HP WDB 5.7 Release Notes

HP-UX 11i v1, HP-UX 11i v2, and HP-UX 11i v3

HP Part Number: 5992-0593
Published: September 2007

Legal Notices

Table of Contents

List of Tables

HP WDB 5.7 Release Notes

Announcement

HP Wildebeest Debugger (WDB) 5.7 is an HP-supported implementation of the open source debugger GDB. It supports source-level debugging of programs written in HP C, HP aC++, and Fortran 90 on Itanium®-based systems running HP-UX 11i v2 or HP-UX 11i v3, and HP 9000 systems running HP-UX 11i v1, HP-UX 11i v2, or HP-UX 11i v3.

This document discusses the most recent product information for HP WDB 5.7

For the latest version of HP WDB, see the HP WDB Technical Resources website at:

http://www.hp.com/go/wdb

What Is New in This Version

This section describes the new features that are introduced in this version of HP WDB.

The following new features are supported in HP WDB 5.7 for Integrity systems:

The following new feature is supported in HP WDB 5.7 for HP 9000 systems:

What is New in This Version of HP WDB for Integrity Systems

The following new features are introduced in this version of HP WDB for Integrity systems:

Enhanced Support for Debugging Optimized Code

HP WDB 5.7 provides the following enhanced support for debugging optimized code:

Printing Values of the Local Variables in Optimized Code Built With –g

HP WDB 5.7 enables you to examine local variables and formal parameters in programs that are compiled with optimization levels above `+O1` (available for compiler versions `A.06.15` and later). This feature is available for live processes, and core files that are created by optimized code. Limited stepping operations in the optimized code continue to be supported with no additional enhancements.

The debugger reports the values of the variables in the optimized source code. However, as a result of optimization, there are code locations where the value of the local variables may not be

available. In such cases, the debugger reports that the value of the local variable is not available because of optimization. The debugger displays the current code address, which is used in the attempt to recover the value of the variable. If the variable is optimized completely out of existence, the debugger reports the corresponding message.

Optimization of code results in the reordering of the instructions and the source line-numbers. Hence, the value of the variable, which is printed by the debugger may not correspond to the reported source code location. The debugger may print the value of the variable at a source code location either before or after the reported source code location.

NOTE: The following limitations apply when debugging optimized code:

- Support for high-level loop transformations such as modulo-scheduled loops, or LNO-optimized loop nests is limited.

  (This limited support includes all loop optimizations that are enabled at `+O3` and above, and some loop optimizations at `+O2` or `-O`.)

- Debug support for local aggregates and arrays is limited.
- Complete debug support for inlined subroutines is not available.
- Variable values that are unavailable at the current code location are reported unavailable, even if these values can be computed from some other values that are available.
- Step operations may include occasional "`backwards`" steps, because of the reordered code during optimization.

Support to Prevent the Debugged Program from Stopping at Instructions that are Predicated False

This enhancement in HP WDB 5.7 provides support to prevent the debugged program from stopping at instructions that are predicated false. The program execution can be stopped by a software breakpoint, a hardware breakpoint, or an asynchronous signal. In the case of optimizations such as if-conversion, the predicated false instructions indicate that an alternate source path is executed. Hence, stopping the program at a predicated false instruction results in the misleading conclusion that the path corresponding to the predicated false instruction is executed.

To prevent this ambiguity, HP WDB 5.7 does not stop at predicated false instructions. The predicated false instructions are equated to NOPs (No OPeration), because these instructions do not modify the processor state.

NOTE: The program stops at asynchronous signal stops even if the reported instruction is predicated false.

The exception to this rule is the use of certain instructions, such as `wtop`, `wexit`, and `frcpa`, which modify the processor state even when predicated false. In such cases, the debugger stops at the instructions irrespective of the predicate value of the instructions.

Assembly and low-level programmers, who require the old behavior of the debugger to stop at the instructions irrespective of the predicate value of these instructions, can explicitly turn off this feature.

To explicitly turn off this feature, enter the following command at the `gdb` prompt:

```
(gdb) set no-predication-handling 1
```

Debugging Macros

HP WDB 5.7 enables you to display and evaluate macro definitions for programs running on Integrity systems. This feature is available only for compiler versions `A.06.15` and later.

HP WDB 5.7 provides the following support for debugging macros:

- **Displaying Macro Definitions**

  HP WDB 5.7 provides the following commands to display macro definitions:

  — `show macro <macro-name>` or `info macro <macro-name>`

  Displays the macro definition, source file name, and the line number.

  — `macro expand <macro-name>`

  Expands the macro and the parameters in the macro. If there are any parameters in the macro, they are substituted in the macro definition when the definition is displayed.

- **Evaluating Macros**

  HP WDB 5.7 enables you to evaluate a macro and display the output. You can evaluate the macro by using the commonly used `gdb` commands for evaluating and displaying expressions, such as `print`.

  HP WDB supports the evaluation of macros with variables, constants, complex algebraic expressions involving variables, nested macros, and function calls.

  HP WDB does not support the evaluation of macros with multiple statements in the macro definitions, or the evaluation of macros with stringifying and pasting tokens in the macro definitions.

Compiler Options to Enable Macro Debugging

To enable macro debugging, the program must be compiled with the `+macro_debug=[`*`all`*`|`*`none`*`|`*`ref`*`]` compiler option and with one of the `-g` options (`-g`, `-g0`, or `-g1`) to enable macro debugging.

For example:

```
cc -g +macro_debug=all -o sample sample.c
```

The following options are available for the `+macro_debug` compiler option:

- *`all`*

  To view and evaluate all the macro expressions in the program, you must compile the program with `+macro_debug=`*`all`*. This option can cause a significant increase in object file size.

- *`ref`*

  To view and evaluate only the reference macros in the program, you must compile the program with `+macro_debug=`*`ref`*. This is the default for `-g`, `-g1`, or `-g0`.

- *`none`*

  To disable macro debugging, you must compile the program with `+macro_debug=`*`none`*.

The macro debugging features are supported for `+objdebug` and `+noobjdebug` compiler options.

Printing the Execution Path Entries for the Current Frame or Thread

HP WDB 5.7 enables you to print the execution path entries in the current frame, or the current thread for programs running on Integrity systems. This feature is supported only for compiler versions `A.06.15` and later. This feature enables the display of the execution path taken across branched modules. The first instruction in each block associated with the executed branch is displayed.

HP WDB 5.7 supports the following commands to print the execution path entries in the current frame, or in the current thread:

- `info exec-path [start_index] [end_index]` (aliased to `info ep`)

  Lists all the local execution path entries in the current frame.

  The `[start_index]` and `[end_index]` indicate the range of table indexes (execution path entries) that must be displayed.

  If `[end_index]` is not specified, the debugger displays the complete table of execution path entries, starting from `[start_index]`.

  If `[start_index]` and `[end_index]` are not specified, the complete table of execution path entries is displayed.

  For example:

  ```
  (gdb) i ep 4 10
  ```

- `info exec-path summary`

  Prints the summary information about all the local execution path entries in the current frame. This command displays the total number of branches for the frame, the number of branches executed in this frame in the last iteration, and the last executed branch number.

- `info global-exec-path [start_index] [end_index]` (aliased to `info gep`)

  Lists all the global execution path entries for the current thread.

  The `[start_index]` and `[end_index]` indicate the range of table indexes (execution path entries) that must be displayed.

  If `[end_index]` is not specified, the debugger displays the complete table of execution path entries, starting from `[start_index]`.

  If `[start_index]` and `[end_index]` are not specified, the complete table of execution path entries is displayed.

- `info global-exec-path summary`

  Prints the summary information about all the global execution path entries in the current frame. This command displays the total number of global execution path entries that can be stored, the number of global execution path entries in this frame in the last iteration, and the last executed global execution path number.

- `exec-path [up] [down] [path_index]` (aliased to `ep`)

  Enables you to select, print, and navigate through the execution path entries. When no arguments are specified, it prints the selected execution path entry. You can specify the argument as an execution path index from the `info exec-path` or the `info global-exec-path` commands. Alternately, you can use the `up` or `down` command to navigate through the execution path entries.

Compiler Dependencies for Printing the Execution Path Entries

The `+pathtrace` compiler option provides a mechanism to record program execution control flow into global path tables, local path tables, or both. This saved information enables the debugger to print the execution path entries in the current thread or frame. To print the execution path entries in the current thread or frame for programs running on Integrity systems, you must set the required sub-options for the `+pathtrace` compiler option.

You must set the following `+pathtrace` compiler option to enable the debugger to print the execution path entries:

```
+pathtrace= [<global|global_fixed_size>:<local>]
```

For more informtion on the `+pathtrace` compiler option, see the *aCC*(1) manpages.

Automatic Preloading of librtc.[sl|so] with the New +mem_check Option for the chatr Command

The new `+mem_check` option for the `chatr` command enables you to automatically preload the `librtc.[sl|so]` runtime library. In addition to automatically preloading the `librtc.[sl|so]` library, it also maps the shared library as private. This enhancement simplifies the current method of explicitly preloading the appropriate `librtc.[sl|so]` library, by using the `LD_PRELOAD` environment variable.

The `+mem_check <enable|disable>` option is available for dynamic linker versions `B.11.61` and later on HP 9000 systems, and dynamic linker versions `B.12.46` and later on Integrity systems.

To set the target application to preload `librtc.[sl|so]`, enter the following command at the HP-UX prompt:

```
$ chatr +mem_check enable <executable>
```

The `+mem_check` option for the `chatr` simplifies the steps for batch mode and attach mode memory debugging.

Simplified Steps for Batch Mode RTC Using the +mem_check Option for the chatr Command

To use batch mode memory leak detection, complete the following steps:

1. Compile the source files.
2. Create an `rtcconfig` file in the current directory with the required batch mode commands.
3. Define the environment variable `BATCH_RTC` at the HP-UX prompt, as follows:

   ```
   export BATCH_RTC=on
   ```

4. To set the target application to preload `librtc.[sl|so]`, enter the following command at the HP-UX prompt:

   ```
   $ chatr +mem_check enable <executable>
   ```

   At the end of the run, output data file is created in `output_data_dir` (if defined in `rtcconfig`), or the current directory.

Simplified Steps for Debugging Memory in the Attach Mode by Using the +mem_check Option for the chatr Command

To debug memory after attaching GDB to a running process, complete the following steps:

1. To set the target application to preload the `librtc.[sl|so]` runtime library, enter the following command at the HP-UX prompt:

   ```
   $ chatr +mem_check enable <executable>
   ```

2. Run the program.
3. Start the debugging session, as follows:

   ```
   gdb -leaks <executable-name> <process-id>
   ```

NOTE: To attach and find leaks for PA-32 applications from the start-up, the environment variable `RTC_INIT` should be set to `on` in addition to preloading the `librtc.[sl|so]` library before starting the application, as follows:

```
$ RTC_INIT=on <executable>
```

If `RTC_INIT` is turned on, `librtc.[sl|so]` starts recording heap information for PA–32 process, by default. Hence, you must set this environment variable only when memory debugging is required, and you must not export this environment variable for shell.

Support for Aborting a Command Line Call

When a command line call is issued and it is interrupted by a breakpoint or a signal before completing the program execution, the `abort` command enables the user to abort the command line call without allowing the signal to modify the state of the debugged process.

When a signal interrupts program execution, it can modify the process state of the debugged program and result in an abrupt termination of the program (due to adressing errors from a call that is not a part of the source program). In such cases, the `abort` command is particularly useful in exiting the command line call without modifying the process state of the debugged program.

Improved Batch Mode Memory Debugging

HP WDB 5.7 provides enhanced batch mode RTC (Run Time Checking) with the following improvements:

- Improved quality and accuracy of the RTC reports
- Improved reliability in the batch mode RTC to address multiple corner case scenarios

What is New in This Version of HP WDB for HP 9000 systems

The following new features are introduced in this version of HP WDB for HP 9000 systems:

- "Automatic Preloading of librtc.[sl|so] with the New +mem_check Option for the chatr Command" (page 11)
- "Support for Aborting a Command Line Call" (page 12)
- "Improved Batch Mode Memory Debugging" (page 12)

Patches and Defect Fixes in HP WDB 5.7

The following defects are fixed in HP WDB 5.7:

- GDB crashes when viewing the type information of certain templated C++ classes.
- GDB occasionally fails to continue program execution due to incorrect break insertions.
- RTC does not handle heap exhaustion events.
- RTC scans some special heap regions, thus masking leaks.
- GDB does not detect the executable and the core file name mismatch.
- GDB does not detect some frame-less invocations.
- Batch Mode RTC does not work for certain `LIBRTC_SERVER` settings.
- GDB dumps core when printing array elements, while using array slices.
- The `files` command in Batch Mode memory debugging is ignored if the filename of the executable exceeds 80 characters.
- Printing '`if`' in Fortran causes a syntax error in GDB.
- GDB crashes when attempting to print Fortran variables when Fortran assumed arrays are passed as arguments to functions.
- The debugger dumps core on `info type`, or `ptype` command when debugging optimized code.

- Thread safety issues and heap corruption issues occur in an RTC function, `rtc_split_special_region`.
- Obsolete message text is displayed on attach failure with `+Oprofile=collect` executables.
- When GDB attaches to HP Caliper, it alters the behavior of the target.
- GDB erroneously display a warning message for `pre-init mmap` blocks.
- The makefiles and configure scripts must be updated to remove old `librtc.a` and `librtc64.a` builds.
- An additional parentheses misleads GDB into expecting an identifier before the left parenthesis in the symbol table lookup.
- Implemented the `Itanium C++ ABI` (and the `aCC6/EDG` naming convention for the hidden reference parameter) for `struct-valued` return types in functions other than operators.
- Replaced fixed-allocation buffers with dynamic allocation and passed it bottom-up for concatenation because parts of long (> 1K bytes) C++ template names were assembled for printing.
- Implemented location list support for the `DW_AT_frame_base DIE`, which `gcc` versions 4.x use for most functions.
- Function name is not demangled in `File/Function/Line/Pc bar` in Firebolt
- More helpful warning message issued instead of "`Unwind failed for lack of lmdp`"
- Breakpoints in shared libraries multiply on re-rerun.
- GDB does not display cold function names.
- GDB RTC erroneously reports unallocated frees.
- Bad return values are printed for command line calls to functions with `struct` return value.
- GDB crashes when evaluating watchpoints.
- SIGBUS signal messages is improved by providing link for help text.
- The ptype of arrays is erroneously displayed when a subroutine has multiple entries in Fortran.
- The scope for macro debugging must be updated when the user switches across frames.

Notes, Cautions, and Warnings

Following are some notes, cautions, and warnings related to WDB 5.7:

- The Run Time Checking feature (Interactive and Batch Mode) of WDB cannot be used with applications that redefine or override the default system-supplied versions of the standard library routines (under `libc.[sl|so]` and `libdld.[sl|so]`).

  Table 1 lists the dependent library routines for Run Time Checking using HP WDB.

Table 1 Dependent Library Routines for Run Time Checking using WDB

abort	atoi	chdir	clock_gettime
creat	ctime	uwx_register_callbacks	strstr
dlhook	U_STACK_TRACE	uwx_get_reg	write
execl	exit	fclose	fopen
fprintf	fscanf	getcwd	getenv
getpid	lseek	memchr	open
printf	rand	pthread_self	putenv
shl_findsym	shl_get_r	shl_load	shl_unload
sprintf	srand	sscanf	strcasecmp
strdup	strlen	strchr	strtok_r

Table 1 Dependent Library Routines for Run Time Checking using WDB *(continued)*

time	unlink	uwx_self_copyin	strchr
uwx_step	uwx_init	perror	shmctl
write	strcmp	shl_get	close
dlgetname	environ	fork	getpagesize
uwx_self_init_context	pthread_getschedparam	uwx_self_init_info	uwx_register_alloc_cb
strstr	uwx_register_callbacks	uwx_self_lookupip	

The Run Time Checking (dynamic memory, libraries, and pthreads checking) in WDB is dependent on the semantics and the standard behavior of these library routines. Run Time Checking in WDB results in unexpected and unpredictable behavior when used with applications that substitute or redefine these library routines.

Before enabling the Run Time Checking feature in WDB, use the `nm` command to determine if your application or the dependent libraries in your application redefine or substitute these library routines.

- From HP WDB 5.7 onwards, the archive version of the run time check library, `librtc.a`, is not available. You must use the shared version of the library, `librtc.[sl|so]`, instead.
- Batch Mode RTC displays one of the following errors and causes the program to temporarily hang if the version of WDB and `librtc.[sl|so]` do not match, or if WDB is not available on the system:

  ```
  /opt/langtools/bin/gdb: unrecognized option `-brtc'
  Use `/opt/langtools/bin/gdb --help' for a complete list of options.
  ```

  or

  ```
  execl failed. Cannot print RTC info: No such file or directory
  ```

  This error does not occur under normal usage where WDB or `librtc.[sl|so]` is used from the default location at `/opt/langtools/...`

  However, this error occurs if `GDB_SERVER` and/or `LIBRTC_SERVER` are set to a mismatched version of WDB or `librtc.[sl|so]` respectively.

Known Problems and Workarounds

This section describes known problems and the suggested workarounds in this release of HP WDB.

- **Debugging an attached process that is not compiled for debugging may generate warnings**

  On attaching the debugger to a program that is not compiled for debugging, the process may stop in a system call and the following warning message about the various registers is displayed:

  ```
  No data warning: reading 'r3' register: No data warning:
  reading 'r4' register: No data warning: reading 'r5'
  register: No data warning: reading 'r6' register:
  No data warning
  ```

  This warning occurs when executing the `step` command, the `backtrace` command, or when attempting to view the register information. To avoid this warning message, use the `finish` command to execute the process until the system call returns. This warning message is generated when the process stops at a system call, and the registers cannot be read by the debugger. When the debugger calls the routine `ttrace`, it returns this warning.

- **Error attaching WDB to a process that is traced by tools using `ttrace`**

  HP WDB cannot attach to a process that is traced by tools that use `ttrace`, such as Caliper, adb, and tusc.

The debugger displays the following error message while attempting to attach to such a process:

```
Attaching to process <pid> failed.
Hint: Check if this process is already being traced by another
gdb or other ttrace tools like caliper and tusc.
Hint: Check whether program is on an NFS-mounted file-system.
If so,you will need to mount the file system with the "nointr"
option with mount(1) or make a local copy of the program to
resolve this problem.
```

- **GDB reports an incorrect stack trace after `dlclose` or `shl_unload`, and a subsequent `dlopen` or `shl_load`**

  The `librtc.[sl|so]` runtime library reports an incorrect stack trace after a `dlclose` or `shl_unload`, and a subsequent `dlopen` or `shl_load`. The leaks are displayed erroneously when the memory address range overlaps between the newly loaded shared library, and the recently unloaded shared library.

  Workaround: Place a breakpoint at `dlclose` or `shl_unload`, and enter the `info leaks` command to view the leaks accurately when a shared library is unloaded.

- **When displaying information about floating point registers, GDB does not display whether the register holds a `NaT` value or not.**

Installation Requirements and Compatibility Information

This section discusses the installation information for HP WDB.

Table 2 lists the `swinstall` products for HP WDB.

Table 2 HP WDB Installation

Product Name	Description
HP WDB	HP WDB — The HP implementation of the open source debugger GDB
HP WDB GUI	Optional graphical user interface component for HP WDB

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

Compatibility

HP WDB is not supported on releases of the HP-UX operating system prior to 11i v1.

Supported Configurations

The following configurations are supported by HP WDB:

- Installing and running locally on an HP-UX 11i v1, HP-UX 11i v2, or HP-UX 11i v3 operating system.
- Installing on an HP-UX 11i v1, HP-UX 11i v2, or HP-UX 11i v3 system, with the display redirected to a remote HP-UX 11.x node.

Filesets

The HP WDB product contains the following components:

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

NOTE: DEBUG-PRG and SENTINEL are co-requisite filesets for HP WDB and are automatically selected during installation.

The WDB GUI product contains the following sub-products:

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

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

Disk Space Requirements

For information on the disk space requirements to install HP WDB 5.7, see the Downloads website at:

http://www.hp.com/go/wdb/

Installation Instructions

To install HP WDB, 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 specific installation instructions, see the HP WDB Technical Resources website at:

http://www.hp.com/go/wdb/

For more information on installation procedures and related issues, see the *Managing HP-UX Software with SD-UX* and other README, installation, and upgrade documentation provided with the HP-UX 11.x operating system package.

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

ERROR: A later revision (one with a higher revision number) of fileset "WDB-GUI.WDB-GUI-HELP,r=B.11.31" has already been installed. Either remove this fileset or change the "allow_downdate"

To retain the later version of HP WDB, ignore this message. The new products are installed, and the latest version of HP WDB continues to be available.

Debugging Core Files from a Different System

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

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

For more information about these variables, see the *Debugging with GDB* manual available at:

http://www.hp.com/go/wdb

Table 3 lists the supported systems for debugging PA-RISC core files.

Core files produced by Integrity systems can be debugged on any Integrity system with an HP-UX version greater than or equal to the HP-UX version on the system where the core file was produced.

Table 3 Supported Systems for PA-RISC Core File Debugging

Type of core files produced	Supported systems for debugging
Core files produced by 32–bit executables	Any PA-RISC 1.1 or PA-RISC 2.0 system with an HP-UX version greater than or equal to the HP-UX version on the system where the core file was produced.
Core files produced by 64–bit executables	Other PA-RISC 2.0 systems with HP-UX versions greater than or equal to the HP-UX version on the system where the core file was produced.

Features Supported in Previous Versions of HP WDB

For information on previous HP WDB releases, see the HP WDB Technical Resources website at:

http://www.hp.com/go/wdb

Related Documentation

HP WDB documentation is available at the following location:

`/opt/langtools/wdb/doc`

Table 4 lists the documents available for HP WDB.

Table 4 HP WDB Documentation

Document	Format	Location

Debugging with GDB	PDF	`/opt/langtools/wdb/doc/gdb.pdf`
		Emacs:
		`/opt/langtools/wdb/doc/gdb.info`
		(Copy the files to your info directory first.)

GDB Quick Reference Card	PDF	`/opt/langtools/wdb/doc`
Getting Started with HP WDB	HTML	`/opt/langtools/wdb/doc/html/wdb/C/GDBtutorial.html`
XDB to WDB Transition Guide	HTML	`/opt/langtools/wdb/doc/index.html`
Using the HP WDB Terminal User Interface	HTML	`/opt/langtools/wdb/doc/index.html`
GDB manpage	*gdb*(1)	

Software Availability in Native Languages

HP WDB 5.7 is available in the English language only.

WDB Mailing Lists

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

Send an electronic mail message to:

`majordomo@cxx.cup.hp.com`

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

```
subscribe wdb-announce
```

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

```
unsubscribe wdb-announce
```