Troubleshooting Guide for Java SE 6 with HotSpot VM

Chapter 7
Submitting Bug Reports

This chapter provides guidance on how to submit a bug report. It includes suggestions about what to try before submitting a report and what data to collect for the report.

7.1 Checking for Existing Fixes in Update Releases

The current platform is Java SE 6. Regularly scheduled updates to this release contain fixes for a set of critical bugs identified since the initial release of the platform. When an update release becomes available, it becomes the default download at the Java SE Downloads site.

The download site includes release notes that list the bug fixes in the release. Each bug in the list is linked to the bug description in the bug database. The release notes also include the list of fixes in previous update releases. If you encounter an issue, or suspect a bug, then, as an early step in the diagnosis, check the list of fixes that are available in the most recent update release.

Sometimes it is not obvious if an issue is a duplicate of a bug that is already fixed. Therefore, where possible, test with the latest update release to see if the problem persists.

7.2 Preparing to Submit a Bug Report

Before submitting a big report, consider the following recommendations:

- Collect as much relevant data as possible. For example, generate a thread-dump in the case of a deadlock, or locate the core file (where applicable) and `hs_err` file in the case of a crash. In all cases it is important to document the environment and the actions performed just before the problem is encountered.
- Where applicable, try to restore the original state and reproduce the problem using the documented steps. This helps to determine if the problem is reproducible or an intermittent issue.
- If the issue is reproducible, try to narrow down the problem. In some cases, a bug can be demonstrated with a small standalone test case. Bugs that are demonstrated by small test cases will typically be easy to diagnose when compared to test cases that consist of a large complex application.
- Search the bug database to see if this bug or a similar bug has been reported. If the bug has already been reported, the bug report might have further information, such as the following:
    - If the bug has already been fixed, the release in which it was fixed.
    - A workaround for the problem.
    - Comments in the evaluation that explain, in further detail, the circumstances that cause the bug to arise.

The bug database is located at https://bugs.java.com/bugdatabase/index.jsp.

- If you conclude that the bug has not already been reported, submit a new bug.

Before submitting a bug, verify that the environment where the problem arises is a supported configuration. See the Supported System Configurations site.

In addition to the system configurations, check the list of supported locales. See the Supported Locales web page.

In the case of the Solaris Operating System, check the recommended patch cluster for the operating system release to ensure that the recommended patches are installed.

7.3 Collecting Data for a Bug Report

In general it is recommended to collect as much relevant data as possible when you create a bug report or submit a support call. This section suggests the data to collect and, where applicable, it provides recommendations for the commands or general procedure for obtaining the data.

The following data can be collected prior to submitting a bug report:

- Hardware details
- Operating system details
- Java SE version information
- Command-line options
- Environment variables
- Fatal error log (in the case of a crash)
- Core or crash dump (in the case of a crash and possibly a hang)
- Detailed description of the problem, including test case (where possible)
- Logs or trace information (where applicable)
- Results from troubleshooting steps

The following sections present more detail for each type of data.

7.3.1 Hardware Details

Sometimes a bug arises or can be reproduced only on certain hardware configurations. If a fatal error occurs, the error log might contain the hardware details. If an error log is not available, document in the bug report the number and the type of processors in the machine, the clock speed, and, where applicable and if known, some details on the features of that processor. For example, in the case of Intel processors, it might be relevant that hyper-threading is available.

7.3.2 Operating System

On the Solaris Operating System, the `showrev -a` command prints the operating system version and patch information.

On Linux, it is important to know which distribution and version is used. Sometimes the `/etc/*release` file indicates the release information, but as components and packages can be upgraded independently, it is not always a reliable indication of the configuration. Therefore, in addition to the information from the `*release` file, collect the following information:

- The kernel version. This can be obtained using the `uname -a` command.
- The `glibc` version. The `rpm -q glibc` command indicates the patch level of `glibc`.
- The thread library. There are two thread libraries for Linux, namely `LinuxThreads` and `NPTL`. The `LinuxThreads` library is used on 2.4 and older kernels and has “fixed stack” and “floating stack” variants. The Native POSIX Thread Library (`NTPL`) is used on the 2.6 kernel. Some Linux releases (such as RHEL3) include backports of `NPTL`to the 2.4 kernel. Use the command `getconf GNU_LIBPTHREAD_VERSION` to determine which thread library is used. If the `getconf` command returns an error to say that the variable does not exist, then it is likely that you are using an old kernel with the `LinuxThreads` library.

7.3.3 Java SE Version

The Java SE version string can be obtained using the `java -version` command.

Multiple versions of Java SE may be installed on the same machine. Therefore, ensure that you use the appropriate version of the `java` command by verifying that the installation `bin` directory appears in your `PATH` environment variable before other installations.

7.3.4 Command-Line Options

If the bug report does not include a fatal error log, it is important to document the full command line and all its options. This includes any options that specify heap settings (for example, the `-mx` option) or any `-XX` options that specify HotSpot specific options.

One of the features in Java SE is garbage collector ergonomics. On server-class machines the `java` command launches the HotSpot Server VM and a parallel garbage collector. A machine is considered to be a server machine if it has at least two processors and 2GB or more of memory.

The `-XX:+PrintCommandLineFlags` option can be used to verify the command-line options. This option prints all command-line flags to the VM. The command-line options can also be obtained for a running VM or core file using the `jmap` utility.

7.3.5 Environment Variables

Sometimes problems arise due to environment variable settings. When creating the bug report, indicate the values of the following Java environment variables (if set).

- `JAVA_HOME`
- `JRE_HOME`
- `JAVA_TOOL_OPTIONS`
- `_JAVA_OPTIONS`
- `CLASSPATH`
- `JAVA_COMPILER`
- `PATH`
- `USERNAME`

In addition, collect the following operating-system-specific environment variables.

- On Solaris OS and Linux, collect the values of the following environment variables.
    - `LD_LIBRARY_PATH`
    - `LD_PRELOAD`
    - `SHELL`

  - `DISPLAY`
  - `HOSTTYPE`
  - `OSTYPE`
  - `ARCH`
  - `MACHTYPE`

- On the Linux operating system, collect the values of the following environment variables.
  - `LD_ASSUME_KERNEL`
  - `_JAVA_SR_SIGNUM`
- On the Windows operating system, collect the values of the following environment variables.
  - `OS`
  - `PROCESSOR_IDENTIFIER`
  - `_ALT_JAVA_HOME_DIR`

7.3.6 Fatal Error Log

When a fatal error occurs, an error log is created. See Appendix C, Fatal Error Log for detailed information about this file.

The error log contains much information obtained at the time of the fatal error, such as version and environment information, details on the threads that provoked the crash, and so forth.

If the fatal error log is generated, be sure to include it in the bug report or support call.

7.3.7 Core or Crash Dump

Core and crash dumps can be very useful when trying to diagnose a system crash or hung process. The procedure for generating a dump is described in 7.4 Collecting Core Dumps.

7.3.8 Detailed Description of the Problem

When creating a problem description, try to include as much relevant information as possible. Describe the application, the environment, and most importantly the events leading up to the time when the problem was encountered.

- If the problem is reproducible, list the stepsthat are required to demonstrate the problem.
- If the problem can be demonstrated with a small test case, include the test case and the commands to compile and execute the test case.
- If the test case or problem requires third-party code (for example, a commercial or open source library or package), provide details on where and how to obtain the library.

Sometimes the problem can be reproduced only in a complex application environment. In this case, the description, coupled with logs, core file, and other relevant information, might be the sole means to diagnose the issue. In these situations the description should indicate if the submitter is willing to run further diagnosis or run test binaries on the system where the issue arises.

7.3.9 Logs and Traces

In some cases, log or trace output can help to quickly determine the cause of a problem.

For example, in the case of a performance issue the output of the `-verbose:gc` option can help in to diagnosing the problem. (This is the option to enable output from the garbage collector.)

In other cases the output from the `jstat` command can be used to capture statistical information over the time period leading up to the problem.

In the case of a deadlock or a hung VM (for example, due to a loop) the thread stacks can help diagnose the problem. The thread stacks are obtained using Ctrl-\ on Solaris OS and Linux and Ctrl-Break on the Windows operating system.

In general, include all relevant logs, traces and other output in the bug report or support call.

7.3.10 Results from Troubleshooting Steps

Before submitting the bug report, be sure to document any troubleshooting steps that were performed.

For example, if the problem is a crash and the application has native libraries, you might have already run the application with the `-Xcheck:jni` option to reduce the likelihood that the bug is in the native code. Another case could be a crash that occurs with the HotSpot Server VM (`-server` option). If you have also tested with the HotSpot Client VM (`-client` option) and the problem does not occur, this gives an indication that the bug might be specific to the HotSpot Server VM.

In general, include in the bug report all troubleshooting steps and results that have already occurred. This type of information can often reduce the time that is required to diagnose an issue.

7.4 Collecting Core Dumps

This section explains how to generate and collect core dumps (also known as crash dumps). A core dump or a crash dump is a memory snapshot of a running process. A core dump can be automatically created by the operating system when a fatal or unhandled error (for example, signal or system exception) occurs. Alternatively, a core dump can be forced by means of system-provided command-line utilities. Sometimes a core dump is useful when diagnosing a process that appears to be hung; the core dump may reveal information about the cause of the hang.

When collecting a core dump, be sure to gather other information about the environment so that the core file can be analyzed (for example, OS version, patch information, and the fatal error log).

Core dumps do not usually contain all the memory pages of the crashed or hung process. With each of the operating systems discussed here, the text (or code) pages of the process are not included in core dumps. But to be useful, a core dump must consist of pages of heap and stack as a minimum. Collecting non-truncated good core dump files is essential for postmortem analysis of the crash.

7.4.1 Collecting Core Dumps on Solaris OS

With the Solaris Operating System, unhandled signals such as a segmentation violation, illegal instruction, and so forth, result in a core dump. By default, the core dump is created in the current working directory of the process and the name of the core dump file is `core`. The user can configure the location and name of the core dump using the core file administration utility, `coreadm`. This procedure is fully described in the man page for the `coreadm` utility.

The `ulimit` utility is used to get or set the limitations on the system resources available to the current shell and its descendants. Use the `ulimit -c` command to check or set the core file size limit. Make sure that the limit is set to `unlimited`; otherwise the core file could be truncated. Note that `ulimit` is a Bash shell built-in command; on a C shell, use the `limit` command.

Ensure that any scripts that are used to launch the VM or your application do not disable core dump creation.

The `gcore` utility can be used to get a core image of running processes. This utility accepts a process id (pid) of the process for which you want to force core dump.

To get the list of Java processes running on the machine, you can use any of the following commands:

- `ps -ef | grep java`
- `pgrep java`
- `jps` command. The `jps` command-line utility does not perform name matching (that is, looking for “java” in the process command name) and so it can list Java VM embedded processes as well as the Java processes.

7.4.1.1 Using the `ShowMessageBoxOnError` Option on Solaris OS

A Java process can be started with the `-XX:+ShowMessageBoxOnError` command-line option. When a fatal error is encountered, the process prints a message to standard error and waits for a `yes` or `no` response from standard input. Below is an example of output when an unexpected signal occurs.

Copy

```
=======================================================================

Unexpected Error
-----------------------------------------------------------------------
SIGSEGV (0xb) at pc=0xfeba31ac, pid=8677, tid=2
Do you want to debug the problem?
To debug, run 'dbx - 8677'; then switch to thread 2
Enter 'yes' to launch dbx automatically (PATH must include dbx)
Otherwise, press RETURN to abort...
=======================================================================
```

Before answering `yes` or pressing RETURN, use the `gcore` utility to force a core dump. Then you can type `yes` to launch the `dbx` debugger.

7.4.1.2 Suspending a Process using `truss`

In situations where it is not possible to specify the `-XX:+ShowMessageBoxOnError` option, you might be able to use the `truss` utility. This Solaris OS utility is used to trace system calls and signals. You can use this utility to suspend the process when it reaches a specific function or system call.

The following command shows how to use the `truss` utility to suspend a process when the exit system call is executed (in other words, the process is about to exit).

Copy

```
$

        truss -t \!all -s \!all -T exit -p               pid
```

When the process calls `exit`, it will be suspended. At this point, you can attach the debugger to the process or call `gcore` to force a core dump.

7.4.2 Collecting Core Dumps on Linux

On the Linux operating system, unhandled signals such as segmentation violation, illegal instruction, and so forth, result in a core dump. By default, the core dump is created in the current working directory of the process and the name of the core dump file is `core.` *pid*, where *pid* is the process id of the crashed Java process.

The `ulimit` utility is used to get or set the limitations on the system resources available to the current shell and its descendants. Use the `ulimit -c` command to check or set the core file size limit. Make sure that the limit is set to `unlimited`; otherwise the core file could be truncated. Note that `ulimit` is a Bash shell built-in command; on a C shell, use the `limit` command.

Ensure that any scripts that are used to launch the VM or your application do not disable core dump creation.

You can use the `gcore` command in the `gdb` (GNU Debugger) interface to get a core image of a running process. This utility accepts the pid of the process for which you want to force the core dump.

To get the list of Java processes running on the machine, you can use any of the following commands:

- `ps -ef | grep java`
- `pgrep java`
- `jps` command. The `jps` command-line utility does not perform name matching (that is, looking for "java" in the process command name) and so it can list Java VM embedded processes as well as the Java processes.

7.4.2.1 Using the `ShowMessageBoxOnError` Option on Linux

A Java process can be started with the `-XX:+ShowMessageBoxOnError` command-line option. When a fatal error is encountered, the process prints a message to standard error and waits for a `yes` or `no` response from standard input. Below is an example of output when an unexpected signal occurs.

Copy

```
=======================================================================

Unexpected Error
-----------------------------------------------------------------------
SIGSEGV (0xb) at pc=0x06232e5f, pid=11185, tid=8194
Do you want to debug the problem?
To debug, run 'gdb /proc/11185/exe 11185'; then switch to thread 8194
Enter 'yes' to launch gdb automatically (PATH must include gdb)
Otherwise, press RETURN to abort...
=======================================================================
```

Type `yes` to launch the `gdb` (GNU Debugger) interface, as suggested by the error report shown above. In the `gdb` prompt, you can give the `gcore` command. This command creates a core dump of the debugged process with the name `core.` *pid*, where *pid* is the process ID of the crashed process. Make sure that the `gdb gcore` command is supported in your versions of `gdb`. Look for `help gcore` in the `gdb` command prompt.

7.4.3 Reasons for Not Getting a Core File

The following list explains the major reasons that a core file might not be generated. This list pertains to both Solaris OS and Linux, unless specified otherwise.

- The current user does not have permission to write in the current working directory of the process.
- The current user has write permission on the current working directory, but there is already a file named `core` that has read-only permission.
- The current directory does not have enough space or there is no space left.
- The current directory has a subdirectory named `core`.
- The current working directory is remote. It might be mapped by NFS (Network File System), and NFS failed just at the time the core dump was about to be created.
- Solaris OS only: The `coreadm` tool has been used to configure the directory and name of the core file, but any of the above reasons apply for the configured directory or filename.
- The core file size limit is too low. Check your core file limit using the `ulimit -c` command (Bash shell) or the `limit -c` command (C shell). If the output from this command is not `unlimited`, the core dump file size might not be large enough. If this is the case, you will get truncated core dumps or no core dump at all. In addition, ensure that any scripts that are used to launch the VM or your application do not disable core dump creation.
- The process is running a `setuid` program and therefore the operating system will not dump core unless it is configured explicitly.
- Java specific: If the process received `SIGSEGV` or `SIGILL` but no core dump, it is possible that the process handled it. For example, HotSpot VM uses the `SIGSEGV` signal for legitimate purposes, such as throwing `NullPointerException`, deoptimization, and so forth. The signal is unhandled by the Java VM only if the current instruction (PC) falls outside Java VM generated code. These are the only cases in which HotSpot dumps core.
- Java specific: The JNI Invocation API was used to create the VM. The standard Java launcher was not used. The custom Java launcher program handled the signal by just consuming it and produced the log entry silently. This situation has occurred with certain Application Servers and Web Servers. These Java VM embedding programs transparently attempt to restart (fail over) the system after an abnormal termination. In this case, the fact that a core dump is not produced is a feature and not a bug.

7.4.4 Collecting Crash Dumps on Windows

On the Windows operating system there are three types of crash dumps.

- Dr. Watson logfile, which is a text error log file that includes faulting stack trace and a few other details.
- User minidump, which can be considered a “partial” core dump. It is not a complete core dump, because it does not contain all the useful memory pages of the process.
- Dr. Watson full-dump, which is equivalent to a Unix core dump. This dump contains most memory pages of the process (except for code pages).

When an unexpected exception occurs on Windows, the action taken depends on two values in the following registry key.

Copy

```
\\HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\AeDebug
```

The two values are named `Debugger` and `Auto`. The `Auto` value indicates if the debugger specified in the value of the `Debugger` entry starts automatically when an application error occurs.

- A value of 0 for `Auto` means that the system displays a message box notifying the user when an application error occurs.
- A value of 1 for `Auto` means that the debugger starts automatically.

The value of `Debugger` is the debugger command that is to be used to debug program errors.

When a program error occurs, Windows examines the `Auto` value and if the value is 0 it executes the command in the `Debugger` value. If the value for `Debugger` is a valid command, a message box is created with two buttons: OK and Cancel. If the user clicks OK, the program is terminated. If the user clicks Cancel, the specified debugger is started. If the value for the `Auto` entry is set to 1 and the value for the `Debugger` entry specifies the command for a valid debugger, the system automatically starts the debugger and does not generate a message box.

7.4.4.1 Configuring Dr. Watson

The Dr. Watson debugger is used to create crash dump files. By default, the Dr. Watson debugger (`drwtsn32.exe`) is installed into the Windows system folder (`%SystemRoot%\System32`).

To install Dr. Watson as the postmortem debugger, run the following command.

Copy

```
drwtsn32 -i
```

To configure name and location of crash dump files, run `drwtsn32` without any options.

Copy

```
drwtsn32
```

In the Dr. Watson GUI window, make sure that the Create Crash Dump File checkbox is set and that the crash dump file path and log file path are configured in their respective text fields.

Dr. Watson may be configured to create a full dump using the registry. The registry key is as follows.

Copy

```
System Key: [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DrWatson]

Entry Name: CreateCrashDump
Value: (0 = disabled, 1 = enabled)
```

Note that if the application handles the exception, then the registry-configured debugger is not invoked. In that case it might be appropriate to use the `-XX:+ShowMessageBoxOnError` command-line option to force the process to wait for user intervention on fatal error conditions.

7.4.4.2 Forcing a Crash Dump

On the Windows operating system, the `userdump` command-line utility can be used to force a Dr. Watson dump of a running process. The `userdump` utility does not ship with Windows but instead is released as a component of the OEM Support Tools package.

An alternative way to force a crash dump is to use the `windbg` debugger. The main advantage of using `windbg` is that it can attach to process in a non-invasive manner (that is, read-only). Normally Windows terminates a process after a crash dump is obtained but with the non-invasive attach it is possible to obtain a crash dump and let the process continue. To attach the debugger non-invasively requires selecting the Attach to Process option and clicking the Noninvasive checkbox.

When the debugger is attached, a crash dump can be obtained using the following command.

Copy

```
.dump /f crash.dmp
```

The `windbg` debugger is included in the “Debugging Tools for Windows” download.

An additional utility in this download is the `dumpchk.exe` utility, which can verify that a memory dump file has been created correctly.

Both `userdump.exe` and `windbg` require the process id (pid) of the process. The `userdump -p` command lists the process and program for all processes. This is useful if you know that the application is started with the `java.exe` launcher. However, if a custom launcher is used (embedded VM), it might be difficult to recognize the process. In that case you can use the `jps` command line utility as it lists the pids of the Java processes only.

As with Solaris OS and Linux, you can also use the `-XX:+ShowMessageBoxOnError` command-line option on Windows. When a fatal error is encountered, the process shows a message box and waits for a yes or no response from the user.

Before clicking Yes or No, you can use the `userdump.exe` utility to generate the Dr. Watson dump for the Java process. This utility can also be used for the case where the process appears to be hung.