file.
  - — Install the linker patch, PHSS_28870 (for 11.0) or PHSS_28871 (for 11i).
  - — Execute the following commands:

```
cc -c file.c
ld /opt/langtools/lib/crt0.o file.o +ea /opt/langtools/lib/end.o
-lc -s
```

❑ **For 64-bit applications:**

- To perform command line calls in a stripped executable, linked with `end.o`, you need to do the following:
  - — In the `+std` link mode, GDB supports this feature without any changes. You must export the `__wdb_call_dummy` symbol as shown in the next line.

— In the `+compat` link mode, execute the following command:

```
cc +DD64 -c file.c -Wl,+ee __wdb_call_dummy -o a.out
```

- You cannot perform command line calls from a 64-bit program that is not linked with `end.o`.

Summary of Changes in HP WDB 3.1.5

HP WDB 3.1.5 includes the following new functionality:

- Support for the `steplast` command for C.
- Additional support for procedural breakpoints.
- Support for dumping an array into an ASCII file.
- Support for the FORTRAN array slices.
- Support for a new command line option (`-pid` or `-p`).
- Change in handling the command line arguments.
- Support for command line calls in a stripped executable.

Support for the `steplast` Command for C

Typically, if a function call has arguments that make further function calls, executing a simple `step` command in GDB steps into the argument evaluation call.

HP WDB 3.1.5 provides a new command, `steplast`, which helps to step into a function without stepping into the argument evaluation function calls.

If the `steplast` command is not meaningful at the current line, GDB displays the following warning message:

```
Steplast is not meaningful for the current line; behavior undefined.
```

The following example shows how GDB behaves when you execute the steplast command:

```
(gdb)
16 foo (bar ()); ---> bar() will return 10
(gdb) steplast
foo (x=10) at foo.c:4
4 int k = 10;
(gdb) steplast
warning: Steplast is not meaningful for the current line; behaviour undefined.
```

```
5 }
```

Additional Support for Procedural Breakpoints

HP WDB 3.1.5 enables you to set permanent breakpoints at the entry and exit (first and final executable statement) of every function that can be debugged.

The new breakpoint commands are `xbp` and `xdp`.

- **`xbp`:** Sets breakpoints at the first and final executable statement in all the functions, which can be debugged, in all the source files. These shared files also include shared libraries that are already loaded. After you set these breakpoints, you can manage them like any standard breakpoints. You can delete them, disable them, or make them conditional.
- **`xdp`:** Deletes all the breakpoints set by the `xbp` command.

Here is an example of how to set a breakpoint at the start and end statement of each procedure:

```
(gdb) file a.out
Reading symbols from a.out...done.
(gdb) xbp
Breakpoints set from 3 to 8
(gdb)
```

Support for Dumping an Array into an ASCII File

HP WDB 3.1.5 supports a new functionality for dumping an array into an ASCII file.

The array elements are stored in Array format of Matrix Market in a predefined (column-major order for Fortran arrays) order. The objective is to provide a simple mechanism to facilitate the exchange of matrix data and to enable easier parsing of the array elements.

For common file formats, see http://math.nist.gov/MatrixMarket/formats.html.

To dump an array, `ARRAY`, to a file named DUMPFILE, use the following command:

```
(gdb) dump2file ARRAY DUMPFILE
```

The entries of `ARRAY` are dumped into an ASCII file named DUMPFILE in the array format. The file is created in the current working directory. The content of the file has the following format:

```
%%ArrayBrowsing matrix array ARRAY
% A 5x5 matrix
5 5
0
2
```

```
4
6
8
2
..
..
```

where, ARRAY is the name of the array, and its size is 5x5.

The first two lines are comments about this file and the array. The third line denotes the array coordinates. From the fourth line, the elements of the array are listed.

NOTE This feature is not supported for the FORTRAN array slices.

Support for FORTRAN Array Slices

HP-WDB 3.1.5 prints the FORTRAN array slices like DDE. Instead of an array subscript, you can specify a range of elements (see Table 1-4).

Table 1-4 Mapping of Array Subscript to Element Range

Subscript	Element Range
:	Lower bound to upper bound
expr1:expr2	Given range
:expr2	Lower bound to given expr2
expr1:	Given expr1 to upper bound
::	Lower bound to upper bound, stride of 1
::expr3	Lower bound to upper bound, stride of expr3
expr1:expr2:	Given range, stride of 1
expr1:expr2:expr3	Given range, stride of expr3
:expr2:expr3	Lower bound to given expr2, stride of expr3
expr1::expr3	Given expr1 to upper bound, stride of expr3

Consider the following example:

```
(gdb) ptype a
type = real*4 (3,3)
```

```
(gdb) print a(1:2:1,:2)
$3 = (1,1) = 11
(2,1) = 21
(1,2) = 12
(2,2) = 2
```

Support for a New Command Line Option (-pid or -p)

HP-WDB 3.1.5 accepts `-pid` or `-p` followed by a process ID to attach a running process to GDB.

Change in Handling the Command Line Arguments

HP-WDB 3.1.5 handles the command line arguments in a different manner. The first non-flag argument is always a program to debug. The second non-flag argument can either be a corefile or a process ID.

Earlier, WDB attempted to open the second argument as a corefile. If that failed, it displayed an error message, and then attempted to attach it as a process.

Now, if the second argument is numeric, and if there is a running process with that process ID, it is attached to GDB; otherwise, it is treated as a corefile.

Support for Command Line Calls in a Stripped Executable

HP-WDB 3.1.5 enables you to perform command line calls in a stripped executable.

To perform command line calls in a shared library without the help of dynamic linker (using `end.o`), you must execute the following command:

```
chatr -B immediate <executable>
```

In addition, modify all the calls to `shl_load()` to specify `BIND_IMMEDIATE`.

To perform command line calls after attaching a running process to GDB, you must execute one of the following commands:

- `/opt/langtools/bin/pxdb -s on <executable>`
- `chatr +dbg enable <executable>`

NOTE Command line calls do not work for the following:

- Stripped executable (32 bit) on attach with shared mapping of libraries
- 64-bit applications

Summary of Changes in HP WDB 3.1.1

HP WDB 3.1.1 includes the following new functionality and defect fixes:

- Support for stack traces in Java, C, and C++ programs in 64-bit mode.
- Support for co-variant return type.
- Support for HP-UX 11.00 to 11.11 only.

Support for Stack Traces in Java, C, and C++ Programs in 64-bit Mode

This release of HP WDB can show stack traces of mixed Java, C, and C++ programs in the 64-bit mode. The stack trace functionality requires Java SDK version 1.4 or later for HP-UX. Please check the HP web site for Java, http://www.hp.com/go/java, for availability of Java SDK version 1.4 or later.

To enable this feature, set the environment variable `GDB_JAVA_UNWINDLIB` to the path to a `libjunwind.sl`. This library is part of the Java SDK version 1.3.1.02 or later for HP-UX. When `GDB_JAVA_UNWINDLIB` is set to the path for a valid Java unwind library, stack traces will show Java and C/C++ frames.

In this version, even if `GDB_JAVA_UNWINDLIB` is not set, HP WDB uses the `libjunwind.sl` specified by the Java Virtual Machine.

The types of Java frames supported are as follows:

- Interpreter
- Compiled frames
- Adapter frames

For specific details on how to use this feature, refer to the HP Web site for Java, http://www.hp.com/go/java.

Support for Co-variant Return Type

HP WDB 3.1.1 can step into a co-variant function. The compiler-generated function called `thunks`, which is used internally by the compiler to support co-variant return type, will not be shown when you do a "backtrace" or switch from one frame to another frame. Similarly, using a `finish` or `return` command at a co-variant callee function directly returns the control back to the caller of `thunks`.

Summary of Changes in HP WDB 3.1

HP WDB 3.1 includes the following new functionality and defect fixes.

- Support for output logging.
- Support for xdb-like procedural breakpoints.

Support for Output Logging

The Visual Interface for HP WDB terminal user interface (TUI) mode supports a new command, `log` *`logfile_name`*, that saves the content of a session to the specified log file.

When you use the `log` command, the debugger saves a snapshot of the current session, from the start of the session to the point where you issued the `log` command. Each time you use the log command, HP WDB overwrites the specified log file with a new snapshot from the start of the session.

To run the Visual Interface for HP WDB, use the following command:

```
$v  db -tui
```

To redirect HP WDB output to a log file named `mylogfile`, use the `log` command in the following manner:

```
(gdb) log mylogfile
```

The Visual Interface for HP WDB stores the log file, `mylogfile`, in the current directory.

To view the log file from Visual Interface for HP WDB, start a shell process and use the following command:

```
(gdb) shell vi mylogfile
```

Support for Procedural Breakpoints

HP WDB 3.1 allows you to set breakpoints at the beginning (first executable line) of every function that can be debugged. In addition, it allows you to specify a set of commands to be performed when the breakpoint is reached. These breakpoints work like procedural breakpoints in the `xdb` debugger.

The new breakpoint commands are `rbp` and `rdp`.

- **`rbp`:** Sets breakpoints at the first executable statement in all the functions that can be debugged, including any shared libraries that are already loaded. The `rbp` command sets breakpoints in all the functions, which can be debugged, in all the source files.

  After you set these breakpoints, you can manage them like any standard breakpoints. You can delete them, disable them, or make them conditional.

Each time you use the `rbp` command, HP WDB adds an additional breakpoint at the beginning of each function that performs the commands you specify, if any.

- **`rdp`:** Deletes all the breakpoints set by the `rbp` command.

Here is an example of how to set a breakpoint at the start of each procedure that displays information at the breakpoint.

```
(gdb) file a.out
Reading symbols from a.out...done.
(gdb) rbp
Breakpoints set from 170 to 211
Type commands to execute when the breakpoint is hit (one command per
line).
End with a line saying just "end".
>info break
>end
(gdb)
```

Summary of Changes in HP WDB 3.0.01

HP WDB 3.0.01 includes new functionality as well as numerous defect fixes.

- Support for debugging large core files (> 2GB).
- Support for Java, C, and aCC stack unwinding.
- Many defects are fixed. See "Defects Fixed in HP WDB 3.0.01" on page 35 for details.

Support for Stack Traces in Java, C, and C++ Programs

This release of HP WDB can show stack traces of mixed Java, C, and C++ programs. The stack trace functionality requires Java SDK version 1.3.1.02 or later for HP-UX. Please check the HP web site for Java, http://www.hp.com/go/java, to know the availability of Java SDK version 1.3.1.02 or later.

To enable this feature, set the environment variable, `GDB_JAVA_UNWINDLIB`, to the path to `libjunwind.sl`. This library is part of the Java SDK version 1.3.1.02 or later for HP-UX. When `GDB_JAVA_UNWINDLIB` is set to the path for a valid Java unwind library, stack traces display Java and C/C++ frames.

The types of Java frames supported are as follows:

- Interpreter
- Compiled frames
- Adapter frames

For specific details on how to use this feature, visit the HP web site for Java, http://www.hp.com/go/java.

Summary of Changes in HP WDB 3.0

HP WDB 3.0 includes new functionality as well as numerous fixed defects.

Enhanced Memory Leak Detection

HP WDB 3.0 has improved memory leak detection and heap analysis capabilities, also known as run-time checking (RTC):

- If you are using dynamic linker version B.11.19 or later, you do not have to explicitly link with `librtc.sl`. The debugger automatically loads this library, which is needed for detecting memory leaks and analyzing the heap of program. This is true for both the PA-32 and PA-64 applications.
- HP WDB 3.0 can perform heap analysis of EXEC_MAGIC programs.
- HP WDB 3.0 can perform heap analysis of OGL programs.
- HP WDB 3.0 does not hang when memory leak detection is turned on and the program calls `vfork()`.
- Multi-threaded programs do not receive a `SIGILL` signal when memory leak detection is turned on.

New Terminal-Based Interface

HP WDB 3.0 uses a new HP-supported user interface with both the graphical and terminal modes. Visual Interface for HP WDB is based on Vim 5.7 and WDB. This interface replaces the `-tui` mode on Itanium-based systems. To use the interface with HP WDB, type `vdb`.

When you are using the interface, you are actually using `vim`, which is a vi-compatible editor. With the new interface you can use `vi` commands to browse in the WDB display.

Most of the interface functionality is also available for emacs users and does not require knowledge of vi commands.

For details about the interface, refer to the *Debugging with GDB* guide, which is included with the software.

Summary of HP-Enhanced Features

HP WDB provides the following features in addition to the standard GDB features:

- Support for debugging incrementally linked 64-bit programs.

This feature requires linker version B.11.25 or later on HP-UX 11.0 and B.11.18 or later on HP-UX 11.11.

- Support for Fix and Continue debugging.
- Support for memory leak checking and heap profiling.
- Support for debugging 32-bit and 64-bit programs.
- Support for a Terminal User Interface (TUI) in addition to the GDB line-mode interface, enabled with the `-tui` option.
- Visual Interface for HP WDB, a terminal mode and graphical mode interface to the debugger.
- Support for automatic loading of debug information from object modules when an application is compiled with the `+objdebug` option. The benefits of this option are only available if you have installed a recent linker patch. See "HP WDB Needs Linker Patch to Use +objdebug" on page 28.
- Support for debugging optimized code (compiled with both `-g` and `-O`).
- Support for debugging code in shared libraries and archive libraries.
- Support for kernel threads, including thread local storage (HP-UX 11.0 to 11.11 only).
- Support for user threads on HP-UX 11.0 to 11.11.
- Support for hardware watchpoints on HP-UX 11.0 to 11.11.
- Support for assembly-level debugging (PA-RISC 1.*x* and PA-RISC 2.0).
- Support for a subset of XDB commands, enabled with the `-xdb` option.

Known Problems and Workarounds

`set threadverbose` Functionality Not Available

The `set threadverbose` functionality is not available in HP WDB 3.3.

Limitations of the `steplast` Command

The `steplast` command has the following limitations:

- In C++ applications, if you execute the `steplast` command at the following line, it steps into the constructor itself:

  ```
  string hello = "hello gdb";
  ```

- In C applications, if you execute the `steplast` command at the following line, it steps into `c()`, and not into `a()`:

  ```
  a(b()) + c(d());
  ```

Preventing Program Hangs while Disabling Signals

HP WDB has the ability to detect any occurrence of a signal in your program. You can notify HP WDB in advance what to do for each kind of signal.

Normally, HP WDB ignores non-erroneous signals like `SIGALRM` (to not interfere with their role in the functioning of your program but to stop your program immediately whenever an error signal happens). You can change these settings with the `handle` command.

Use caution if you disable all signals from certain processes. Disabling `SIGTRAP` in your program may cause your program to hang on HP-UX 11.00 and 11.11, because breakpoint events are treated as `SIGTRAP` signals by the operating system.

HP-UX 11.00 and 11.11 use `SIGTRAP` to communicate with the debugger. If you disable all signals from certain processes (to allow the signals to be delivered to the right process), your program may hang when you try to debug it.

This behavior occurs because if you disable `SIGTRAP`, the debugger no longer receives notification of events such as breakpoint hits and loading or unloading of shared libraries. This behavior does not occur with breakpoints and `SIGTRAP` on HP-UX 11.20 and later versions of HP-UX. This is because of changes in the `ttrace` calls that the operating system uses to communicate with the debugger.

To prevent this problem, do the following:

Ensure to set the following flag:

```
(gdb) set complain-if-sigtrap-disabled on
```

In addition, ensure that the following warning message is not displayed by the debugger before your program execution halts:

```
Warning: Thread %d (in process %d) has disabled SIGTRAPs.
Debugging this thread is probably impossible.
If you do not want to see this message again, use:
"set complain-if-sigtrap-disabled 0"
```

Debugging an Attached Process may Generate Warnings

If you attach to a program that has not been compiled for debugging, you may see the following warning messages about various registers when you step, request a back trace, or request register information from the debugger, and the process is stopped in a system call:

```
warning: reading ‘r3’ register: No data
warning: reading ‘r4’ register: No data
warning: reading ‘r5’ register: No data
warning: reading ‘r6’ register: No data
```

To avoid these warning messages, use the `finish` command to execute the process until the system call returns.

The reason for this message is that when a process stops at a system call, all registers are not readable by the debugger. The debugger’s call to the system routine, `ttrace`, returns this message when it tries to access such non-readable registers. The debugger then prints this error message from `ttrace`.

Shared Library Debug Information Missing in Stripped Program

HP WDB cannot find debug information in dynamically loaded shared libraries if the main program has been stripped. The main program is most likely stripped if HP WDB displays the following message when you load the program.

```
no debugging symbols found
```

A workaround is to use this process is as follows:

Step 1. Load the program to debug.

Step 2. Use the following command before running your program to prevent autoloading of the debug information for all shared libraries.

```
set auto-solib-add 0
```

Step 3. Run your program.

Step 4. After the program loads the relevant libraries, interrupt the program.

Step 5. Use the following command:

```
share .
```

This loads the debug information for all the shared libraries that match the "." pattern. This pattern is for all the shared libraries.

Debugging CMA Programs on HP-UX Version 11.0 and 11.11

To debug CMA programs on both HP-UX 11.0 and 11.11, build the program with `/usr/lib/libcma.sl` (there is no archive library in `/usr/lib`).

HP WDB Problem in Stepping Out of Fortran Functions

HP WDB does not reliably step out of functions with both the `step` and `next` commands. A workaround is to use the `finish` command at the end of a function.

HP WDB Fortran Support Limitations

The following Fortran features are not supported:

- Complex number arithmetic
- Type casting/conversion
- Nested routines
- Module variables

HP WDB Limitation while Debugging C++ Inlined Functions

You must compile with the `+d` option to turn off inlining if you want to call, step into, or set breakpoints in C++ functions that have been inlined.

For example, you might see the following message while debugging:

```
(gdb) p bar.head()
Cannot evaluate function -- may be inlined
```

For the call to succeed, you must compile your code with the `+d` option to get the correct result. After the `+d` option is used, the compilation displays the following message:

```
(gdb) p bar.head()
$1 = 0x4007e6f8 "inline function head called"
```

HP WDB Limitation in Accessing Virtual Classes

If a virtual class is defined in a shared library that is not compiled for debugging (`-g0`), the debug information is not available in the executable. If you try to access the class, HP WDB displays the `<no data fields>` message.

For example, `RWCollectableDate` is defined in `librwtool.sl`, which is not compiled with `-g0`.

The following line creates an object named `d3`:

```
RWCollectableDate d3(15,5,2001);
```

If you try to print the variable or class, HP WDB returns the following message:

```
(gdb) p d3
$3 = {<No data fields>}
(gdb) ptype RWCollectableDate
type = class RWCollectableDate {
    <no data fields>
}
```

HP WDB Problem with 64-bit `vfork()` Debugging

The `gdb64` component of HP WDB cannot step over call to `vfork()` because of an HP-UX limitation. The limitation is fixed in the HP-UX 11.0 Extension Pack, December 1998.

HP WDB Limitation Regarding Catchpoints

If you restart a program using the `run` command, any catchpoints you have set on C++ exceptions in shared-linked executables are deleted, and a warning message is displayed. You can recreate the catchpoints using the `catch` command.

HP WDB Limitation on Window Resizing with TUI

HP recommends that you do not resize an `hpterm` or `xterm` terminal window while using the Terminal User Interface (TUI). If you do so, the following problems occur:

- The command window contents and (`gdb`) prompt are erased. Issue the `refresh` (`ref`) command, and then press the Return key to recover the prompt.
- It is no longer possible to scroll in windows by using the Page Up, Page Down, and arrow keys. You must use the `+`, `-`, `>`, and `<` commands instead.

HP WDB Needs Linker Patch to Use +objdebug

You need to install the linker patch, PHSS_19866 for HP-UX 11.00 (or a later linker patch), to generate object modules that enable faster linking and smaller executable file sizes for large applications. Refer to your Compiler Release Notes for more details.

Fix and Continue Requires Valid TERM Type

If the value of the shell environment variable `TERM` is not set, HP WDB attempts to use settings valid for an xterm display. If the value for `TERM` is invalid, the shell prints the following message:

`sh: termtype: not found`

You must set the value of `TERM` to a valid type and restart HP WDB.

Backtrace Request Generates “Out of Memory” Error

If you request a backtrace and one of the functions was called with a very large array as an argument, HP WDB attempts to get memory to hold the array and reports that it is out of virtual memory.

The workaround is to use the following command:

```
set print args off
```

GUI Editor Windows do not Support Undo

The edit windows that WDB-GUI opens do not have undo functionality.

GUI Leak View does not Display All Items

The update pop-up menu item in Leak View collapses all items, because the memory check of HP WDB does not guarantee consistent leak numbers from subsequent `info leaks` output.

GUI may Show Incorrect Line Numbers after Fix Command

Fixing the target after deleting lines from current procedure can cause line numbers to be incorrect.

Compiling from Source Fails with Unexpected Symbol Error

When you try to compile `gdb` from the source available here, it fails to compile and generates the following error:

```
cc: "../wdb-2.1/gdb/infrtc.c", line 465: error 1000: Unexpected symbol: "mutex".
cc: error 2017: Cannot recover from earlier errors, terminating.
```

You can safely ignore this compile failure. This error occurs after `gdb` is built successfully. The error is caused by a reference to an HP-internal header file that is used to build a library that is not required to build `gdb`.

The compile generates a working `gdb` binary regardless of this error message.

Compatibility Information and Installation Requirements

This section contains installation information for HP WDB.

The HP WDB debugger consists of the following `swinstall` products:

Table 1-5

Product Name	Description
HP WDB	HP WDB Debugger
HP WDB-GUI	Optional graphical user interface component for the WDB Debugger

If you install WDB GUI on a system where HP WDB has not yet been installed, HP WDB will be installed automatically.

Compatibility

HP WDB is not supported on releases of the HP-UX operating system prior to 11.00. HP WDB 3.x is not supported on Itanium-based systems. Check the HP WDB web page, http://www.hp.com/go/wdb, for a version of HP WDB compatible with Itanium-based systems.

If you have installed linker patch PHSS_26559 on HP-UX 11.0 or PHSS_26560 on HP-UX 11.11, you must upgrade to HP WDB 3.x.

If you try to debug 64-bit shared programs with those linker patches, HP WDB versions earlier than 3.x will fail, generating the following message:

```
Version of shared library shared_library_name not understood;
there may be new features in the library that gdb does not support
```

Installation Instructions

To install your software, run the SD-UX `swinstall` command. It invokes a user interface that leads you through the installation. It also gives you information about disk space requirements, version numbers, product descriptions, and dependencies.

Specific installation instructions are available on the WDB download page. Refer to http://www.hp.com/go/wdb/.

For more information on installation procedures and related issues, refer to *Managing HP-UX Software with SD-UX* and other README, installation, and upgrade documentation provided or described in your HP-UX 11.*x* operating system package.

NOTE If you install a compiler product that includes a version of HP WDB earlier than this release, swinstall generates the following message:

```
ERROR:    A later revision (one with a higher revision number) of
          fileset "WDB-GUI.WDB-GUI-HELP,r=B.11.11.02" has already been
          installed.
          Either remove this fileset or change the "allow_downdate"
```

If you want to keep the later version, you can ignore the message. The new products will be installed, and the latest version of HP WDB will remain unchanged.

Disk Space Requirements

The approximate disk space required for HP WDB is 10 megabytes.

The HP WDB GUI requires an additional 7 megabytes of disk space.

Filesets

HP WDB is the product name for the HP WDB debugger. The HP WDB product contains the following components:

- Runtime contains one fileset: `WDB`
- Documentation contains one fileset: `WDB-DOC`
- Manuals contains one fileset: `WDB-MAN`

NOTE On HP-UX 11.0 and later, `DEBUG-PRG` is a co-requisite fileset for HP WDB and is automatically selected during installation.

The WDB GUI product contains the following sub-products:

- Runtime contains one fileset: `WDB-GUI-RUN`
- Help contains one fileset: `WDB-GUI-HELP`
- Manuals contain one fileset: `WDB-GUI-MAN`

HP WDB GUI requires the WDB product. If WDB is not already installed, it will be automatically selected during installation.

Supported Configurations

The following configurations are supported by HP WDB:

- Installing and running locally from an HP-UX 11.00 to 11.11 system
- Installing on an HP-UX 11.00 to 11.11system, with the display redirected to a remote HP-UX 9.*x*, 10.*x*, or 11.*x* node

Debugging Core Files from a Different System

Debugging a core file on system other than the one on which it was originally produced is supported under the following conditions:

1. The correct system and user shared libraries are copied with the executable and core file to the other system, and the location of the shared libraries is defined by setting `GDB_SHLIB_PATH` or `GDB_SHLIB_ROOT` before debugging the core file.

   For details about these variables, refer to *Debugging with GDB*, Section 12.2.

2. | Core files produced by: | Can be debugged on: |
   |---|---|
   | 32 bit executables | Any PA-RISC 1.1 or PA-RISC 2.0 system with an HP-UX version greater than or equal to the HP-UX version on the machine where the core file was produced. |
   | 64 bit executables | Other PA-RISC 2.0 Wide systems with HP-UX versions greater than or equal to the HP-UX version on the machine where the core file was produced. |

Patches and Fixes in This Version

This section lists major defects fixed in recent releases.

Defects Fixed in HP WDB 4.5

- `info locals` does not print once an uninitialized dynamic array is print.
- GDB is not able to print the value of external variables of FORTRAN applications.
- Shared library breakpoints `non-g` do not work.
- GDB fails in attaching a process from different directory .
- GDB gets assertion failure with shared command after run.
- Command line expression parse: Error accessing memory address 0x0.
- GDB confuses a variable with a type if they have the same name.
- GDB call function doesnot work when attaching to program.
- Temporary breakpoints do not execute their command.
- GDB is refusing to print a variable with the same name as a struct type.
- GDB on PA shows incorrect size info for `wchar_t` variables.
- `SLT_ASSIST` problem handling nested Fortran routines.
- GDB disassembly output should print line numbers on the source lines.
- Warning: The shared libraries are not privately.. for corefile.
- Unclear Internal Error: unexpected DNTT kind 0 encountered as field.
- The selected stack frame changes when switching between Views.

Defects Fixed in HP WDB 3.2

- HP WDB 3.2 checks the `dir` path before the `rooted` path for the source files.
- You can use HP WDB 3.2 to disassemble large functions.
- HP WDB 3.2 supports `Info type` for printing types information.
- HP WDB 3.2 can print the global variables defined in a shared library compiled using `-g` and `-O` (DOC mode) options.

- HP WDB 3.2 displays more explanatory error messages while debugging 64-bit applications on 32-bit machines.
- In HP WDB 3.2, the following improvements have been made for heap analysis:
  - Enables you to perform `info heap` in the child process created by the running parent process after attaching to the process.
  - Enables you to use `heap check` for stripped executables.
  - Supports `info leak` after attaching to a process.
  - Supports `memory check` in the middle of execution.
  - Supports `memory check` in a multi-threaded applications.

Defects Fixed in HP WDB 3.1.1

- The 32-bit version HP WDB 3.1.1 is now capable of recognizing new libraries being loaded by `shl_load` after a `vfork` call.

Defects Fixed in HP WDB 3.1

- HP WDB 3.1 can debug a dynamically loaded shared library even when `end.o` is not present.
- You can now print global variables declared in a shared library when you are debugging a core file.
- HP WDB 3.1 can display large Fortran arrays faster than before.
- The Visual Interface for HP WDB `shell` command does not cause the debugger to hang.
- In case of multiple inheritance, HP WDB 3.1 correctly applies “this pointer adjustment” when a pointer to a secondary base class object is cast into a pointer to the derived class.
- You can print the contents of register names by specifying the register alias names. For example:

  ```
  (gdb) print $cr27
  ```

  In the example, `cr27` is an alias to `mpsfu_high` register.
- HP WDB 3.1 can set breakpoints at functions in `typedef`’ed classes.
- Start-up time has been tuned for programs that load a large number of shared libraries.
- The debugger start-up time has been tuned for programs that mix HP aCC++ code with `gcc` code.

- Memory leak checking performance has been improved.
- The problem that sometimes caused `next` to become `continue` or `step` while debugging large shared libraries has been corrected.
- The debugger has been enhanced to automatically load the debug information from shared libraries when symbols from the libraries are referenced. For example, as arguments to the `break` command, or when you view the stack backtrace.
- The debugger has been enhanced to understand and honor the `share` command from the `.gdbinit` file.
- In some situations, backtraces displayed incorrect frames of the form `$L123`. This has been fixed.
- In some situations, the debugger did not display the source code for template functions. Arguments for such functions will be missing in the stack traces. This has been fixed.

Defects Fixed in HP WDB 3.0.01

- HP WDB correctly unwinds beyond `alloca` frames and displays a full stack trace when debugging core files.
- While debugging a core file, HP WDB searches for shared libraries and displays a warning message for each shared library it is unable to locate, not just for the first shared library.
- Some earlier versions of HP WDB were unable to handle stripped core files. This has been fixed. Now, you will be able to get a stack trace and disassembly of the program counter at the point where the program stopped.
- HP WDB no longer aborts when trying to print a variable or set a break point in a function that is located in a shared library compiled with `+objdebug`.The failure often seemed dependent on the names of the variables and functions. HP WDB now properly accesses these shared library symbols.
- Several defects fixed in the `info registers` command:
  - While debugging a core file, the `info registers` command always gave incorrect value for register 3.
  - When debugging 2.0 Narrow core files, HP WDB's `info registers` command no longer prints 32-bit values instead of 64-bit values for registers. HP WDB now prints the true 64-bit wide register values where they are appropriate.

— The `info registers` command now takes into account the screen width, controlled by the `set width` command, to determine the number of registers to display on each row. If the width has not been set, HP WDB uses the termcap database together with the value of the `TERM` environment variable and the `stty cols` setting or the `COLUMNS` environment variable to determine the screen width.

Defects Fixed in HP WDB 3.0

- Improved start-up performance when debugging programs with a large number of shared libraries.
- Class statics defined in shared libraries can be accessed properly.
- The following step after fork is cleaned up: the detached-process does not die when loading the shared libraries, and the followed-process properly gives notifications to HP WDB when HP WDB loads shared libraries.
- You can set the `GDB_SHLIB_PATH` environment variable to a colon-separated list of directory paths instead of just one shared library path, allowing more flexibility when you need to debug core files on a system other than the one that produced the core file.
- The `dir` command of HP WDB now automatically refreshes the TUI source window if you add a directory that contains currently needed source files that were previously unavailable.

Defects Fixed in HP WDB 2.1

HP WDB 2.1 contains the following fixes:

- The problem encountered when inserting breakpoints from the GUI using point and click is fixed.
- Memory leak detection fails with a message "`cannot detect leaks in a Open Graphics application`" even though the application does not use Open Graphics.
- When a secondary thread forks, WDB reports a `SIGTRAP` and hangs on continue.
- WDB hangs or misbehaves shortly after the following message: `Internal error: Continuing unhandled thread`.
- WDB intermittently hangs or stops with the following message: `Program received SIGTRAP`.
- WDB fails to attach to a process when `attach` command is invoked with a process ID.
- WDB sometimes fails to attach to a process if `SIGTRAP`s are disabled by all threads.
- The `kill` command detaches multi-threaded programs instead of killing.

- In a multi-threaded program, breakpoints trigger repeatedly. This behavior was incorrect.
- Deleted breakpoints trigger in a multi-threaded program.
- WDB may hang when the program being debugged receives a signal and is continued.

Defects Fixed in HP WDB 1.2.01

HP WDB 1.2.01 contains several fixes including:

After printing a class, HP WDB no longer core dumps when you type `q` in response to the following:

```
---Type <return> to continue, or q <return> to quit---
```

Defects Fixed in HP WDB 1.2

HP WDB 1.2 contains numerous fixes and improvements including:

- Correctly debugging a CMA thread program that forks.
- Correctly stepping over an indirect call from inside a shared library.
- Correctly handling a signal during a `next` command.
- Correctly stepping over an OpenGL call on HP-UX 10.20.
- Detaching after an interrupt does not kill the target process.
- Correctly stepping while in a system routine.
- HP WDB does not fail with `SIGSEGV` when performing traceback.
- More robust support for stack unwind, `fork/vfork/exec` notification, following child processes, and executing command line calls.
- Start-up performance improved for large C++ programs.

Relevant Documentation

This document, HP WDB Release Notes, provides overview, installation, documentation, reported defects, and troubleshooting for HP WDB.

HP WDB documentation is available online in the following directory:

`/opt/langtools/wdb/doc`

For the latest information and updates, visit the HP WDB web page, http://www.hp.com/go/wdb/

You can print the following PostScript documents, which constitute the official documentation for GDB (on which HP WDB is based):

- *Debugging with GDB*, `/opt/langtools/wdb/doc/gdb.ps`
- *GDB Quick Reference*, `/opt/langtools/wdb/doc/refcard.ps`

The PDF version of *Debugging with GDB* is available at `/opt/langtools/wdb/doc/gdb.pdf`. You need to have Adobe Acrobat Reader installed on your computer to view the PDF documents. You can download Acrobat Reader from http://www.adobe.com/products/acrobat/readstep2.html.

In addition, all of the HP WDB documentation is available through your Web browser from `/opt/langtools/wdb/doc/index.html`.

In addition to the standard GDB documentation, HP provides the following HTML documents:

- *Getting Started with HP WDB*, `/opt/langtools/wdb/doc/html/wdb/C/GDBtutorial.html` (if your browser does not support frames, view `/opt/langtools/wdb/doc/html/wdb/C/tutmain.html`)
- *XDB to WDB Transition Guide*, `/opt/langtools/wdb/doc/index.html`
- *Using the HP WDB Terminal User Interface*, `/opt/langtools/wdb/doc/index.html`

Use EMACS to view the following info files, an online version of *Debugging with GDB*:

Debugging with GDB, `/opt/langtools/wdb/doc/gdb.info*`

(Copy the files to your `info` directory first.)

The man page is `gdb(1)`.

To receive an electronic mail message only when HP releases a new version of HP WDB, join the product news mailing list.

Send an electronic mail message to `majordomo@cxx.cup.hp.com`.

To add yourself to the list, type the following in the body of the message:

`subscribe wdb-announce`.

To remove yourself from the list, type the following in the body of the message:

`unsubscribe wdb-announce`.