HP WDB 5.6 Release Notes

HP 9000 and HP Integrity Systems

Manufacturing Part Number: 5991-7500
January 2007

Legal Notices

1 HP WDB 5.6 Release Notes

HP Wildebeest Debugger (WDB) 5.6 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 PA-RISC systems running HP-UX 11.0, HP-UX 11i v1, HP-UX 11i v2, or HP-UX 11i v3. This version of WDB supports programs compiled for Integrity systems (Itanium) and HP 9000 (PA-RISC) systems.

For the latest version of HP WDB, see the technical resources website at:

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

This document addresses the following topics:

What is New in This Release

The following features are introduced in this version of HP WDB

- **Consistent Format for Batch Mode Options and Interactive Mode Commands**

  The new format for batch mode options in the configuration file are consistent with the equivalent interactive mode commands. The old batch mode options are also supported.

  Table 1-1 lists the new format for Batch Mode options in the configuration file.

Table 1-1 New Format for Batch Mode Options

Old Format for Batch Mode Options in the Configuration File (`rtcconfig`)	New Format for Batch Mode Options in the Configuration File (`rtcconfig`)
`check_heap=<on/off> (or) heap=<on/off>`	`set heap-check <on/off>`
`check_leaks=<on/off> (or) leaks=<on/off>`	`set heap-check leaks <on/off>`
`check_string=<on/off>`	`set heap-check string <on/off>`
`check_bounds=<on/off>`	`set heap-check bounds <on/off>`

Table 1-1 New Format for Batch Mode Options

Old Format for Batch Mode Options in the Configuration File (`rtcconfig`)	New Format for Batch Mode Options in the Configuration File (`rtcconfig`)
`check_free=<on/off>`	`set heap-check free <on/off>`
`scramble_block=<on/off>`	`set heap-check scramble <on/off>`
`frame_count=<num>`	`set heap-check frame-count <num>`
`min_leak_size=<num>`	`set heap-check min-leak-size <num>`
`min_heap_size=<num>`	`set heap-check min-heap-size <num>`
`files=<executable-name>`	`files=<executable-name>`
`output_dir=<path>`	`output_dir=<path>`

- **Support for Invoking GDB Before a Program Aborts**

  The `-crashdebug` option enables GDB to monitor the execution of a process or a program. It invokes GDB when the program execution is about to abort. Once the debugger is invoked, you can debug the application using the normal debugger commands. This option enables you to debug a live process instead of getting a core dump if the program is about to abort. You can examine the state of the process, make changes to the state and continue program execution, force a core dump, or terminate execution. It enables you to control program execution under the debugger if the program is about to abort. You can load a new process or attach to a running process for monitoring.

  To monitor a new process, enter the following command:

  ```
  gdb -crashdebug <command> <options>
  ```

  To monitor a running process, attach to the process by entering the following command:

  ```
  gdb -crashdebug -pid <pid>
  ```

- **Support for Setting Breakpoints on Inline Functions in Integrity Systems**

  HP WDB 5.6 enables you to set and modify breakpoints on inline functions for applications compiled with optimization levels less than +O2. Compiler versions A.06.02 and later support the debugging of inline functions in Integrity systems.

  The breakpoint features are introduced as advanced options for the `set inline-debug` command. You can toggle the options by entering either of the following commands:

  ```
  $ gdb --inline= <option>
  ```

  or

  ```
  (gdb) set inline-debug <options>
  ```

  New Command Line Options for Setting Inline Breakpoints

  Inline breakpoint features are available as options for the `set inline-debug` command. The following options are available for the `set inline-debug` command:

  — `on`

  The `set inline debug on` command enables the inline debugging feature without the inline breakpoint option in Integrity systems. This command is enabled by default.

— `off`

The `set inline-debug off` command disables the inline debugging feature. You can disable inline debugging by entering this command before loading the application to the debugger.

— `inline_bp_all`

The `set inline-debug inline_bp_all` command enables you to set and modify breakpoints on all instances of a particular inline function. It also enables the inline debugging feature. A single instance of the specified inline function is displayed as a representative instance for all the instances of the specified inline function. This creates a single-breakpoint illusion for multiple instances of the inline function. You can set and modify breakpoints on all the instances of the inline functions by setting and modifying breakpoints on the displayed instance of the inline function. You must enter this command before loading the application to the debugger.

— `inline_bp_individual`

The `set inline-debug inline_bp_individual` command enables you to set and modify normal breakpoints on a specific instance of an inline function. It also enables the inline debugging feature. All instances of the inline function are displayed separately with individual breakpoint occurrences. You can set or delete individual breakpoints on a specific instance of an inline function without modifying the breakpoints on other instances of the inline function. You must enter this command before loading the application to the debugger.

Limitations:

— This inline breakpoint features are not available for programs that are compiled with +O2 optimization level and above.

— The inline breakpoint features can degrade the performance of the debugged application. You can explicitly disable the breakpoint features when this feature is not required and continue to use other inline debugging features, such as `step` and `next`.

For more information on inline debugging and the inline breakpoint options, see *Debugging with GDB*.

- **Enhanced thread debugging support**

HP WDB 5.6 provides advanced thread-debugging options to detect the following thread-related conditions:

— The thread attempts to acquire a non-recursive mutex that it currently holds.

— The thread attempts to unlock a mutex or a read-write lock that it has not acquired.

— The thread waits (blocked) on a mutex or a read-write lock that is held by another thread with a different scheduling policy.

— Different threads non-concurrently wait on the same condition variable, but with different associated mutexes.

— The thread terminates execution without unlocking the associated mutexes or read-write locks.

— The thread waits on a condition variable with an associated mutex that is not locked.

— The thread terminates execution, and the resources associated with the terminated thread continues to exist in the application because the thread is not joined or detached.

— The thread uses more than the specified percentage of the stack allocated to the thread.

— The number of threads that wait on any `pthread` object exceeds the specified threshold number.

New command line options for thread-debugging

The thread debugging enhancements are introduced as advanced options for the following command:

`set thread-check <on/off> <option> <on/off> <num>`

The following options are available for the `set thread check` command:

— `recursive-relock <on/off>`

This option checks if a thread has attempted to acquire a non-recursive mutex that it currently holds. Re-locking a non-recursive mutex results in a deadlock. At run-time, the debugger keeps track of each mutex in the application and the thread that currently holds each mutex. When a thread attempts to acquire a lock on a non-recursive mutex, the debugger checks if the thread currently holds the lock object for the mutex. If the thread currently holds the lock object for the non-recursive mutex that it is attempting to acquire, the debugger transfers the execution control to the user and prints a warning message.

— `unlock-not-own <on/off>`

This option checks if a thread has attempted to unlock a mutex or a read-write lock that it has not acquired. The debugger transfers the execution control to the user and prints a warning message when this condition is detected.

— `mixed-sched-policy <on/off>`

This option checks if a thread is waiting (blocked) on a mutex or a read-write lock that is held by a thread with a different scheduling policy. The debugger transfers the execution control to the user and prints a warning message when this condition is detected. This is not an application error and does not always result in deadlock. However, it degrades the application performance.

— `cv-multiple-mxs <on/off>`

This option checks if an application uses the same condition variable in multiple calls, made by different threads to `pthread_cond_wait()` or `pthread_cond_timedwait()`, but specifies different mutexes. The debugger transfers the execution control to the user and prints a warning message when this condition is detected.

All threads that concurrently wait on any single condition variable must specify the same associated mutex. For example, the `pthread` implementation does not allow `thread 1` to wait on c`ondition variable A` by specifying `mutex A` , while `thread 2` waits on the same `condition variable A` by specifying `mutex B`. This returns an `EINVAL` error to the application. In contrast to this obvious EINVAL error, the `cv-multiple-mxs` command detects the less-obvious and non- concurrent use of multiple mutexes with the same condition variable by different threads. This option detects potential `EINVAL` errors that exist as a result of different timings or execution paths.

This option is used to detect unintentional use of multiple mutexes with the same condition variable by different threads. In the case of applications that use a dynamic pool of mutexes, the `cv-multiple-mxs` option is not required because this usage is normal and expected application behavior.

— `cv-wait-no-mx <on/off>`

This option checks if the associated mutex of a condition variable is locked when the thread calls the `pthread_cond_wait()` routine. The debugger transfers the execution control to the user and prints a warning message when this condition is detected.

This check is not a POSIX.1 standard requirement for the `pthread_cond_wait()` routine. It is an additional check provided by WDB.

— `thread-exit-own-mutex <on/off>`

This option checks if any thread has terminated execution without unlocking the mutexes or read-write locks that are associated with the thread. The debugger transfers the execution control to the user and prints a warning message when this condition is detected. This situation can result in deadlocks if other threads are waiting to acquire the locked mutexes or read-write locks.

— `thread-exit-no-join-detach <on/off>`

This option checks if a thread has terminated execution (either successfully or because of an exception, or a cancel) without joining or detaching the thread. The debugger transfers the execution control to the user and prints a warning message when this condition is detected. The resources associated with a terminated thread continue to exist in the application if the thread is not joined, or detached. The thread must be explicitly joined or detached, or the thread must be created with the detach attribute. When an application repeatedly creates threads without a join or detach operation, the application leaks resources. This may result in application failure.

— `stack-util <num>`

This option checks if any thread has used more than the specified percentage `<num>` of the stack allocation. The debugger transfers the execution control to the user and prints a warning message when this condition is detected. You can use this option to verify if there is a margin of safety in stack utilization by the threads. The application must ensure that the thread stack size is sufficient for all the operations of the thread. Each thread is assigned a stack allocation when it is created. If the stack allocation is not specified for a thread, the default value is used. The stack allocation cannot be modified after a thread is created. If a thread attempts to use more stack space than the allocated stack space, it results in a stack overflow. Stack overflow can result in memory access violations, bus errors, or segmentation faults.

— `num-waiters <num>`

This option checks if the number of threads waiting on any `pthread` object exceeds the specified threshold number `<num>`. The debugger transfers the execution control to the user and prints a warning message when this condition is detected.

NOTE

— The advanced thread debugging features are supported only on HP-UX 11i v2 and later on both PA-RISC and Integrity systems.

— The tracing `pthread` library is required for advanced thread-debugging. The `pthread tracer` library is available by default on systems running on HP-UX 11i v2 or later. You must install HP WDB 5.5 or later versions of the debugger to support enhanced thread debugging. The installation scripts for HP WDB 5.5 and later versions of the debugger automatically add links at `/opt/langtools/lib/` to replace the standard `libpthread` library with `libpthread tracer` library at run time.

— The thread debugging feature in HP WDB is dependent on the availability of the dynamic Linker Version `B.11.19`.

— HP WDB uses `librtc.sl` to enable thread debugging support. If the debugger is installed in a directory other than the default `/opt/langtools/bin` directory, you must use the environment variable, `LIBRTC_SERVER`, to export the path of the appropriate version of `librtc.sl`.

— HP WDB does not support debugging of programs that link with the archive version of the standard C library `libc.a` or the core library `libcl.a`. The programs must be linked with `libc.sl`.

— The advanced thread debugging commands work only if `set thread-check` is set to `on`.

— For PA-RISC 32-bit applications, the dynamic library path look-up must be enabled for advanced thread debugging.

To enable dynamic library path look-up for advanced thread debugging, enter the following command at HP-UX prompt:

```
# chatr +s enable  <PA32-bitApp>
```

This command automatically enables dynamic library path look-up. No additional environmental variables are required to be set.

- Detection of string corruption with `set heap-check string` extended to `strcat()` and `strncat()`
- The `info heap` command provides additional information on the start of heap, the end of heap, and the total heap size.
- **Enhanced debugging support for `gcc/g++` binaries on HP 9000 systems**

  WDB 5.6 provides improved support for the basic debugging of `gcc/g++` binaries.

  The following limitations apply when debugging `gcc/g++` binaries:

  — Handling of command-line calls

  — Printing type information for certain combinations of structs, unions and bit fields, for 64-bit PA-RISC applications

  — Printing of frame arguments for 32-bit PA-RISC applications

  — Handling of overloaded methods for `g++` binaries

  — HP-specific advanced debugging features (such as runtime memory debugging and thread checking)
- **Enhanced Filtering for Heap Reporting Using the `min-heap-size` Option**

  The `set heap-check min-heap-size <num>` command reports the heap allocations that exceed the specified number `<num>` of bytes based on the cumulative number of bytes that are allocated at each call-site, which is inclusive of multiple calls to `malloc()` at a particular call site.
- **Enhanced heap checking**

  The high water mark records the number of times the break value changes. The following commands are supported:

  — `info heap high-mem`

  This command displays the number of times that the break value changes for a given run.

  — `set heap-check high-mem-count <X_number>`

  This command stops program execution when the break value has moved `X_number` of times

  Limitations

  — High-mem commands are not supported in the batch mode.

  — This feature assumes that an application has a deterministic memory allocation pattern from one run to another.

NOTE The Run Time Checking feature (Interactive and Batch Mode) of WDB cannot be used with applications that re-define or override the default system-supplied versions of the standard library routines (under `libc.so` and `libdld.so`). Table 1-1 lists the dependent library routines for Run Time Checking using WDB.

Table 1-2 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()	uwx_self_lookupip()	strrchr()
sprintf()	srand()	sscanf()	strcasecmp()
strdup()	strlen()	strchr()	strtok_r()
time()	unlink()	uwx_self_copyin()	strcat()
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()

The runtime checking (dynamic memory, libraries, and `pthreads` checking) in WDB relies on the semantic and standard behavior of these library routines. Run Time Checking in WDB results in unexpected and unpredictable behavior when used with applications that substitute or re-define these library routines.

Before enabling the Run Time Checking feature in WDB, use the `nm(1)` command to determine if the application or the dependent libraries in the application re-define or substitute these library routines.

Patches, Defects, and Fixes

This section describes the defects fixed in this version of HP WDB.

- Printing of floating point register values fails for MI.
- The `info frame` command dumps the saved register-state for a signal-handler caller-frame in Integrity systems.
- When `set follow-fork-mode` is set to serial, the debugger may show spurious `SIGTRAP` signals after re-attaching to the parent in Integrity Systems.
- During runtime-checking, `mmap()` calls fail on PA-RISC 32-bit systems for WDB 5.5 .
- HP WDB occasionally crashes when running in a core-file debugging session on HP-9000 systems.
- The start-up in HP WDB 5.5 is slow while debugging JVM.
- HP WDB goes into a tail spin while unwinding from frameless invocations, such as `sigprocmask`.
- The scramble feature must not be enabled by default.
- The warning message must trigger only once when using the `define` command in a module that is compiled with an optimization level greater than 1.
- Handling of how GDB stops at predicated-false instructions in Integrity systems.
- GDB must support short circuiting when evaluating expressions.
- GDB reports incorrect call sites when debugging memory if `frame count` is set to 1.
- Stack growth can fail when using the `info leaks` command.
- The previously set `LD_PRELOAD` value is not saved for the subsequent RTC sessions.
- The `ptype` command prints the menu for a method of a templated class twice.

  GDB must support the addition of a new symbol file in the midst of live debugging in HP 9000 systems.
- GDB 5.5 does not allow modification of registers in HP 9000 systems.
- GDB cannot attach to an executable when the pathname of the executable exceed the length limitation in HP 9000 systems.
- `Scramble.exp` fails by issuing a `SIGSEGV` on HP 9000 systems running on 11i v2.
- The `info heap` and `info leaks` commands fail to generate reports when the current thread is blocked.
- The filename of the process that is being attached to GDB must be less than 60 characters in length.
- GDB must automatically set the `GDB_JAVA_UNWINDLIB` environment variable.
- The `continue` command fails after attaching to a daemon.
- All temporary files that are created during RTC must be deleted before the debugging session is completed.
- Problems in getting the `gdb` prompt when running RTC in batch-mode.
- Batch RTC fails to create leak report and heap report for executables that are located a directory where the pathname is longer than 64 characters.
- The Run Time Checking in WDB must not scan non-anonymous memory-mapped regions.
- Fixed RTC to intercept calls to `memcpy(),memset()`, and `memmove()` in Integrity Systems.

- WDB displays bogus corruption warning for calls to `strncpy()`, `memcpy()` and `strncat()`.
- GDB crashes at `down_load_data()` on `info heap` command.
- GDB errors out with "`Current thread is inside a syscall. Cannot detect leaks now.`" during batch RTC or with `+check=malloc`.
- GDB prints incorrect number of stacked registers for upper frames with the `info reg` command.
- The `+check=malloc` compiler option reports spurious leaks.
- The `info leaks` command crashes after a `shl_unload` of an application.
- The `quit` command fails and takes the user back to the `gdb` prompt when the debugged process has been killed.
- The `execl()` command intermittently fails when spawning GDB from `librtc64.so.1`.
- For application which `fork()`or `exec()`, `LD_PRELOAD_ONCE` must be used for loading the `librtc.sl` library.
- The `info reg [reg-id]` command must display `NaT` instead of the register value if the corresponding `NaT` bit is set.
- GDB must display appropriate error message when attached to a process that is traced by tools that use `ttrace`.
- Printing of variables fails with bad address errors.
- Duplicate code interferes with applications during run time checking.
- File access errors encountered in Batch RTC.
- Error encountered when attempting to disassemble a 32-bit shared library symbol on Integrity Systems.
- GDB errors out with `Internal error: unknown thread <pid>` on info leaks command.
- Application hangs due to problems with `pthread tracer` routine.
- Bus error in `ksh` when running a program.
- The `mmap` regions must not be displayed under `info heap`.
- GDB prints incorrect values for `.rodata` arrays in shared libraries.
- Placing internal breakpoints in a `pthread` library fails on attaching to multi-threaded applications in Integrity systems, where the shared libraries are not mapped private.
- WDB 5.5 installation script creates `libpthread` links in `/opt/langtools/lib` that cause problems when using `gcc`–compiled multi-threaded applications.
- Run Time Checking on multi-threaded programs crashes for MxN threads in Integrity systems.

Known Problems and Workarounds

This section describes known problems and 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 be stopped 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 system routine, `ttrace` returns this warning.

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

  HP WDB cannot be attached to a process that is traced by tools that use `ttrace`, such as Caliper, adb, and tusc. The debugger displays the following error message on 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.
```

Compatibility Information and Installation Prerequisites

This section discusses installation information for HP WDB.

Table 1-2 lists the `swinstall` products for HP WDB.

Table 1-3 HP WDB Products

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 WDB GUI on a system where HP WDB is not installed, HP WDB is installed automatically.

Compatibility

HP WDB is not supported on releases of the HP-UX operating system prior to 11i v0.

For more information on versions of HP WDB that are compatible with Integrity systems, see the technical resources website at:

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

Supported Configurations

HP WDB supports the following configurations:

- Installing and running locally on an HP-UX 11i v1, HP-UX 11i v2, or HP-UX 11i v3 operating system

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

Filesets

The HP WDB product contains the following filesets:

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

NOTE On HP-UX 11.0 or later, `DEBUG-PRG` and `SENTINEL` are pre-requisite filesets for HP WDB and are automatically selected during installation. These filesets are internally invoked by HP WDB.

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

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

Disk Space Requirements

For information on disk space requirements, see the installation guidelines that are available on the HP WDB download webpage at:

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

Installing HP WDB

To install HP WDB, enter the SD-UX `swinstall` command. This command invokes a user interface that assists you through the installation process. It also provides information about disk space requirements, version numbers, product descriptions, and dependencies.

For specific installation instructions, see the technical resources website at:

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

For more information on installation procedures and related issues, see *Managing HP-UX Software with SD-UX,* `README`, *Installation and Upgrade* documents that are described in your HP-UX 11.x operating system package.

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

```
ERROR: A later revision (one with a higher revision number) of fileset
"WDB-GUI.WDB-GUI-HELP,r=B.11.31" has already been installed.

Either remove this fileset or change the "allow_downdate"
```

Ignore this message to retain the later version of HP WDB. The new products are installed, and the latest version of HP WDB is retained.

Debugging Core Files from a Different System

Using WDB, you can debug a core file even if it was originally generated on a different system. Befor debugging a core file that was created on a different system, you must ensure that the corresponding system and user shared libraries are copied along with the executable and the core file to the other system. The location of the shared libraries must be defined by setting `GDB_SHLIB_PATH` or `GDB_SHLIB_ROOT` before debugging the core file.

For more information about these variables, see *Debugging with GDB*.

Table 1-4 Supported Systems for Debugging Core Files

Type of Core File	Supported Systems for Debugging
Core files created 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 generated.
Core files created by 64 bit executables	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 generated.

Features Supported in Previous Versions of HP WDB

For information on previous HP WDB releases, see the technical resources website at:

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

Related Information

HP WDB documentation is available at:

`/opt/langtools/wdb/doc`

Table 1-5 lists the documentation available for HP WDB.

Table 1-5 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 the `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		`/opt/langtools/wdb/doc/index.html`
GDB manpage	`gdb(1)`	

WDB Mailing Lists

To receive an electronic mail message 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 line of the message:

```
subscribe wdb-announce
```

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

```
unsubscribe wdb-announce
```