HP WDB 5.2 Release Notes

HP 9000 and HP Integrity Systems

Manufacturing Part Number: 5991-1264

September 2005

Printed in the United States

Legal Notices

1 Announcement

This document describes the latest changes to and features of the HP WDB version 5.2 debugger for HP-UX systems running on Itanium® and PA-RISC processors.

The latest information on HP WDB can be obtained from http://www.hp.com/go/wdb/.

What Is HP WDB 5.2?

HP WDB 5.2 is an HP-supported implementation of the GDB version 5.1 debugger. It supports source-level debugging of programs written in HP C, HP aC++, and Fortran 90 on HP-UX 11i v1.6 and later for Itanium systems, and HP-UX 11.0 and later for PA-RISC systems. This release works on programs compiled for Intel® Itanium and PA-RISC architectures.

Summary of Changes in WDB 5.2

WDB 5.2 includes the following enhancements and defect fixes:

- Enhanced inline support:

  WDB 5.2 (and later) provides improved support for debugging inlined functions in applications compiled with -g, without needing the `+d` option to disable inlining. This support is enabled for PA-RISC applications compiled with +inline_debug option, added in the A.03.65 HP aC++ compiler. Itanium applications compiled with `-g` by the A.06.02 (and later) compilers support this feature with no additional option needed. Set inline-debug on at gdb command prompt to enable debugging of inlined code. You can also use `-inline` option when starting up the gdb to enable this. By default, debugging of inlined code is disabled in gdb 5.2.

  (gdb) set inline-debug on

  or

  ```
  $ gdb --inline a.out
  ```

NOTE For PA, the current implementation will not support stepping into any function inlined through an implicit call.

The following commands have been added to enhance inline support for applications compiled with +O1 -g option:

— `step`

— `next`

— `list`

— `backtrace`

— `frame <n>`

— `info locals`

— `info args`

The `+d` option, to disable inlining need not be used.

The following commands do not work with inlined functions:

— `breakpoint`

— `info frame`

— `disassembly`

- WDB supports a new `-mapshared` option:

  Implemented the `-mapshared` option to suppress mapping all shared libraries in a process private. This option enables new facilities in the dynamic loader (patch `PHSSS_33110` or later) to designate individual shared libraries for debugging. By default, HP WDB instructs the shared library dynamic loader, `dld.sl(5)`, to map shared libraries in a process private, regardless of whether the `chatr` command is run for a particular `shlib` with `+dbg` or not.

  The `-mapshared` option is used to save virtual memory for debugging applications with large amounts of code in shared libraries, on machines with simultaneous debug sessions. The `chatr+dbg` option, and the `_HP_DLDOPTS` environment variable are used to identify shared libraries for debugging. The `-mapshared` option ensures that the text segments of all other shared libraries is shared across the system. The shared libraries are not mapped private and cannot have breakpoints in them.

  The `set mapshared on` command can be used to change modes from the `(gdb)` prompt.

```
(gdb) set mapshared on
```

  The `set mapshared off` command can be used to load shared libraries after the current point is mapped private.

```
(gdb) set mapshared off
```

  The `-mapshared option` is implemented on both PA-RISC and Itanium platforms in HP WDB 5.2. This option is provided in the WDB GUI, `wdb(1)`, as well as HP WDB, `gdb(1)`. The default behavior does not change if the `-mapshared` option for all shared libraries in processes started under the debugger, (or dynamically loaded after an attach) are mapped private.

- New command `watch_target`:

  The `watch_target` command takes an expression as an argument, evaluates it to an address, and watches the target of that address.

  For example:

```
(gdb) watch_target current_frame
```

  This is equivalent to executing:

```
(gdb) print current_frame
$1 = (struct frame_info *) 0x7fdf78
(gdb) watch *(struct frame_info *) 0x7fdf78
```

- Support for `__fpreg` data type on IPF:

WDB internally converts this data type into a `long double` data type, to evaluate an expression or print the value of the expression. Since long double has only 15 bit exponent as opposed to 17 bit exponent of `__fpreg`, precision is lost when the exponent is larger than that can fit in 15 bits.

- Support for hardware breakpoints on IPF:

  The `hbreak` command sets hardware assisted breakpoints.

  ```
  hbreak args
  ```

  The arguments `(args)` is same as that for the `break` command and the breakpoint is set in the same way. However, the breakpoint uses hardware assisted breakpoint registers. There are only 2 hardware breakpoints that can be set on IPF. This is useful in ROM code debugging and shared library debugging for libraries, including `dld`, that are not loaded private.

  The normal breakpoints are converted to a hardware breakpoint when WDB cannot set a normal breakpoint in the shared library.

- Enhanced `info symbol ADDRESS` command:

  The `info symbol` command has been enhanced to search the stack for a symbol at the given address. Previously, the `info symbol` command could be used only to search the global namespace.

- New command `info module ADDRESS`:

  The `info corruption` command can detect memory corruption in an application. That is, it reports all the memory blocks that have over-writes and under-writes.

  Syntax:

  ```
  info corruption [<file name>]
  ```

  The run time memory checking must be enabled before using `info corruption` command to detect memory corruption.

  The corruption information is written to a file specified in the file name argument if provided. Otherwise, it is printed to the `stdout`.

- New command `show envvars`:

  The `show envvars` command prints information on environment variables supported by WDB.

  Syntax:

  ```
  show envvars
  ```

- New command `info corruption`:

  The `info corruption` command can detect memory corruption in an application.

  Syntax :

  ```
  info corruption <file name>
  ```

  The run time memory checking must be enabled before using info corruption command to detect memory corruption.

  The corruption information is written to the file specified in the file name argument.

- Enhanced `nexti` and `stepi` commands:

  During instruction level stepping with `nexti` and `stepi`, WDB now prints the assembly instruction along with the next source line.

  ```
  (gdb) stepi

  0x101530:0 st4 [r9]=r34
  ```

```
1337 args.argc = argc;
```

It also prints DOC line information, which includes actual line number and the column number, when debugging a `-g -O` binary.

```
(gdb) stepi

;;; [8, 1]

0x4000820:1 nop.m 0x0
```

- New option `+check`:

  The HP aC++ compiler supports a new option `+check`. The `+check` option can be used to invoke batch RTC to determine run time memory problems. For more information on the `+check` option refer to *HP aC++ Release Notes*.

- Enhanced batch RTC support:

  The batch RTC support in WDB has been enhanced to support better reporting, and old options have replaced with new ones.

  The enhancements to the batch RTC support are:

  — Memory corruption events written to `<a.out>.<pid>.mem` file.

  — New environment variable `RTC_MALLOC_CONFIG`: `RTC_MALLOC_CONFIG=config_string1[;config_strings]`. Specify `config_string` for `+check=malloc` or wdb memory check batch mode. This overwrites the default `config` and `rtcconfig` file settings. The `config` strings are separated by ;

  — Added new `config_ strings`:

Table 1-1 New config_string

config_strings	Description
`abort_on_bounds=[01]`	Aborts execution when heap objects bounds check fail, value is 1, and `RTC_NO_ABORT` is not set
`abort_on_bad_free=[01]`	Aborts execution when free or realloc a non valid heap object, value is 1, and the environment variable `RTC_NO_ABORT` is not set
`abort_on_nomem=[01]`	Abort execution when out of memory if value is 1, and `RTC_NO_ABORT` is not set

`leak_logfile=stderr[+]filename`
`mem_logfile=stderr[+]filename`
`heap_logfile=stderr[+]filename`
	Specify the log file for batch mode.
	here:
	`stderr`: error message goes to stderr
	`[+]filename`: error message goes to filename, + means output is appended to the file

  — The batch rtc mode continues to run if the environment variable `RTC_NO_ABORT` is set.

  — The batch mode can be enabled using IPF C and C++ compiler option `+check=malloc`. For more information refer to *HP C/HP aC++ Release Notes*.

  — The `BATCH_RTC` environment variable valid values have been changed from `BATCH_RTC=1|0 to BATCH_RTC=on|off`.

NOTE A detailed section on WDB support of debugging optimized code has been added to the user manual.

- Support for deferred breakpoints on a `dlopened` and `shl_loaded` libraries with stripped main program:

  WDB triggers deferred breakpoints on `dlopened` and `shl_loaded` libraries for PA 32-bit applications with striped main program. This support is available for PA 64-bit and IPF applications.

- Support for Debugging PA-RISC Applications on Itanium-based Systems:

  This feature supports debugging PA-RISC applications and core files on Itanium-based systems in compatibility mode under Aries. Loading a PA executable on IPF system, and the debug target being a PA-RISC binary, the PA version of WDB is invoked.

  For example:

  ```
  $ gdb ./a.out
  ```

  where `a.out` file is a PA binary. When this command is run, the PA version of WDB is invoked on the IPF system. Debugging the PA binary on IPF, involves the following steps:

  — setting the `PA_DEBUG` environment variable to 1.

  — setting the `SHELL` environment variable to a PA shell.

  The PA shell is required as WDB on PA invokes it in order to parse any argument passed to the debugged application. Any shell from a PA system can be copied over to IPF systems for this purpose. All the features of WDB on PA are supported on IPF on Aries. However, the support for `MxN` debug capabilities of WDB on PA will be discontinued on Aries. On HP-UX IPF systems, PA version of WDB is packaged under `/usr/ccs/bin`. The `PATH` environment variable must include the `/usr/ccs/bin` directory.

- Enhanced commands for core files:

  WDB 5.2 introduces a set of commands for the following operations on core files:

  — Saving a core file and related information.

  — Examining the core file on a system other than the one on which the core file was created or updated.

  The following new commands have been added:

  — `packcore`: This creates a file with information about the executable and the core file being examined (e.g. `packcore.tar.Z`).

  Syntax:

  ```
  packcore [ basename ]
  ```

  — `unpackcore`: The `unpackcore` command unpacks the file made by `packcore` and loads the executable and the core file with `GDB_SHLIB_PATH` to allow backtrace and other commands to examine the core file.

  Syntax:

  ```
  unpackcore [ packore_file ] [ core_file ]
  ```

  — `getcore`: The `getcore` command can be used to examine a core file in a directory previously created by `unpackcore`.

  Syntax:

  ```
  getcore [ packcore_directory ]
  ```

- Enhanced Java debugging support:

  The following commands have been added to enhance Java debugging support:

Table 1-2 Enhancements to Java Debugging Support

Command	Description
`backtrace`	Prints `backtrace` of mixed java and native frames. Standard `backtrace` command of gdb has been enhanced to work with the mixed java and native stack frames.
`info frame`	Prints Java frame specific information for a Java frame. Standard frame command of gdb has been enhanced to interpret a java stack frame.
`info threads`	Prints state information for all Java threads.
`thread`	Prints detailed state information for the current Java thread.

- New commands for examining Java Virtual Machine (JVM) internals:

  The following commands can be used to obtain information on Java VM internals:

Table 1-3 Commands to obtain information on Java VM internals

Command	Description
`help java`	Prints list of Java and JVM debugging commands
`java`	Prints list of java subcommands

- New Java subcommands:

  The following Java subcommands have been added:

Table 1-4 Java Subcommands

Subcommand	Description
`java args <frame-number>`	Prints the current or specified Java frame arguments information
`java bytecodes <methodOop>`	Disassembles the given Java method bytecode
`java heap-histogram`	Displays the Java heap object histogram
`java instances <klassOop>`	Locates the instances of the given `klassOop` in the Java heap
`java jvm-state`	Prints current status of Java virtual machine internal states
`java locals`	Prints current or specified Java frame locals information
`java mutex-info`	Prints details of the static mutexes
`java object <object-ptr>`	Prints the given Java object field information
`java oop <Java_heap_address>`	Locates Java object oop of the given Java heap address
`java references <oop>`	Locates references to the given Java object in the Java heap
`java unwind-info <pc>`	Prints unwind information of the code where the given PC is located

Table 1-4 Java Subcommands (Continued)

Subcommand	Description
`java unwind-table`	Prints the dynamically generated Java Unwind Table

- Support for `stop in/at dbx` commands:

  The commands `<stop in function/address>` and `<stop at line>` are the dbx equivalent of `<break function/address/line>` command. WDB supports the `<stop in/at>` command in non-dbx mode.

  For example :

```
$ gdb a.out
(gdb) stop in main
Breakpoint 1 at 0x2a34: file list.c, line 18 from /tmp/a.out.
(gdb) stop at 25
Breakpoint 2 at 0x2a6c: file list.c, line 25 from /tmp/a.out.
(gdb)
```

- Support for GNU GDB logging commands:

  The following new commands have been added:

Table 1-5 GNU GDB Logging Commands

Command	Description
`set logging file`	Set the current logfile
`set logging off`	Set logging off
`set logging on`	Set logging on
`set logging overwrite[on|off]`	Set whether logging overwrites or appends to the log file
`set logging redirect [on|off]`	Set the logging output mode

Summary of Changes in WDB 5.1

This version includes corrections to defects and several enhancements.

- Support for Command-line Calls without end.o for PA64 application

  GDB 5.1 can now perform command line calls for PA64 applications that are not linked with end.o. This support is already available for PA 32-bit applications.

- Support for heap profiling/leak detection from startup of PA32 application

  PA32 RTC is enhanced to collect leaks/heap information from startup of PA32 application when environment variable RTC_INIT is turned on and RTC library LD_PRELOADed.

  $ LD_PRELOAD=/opt/langtools/lib/librtc.sl RTC_INIT=on <pa32-exe> &

Attach the process to gdb to view the complete heap profile and leak information from its startup. This is already the behavior for PA64 and IPF applications.

NOTE 1. RTC_INIT environment variable is PA32 specific and not required to be turned on for PA64 and IPF applications.

2. If RTC_INIT is set, PA32 RTC starts recording heap information by default for all PA32 processes. So, user should avoid exporting this environment variable for shell and set only when appropriate. The recommendation is to set RTC_INIT in the command line while invoking the PA32 application.

- Support for `catch nomem`: GDB 5.1 can detect and report `malloc` returning NULL pointer. The functionality is available with the RTC library.GDB can also detect NULL pointer return from `calloc`, `valloc` and `realloc`.

```
(gdb) set heap-check on

[...]   ---> till librtc.sl gets loaded

(gdb) catch nomem

(gdb)

[...]   ---> continue; when a NULL pointer is returned from malloc,

warning: Memory Allocation is returning 0x00000000 value __rtc_nomem_event (ecode=RTC_MEM_NULL,
pointer=0x0)

    at ../../../Src/gnu/gdb/infrtc.c:688


688       switch (ecode)
```

- Enhanced `ptype` command to show field layout of `struct/union/class`: A new option `-v` in `ptype` command will now display the field offset and size information of a `struct/union/class` in addition to the default type information. This option also displays filler bits used in the layout of fields.

```
Syntax: ptype [-v] [struct|union|enum|class] <TYPE-TAG|NAME>


Example:

(gdb) ptype -v struct info

type = struct info { /* off 0 bits, len 512 bits */

    int i;

        /* off 0 bits, len 32 bits */

    char a[20];

        /* off 32 bits, len 160 bits */

    struct details d;

        /* off 192 bits, len 256 bits */

    int b : 2;

        /* off 448 bits, len 2 bits */

    int c : 3;

        /* off 450 bits, len 3 bits */

    <filler>
```

```
        /* off 453 bits, len 27 bits */
    float f;
        /* off 480 bits, len 32 bits */
}
```

- For PA 64-bit applications, GDB can now step into shared library bind-on-reference calls. This support is already available for PA 32-bit and IPF applications.
- RTC now supports incremental heap profiling. It allows users to get heap growth starting from a particular time. This feature requires users to specify two parameters:
  - Interval: The user specifies the interval period in seconds. WDB collects the heap growth during this time interval.
  - Repeat Count: WDB collects the heap growth for the number of repeat count specified by the user.

  User can at any time use `info heap-interval` command to get the heap growth report. This option is similar to `info heap` command.

New Commands in HP WDB 5.1

- `set heap-check interval <nn>`

  This command starts incremental heap growth profile. All allocations prior to the execution of this command are ignored from report. If incremental heap growth profile is already on, executing this command will reset the counters and start a fresh collection.Interval is specified in seconds.

- `set heap-check repeat <nn>`

  This allows user to specify the number of intervals for which WDB should collect the incremental heap growth. The default value is 100. Every repeat of the interval tracks heap allocation during that interval.

  e.g. <gdb> `set heap-check interval 10`

  <gdb> set heap-check repeat 100

  Here WDB will create 100 incremental heap profiles which are 10 seconds apart.

- `set heap-check reset`

  GDB stores incremental heap growth data in an internal file. During each session data is appended to this file. If a session is very long, it is possible that this file may become very large. In such condition, the command discards the data existing in the file and creates a new data file. Once this command is executed, you cannot see the old data .

- `info heap-interval <file name>`

  This command creates the report of heap growth. The data for each interval has the start and end time of the interval. If file name is mentioned a detailed report is written in the file.

Summary of Changes in HP WDB 5.0

This version includes corrections to defects as well as several enhancements.

- HP WDB 5.0 supports the following commands to support debugging exception handlers in C++ on Itanium-based systems:

— `catch throw`: Catches all the exceptions when they are thrown and stops the program when an exception is thrown.

— `catch catch`: Catches all the exceptions when caught and stops the program when an exception is caught.

— `info catch [all | *address | filename:line | line]`: Prints a list of all the exception handlers that are active in the current stack frame at the current point of execution. To see other exception handlers, visit the associated frame (using the `up`, `down`, or `frame` commands); then type `info catch`. Use the argument `all` to see the exception handlers of all the frames. Use argument `*address` to see the exception handlers for a particular PC location. Use argument `filename:line` or `line` to see the exception handlers associated with a particular line.

NOTE Exception handling support on Itanium is only available with the aCC compiler A.06.00 and later. For details of the patches that you must install to use this feature, see “Software Requirements” on page 48

- Support for debugging C++ programs running under LRE (Linux Runtime Environment - IPF Linux binaries running on HP-UX): All of the basic functionality for debugging C and C++ applications is supported with the following exceptions:

  — Examining core files

  — Attaching to a running LRE process

  — Support for any language other than C and C++ (e.g.: FORTRAN, Java)

  — Run Time Checking (RTC) of heap allocations

NOTE The reference Linux platform for this feature is Red Hat 7.2

- Support for debugging of executables with method and expressions involving covariant return types: Users can step into, step out of, and step over such methods transparently without encountering covariant return adjusting thunks. This feature is supported in HP aCC5 and aCC6 compilers on HP-UX 11.22 and above, but not in g++ on HP-UX or LRE.

  Limitations: Expressions from the WDB command line that invoke methods needing a covariant return type adjustment are not supported.

- Support for FORTRAN nested routines: GDB 5.0 supports source-level debugging of fortran applications that contain nested routines. GDB will be able to set breakpoint, step-in, step-next, step-out of the nested routines and access the nested routines local variables.

- Support for template breakpoints: With GDB 5.0, you can set breakpoints on member functions in all instantiations of the template class by just specifying the template name with member function name.

  It is not necessary to specify the instantiation type.

  Setting a breakpoint on a template method with multiple instantiations displays a menu showing all instantiations and the user can choose to set breakpoints on all or any one or none.

  For example:

```
(gdb) file test

Reading symbols from test...done.
```

```
(gdb) b MyClass::MyMember
[0] cancel
[1] all
[2] MyClass::MyMember(int, int) at test.C:14
[3] MyClass::MyMember(int, float) at test.C:14
[4] MyClass::MyMember(int, double) at test.C:14
```

- Release RTC library renamed: Starting WDB 5.0, the name of the release RTC library has been changed to `librtc.so.1`. `librtc.sl` will be linked to `librtc.so.1` for backward compatibility. `Librtc.so` will also be linked to `librtc.so.1`.

Summary of Changes in HP WDB 4.5

This version includes corrections to defects as well as several enhancements.

- Support for the `backtrace_other_thread` command: Prints the backtrace of all stack frames for a thread with stack pointer SP, program counter PC and address of `gr32` in the backing store BSP.

  Usage: `backtrace_other_thread SP PC BSP`

  This is useful to get stack traces when the stack is corrupted.

NOTE Use this command carefully and make sure that you are giving reasonable SP, PC, and BSP 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 the usage information by executing the `help packcore` on command at the `gdb` prompt.
- 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 1-6

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

Table 1-6 (Continued)

Variable	Description
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>
```

NOTE This works only on PA-RISC systems because `pxdb` does not exist on Itanium systems.

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 PA-RISC 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
```

NOTE This pathname is applicable only to PA-RISC systems.

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 1-7

Variables	Description
GDBRTC_CONFIG	Specifies the location of `rtcconfig`.

Table 1-7 (Continued)

Variables	Description
`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 1-8

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

  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.

- 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.

Summary of Changes in Previous Versions of HP WDB

The following features and changes were part of the previous releases of HP WDB for Itanium and PA-RISC architectures:

- HP WDB provides a prototype for Linux Runtime Environment (LRE) debugging, which allows you to debug applications ported from Linux that run under LRE. This provides a minimal debugging capability for LRE.

  LRE allows many Itanium-based Linux binaries to be copied to an HP-UX Itanium-based system and executed without modification.

- 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.

- The `info threads` command output now shows the priority of a thread along with other thread information.

  For example:

  ```
  6 system thread 2345 Priority:178 MyFunction () at ...
  ```

- A 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.

- 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.so` version B.12.21 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.

- 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
>
```

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`.

Support for Debugging Stripped Binaries

HP WDB provides limited support for debugging stripped binaries.

Printing of Locals and Globals

HP WDB cannot print the locals and statics declared in a module that has been stripped. HP WDB, however, prints the exported symbols because exported symbols are not stripped by the strip command, they remain in `dynsym`.

HP WDB can access the globals or locals defined in other non-stripped shared libraries loaded into the stripped executable when you are in the appropriate scope.

Breakpoints

HP WDB places breakpoints using symbol names in the unstripped shared libraries loaded into the stripped executable. In stripped binaries, however, HP WDB cannot insert breakpoints using line numbers or symbolic names of the symbols not in the export list.

Backtraces

HP WDB can backtrace properly stripped frames, however, arguments will not be displayed, just as with non `-g` binaries. For fully archived stripped binaries, PCs, but not function names, will be displayed.

A known problem is that HP WDB displays improper function names instead of `unknown_procedure` for the symbols that are not known.

HP WDB versions earlier than 4.2 cannot backtrace through stripped archived binaries.

Command-line Calls

Command-line calls to the non-stripped library work correctly when called while stopped in a stripped binary, or elsewhere. Command-line calls to the functions (exported symbols) in the stripped binary work correctly.

Step and Continue

HP WDB cannot `step` into a function that does not contain debug information, nor can you use `next` to go past a line when there is no debug information. The `continue` command works correctly.

Disassembly

HP WDB cannot display disassembly for fully archived stripped binaries. You can, however, examine the addresses in assembly format with the HP WDB `x/i` *address* command:

```
(gdb) x/i 0x20000000792993c0
0x20000000792993c0:0 <time>:          mov.m          r31=ar.k7
```

The src-no-g Command

The `src-no-g` does not work for stripped binaries because the line table gets stripped.

The `--pid` Command-line Option

HP WDB now accepts `--pid` or `-p` followed by a process id on the command line.

HP WDB makes a slight change in the way it handles command-line arguments. The first non-flag argument is always a program to debug, but the second non-flag argument may either be a core file or a process ID.

In earlier versions, HP WDB would attempt to open the second argument as a core file, and if that failed, would issue an error message and then attempt to attach it as a process.

Now, if the second argument begins with a non-digit, it will be treated as a core file. If it begins with a digit, HP WDB will attempt to attach it as a process, and if no such process is found, will then attempt to open it as a core file.

Additional Support for Procedural Breakpoints

HP WDB 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 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.

Arguments Supported for Target Programs

You can use the new `--args` option to specify command-line arguments for the target program from the HP WDB command line.

Support for “array slicing” for Fortran Programs

HP WDB supports “array slicing” for Fortran programs. Array slicing allows you to give a range of elements in place of an array subscript.

The syntax is:

`:`	Lower bound to upper bound
`expr1``:`*`expor2`*	Given range
`:`*`expr`*	Lower bound to given *`expr`*
`expr``:`	Given *`expr`* to upper bound
`::`	Lower bound to upper bound, stride of 1
`::`*`expr`*	Lower bound to upper bound, stride of *`expr`*
`expr1``:`*`expr2`*`:`	Given range, stride of 1
`expr1``:`*`expr2`*`:`*`expr3`*	

Given range, stride of *expr3*

`:`*`expr1`*`:`*`expr3`*

Lower bound to *expr2*, stride of *expr3*

`expr1``::`*`expr3`*

Given *expr1* to upper bound, stride of *expr3*

Here is an example:

For this array: `INTEGER*4 A(3,3)`

The HP WDB command `print A(1:2:1, :2)` prints these values:

A(1,1), A(2,1), A(1,2) and A(2,2)

Commands to Enable and Disable Threads for DDE Compatibility

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

- The `thread disable` command prevents specified threads from running until they are enabled again using `thread enable`.
- The `thread enable` command allows the specified thread to run when you issue a `continue` or `step` command. By default, all threads are enabled. Use `thread enable` to reactivate a disabled thread.

Support for Unwinding through Corrupted Program Counter

You can perform stack tracing when the program counter (PC) is corrupted due to a call through bad function pointer.

Runtime Memory Checking is Supported

Refer to *Debugging with GDB* for details on how do memory leak detection.

With HP WDB, you can also do memory checking on the attached processes, which was not supported previously.

- To start memory checking from the time you attached:

  Attach to the process with `-leaks`, or `set heap-check on` at the gdb prompt.

```
gdb -leaks a.out pid
```

or

```
gdb a.out pid
(gdb) set heap-check on
```

- To start memory checking when the target program starts:

Step 1. To use `attach`, you need to modify the your code to call `__rtc_init_leaks(`*`int`*`)`. The value you provide controls the number of frames that HP WDB collects.

Add this code to your `main()`:

```
void __rtc_init_leaks (int frame_count);
int main()
   {
    /* Before you do anything else.. */
    __rtc_init_leaks (10);
    /* your code... */
    ....
   }
```

Step 2. Also modify your link command to link in this library:

`/opt/langtools/lib/hpux[32/64]/librtc[64].sl.`

Make sure that this is the first library in your link line.

Step 3. During attach, use `-leaks` on the command line or use `set heap-check on` at the gdb prompt.

Printing CFM and PFS Registers

HP WDB prints Current Frame Marker (CFM) and Previous Frame State (PFS) `ar64` registers twice: first as raw values as are other registers and in special formats identifying the size of rotating registers, frame and locals.

For example,

```
ar64: 0xc00000000000050c
      (sor:0, sol:10, sof:12)
cfm: 0x800000000000450a
      (sor:1, sol:10, sof:10)
```

Showing Unwind Information with the `maint info unwind` Command

The `maint info unwind` command prints the unwind information for the unwind regions at the given address expression.

Usage:

`maint info unwind` *exp*

where *exp* is an address expression.

For example:

```
(gdb) maint info unwind $pc
modsched:
0x4000930 .. 0x4000a20, end_prologue@0x4000970
Info block version:0x0, flags:0x0, length:4 * 4 == 16

0x40172b20: (0c) R1prologue    rlen=12
0x40172b21: (e8) P7preds_when  t=11
0x40172b23: (b1) P3preds_gr    gr=41
0x40172b25: (ea) P7lc_when     t=7
0x40172b27: (b2) P3lc_gr       gr=40
0x40172b29: (61) R3body        rlen=33
0x40172b2b: (81) B1label_state label=1
0x40172b2c: (c0) B2epilogue    t=44
0x40172b2e: (00) R1prologue    rlen=0
0x40172b2f: (00) R1prologue    rlen=0
```

Support for Debugging typedefs

When you have a typedef class as a template parameter you can set a breakpoint on a member function by using the command:

```
break Class<typedef_classB>::memfunc
```

- Support for stack traces in Java/C/C++ programs.
- Enhanced support for C++ templates.
- New command: `dumpcore`.
- Info threads command output changed.

Support for Stack Traces in Java/C/C++ Programs

This debugger release can show stack traces of mixed Java/C/C++ programs.

The stack trace functionality requires Java SDK version 1.3.1.06 or later for HP-UX. Please check the HP web site for Java, http://www.hp.com/go/java/, for availability.

The debugger uses the default `libjunwind.sl` library specified by the JVM. This library is part of the Java SDK version 1.3.1.06 or later for HP-UX. To override the default setting, set the environment variable `GDB_JAVA_UNWINDLIB` to the path of your `libjunwind.sl`.

The debugger supports these types of Java frames:

- interpreter
- compiled
- adapter

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

Enhanced Support for C++ Templates

HP WDB includes these features to support C++ templates:

- Setting breakpoints in template class functions and template functions without having to specify details about the instantiation.
- The `ptype` command shows any one of the class instantiations.

Setting Breakpoints in Template Class Member Functions

You can set a breakpoint in template class member functions and template functions by not specifying the type enclosed by angle brackets.

For example, if you have a template class named myClass with three instantiations:

- `myClass<char>`
- `myClass<int>`
- `myClass<volatile char* >`

To set a breakpoint on the member function `bar()` you can use the following command and pick from the menu.

```
(gdb) break myclass::bar

[0] cancel
[1] all
[2] myClass< char  volatile* >::myClass( int, char  volatile* ) at /gdb.c++/templates.cc:567
[3] myclass< char >::myclass( int, char ) at /gdb.c++/templates.cc:567
[4] myClass< int >::myclass( int, int ) at /gdb.c++/templates.cc:567
```

Displaying an Instantiation of a Template Class

To display an instantiation of a template class use the `ptype` command.

For example:

```
(gdb) ptype myclass
type = class myClass<char> {
  public:
    int x;
    signed char t;

    signed char myClass( int, char );
}
```

The dumpcore Command

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

The `dumpcore` command takes no arguments and saves the core image for the current process being debugged in the file named `core.`*`pid`*, where *`pid`* is the process ID number.

To analyze this core file with HP WDB on HP-UX B.11.22 or earlier systems, you need to do the following:

When starting from HP WDB command line:

```
(gdb) set live_core 1
(gdb) core-file core.pid
```

When starting from shell prompt:

```
% gdb --lcore a.out core.pid
```

NOTE HP WDB cannot debug the core file generated by the `dumpcore` command on a threaded program on HP-UX B.11.22.

The info threads Command Output Changed

The `info threads` command output is different than on WDB 1.4.00:

In 1.4.00 the output lines show:

```
1 system thread 2345 user thread 6 MyFunction () at ...
```

In WDB 1.4.01 the output lines show:

```
6 system thread 2345 MyFunction () at ...
```

In the WDB 1.4.01 output, the left-most number is the user thread ID, which stays constant throughout the execution of the program.

When `show-all-kthreads` is set to “1”, the `info threads` command includes threads that have no user threads attached to them. For these threads, the left-most number is a `-ve` number (`-ve` system thread ID).

Support for Assembly-level Debugging of Optimized Code

HP WDB supports assembly-level debugging of optimized code.

If a function is compiled with optimization level 2, the disassembly also displays the actual source line and column information for each assembly instruction.

Stack traces show line-number information for optimized functions.

You can set breakpoints and single-step at the source statement level for optimized functions.

Locals and argument values are not available for optimized functions.

NOTE When debugging optimized code, Fortran, C, and C++ compilers emit column numbers that represent relative position in the source line, not actual column numbers. For example, column 4 represents a location in the source line that is guaranteed to occur before column 5, though neither column 4 nor 5 are necessarily refer to the correct columns in the source line. For legacy HP C compilers, however, the column positions are correct.

Support for Debugging MxN Threaded Programs

HP WDB provides additional support for programs that use the MxN model for threaded programs on HP-UX B.11.22.

1. The `info threads` command now displays both the `utid` (user thread) and `lwpid` (system thread) for each thread. All user threads are displayed.
2. You can modify only the registers of threads that have a non-zero `lwpid` associated with them (i.e. system thread != 0).
3. You can place thread-specific breakpoints as before, but the breakpoint is associated with the `utid` of that thread.

   For example:

   ```
   6 system thread 818664 user thread 2
   ```

   When you put a breakpoint on thread 6 it will be associated with user thread 2.

Support for _Complex Variables in HP C

HP C supports a `_Complex` data type built from any of the floating point types.

A `_Complex` number holds a pair of floating point numbers, the first is the “real part” and the second is the “imaginary part”.

Here are examples of declarations and initializations using _Complex numbers:

```
float _Complex glob_float_complex;
double _Complex glob_double_complex = 6;
long double _Complex glob_long_double_complex = _Imaginary_I;
__float80 _Complex glob_float80_complex = 8 + 9 * _Imaginary_I;
```

`_Imaginary_I` is a keyword which represents the square root of -1.

The debugger has limited support for `_Complex` variables. No arithmetic operations are allowed with `_Complex` numbers. A `_Complex` number may be cast or assigned to any numeric data type and vice versa.

A `_Complex` variable can be initialized with an expression of the form:

```
A + B * _Imaginary_I
```

Where A and B are ordinary numeric expressions, perhaps in parentheses.

This is also the format in which gdb displays a `_Complex` value.

Complex numbers cannot be used in arithmetic expressions in the debugger.

For more information of `_Complex` type, refer to the HP C/ANSI C documentation.

Support for Debugging Namespaces

HP WDB for Itanium provides full support for debugging namespaces.

You do not need to use fully qualified names to access symbols within a namespace. The debugger can compile a list of namespaces active in the scope that you are in and when possible choose an appropriate symbol.

The debugger recognizes using declarations, using directives, namespace aliases, nested namespaces and unqualified lookup within a namespace. It also recognizes using directives, aliases and using declarations within namespaces.

When the debugger has a list of possible resolutions for a given symbol, it will display a menu that shows all names fully qualified whenever namespaces are involved. You can choose the appropriate symbol from the list.

For example, if you stop the debugger in a function that contains an `int i` using directive for a namespace such as:

```
using namespace A::AB::ABC::ABCD
```

You can use the command `print i` and if the only possible resolution for i is A::AB::ABC::ABCD::i the debugger prints out the name of the symbol and its value.

If, however, a global `i` exists, the debugger will display a menu from which to choose:

```
(1) i
(2) A::AB::ABC::ABCD::i
>
```

Setting breakpoints on functions works in the same way.

The debugger also allows semi-qualified names. For example, if you stop in a function in namespace B, which is nested in namespace A, and namespace A has an `int i`, you can use `print B::i` to display the value of `A::B::i`.

To disable namespace support, use the command:

```
(gdb) set namespaces-enabled off
```

Support for Debugging PA-RISC Programs on Itanium-Based Systems

You can debug PA-RISC applications and core files on Itanium-based systems. When you start HP WDB, if the debug target is a PA-RISC binary program, the debugger automatically loads HP WDB 3.0.01. The PA-RISC version of HP WDB is provided as part of the HP-UX Operating System.

Debugging a PA-RISC Core File on Itanium-Based Systems

When a PA-RISC application creates a core file on an Itanium-based system, the file name must be core.*program_name*, for example, `core.myprog`.

To debug the core file for the PA-RISC application, use the normal HP WDB command, for example:

```
$gdb myprog core.myprog
```

HP WDB will automatically use the PA-RISC version of the debugger, WDB 3.0.01, and you can debug the core file as you would on a PA-RISC system.

NOTE A core file named `core` or `core.`*`pid`* is not a core file of the PA-RISC program you are running. These core files were created by the PA-RISC emulator and cannot be used to debug your PA-RISC application.

Debugging a PA-RISC Core File on a Different Itanium-Based System

To debug a core file generated by a PA-RISC program on a machine other than the one on which the core file was generated, follow these steps:

Step 1. Transfer the executable program, core file and all shared libraries used by the PA-RISC application to the target machine.

Step 2. Set the `GDB_SHLIB_PATH` environment variable to a colon-separated list of directory path names on the system where the transferred shared libraries reside.

Step 3. Use the debugger to examine the core file.

Debugging a PA-RISC Program on Itanium-Based System

To debug a live PA-RISC program on Itanium-based systems, use these steps:

Step 1. Set the environment variable `PA_DEBUG` to 1.

Step 2. Set the environment variable `SHELL to point to a PA-RISC shell, which needs to be present on the IPF machine on which debugging is being performed.`

You can copy PA-RISC shells from the `/usr/bin` directory on HP-UX PA-RISC machines.

Step 3. Make sure `/usr/ccs/bin` is in your `PATH` environment variable. All PA-RISC HP WDB binaries on Itanium-based systems are in this directory.

Step 4. Use the debugger to examine your program.

```
$gdb a.out
```

HP WDB automatically chooses the appropriate PA-RISC version of the debugger.

Step 5. After debugging is finished, unset the `PA_DEBUG` environment variable and restore the original value of the `SHELL` environment variable.

Minimal Support for Programs that do not Contain Debugging Information

HP WDB for Itanium-based systems includes minimal support for source-level debugging of programs compiled without the `-g` option at any optimization level.

The appropriate debugger commands enable these features on programs compiled without `-g`:

- The ability to use the `list`, `step`, `next` commands.
- The disassembly command shows interleaved source lines.
- Stack traces display line numbers.

Local variable and type information, however, will still not be available without compiling with `-g`.

To enable debugging support for programs that have not been compiled with `-g`, use the `set src-no-g` command. Here are the options for the command:

```
set src-no-g = none | no_sys_libs | all
```

`none`	No source level debugging possible without `-g`. Default.
`no_sys_libs`	For all source files except those in system libraries.
`all`	For all source files.

To see the current settings, use `show src-no-g`.

Hardware Support Limited to Four Watchpoints

HP WDB includes hardware support for watchpoints. You should not set more than four watchpoints to get the best results. Setting a fifth watchpoint causes HP WDB to set a software watchpoint that can slow down in the execution of your program.

Support for Co-Variant Return Type

HP WDB 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`.

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
```

New Functionality for Visual Interface for HP WDB

Visual Interface for HP WDB includes these changes:

- A new disassembly window provides added visibility and support for monitoring programs at the instruction level. You can now easily switch between assembly and source level debugging

  To see the new disassembly window,

  — Run Visual Interface for HP WDB (`vdb`) on a program and click the `DisAsm` button in the right corner of the window

  — Run the terminal version (`vdb -tui`) and type `layout src` and `layout asm` to switch between source and assembly level debugging.

  For more information, click the `faq` button in the lower left corner of any graphical version window or see "Visual Interface for HP WDB" on page 32.

- The graphical version of Visual Interface for HP WDB adds common HP WDB commands such as `stepi`, `nexti`, `break` (on an address location), and others available via buttons and menu selections.

- You can control the automatic printing of changed registers when in disassembly mode.

  Turn this feature on by typing this command at the HP WDB prompt:

  ```
  (gdb) set print-changed-registers 1
  ```

  Turn this feature off by typing this command at the HP WDB prompt

  ```
  (gdb) set print-changed-registers 0
  ```

  When register printing is on, HP WDB prints old and new values for changed registers during assembly debugging.

- Instructions on how to copy Visual Interface for HP WDB to a location on your systems other than the HP default (`/opt/langtools/bin`) are available in the online Visual Interface for HP WDB faq, which you can view by clicking on `faq` button in any Visual Interface window.

New Terminal-Based Interface

HP WDB 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.

Additional Features in HP WDB

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

- Support for debugging 32-bit and 64-bit programs.
- Support for automatic loading of debug information from object modules.
- Support for debugging code in shared libraries and archive libraries.
- Support for assembly-level debugging.
- Support for a subset of XDB commands, enabled with the `-xdb` option.

Referencing Instructions in a Memory Location

Changes in the architecture require that you use slightly different syntax to display the instructions in a memory location or setting a breakpoint to an instruction in a memory address.

In the Itanium architecture, the memory address represents the location of a “bundle” of three instructions. Unlike PA-RISC, the Itanium architecture stores three instructions at the memory location, called the bundle address.

Each instruction at the bundle address is referenced by a slot number, either 0, 1, or 2.

On Itanium-based systems the pc register value does not represent a byte address. It represents a byte address of a bundle plus the slot number of the instruction.

For example, if you print the program counter in hex with the print command you would see:

```
(gdb) p /x $pc
$2 = 0x4000941
```

In the example,

0x4000940	represents the bundle address
1	represents the slot number.

You can use the value just as you would use any other byte address.

Other examples of addresses you might see are:

These two representations are equivalent:

- 0x40009400
- 0x4000940:0

These two representations are also equivalent:

- 0x40009402
- 0x4000940:2

You need to use the bundle address and slot number format to specify breakpoints at a specific instruction. For example:

```
(gdb) b *0x400a980:1
Breakpoint 6 at 0x400a980:1: file Hand.C, line 103.
```

This command sets a breakpoint at the second instruction in the bundle at the specified address. The output you would see at the breakpoint is:

```
Breakpoint 6, 0x400a980:1 in Show (this=0x7ffff988) at Hand.C:103
```

Debugging HP aC++ Programs Using Old Application Binary Interface

HP aC++ version A.05.01.00 adds support for new industry standard application binary interface (ABI). By default, HP WDB for Itanium-based systems correctly interprets the new ABI format.

If you need to debug programs compiled with a version of HP aC++ earlier than A.05.01.00, you must use the command `set abi old` in HP WDB before loading the file.

To debug a core file generated by a program compiled with a version of HP aC++ earlier than A.05.01.00, you must use the `-old_abi` command-line option when starting HP WDB:

```
gdb a.out core -old_abi
```

To confirm if a program uses the new ABI, use the command:

```
nm a.out | grep " _Z"
```

If the result shows mangled names starting with `_Z`, the program is a new ABI program.

Visual Interface for HP WDB

HP WDB includes an HP-supported Visual Interface for HP WDB with both graphical and terminal modes. The interface is based on Vim 5.7 and HP WDB. This interface replaces the `-tui` mode on Itanium-based systems.

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

Most of Visual Interface for HP WDB functionality is also available for `emacs` users. Visual Interface for HP WDB does not require knowledge of `vi` commands.

Visual Interface for HP WDB identifies you as an emacs user by looking at the environment variable `$EDITOR`. If this variable has a value that matches `emacs`, or `gmacs`, or `xemacs`, then Visual Interface for HP WDB starts in `emacs` mode automatically.

NOTE If your program expects unbuffered input or uses `curses`, `termcap`, or `terminfo` or otherwise transmits escape or control sequences to the terminal, you must use one of the following methods to run Visual Interface for HP WDB:

1. Start the process in one terminal and attach to it with Visual Interface for HP WDB.
2. Use the `tty` command at the debugger prompt so the program's input and output are directed to another terminal.

If the underlying GDB terminates abnormally when you are using Visual Interface for HP WDB, do not close the Visual Interface for HP WDB window. Wait for a minute or two. Visual Interface for HP WDB captures the stack trace and the debugging session details and sends you an e-mail. You can then forward this to HP when you report the problem. This is helpful to HP in reconstructing the crash scenario.

Starting and Stopping Visual Interface for HP WDB

You can use Visual Interface for HP WDB in either of two modes:

- X-window -based graphical interface: Supports mouse and keyboard commands.
- Terminal interface: Supports keyboard commands only.

Visual Interface for HP WDB accepts the same command-line arguments as GDB so you can add options to the startup command. See the manual page for GDB for the list of arguments.

- To start Visual Interface for HP WDB in graphical mode with mouse support, run Visual Interface for HP WDB with the command:

  ```
  /opt/langtools/bin/vdb
  ```

- To start Visual Interface for HP WDB in terminal user interface mode run Visual Interface for HP WDB with the command:

  ```
  /opt/langtools/bin/vdb -tui
  ```

- To stop Visual Interface for HP WDB, type `quit` on the HP WDB command line.

  ```
  (wdb) quit
  ```

Navigating the Visual Interface for HP WDB Display

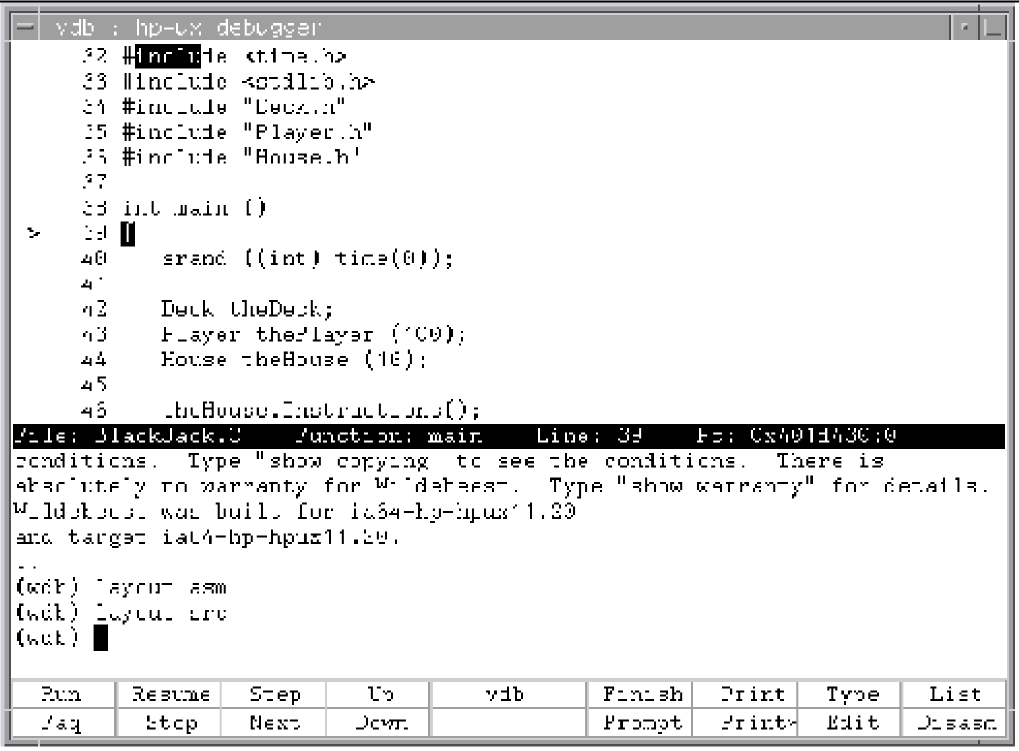
The Visual Interface for HP WDB screen consists of two areas:

- Source pane at the top
- Debugger pane at the bottom

You can use the arrow and pagination keys on the keyboard to move the cursor:

- Pagination keys always move the cursor in the source window, at the top, above the status line.
- Holding the shift key down while using the pagination keys moves the cursor in the debugger window.
- The up and down arrow keys move the cursor in the source window.
- Holding the shift key down while using the up and down arrow keys move the cursor in the debugger window.
- The left and right arrow keys always move the cursor in the debugger window.
- Two rows of labeled softkeys at the bottom of the display give you quick access to common commands.

Figure 1-1 Visual Interface for HP WDB GUI Display

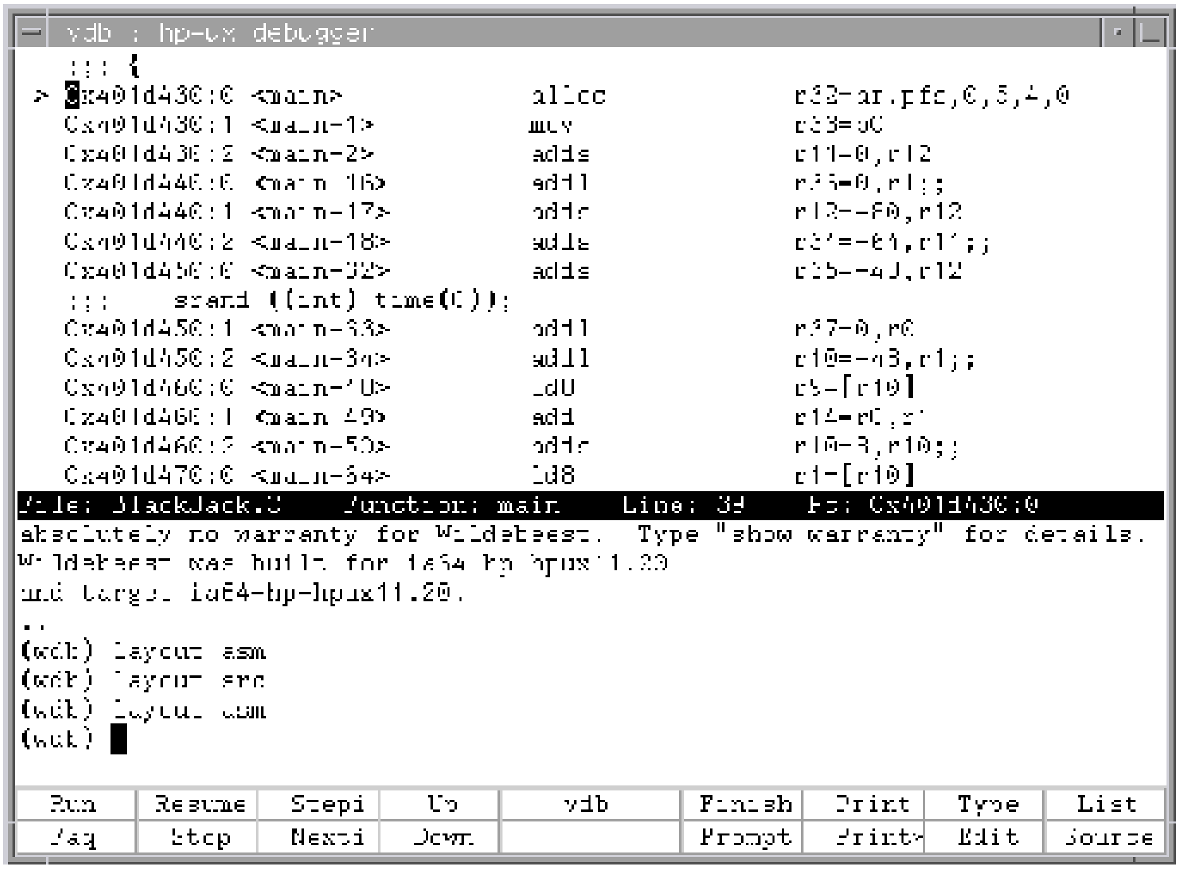


```
vdb : hp-ux debugger
#include <time.h>
#include <stdlib.h>
#include "Deck.h"
#include "Player.h"
#include "House.h"

int main ()
{
   srand ((int) time(0));

   Deck theDeck;
   Player thePlayer (100);
   House theHouse (16);

   theHouse.Instructions();
File: BlackJack.C     Function: main     Line: 39     Pc: 0x4014430:0
conditions.  Type "show copying" to see the conditions.  There is
absolutely no warranty for Wildebeest.  Type "show warranty" for details.
Wildebeest was built for ia64-hp-hpux11.20
and target ia64-hp-hpux11.20.
..
(wdb) layout asm
(wdb) layout src
(wdb)
```

You can click on the softkey or press a function key on your keyboard to invoke the command. The function keys F1 through F8 correspond to the bottom row of softkeys. The function keys f9 and up correspond to the top row.

This display show the Disassembly view.

Figure 1-2 Visual Interface for HP WDB GUI Disassembly Display



```
   ::: {
 > 0x401d430:0 <main>                alloc          r32=ar.pfs,0,5,4,0
   0x401d430:1 <main+1>              mov            r33=b0
   0x401d430:2 <main+2>              adds           r11=0,r12
   0x401d440:0 <main+16>             addl           r35=0,r1;;
   0x401d440:1 <main+17>             adds           r12=-80,r12
   0x401d440:2 <main+18>             adds           r34=-64,r11;;
   0x401d450:0 <main+32>             adds           r35=-40,r12
   :::     srand ((int) time(0));
   0x401d450:1 <main+33>             addl           r37=0,r0
   0x401d450:2 <main+34>             addl           r10=-43,r1;;
   0x401d460:0 <main+48>             ld8            r5=[r10]
   0x401d460:1 <main+49>             add            r14=r0,r1
   0x401d460:2 <main+50>             adds           r10=8,r10;;
   0x401d470:0 <main+64>             ld8            r1=[r10]
File: BlackJack.C     Function: main     Line: 39     Pc: 0x401d430:0
absolutely no warranty for Wildebeest.  Type "show warranty" for details.
Wildebeest was built for IA64 hp hpux11.20
and target ia64-hp-hpux11.20.
..
(wdb) layout asm
(wdb) layout src
(wdb) layout asm
(wdb) █
```

Specifying Foreground and Background Colors

To change the foreground and background colors, update the `.Xdefaults` file in your home directory. The resources are the same as for `hpterm`.

Here is a sample entry:

```
HPterm*foreground:                white
HPterm*background:                rgb:68/90/C4
```

Using the X-window Graphical Interface

To start Visual Interface for HP WDB in graphical mode with mouse support, run Visual Interface for HP WDB with the command:

```
/opt/langtools/bin/vdb
```

Visual Interface for HP WDB opens an `hpterm` window, ignoring the value of the `TERM` environment variable, for debugging a program.

With a mouse you can do the following:

- You can left-click on the line number to insert or remove breakpoints.

- You can left-click on an identifier to select the identifier as an operand for the `Print`, `Print*`, `Type`, `List` softkeys.
- Where necessary you can manually select an expression by dragging the cursor over it.
- You can right-click on the line number to activate a pop-up menu with several useful commands.
- You can right-click on an identifier to automatically select it and use the selection as an operand for the pop-up window that appears.
- You can right-click on an empty region for a third pop-up menu with several useful actions. For example, see Saving session to file.
- You can left-click on the command softkeys at the bottom of Visual Interface for HP WDB window.
- You can click middle button to paste the selection.
- You can drag the status bar with the mouse to resize the debugger window relative to the source window.

Using the TUI Mode

To start Visual Interface for HP WDB in terminal user interface mode, run Visual Interface for HP WDB with the command:

```
/opt/langtools/bin/vdb -tui
```

This mode works well with `hpterm` and `xterm` and fairly well with `dtterm` and `vt100 (telnet)` terminals.

NOTE A defect in `dtterm` may truncate the display of lines that do not fit within the window. To work around this defect you can refresh the display with CTRL-L or widen your terminal window so source lines will not wrap.

If you use `xterm` and `dtterm`, you should update your `.Xdefaults` file with keyboard translations to get the shifted arrows and shifted paging keys to work.

For `xterm` use these settings:

```
*XTerm*vt100.translations: #override \
Shift <Key>Prior:    string(0x2)  \n \
Shift <Key>Next:     string(0x6)  \n \
Shift <Key>Up:       string(0x5)  \n \
Shift <Key>Down:     string(0x19) \n \
Shift <Key>Left:     string(0x1b) string(i) \n \
Shift <Key>Right:    string(0x1b) string(la)
```

For DtTerm use these settings:

```
*DtTerm*Translations: #override\n \
Shift <Key>osfPageUp: string(0x2)  \n \
Shift <Key>osfPageDown: string(0x6)  \n \
Shift <Key>osfUp:       string(0x5)  \n \
Shift <Key>osfDown:     string(0x19) \n \
Shift <Key>osfLeft:     string(0x1b) string(i) \n \
Shift <Key>osfRight:    string(0x1b) string(la)
```

Mouse operations are not supported in the `-tui` mode. Also the paging and shift keys do not work well with vt100 terminals.

Changing the Size of the Source or Debugger Pane

If you are using the GUI mode:

1. Escape to `vi` command mode.
2. Drag the status bar to the desired position using the mouse.

If you are using `-tui` mode, use these commands to change the size of the current window:

CTRL-W +	to increase
CTRL-W -	to decrease

In Visual Interface for HP WDB, the current window is usually the debugger window.

Using Commands to Browse Through Source Files

Visual Interface for HP WDB is based on `vim`, so you can also use `vi` commands to browse. For example, CTRL-B, CTRL-F, CTRL-D, CTRL-U are useful for browsing the debugger window. These commands work whether or not you escape to `vi` mode.

These `vim` commands require you to escape to `vi` mode.

`/`	Search forward
`?`	Search backward
`n` `N`	Repeat search
`%`	Match braces
`[[ ]]`	Skip to the next procedure
`:`*`line number`*	Go to any line number

All these commands require you to escape to `vi` command mode first. When you are done, type a for append or i for insert or other `vi` commands to return to text insertion mode.

You can also simply click on the `Prompt` softkey.

Loading Source Files

Escape to `vi` command mode and use the `:e` command to load a source file.

```
:e filename
```

When the source files are located in multiple directories, you can simply specify the base name alone as long as file names are unique and the appropriate `dir` commands have been executed.

Pressing the `Prompt` softkey takes you to the command prompt and also updates the source window so that the cursor is left at the point where the program is stopped.

Editing Source Files

To edit a file, kill the process then click on the `Edit` button. If you do not kill the process, the source file and binaries could get out of sync.

Editing the Command-line and Command-line History

Visual Interface for HP WDB preserves the entire session's transactions so you can browse through these at any time.

To edit the command line press ESC to enter `vi` mode and then use `vi` commands. You can recall previous commands in history by using `[jk^P^N]`. Complete command lines using TAB.

Saving Contents of a Debugging Session to a File

You can save the contents of the current debugging session, including debugger input/output and program input/output) to a file.

To save a session to a file:

1. Right-click on an empty region of the source or debugger pane.
2. Choose `Save Session to vdb.pid` from the pop-up menu.

The debugger writes the input and output to a file whose name ends in the process ID (pid) of the debugger. If you save the session more than once, the new transactions are appended to the file.

2 Known Problems and Workarounds

This section describes known problems and suggested workarounds.

Known Problems in WDB 5.2

- **Disassembly and breakpoints do not work for inline functions by line number:**

  The inline functions cannot be disassembled by line number. Also, breakpoints cannot be set in inlined code by line number.

  Work Around

  A workaround is to use addresses or to disassemble the containing function.

- **Debugging longjmp targets:**

  The process being debugged does not stop over a longjmp call.

  Work Around

  A breakpoint is placed at the longjmp target. The process stops when the breakpoint is reached rather than after completing execution.

Known Problems in WDB 5.1

The internal format of the register context changed between 11.22 and 11.23 to support new scratch registers (ar.csd/ar.ssd) and lazy FP. Libuca changed to match the new format and does not support the old format.

Work Around

- If you need to debug an 11.22 core file, you need an 11.22 libuca.
- If you need to debug an 11.23 core file, you need an 11.23 libuca.

PA-RISC Debugger Features Not Available in WDB 5.0

This release of WDB does not support these PA-RISC debugger features:

- The `-tui` mode. This release supports a command-line interface, HP Visual Interface for HP WDB, and the HP WDB GUI.
- Debugging with Fix and Continue.
- Conditional reporting of system call entry and exit events

Known Problems on Itanium

- When following a child process created using `vfork()`, you cannot get control of the child process before it execs.

  The earliest you can get control is at the exec point by using the command `catch exec`. The debugger ignores all breakpoints in the child prior to the exec. If both `catch vfork` and `catch exec` are set, only the exec event might be caught.

- HP WDB may show incorrect information for `ptype` of a structure that has the same name as a variable, as in this program:

  ```
  int XYZ;

  int main ()
  {
    struct XYZ {
      int ii;
    } ny;
    XYZ = 10;
    ny.ii = 11;
  }
  ```

  When stopped in `main()`, if you use **`ptype XYZ`** (c++ style), HP WDB prints the `ptype` of the global `XYZ`, not the `struct XYZ` defined in this scope.

  The workaround is to use **`ptype struct XYZ`** to get the type of `struct XYZ`.

- When debugging a program compiled on HP-UX 11i v1.6 with the `+noobjdebug` option, stepping into a static function may cause the debugger to behave as if `next` was used.

  The workaround for this problem is to set a breakpoint on the function and then use the step command:

  ```
  (gdb) break static_function
  (gdb) step
  ```

  This causes the debugger to hit the breakpoint in the function.

  You can also minimize the problem by installing the patches listed in “Software Requirements” on page 48 and recompiling your program.

- You cannot execute a function while a system call is interrupted.

  HP WDB may print this message when you attempt to use a command such as `print my_function()`:

  ```
  You may not execute a function while a system call is interrupted.
  ```

  HP WDB displays the message when your program was interrupted in the middle of a system call, for example, `sleep()` or `read()`, by some signal, such as one caused by pressing Ctrl-C.

  The workaround for this limitation is to use these commands when you see the error:

  ```
  (gdb) up
  (gdb) tb
  (gdb) c
  (gdb) print my_function()
  ```

The command `tb` is an abbreviation for `tbreak`, a temporary breakpoint and `c` is an abbreviation for `continue`.

These commands set a breakpoint on the return address of the system call and continue. When the program hits the breakpoint, it will be removed and you will get a debugger prompt. You can now call `my_function()`.

- Disabling threads can lead to deadlocks in your program that prevent Ctrl-C from stopping your program. If this happens, you might be able to stop your program by using fatal signals such as `SIGSEGV` or `SIGBUS`.

  You can use these steps to try to stop your program:

  1. Send this command from another shell.

     `kill -SIGSEGV` *`pid`*

     This causes HP WDB to report the signal and show the command prompt.

  2. At the command prompt, examine the threads.
  3. Enable one or more threads to get out of the deadlock.
  4. Issue `signal 0` to continue without delivering the signal to the program.

- HP WDB cannot properly debug programs compiled with HP aC++ versions earlier than A.05.36.

  HP WDB displays this warning when you try to print information on variables in virtual classes or virtual member functions:

  ```
  (gdb) print x
  warning: x ambiguous; using JV::LV::A1::x. Use a cast to disambiguate.
  $1 = 1610612736
  ```

  Where *x* is a element in a virtual class.

  The workarounds include the following:

  — In HP WDB, use the command `set old-vtable on`, then reload your file with the `file` command.

  — Use WDB 1.3.02 to debug your aC++ program.

  — Re-compile your aC++ program with HP aC++ version A.05.36 or later.

- HP WDB does not properly handle a command line call of a function that returns an aggregate consisting of 8 or fewer float or double numbers, all of the same size. The Itanium architecture calling conventions handle these differently than other aggregates.

- When you have multiple shared libraries with the same function name and you set a deferred breakpoint on that function name, HP WDB sets a breakpoint only in the first shared library loaded that contains the function. HP WDB does not set breakpoints when the given function is defined in any subsequently loaded shared libraries.

  To work around this problem, use the `catch load` command to take control when HP WDB loads the shared library that contains the function on which you want to set a breakpoint. For example:

  ```
  (gdb) catch load libtest1.sl
  ```

  The `catch load` command stops the debugger after the library `libtest1.sl` loads. You can now set breakpoints on any function in the shared library.

- The `set threadverbose` functionality is not available.

Known Problems on PA-RISC Systems

`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.

3 Compatibility Information and Installation Requirements

This chapter contains installation information for HP WDB and HP WDB GUI.

HP WDB is bundled with the HP C, HP ANSI C++, and Fortran compilers.

The HP WDB debugger consists of the following swinstall products:

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.

The WDB GUI is compatible only with the corresponding version of HP WDB.

NOTE Before using the `file` command on programs that contain virtual functions and have been compiled with the B.11.22 compiler or earlier, you must use this command:

```
set old-vtable on
```

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.

For more information about 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.

Software Requirements

To use the exception handling feature on HP-UX 11i v2 (Itanium), you must install the following patches:

- PHSS_31849 - linker + fdp cumulative patch (supersedes PHSS_31087)
  PHSS_31850 - assembler patch (supercedes PHSS_29679)
  PHSS_31851 - Integrity Unwind (supercedes PHSS_31086)
  PHSS_31852 - aC++ Runtime (IA A.05.5?, PA A.03.5?) (supercedes PHSS_30414)
  PHSS_31853 - math library cumulative patch (supercedes PHSS_30719)
- PHSS_31854 - milli cumulative patch (new)

To use the memory leak detection features on HP-UX 11i v1.6, you must install this patch:

- PHSS_28976 CORE libunwind

If you are installing on HP-UX 11i v1.6, these patches are recommended:

- PHSS_28700 C++ Compiler Product
- PHSS_28701 C Compiler Product
- PHSS_28702 u2comp/plugin library patch
- PHSS_28970 C++ library patch
- PHSS_28971 Linker and fdp cumulative patch

When you install the C++ run-time patch PHSS_29659, you must upgrade to HP WDB 4.2.

For debugging PA applications built with `+objdebug` option, use compiler version A.3.55 (or above) and linker version B.11.41 (or above).

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

HP-UX 11i Release Names and Release Identifiers

With HP-UX 11i, HP delivers a highly available, secure, and manageable operating system that meets the demands of end-to-end Internet-critical computing. HP-UX 11i supports enterprise, mission-critical, and technical computing environments. HP-UX 11i is available on both PA-RISC systems and Itanium®-based systems.

Each HP-UX 11i release has an associated release name and release identifier. The *uname* (1) command with the `-r` option returns the release identifier. The following table shows the releases available for HP-UX 11i.

Table 3-1 HP-UX 11i Releases

Release Identifier	Release Name	Supported Processor Architecture
B.11.11	HP-UX 11i v1	PA-RISC
B.11.20	HP-UX 11i v1.5	Intel® Itanium®

Table 3-1 HP-UX 11i Releases (Continued)

Release Identifier	Release Name	Supported Processor Architecture
B.11.22	HP-UX 11i v1.6	Intel® Itanium®
B.11.23	HP-UX 11i v2	Intel® Itanium®

Disk

The approximate disk space required for HP WDB is 4.5 megabytes.

HP WDB GUI requires 7 megabytes of disk space.

Filesets

HP WDB is the product name for the HP WDB debugger.

The HP WDB product contains the subproducts:

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

Supported Configurations

The following configurations are supported by HP WDB:

- Installing and running locally from an HP-UX B.11.22 or later Itanium system, or 11.0 or later PA-RISC system.
- Installing on an B.11.22 or later Itanium system, or 11.0 or later PA-RISC system, with the display redirected to a remote HP-UX 9.x, 10.x, or 11.x node.

Relevant Documentation

This document, HP WDB Release Notes, provides overview, installation, documentation, and problem information for HP WDB for Itanium-based applications.

All of the 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/

NOTE All documentation other than these release notes describes the PA-RISC version of the debugger. Some features described in the documentation may not yet be available in the Itanium architecture version.

You can print the following PostScript documents, which constitute the official documentation for GDB version 5.1 (on which HP WDB for Itanium-based systems 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`, which you can read with the Acrobat Reader, `acroread`. If you do not already have `acroread`, it is available free from Adobe Systems Inc. (http://www.adobe.com).

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/GDBtutorial.html` (if your browser does not support frames, view `/opt/langtools/wdb/doc/html/wdb/C/tutmain.html`)
- "XDB to WDB Transition Guide", contained in *Debugging with GDB*, `/opt/langtools/wdb/doc/html/wdb/C/debug.html`
- "Using the HP WDB Terminal User Interface", contained in *Debugging with GDB*, `/opt/langtools/wdb/doc/html/wdb/C/debug.html`
- Use EMACS to view the following info files, an online version of *Debugging with GDB*:

  `/opt/langtools/wdb/doc/gdb.info*` (Copy the files to your usual info directory first.)
- The man page is "gdb"(1).
- The man page is "wdb"(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, in the body of the message type the following:

  `subscribe wdb-announce`

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

```
unsubscribe wdb-announce
```

4 Patches and Fixes

This chapter describes the defects fixed in various versions of HP WDB.

Defects Fixed WDB 5.2

The following defects have been fixed in WDB 5.2:

For uninitialized ptr arg, WDB does not print Fortran 90 function arguments and array contents.

Defects Fixed in WDB 5.1

The following defects have been fixed in WDB 5.1:

- ptype displays inline member function as static: missing () too.
- `core-file/packcore` displays only 14 characters of the executable name in core.
- `unpackcore` fails when not executed in the `unpack` directory.
- STL code causes debugger problems.
- RTC does not handle `shl_unloads` if they happen in the library initializer.
- `backtrace` does not unwind upto main after command line call in PA32.
- WDB cannot print tls variables from a (DOOM) shared library.
- WDB unable to set breakpoint using absolute path name.
- WDB should list src lines even if exe is moved out of compiled location.
- WDB can't handle largefile core files.

Patches and Fixes in WDB 5.0 (Itanium)

The following defects related to WDB on Itanium systems have been fixed in HP WDB 5.0:

- GDB dies printing a large array
- WDB manpage needs update for 5.0/HP-UX 11i v2 release
- GDB dumps core when debugging gdb at set_demangled_name()
- Usability/documentation problems with heap-checking
- Command line expression parse: attempt to take address of non-lval
- GDB is printing local variables incorrectly
- After up command, $pc is showing as zero
- Backtrace after attach does not work in some cases
- ToT compiler empties prologue, causes function breakpoints set on second line
- info thread problem with MxN core dump

- covariant.exp fails for aCC5, not aCC6 due to wrong instruction decoding
- Bad code in thread.c
- Fix compiler warnings as of 7/723/04
- Executable breaks in .gdbinit left deferred at launch
- GDB crashes when reading Caliper core file
- GDB SIGSEGV/coredump on re-run after modifying NFS-mounted executable
- Attach does not work if the command line is very long
- RTC needs _rtc_init_leaks call in yr code to be able to attach to it
- GDB shows ’next’ behaviour on nested routines of FORTRAN applications
- LRE testing needs to be enabled without special symbolic link on Linux
- GDB not recognizing g++ mangling used on RH7.2
- Until command does not work properly when stepping over a for loop
- Unable to set psr.um
- Cntl-C & bt after resuming an attached process gives "protocol error"
- GDB build is getting compiler warnings (all platforms)
- IPF gdb aborts when issuing bt command when debug of f90 prog
- GDB heap-check does not work with f90 on Itanium systems
- xbreak gets error in simple acc6 program
- With src-no-g on, GDB segmentation violates in solib-d.exp
- anon-union test fails with acc6 because of change in dwarf
- aCC6: member-ptr.exp - dereferencing pointers to class members errs
- aCC6: member-ptr.exp - ptype of a pointer to a member function fails
- aCC6: +noobjdebug fails to set bkpt on nested-template function
- The src-no-g command option is in man page, but not in gdb --help
- GDB only recognizes one of: b Temp::conv(void) and b Temp::conv()
- "help go" is confusing
- GDB gets confused with breakpoints set with unswizzled addresses

Patches and Fixes in WDB 5.0 (PA-RISC)

The following defects related to WDB on PA-RISC systems have been fixed in HP WDB 5.0:

- Switching to IA GDB fails
- WDB 4.5: info reg output is wrong for 32-bit core
- Pathmap does not work when it is set after a combo of file load and list

- GDB crashes for info break - conditional breakpoints
- GDB: breakpoints do not work correctly
- When switching from gdb32 to gdb64, arguments are not being passed properly
- "No Locals" in main scope of (nested routines) FORTRAN application
- Unable to do command line call on attach from shlib space
- GDB on PA shows incorrect type and size info for wchar_t variables
- GDB gets assertion failure with shared command after run
- GDB to support new core file format for HP-UX 11i v2 September 2004 Release and 11.31.
- GDB64 - attached process terminates with SIGBUS on HP-UX 11i v2 September 2004 Release
- usrthfork.exp-fails on doing a next on fork() on HP-UX 11i v2 September 2004 Release
- GDB should list src lines even if exe is moved out of compiled location
- GDB is not able to print the value of external variables of FORTRAN application
- GDB fails in attaching a process from different directory
- New Batch RTC implementation in PA-RISC
- MERGE : release_ux gdb32 fails command line calls returning structs > 8 bytes
- Command line expression parse: Error accessing memory address 0x0
- GDB does not print a variable with the same name as a struct type
- Detect library and corefile mismatches
- Corefile not validated prior to use
- GDB call function does not work when attaching to program
- Shared library breakpoints non-g do not work
- Catch unload does not catch statically loaded libraries in gdb32
- rtc_child.exp testsuite fails in nightlies
- GDB: conditional breakpoints do not work after rerun
- GDB could not find object files and/or source files
- WDB/GDB does not display thread priority in ’info threads’ output
- GDB gives bad stack trace and no warning if shared lib not found
- Enhance GDB to validate the core file and issue a warning if mismatch
- Minor error in README.HP_WDB on wdb 2.1 version
- Temporary breakpoints do not execute their command
- WDB in info shared command should indicate unloaded libraries
- BugUse: 2: Set BP on template member functions across all instantiated template classes

Patches and Fixes in WDB 4.5

The following defects have been fixed in HP WDB 4.5:

- Missing stack traces in leak report
- Call from the gdb command line gets "Attempt to take address of non-lval" message
- gdb 4.2[.01] incorrectly parses `pgm` file name when attaching
- No line 0 in module.C.
- GDB hangs with `stty tostop`
- C++ member functions in namespaces cannot be called from command line, breakpoints set or listed
- `batchrtc` does not work for major ISV
- On rereun multiple choice list for bp does not handle invalid input
- Deferred breakpoints should use demangled name for comparison
- WDB/GDB fails finding source lines mixed C/C++ source file
- Command line call of member function of reference cannot be resolved
- WDB should validate core file being debugged
- `gdb.c++/overload.exp`: Not displaying destructor in `ptype`
- `gdb.c++/infotypes.exp`: Duplicate types emitted in info types
- Comparing pointer to integer silently gives wrong answer
- Report text form of `si_code` instead of or in addition to numeric form
- Incorrect register contents displayed for `fr67` and higher
- GDB sends `SIGBUS` when used `__rtc_init_leaks` and linked with `librtc.sl`
- Breakpoints in shared libraries are not maintained across `exec()` calls
- Problems getting gdb leak detection to work on IA64
- GDB searches local directory for `objdebug` info incorrectly
- Catch load events should use commands and conditions
- WDB disassembly view does not format register name for popup value view
- Allow access to members of anonymous unions
- print/ptype of a template class not supported on IPF, only instantiations
- Getting "unable to find dynamic library list" error when debugging core
- Problems showing `natr` registers
- `info shared` should say when library's symbols not loaded due to `auto-solib-add`

Patches and Fixes in WDB 4.2.01

The following defects have been fixed in HP WDB 4.2.01:

- WDB 4.2.01 cannot unwind the stack through Java stack frames.
- Older versions of WDB do not unwind through signal handlers when the signal handler is called from Java code.
- WDB “steps” into shared libraries broken by linker import stub change.
- WDB crashes during the solib.exp test on wasp and when WDB starts debugging itself.

Patches and Fixes in WDB 4.2

This version of HP WDB includes corrections to numerous defects. These are some of the defects fixed in this version:

- HP WDB stops with a core dump at `set_demangled_name()`.
- Crash on startup when debugging stripped archived executables.
- Unable to backtrace through stripped archived executables.
- Shared library breakpoints on non-`g` libraries on rerun now work properly.
- Hang when attempting to `break` across `execv`.
- Watchpoint change during `syscall` were not reported.
- “Internal error: unknown thread..” error fixed for noticed cases.
- HP WDB can now print a variable with the same name as a struct type. See “Known Problems on Itanium” on page 41 for more information.
- Out of range `enum` values were being printed incorrectly.
- Object files were picked up from the wrong location even after `objectdir` was specified.
- NaN values were incorrectly identified as inf values.
- IP offsets were printed in decimal for disassembled code and in hexadecimal elsewhere. All are now in hexadecimal.

Defects Fixed in WDB 4.1

This version of HP WDB includes corrections to numerous defects.

Defects Fixed in WDB 4.0

- On HP-UX B.11.23, you can unwind through signal handlers when debugging core files generated on an Itanium-based system on a different Itanium-based system.
- HP WDB gives the correct address for a bit field in a structure.
- The debugger supports the environment variables `$GDB_SHLIB_PATH` and `$GDB_SHLIB_ROOT`.
- When displaying disassembly, the debugger shows pseudo instructions and symbolic names for branch destinations where appropriate.
- You do not need to compile with the `+noppu` to debug Fortran 90 executables. You can set a breakpoint using the function name without appending an underscore.
- The debugger no longer hangs when restarting after disabling a watchpoint.
- The debugger is aware of shared libraries loaded after a `vfork ()` call.
- You can restart a program after an “unable to read bundle” error message.
- When you specify a function name, the debugger now looks for multiple occurrences of symbol with the same name.
- The debugger preserves the conditions you set when you set a breakpoint in a shared library.

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 3.0.01

- 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 WDB 1.4.00

- Line number information prints correctly
- Debugger correctly displays information for GR20
- WDB-GUI correctly displays breakpoints in disassembly window
- Debugger correctly handles threads that use `fork()` or `exec()` or both
- Debugger properly handles `libsin.so.1`
- Debugger does not terminate with stack growth failure error
- Debugger correctly displays `$ar17` (`ar.bsp`)
- Debugger correctly steps into shared library that is loaded `BIND_IMMEDIATE`
- Option to specify source file directories works correctly
- Debugger correctly displays function with hex offset in stack traces when line numbers do not print
- Improved performance when displaying arrays

Defects Fixed in WDB 1.3.02.

- Printing of C++ static members now prints correct values
- You can insert breakpoints by point and click in the WDB GUI
- Long double variables can be accessed correctly
- You can make command-line return calls
- You are no longer restricted to be on a bundle boundary when making a command line call
- Breakpoints on different source lines that are in the same bundle work correctly
- Unused variables with no address assigned print <variable optimized out> when accessed
- xdb-mode command `bx n` sets breakpoint on the last line in function at depth n instead of the first line of the function at depth n-1
- Auto-solib-add now works properly

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.