2 Useful System Tools for Java Troubleshooting.................................................................37
  2.1 GlancePlus..................................................................................................................37
  2.2 tusc............................................................................................................................37
  2.3 Prospect......................................................................................................................37

Table of Contents
2.4 HP Caliper..........................................................................................................................................37
2.5 sar......................................................................................................................................................37
2.6 vmstat................................................................................................................................................37
2.7 iostat..................................................................................................................................................37
2.8 swapinfo............................................................................................................................................37
2.9 top.....................................................................................................................................................38
2.10 netstat...............................................................................................................................................38
2.11 Other Tools.......................................................................................................................................38

3 Getting Help from Hewlett-Packard....................................................................................................39
3.1 Problem Report Checklist....................................................................................................................39
3.2 Collecting Problem Data......................................................................................................................40
   3.2.1 Collecting Core File Information..................................................................................................40
      3.2.1.1 Core File Checklist...............................................................................................................40
      3.2.1.1.1 Estimate Core File Size.................................................................................................40
      3.2.1.1.2 Ensure Process Can Write Large Core Files.................................................................40
      3.2.1.1.3 Verify Amount of Disk Space........................................................................................41
      3.2.1.1.4 Check If Directory Supports Large File Systems..........................................................41
      3.2.1.1.5 Ensure Permissions Allow Core Files............................................................................41
      3.2.1.2 Generating a Core File.........................................................................................................42
      3.2.1.3 Verifying a Core File............................................................................................................42
      3.2.2 Collecting Fatal Error Log Information.....................................................................................42
      3.2.3 Collecting Stack Trace Information...........................................................................................43
3.3 Collecting System Information.............................................................................................................43
3.4 Collecting Java Environment Information.............................................................................................44
   3.4.1 Environment Variables................................................................................................................44
   3.4.2 Libraries.....................................................................................................................................44
3.5 Packaging Files...................................................................................................................................45

Glossary....................................................................................................................................................47

Index.........................................................................................................................................................49
List of Figures

1-1   HPjconfig - System Tab.......................................................................................................................20
1-2   HPjconfig - Application Tab................................................................................................................20
1-3   HPjconfig - Patches Tab......................................................................................................................21
1-4   HPjconfig - Tunables Tab....................................................................................................................21
1-5   HPjmeter - Welcome Screen..................................................................................................................23
1-6   HPjmeter - Connecting to Server ........................................................................................................24
1-7   HPjmeter - Connecting to the Java VM................................................................................................25
1-8   HPjmeter - Metrics Tab........................................................................................................................25
1-9   HPjtune Screen..................................................................................................................................26
1-10  jconsole Screen..................................................................................................................................29
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Tools and Options for Crash Analysis</td>
<td>11</td>
</tr>
<tr>
<td>1-2</td>
<td>Tools and Options for Debugging Hung and Deadlocked Processes</td>
<td>12</td>
</tr>
<tr>
<td>1-3</td>
<td>Options for Fatal Error Handling</td>
<td>12</td>
</tr>
<tr>
<td>1-4</td>
<td>Tools and Options for Monitoring Memory Use</td>
<td>12</td>
</tr>
<tr>
<td>1-5</td>
<td>Performance Monitoring Tools</td>
<td>13</td>
</tr>
<tr>
<td>1-6</td>
<td>Miscellaneous Options</td>
<td>13</td>
</tr>
<tr>
<td>1-7</td>
<td>JDK Tools Not Available on HP-UX</td>
<td>13</td>
</tr>
<tr>
<td>1-8</td>
<td>Java Version Information for gdb Java VM Debugging Features</td>
<td>16</td>
</tr>
<tr>
<td>1-9</td>
<td>Java VM Debugging Commands</td>
<td>17</td>
</tr>
<tr>
<td>1-10</td>
<td>Java Subcommands</td>
<td>17</td>
</tr>
<tr>
<td>1-11</td>
<td>Options to the jstat Command</td>
<td>30</td>
</tr>
<tr>
<td>1-12</td>
<td>Garbage Collection Field Information</td>
<td>33</td>
</tr>
<tr>
<td>1-13</td>
<td>Overview of HeapDump Options</td>
<td>34</td>
</tr>
<tr>
<td>3-1</td>
<td>Libjunwind Library Location for PA-RISC Systems</td>
<td>45</td>
</tr>
<tr>
<td>3-2</td>
<td>Libjunwind Library Location for Integrity Systems</td>
<td>45</td>
</tr>
</tbody>
</table>
About This Document

The information in this document will help application developers and support engineers debug their Java applications on HP-UX systems.

1 Intended Audience

This document is intended for application developers and support engineers who are debugging Java applications on HP-UX systems.

2 New and Changed Information in This Edition

This is the first version of this document.

3 Document Organization

This document contains three chapters:

Chapter 1: Diagnostic and Monitoring Tools and Options — This chapter provides information on tools and options useful for Java troubleshooting on HP-UX.
Chapter 2: Useful System Tools for Java Troubleshooting — This chapter provides information about HP-UX system tools to aide in Java troubleshooting.
Chapter 3: Getting Help from Hewlett-Packard — This chapter contains information about collecting necessary data before opening a Java-related support call.

4 Typographic Conventions

This document uses the following typographical conventions:

- %, $, or # A percent sign represents the C shell system prompt. A dollar sign represents the system prompt for the Bourne, Korn, and POSIX shells. A number sign represents the superuser prompt.
- audit(5) A manpage. The manpage name is audit, and it is located in Section 5.
- Command A command name or qualified command phrase.
- Computer output Text displayed by the computer.
- Ctrl+x A key sequence. A sequence such as Ctrl+x indicates that you must hold down the key labeled Ctrl while you press another key or mouse button.
- ENVIRONMENT VARIABLE The name of an environment variable, for example, PATH.
- [ERROR NAME] The name of an error, usually returned in the errno variable.
- Key The name of a keyboard key. Return and Enter both refer to the same key.
- Term The defined use of an important word or phrase.
- User input Commands and other text that you type.
- Variable The name of a placeholder in a command, function, or other syntax display that you replace with an actual value.
- [] The contents are optional in syntax. If the contents are a list separated by |, you must choose one of the items.
- {} The contents are required in syntax. If the contents are a list separated by |, you must choose one of the items.
- ... The preceding element can be repeated an arbitrary number of times.
- Ø Indicates the continuation of a code example.
- | Separates items in a list of choices.
- WARNING A warning calls attention to important information that if not understood or followed will result in personal injury or nonrecoverable system problems.
- CAUTION A caution calls attention to important information that if not understood or followed will result in data loss, data corruption, or damage to hardware or software.
5 Related Information

This document contains information specific to troubleshooting Java problems on HP-UX systems. The *Trouble-Shooting and Diagnostic Guide for Java 2 Platform, Standard Edition 5.0* from Sun Microsystems also contains some information that may be useful.

6 Publishing History

The document printing date and part number indicate the document’s current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The document part number will change when extensive changes are made. Document updates may be issued between editions to correct errors or document product changes. To ensure that you receive the updated or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details. Eventually the latest version of this document will be available online at:

http://www.docs.hp.com

<table>
<thead>
<tr>
<th>Manufacturing Part Number</th>
<th>Supported Operating Systems</th>
<th>Supported Versions</th>
<th>Edition Number</th>
<th>Publication Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>5991-7463</td>
<td>HP-UX 11i</td>
<td>Version 1</td>
<td>Edition 1</td>
<td>October 2006</td>
</tr>
</tbody>
</table>

7 HP Encourages Your Comments

HP encourages your comments concerning this document. We are committed to providing documentation that meets your needs. Send any errors found, suggestions for improvement, or compliments to:

feedback@fc.hp.com

Include the document title, manufacturing part number, and any comment, error found, or suggestion for improvement you have concerning this document.
This chapter describes the tools and options available for postmortem diagnostics, analysis of hung/deadlocked processes, monitoring memory usage, and performance monitoring.

The tools and options are listed in tables by their respective functions in the first section of this chapter. Many of them are listed in multiple tables since they can be used for multiple functions.

The tools and options are described in detail with examples, where applicable, in the remaining sections of this chapter. All the tools and options described in this chapter are either included in the Java 2 Platform Standard Edition Development Kit (JDK 5.0), are included with Hewlett-Packard’s Java product, or are available for download at the Go Java! website:

http://www.hp.com/products1/unix/java

1.1 HP-UX Java Tools and Options Tables

The tools and options are categorized into the following table groupings:

- Crash Analysis Tools
- Hung and Deadlocked Processes
- Fatal Error Handling
- Monitoring Memory Use
- Performance Monitoring Tools
- Miscellaneous Options
- JDK Tools Not Available on HP-UX

1.1.1 Crash Analysis Tools

Several of the options and tools described in this chapter are designed for postmortem diagnostics. These are the options and tools that can be used to obtain additional information if an application crashes. This analysis may either be done at the time of the crash or at a later time using information from the core file. In addition to these tools, many other tools have features useful for crash analysis.

<table>
<thead>
<tr>
<th>Tool or Option</th>
<th>Description and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>wdb/gdb</td>
<td>The HP wdb debugger is an HP-supported implementation of the gdb debugger that has Java support. For simplicity this document will refer to wdb/gdb as gdb from this point forward. gdb can be used to attach to a running process.</td>
</tr>
<tr>
<td>Fatal Error Log (hs.err_pid&lt;pid&gt;.log)</td>
<td>The fatal error log, hs.err_pid&lt;pid&gt;.log, contains information obtained at the time of the crash. It is often one of the first pieces of data to examine when a crash occurs.</td>
</tr>
<tr>
<td>-XX:OnError</td>
<td>This option is used to specify a sequence of user-supplied scripts or commands to be executed when a crash occurs.</td>
</tr>
<tr>
<td>-XX:+ShowMessageBoxOnError</td>
<td>This option is used to suspend the process when a crash occurs. Depending on the user response, it can launch the gdb debugger to attach to the Java VM.</td>
</tr>
<tr>
<td>jdb</td>
<td>jdb is a Java language debugger.</td>
</tr>
</tbody>
</table>

1.1.2 Hung and Deadlocked Processes

The following options and tools can help you debug a hung or deadlocked process.
### Table 1-2 Tools and Options for Debugging Hung and Deadlocked Processes

<table>
<thead>
<tr>
<th>Tool or Option</th>
<th>Description and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>wdb/gdb</td>
<td>The HP wdb debugger is an HP-supported implementation of the gdb debugger that has Java support. For simplicity, this document refers to wdb/gdb as gdb from this point forward. gdb can be used to attach to a running process.</td>
</tr>
<tr>
<td>HPjmeter</td>
<td>HPjmeter may be used to identify and diagnose performance problems in Java applications running on HP-UX. It can also be used to debug thread and heap issues.</td>
</tr>
<tr>
<td>Ctrl-Break Handler</td>
<td>This handler is used to get thread dump information. It also executes a deadlock detection algorithm and reports any deadlocks detected involving synchronized code. Heap dumps are also generated beginning with JDK 1.5.0.05 and SDK 1.4.2.11 when the -XX:+HeapDumpOnCtrlBreak option is specified.</td>
</tr>
<tr>
<td>-XX:+HeapDump and _JAVA_HEAPDUMP Environment Variable, starting with JDK 1.5.0.03 and SDK 1.4.2.10</td>
<td>This option can be used to observe memory allocation in a running Java application by taking snapshots of the heap over time. It can be set by providing the -XX:+HeapDump option or setting the _JAVA_HEAPDUMP environment variable.</td>
</tr>
<tr>
<td>jdb</td>
<td>jdb is a Java language debugger.</td>
</tr>
</tbody>
</table>

### 1.1.3 Fatal Error Handling

The following options are useful for retrieving more information when fatal errors occur.

### Table 1-3 Options for Fatal Error Handling

<table>
<thead>
<tr>
<th>Option</th>
<th>Description and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>-XX:OnError</td>
<td>This option is used to specify a sequence of user-supplied scripts or commands to be executed when a crash occurs.</td>
</tr>
<tr>
<td>-XX:+ShowMessageBoxOnError</td>
<td>This option is used to suspend the process when a crash occurs. After the process is suspended, the user can use gdb to attach to the Java VM.</td>
</tr>
<tr>
<td>-XX:+HeapDumpOnOutOfMemory, starting with SDK 1.4.2.11 and JDK 1.5.0.04</td>
<td>This option enables dumping of the heap when an out of memory error condition occurs in the Java VM.</td>
</tr>
</tbody>
</table>

### 1.1.4 Monitoring Memory Use

The following options and tools are useful for monitoring memory usage of running applications.

### Table 1-4 Tools and Options for Monitoring Memory Use

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPjmeter</td>
<td>This HP tool may be used to identify and diagnose performance problems in Java applications by examining and monitoring the heap and threads.</td>
</tr>
<tr>
<td>HPjtune</td>
<td>HPjtune is HP’s garbage collection (GC) visualization tool for analyzing garbage collection activity in a Java program.</td>
</tr>
<tr>
<td>-XX:+HeapDump and _JAVA_HEAPDUMP Environment Variable, starting with JDK 1.5.0.03 and SDK 1.4.2.10</td>
<td>This option can be used to observe memory allocation in a running Java application by taking snapshots of the heap over time. It can be set by providing the -XX:+HeapDump option or setting the _JAVA_HEAPDUMP environment variable.</td>
</tr>
<tr>
<td>-Xverbosegc (HP only) and -verbose:gc</td>
<td>These options can be used to enable logging of garbage collection information. The HP-only -Xverbosegc option generates additional GC information that is used by HPjtune. It is preferable to use -Xverbosegc instead of -verbose:gc.</td>
</tr>
<tr>
<td>hat</td>
<td>This third-party tool may be used to perform Java heap analysis.</td>
</tr>
<tr>
<td>jconsole (1.5 only)</td>
<td>jconsole enables you to monitor and manage an application launched with a management agent on a local or remote machine.</td>
</tr>
</tbody>
</table>
1.1.5 Performance Monitoring Tools

The following tools are useful for monitoring system and application performance.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description and Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPjmeter</td>
<td>This HP tool may be used to identify and diagnose performance problems in Java applications.</td>
</tr>
<tr>
<td>HPjtune</td>
<td>HPjtune is HP's GC visualization tool for analyzing garbage collection activity in a Java program.</td>
</tr>
<tr>
<td>jstat (1.5 only)</td>
<td>This tool attaches to the Java VM and collects and logs performance statistics.</td>
</tr>
<tr>
<td>jconsole (1.5 only)</td>
<td>jconsole launches a simple console tool enabling you to monitor and manage an application launched with a management agent on a local or remote machine.</td>
</tr>
<tr>
<td>hprof</td>
<td>hprof is a simple profiler agent used for heap and CPU profiling.</td>
</tr>
</tbody>
</table>

1.1.6 Miscellaneous Options

The following options do not fall into any of the previous categories.

<table>
<thead>
<tr>
<th>JAVA_TOOL_OPTIONS Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-verbose:class</td>
<td>Enables logging of class loading and unloading.</td>
</tr>
<tr>
<td>-verbose:jni</td>
<td>Enables logging of JNI (Java Native Interface).</td>
</tr>
<tr>
<td>-Xcheck:jni</td>
<td>Performs additional validation on the arguments passed to JNI functions.</td>
</tr>
</tbody>
</table>

1.1.7 JDK Tools Not Available on HP-UX

Some JDK tools are not available on HP-UX so they are not described in this document. They are provided in JavaSoft JDK as unsupported tools. Equivalent functionality is available via gdb Java support, HPjmeter, and the HeapDump options.

<table>
<thead>
<tr>
<th>Tool</th>
<th>Description and HP-UX Alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>jinfo</td>
<td>This tool prints Java configuration information for a given Java process, core file, or remote debug server.</td>
</tr>
<tr>
<td>jmap</td>
<td>This tool prints shared object memory maps or Java heap memory details of a given process, core file, or remote debug server. Use the HeapDump options or gdb heap dump functionality instead.</td>
</tr>
<tr>
<td>jstack</td>
<td>This tool prints a Java stack trace for a given Java process, core file, or remote debug server. Use gdb stack trace back functionality instead.</td>
</tr>
<tr>
<td>Serviceability Agent (SA)</td>
<td>This tool has not yet been ported to HP-UX.</td>
</tr>
</tbody>
</table>

1.2 Ctrl-Break Handler

A thread dump is printed if the Java process receives a SIGQUIT signal. Therefore, issuing the command `kill -3 <pid>` causes the process with id `<pid>` to print a thread dump to its standard output. The application continues processing after the thread information is printed.

In addition to the thread stacks, the ctrl-break handler also executes a deadlock detection algorithm. If any deadlocks are detected, the ctrl-break handler also prints out additional information on each deadlocked thread. The SIGQUIT signal can also be used to print heap dump information when using the `-XX:+HeapDump` or `-XX:+HeapDumpOnCtrlBreak` options described further on in this chapter.
Following is an example of output generated when SIGQUIT is sent to a running Java process:

Full thread dump  [Thu Oct 12 14:00:56 PDT 2006] (Java HotSpot(TM) Server VM 1.5.0.03 jinteg:02.13.06-21:25 IA64 mixed mode):

"Thread-3" prio=10 tid=0x0a78480 nid=24 lp_id=2669798 runnable [0bfc0000..0bfc0ae0]
at java.lang.Math.log(Native Method)
at spec.jbb.JBButil.negativeExpDistribution(JBButil.java:795)
at spec.jbb.TransactionManager.go(TransactionManager.java:234)
at spec.jbb.JBBmain.run(JBBmain.java:258)
at java.lang.Thread.run(Thread.java:595)

"Thread-2" prio=10 tid=0x09f9b7a0 nid=23 lp_id=2669797 runnable [0c1c0000..0c1c0b60]
at spec.jbb.Order.dateOrderLines(Order.java:341)
  - waiting to lock <444ba618> (a spec.jbb.Order)
at spec.jbb.DeliveryTransaction.process(DeliveryTransaction.java:213)
at spec.jbb.DeliveryHandler.handleDelivery(DeliveryHandler.java:103)
at spec.jbb.DeliveryTransaction.queue(DeliveryTransaction.java:363)
  - locked <154927e8> (a spec.jbb.DeliveryTransaction)
at spec.jbb.TransactionManager.go(TransactionManager.java:431)
at spec.jbb.JBBmain.run(JBBmain.java:258)
at java.lang.Thread.run(Thread.java:595)

"Thread-1" prio=10 tid=0x08fa0a0 nid=22 lp_id=2669796 runnable [0c3c0000..0c3c0de0]
at spec.jbb.infra.Collections.longStaticBTree.get(longStaticBTree.java:1346)
at spec.jbb.Warehouse.retrieveStock(Warehouse.java:307)
at spec.jbb.Orderline.validateAndProcess(Orderline.java:341)
  - locked <48563610> (a spec.jbb.Orderline)
at spec.jbb.Order.processOrderLines(Order.java:289)
  - locked <48563128> (a spec.jbb.Order)
at spec.jbb.NewOrderTransaction.process(NewOrderTransaction.java:282)
at spec.jbb.TransactionManager.go(TransactionManager.java:278)
at spec.jbb.JBBmain.run(JBBmain.java:258)
at java.lang.Thread.run(Thread.java:595)

"Thread-0" prio=2 tid=0x0781240 nid=21 lp_id=2669795 runnable [0c5c0000..0c5c0e60]
at spec.jbb.infra.Util.DisplayScreen.privIntLeadingZeros(DisplayScreen.java:448)
at spec.jbb.infra.Util.DisplayScreen.putDollars(DisplayScreen.java:1214)
at spec.jbb.NewOrderTransaction.secondDisplay(NewOrderTransaction.java:416)
  - locked <154d4628> (a spec.jbb.NewOrderTransaction)
at spec.jbb.TransactionManager.go(TransactionManager.java:279)
at spec.jbb.JBBmain.run(JBBmain.java:258)
at java.lang.Thread.run(Thread.java:595)

"Low Memory Detector" daemon prio=10 tid=0x0778b80 nid=19 lp_id=2669774 runnable [00000000..00000000]

"CompilerThread1" daemon prio=10 tid=0x0772e30 nid=17 lp_id=2669772 waiting on condition [00000000..0a7f728]

"CompilerThread0" daemon prio=10 tid=0x07701f0 nid=16 lp_id=2669771 waiting on condition [00000000..0afff5b8]

"AdapterThread" daemon prio=10 tid=0x076c8d0 nid=15 lp_id=2669770 waiting on condition [00000000..00000000]

"Signal Dispatcher" daemon prio=10 tid=0x076a2e0 nid=14 lp_id=2669769 waiting on condition [00000000..00000000]

"Finalizer" daemon prio=10 tid=0x0530a60 nid=13 lp_id=2669768 in Object.wait() [750c0000..750c0e60]
at java.lang.Object.wait(Native Method)
  - waiting on <11000100> (a java.lang.ref.ReferenceQueue$Lock)
at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:133)
  - locked <11000100> (a java.lang.ref.ReferenceQueue$Lock)
at java.lang.ref.ReferenceQueue.remove(ReferenceQueue.java:149)
at java.lang.ref.Finalizer$FinalizerThread.run(Finalizer.java:197)
1.3 Fatal Error Log (hs_err_pid<pid>.log)

When a fatal error occurs an error log is created in the file hs_err_pid<pid>.log, where <pid> is the process id of the process. The file is created in the working directory of the process, if possible. In the event that the file cannot be created in the working directory (for example, if there is insufficient space, a permission problem, or another issue) then the file is created in the temporary directory,/tmp. The error log contains information obtained at the time of the fatal error. This includes:

- operating exception or signal that provoked the fatal error
- version and configuration information
- details on the thread that provoked the fatal error and its stack trace
- list of running threads and their states
- summary information about the heap
- list of native libraries loaded
- command line arguments
- environment variables
- details about the operating system and CPU

In some cases, only a subset of this information is output to the error log. This happens when a fatal error is so severe that the error handler is unable to recover and report all details.
Java stack unwind enhancements have been added to gdb to enable it to support unwinding across Java frames and provide an effective way to examine stack traces containing mixed language frames (Java and C/C++) of both live Java processes and core files. This has been implemented by adding subcommands for Java VM debugging to gdb.

The following table shows which Java versions on PA-RISC and Integrity systems have the stack unwind and the gdb Java subcommands features. These features are available in gdb version 4.5 and later versions.

**Table 1-8 Java Version Information for gdb Java VM Debugging Features**

<table>
<thead>
<tr>
<th>Platform</th>
<th>Stack Unwind Enhancements</th>
<th>Java Subcommands</th>
<th>GDB Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA-RISC 32-bit</td>
<td>SDK 1.3.1.02+</td>
<td>SDK 1.4.1.05+</td>
<td>4.5+</td>
</tr>
<tr>
<td>PA-RISC 64-bit</td>
<td>SDK 1.4.1.01+</td>
<td>SDK 1.4.1.05+</td>
<td>4.5+</td>
</tr>
<tr>
<td>Integrity 32-bit</td>
<td>SDK 1.3.1.06+</td>
<td>SDK 1.4.1.05+</td>
<td>4.5-5.2</td>
</tr>
<tr>
<td>Integrity 64-bit</td>
<td>SDK 1.4.0.01+</td>
<td>SDK 1.4.1.05+</td>
<td>4.5-5.2</td>
</tr>
<tr>
<td>Integrity 32, 64-bit</td>
<td>SDK 1.4.2.10+</td>
<td>SDK 1.4.2.10+</td>
<td>*5.3+</td>
</tr>
<tr>
<td>Integrity 32, 64-bit</td>
<td>JSDK 1.5.0.03+</td>
<td>JSDK 1.5.0.03+</td>
<td>*5.3+</td>
</tr>
</tbody>
</table>

*gdb version 5.3 requires SDK 1.4.2.10 and later versions or JDK 1.5.0.03 and later versions in order to use the Java VM debugging features.

In order to use this functionality, the `GDB_JAVA_UNWINDLIB` environment variable must be set to the path name of the Java unwind library. The default location of the Java unwind library on various systems is shown following. The examples are for SDK 1.4; if you are using JDK 5.0, substitute `/opt/java1.5` for `/opt/java1.4`.

```
/export/java1.4/jre/lib/PA_RISC/server/libjunwind.sl
/export/java1.4/jre/lib/PA_RISC2.0/server/libjunwind.sl
/export/java1.4/jre/lib/PA_RISC2.0W/server/libjunwind.sl
/export/java1.4/jre/lib/IA64N/server/libjunwind.so
/export/java1.4/jre/lib/IA64W/server/libjunwind.so
```

Following are a few examples. If you are using ksh on a PA-RISC machine, this is how you set the environment variable for a 32–bit Java application:

```
export GDB_JAVA_UNWINDLIB=/opt/java1.4/jre/lib/PA_RISC/server/libjunwind.sl
```

Additionally, this is how you set the environment variable on an Integrity machine for a 32–bit Java application:

```
export GDB_JAVA_UNWINDLIB=/opt/java1.4/jre/lib/IA64N/server/libjunwind.so
```

If the SDK is installed in a location other than the default, substitute the non-default location for `/opt/java1.4` in the previous commands.

### 1.4.1 Java Stack Unwind Features

The Java stack unwind features are useful for troubleshooting problems in the Java VM. Following is a list of the Java stack unwind features:

- View mixed language frames information, including Java frames and C/C++ native frames, in a `gdb` backtrace.
- Distinguish various Java frame types including interpreted, compiled, and adapter frames.
- View Java method name, signature, and class package name for Java method frames.

Additional stack unwind features are available starting with SDK 1.4.2. These features fall into three categories: Java stack unwind enhancements, Java heap support, and Java threads support.
These additional features are available as part of the Java stack unwind enhancements:

- View Java compiled frame inlined methods.
- View Java interpreted or compiled frame specific information.
- View Java interpreted or compiled frame arguments and local variables.
- Disassemble Java method bytecodes.
- Print out the Java unwind table.

These additional features are available as part of the Java heap support:

- View Java heap parameters.
- Dump Java object.
- Print Java heap histogram.
- Find all the instances of a given Java class.
- Find all the references to a given object in the Java heap.
- Find out the object OOP (object-oriented pointer) of the given field address.

These additional features are available as part of Java threads support:

- View Java threads state information.
- View current Java thread information.
- View Java interpreted frame monitors information.

1.4.2 gdb Subcommands for Java VM Debugging

To view the gdb commands that support Java VM debugging, type help java at the gdb prompt.

(gdb) help java
Java and JVM debugging commands.

List of java subcommands:

java args -- Show the current or specified Java frame arguments info
java bytecodes -- Disassemble the given Java method's bytecodes
java heap-histogram -- Show the Java heap object histogram
java instances -- Find all the instances of the given klassOop in the Java heap
java jvm-state -- Show Java virtual machine's current internal states
java locals -- Show the current or specified Java frame locals info
java mutex-info -- Print out details of the static mutexes
java object -- Print out the given Java object's fields info
java oop -- Find the Java object oop of the given Java heap address
java references -- Find all the references to the given Java object in the Java heap
java unwind-info -- Show the unwind info of the code where the given pc is located
java unwind-table -- Print out the dynamically generated Java Unwind Table

Type "help java" followed by java subcommand name for full documentation. Command name abbreviations are allowed if unambiguous.

The following two tables list Java VM debugging commands and Java subcommands.

<table>
<thead>
<tr>
<th>Table 1-9 Java VM Debugging Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>backtrace</td>
</tr>
<tr>
<td>info frame</td>
</tr>
<tr>
<td>info threads</td>
</tr>
<tr>
<td>thread</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 1-10 Java Subcommands</th>
</tr>
</thead>
<tbody>
<tr>
<td>java args</td>
</tr>
<tr>
<td>java bytecodes</td>
</tr>
</tbody>
</table>
Table 1-10 Java Subcommands (continued)

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>java heap-histogram</td>
<td>show the Java heap object histogram</td>
</tr>
<tr>
<td>java instances</td>
<td>find all the instances of the given klassOop in the Java heap</td>
</tr>
<tr>
<td>java jvm-state</td>
<td>show the current internal state of the Java VM</td>
</tr>
<tr>
<td>java locals</td>
<td>show the current or specified Java frame locals information</td>
</tr>
<tr>
<td>java object</td>
<td>print out the given Java object's fields information</td>
</tr>
<tr>
<td>java oop</td>
<td>find the Java object OOP of the given Java heap address</td>
</tr>
<tr>
<td>java references</td>
<td>find all the references to the given Java object in the Java heap</td>
</tr>
<tr>
<td>java unwind-info</td>
<td>show the unwind information of the code where the given pc is located</td>
</tr>
<tr>
<td>java unwind-table</td>
<td>print out the dynamically generated Java unwind table</td>
</tr>
</tbody>
</table>

Type `help java` followed by the subcommand name for full documentation. Command name abbreviations are allowed if they are unambiguous.

Following are examples that illustrate the gdb command-line options for invoking gdb on a core file and on a hung process.

The first set of examples illustrate how to invoke gdb on a core file.

- Invoke gdb on a core file generated when running a 32-bit Java application on an Integrity system with /opt/java1.4/bin/java:
  ```
  $ gdb /opt/java1.4/bin/IA64N/java core.java
  ```

- Invoke gdb on a core file generated when running a 64-bit Java application on an Integrity system with /opt/java1.4/bin/java -d64:
  ```
  $ gdb /opt/java1.4/bin/IA64W/java core.java
  ```

- Invoke gdb on a core file generated when running a 32-bit Java application on PA-RISC using /opt/java1.4/bin/java:
  ```
  $ gdb /opt/java1.4/bin/PA_RISC2.0/java core.java
  ```

- Invoke gdb on a core file generated when running a 64-bit Java application on PA-RISC using /opt/java1.4/bin/java:
  ```
  $ gdb /opt/java1.4/bin/PA_RISC2.0W/java core.java
  ```

When debugging a core file, it is good practice to rename the file from `core` to another name to avoid accidentally overwriting it.

If the Java and system libraries used by the failed application reside in non-standard locations, then the `GDB_SHLIB_PATH` environment variable must be set to specify the location of the libraries.

The following example illustrate how to invoke gdb on a hung process.

- Determine the process id:
  ```
  $ ps -u user1 | grep java
  23989 pts/9  8:52 java
  ```

- Attach gdb to the running process:
  ```
  $ gdb -p 23989
  ```

HP gdb 5.0 for HP Itanium (32 or 64 bit) and target HP-UX 11.2x.
Copyright 1986 - 2001 Free Software Foundation, Inc.
Hewlett-Packard Wildebeest 5.0 (based on GDB) is covered by the GNU General Public License. Type "show copying" to see the conditions to
change it and/or distribute copies. Type "show warranty" for warranty/support.

Reading symbols from /opt/java1.4/bin/IA64N/java...
(no debugging symbols found)...done.
Attaching to program: /opt/java1.4/bin/IA64N/java, process 23989
(no debugging symbols found)...
Reading symbols from /usr/lib/hpux32/libpthread.so.1...
(no debugging symbols found)...done.
Reading symbols from /usr/lib/hpux32/libdl.so.1...
...

A tutorial on gdb may be found at the following website:
http://h21007.www2.hp.com/dspp/tech/tech_TechDocumentDetailPage_IDX/1,1701,1677,00.html

1.5 HPjconfig

HPjconfig is a configuration tool for tuning your HP-UX 11i system to match the characteristics of your application. It provides kernel parameter recommendations tailored to your HP-UX hardware platform and application characteristics. HPjconfig has features for saving and restoring configurations so you can distribute customized recommendations across your customer base.

HPjconfig can also be used to verify that your systems has all the necessary patches required for Java. The patches required for Java can be found at the following website:
http://www.hp.com/products1/unix/java/patches

HPjconfig runs on SDK 1.3.1 and later versions, SDK 1.4.x, and JDK 1.5.0.x. HP-UX 11.00 or later versions is required. All HP-UX 11i HP Integrity and HP 9000 PA-RISC systems are supported.

For more information about HPjconfig including the download, go to:

HPjconfig can be run in either graphical user interface (GUI) mode or non-GUI (command-line) mode. In either mode, it generates a summary of the configuration information in the log file named HPjconfig_<hostname>_<date>_<timestamp>.log. This log file name can be specified using the -logfile option.

Following is usage information for the HPjconfig command:

usage:
   HPjconfig [ options ] -gui
   HPjconfig [ options ] <object> <action>

objects: -patches &| -tunables
actions: -listreq | -listmis | -listpres | -apply

options:
   -patches operate on java-specific patches
   -tunables operate on java-specific tunables
   -listreq list all java required patches or tunables that are applicable to this system
   -listmis list missing java-specific patches or tunables on the system
   -listpres list applied (installed) java-specific patches or tunables on the system
   -apply apply (install) missing java-specific patches or tunables on the system
   -javavers s java versions for selecting patches e.g 1.2, 1.3, 1.4, 5.0
   -[no]gui run in GUI mode
   -logfile s name of log file
   -proxyhost s HTTP proxy host name for accessing live data
   -proxyport s HTTP proxy port for accessing live data
   -help show help string and exit
   -version show version string
Following are examples of invoking HPjconfig in GUI mode from the csh and the ksh:

(csh) $ setenv DISPLAY <Display's IP Address>:0.0  
    $ setenv PATH $PATH:/usr/sbin  
    $ java -jar HPjconfig.jar

(ksh) $ export DISPLAY=<Display's IP Address>:0.0  
    $ export PATH=$PATH:/usr/sbin  
    $ java -jar HPjconfig.jar

The following four figures show the System, Application, Patches, and Tunables tabs for the HPjconfig tool.

**Figure 1-1** HPjconfig - System Tab

![HPjconfig - System Tab]

**Figure 1-2** HPjconfig - Application Tab

![HPjconfig - Application Tab]
Following are the commands for invoking HPjconfig in non-GUI mode. The -help option lists options you can use in this mode.

$ cd <hpjconfig_installation_dir>
$ java -jar ./HPjconfig.jar -nogui -help

Following is an example using HPjconfig in non-GUI mode to list missing patches for Java SDK 1.4:

$ java -jar HPjconfig.jar -nogui -patches -listmis -javavers 1.4
Log written to HPjconfig_mutant_20060915_040458.log
List of missing patches:
PHSS_34201 solves problem emulating floating point conversion when running PA2.0 Java on an IPF system. solves problem with Aries signal handling that overlaps Java signal handling. solves problem emulating floating point conversion when running PA2.0 Java on an IPF system. solves problem with Aries signal handling that overlaps Java signal handling.

Following is an example using HPjconfig to show the values for HP-UX tunables required by Java:

$ java -jar HPjconfig.jar -nogui -tunables -listreq
Log written to HPjconfig_mutant_20060915_040934.log
List of required tunables:
<table>
<thead>
<tr>
<th>Name</th>
<th>Recommended value</th>
</tr>
</thead>
<tbody>
<tr>
<td>nproc</td>
<td>2048+20</td>
</tr>
<tr>
<td>max_thread_proc</td>
<td>3000</td>
</tr>
</tbody>
</table>

Figure 1-3 HPjconfig - Patches Tab

Figure 1-4 HPjconfig - Tunables Tab
nkthread                        6000
nfile                           30000
maxfiles                        2*1024
maxfiles_lim                    2*1024
maxdsiz                         2000*1024*1024

Following is an example of using HPjconfig to display tunables that are set to values less than those recommended:

$ java -jar HPjconfig.jar -nogui -tunables -listmis
Log written to HPjconfig_mutant_20060915_040955.log
List of tunables whose values are less than the recommended values:
Name                            Recommended value
max_thread_proc                 3000
maxdsiz                         2000*1024*1024

Following is an example log file produced by HPjconfig:

$ more HPjconfig_server1_20060915_042600.log
Fri Sep 15 16:26:00 PDT 2006
HPjconfig 3.0.01 (Thu Jul 21 14:52:47 2005)

Machine name:  server1
IP address:    15.244.94.25
System type:   ia64 hp server rx5670
Architecture:  IA64N
OS name:       HP-UX
OS version:    B.11.23
Processors:    4
Java version:  1.4

Reading required patches/tunables information from /tmp/HPjconfig.xml
Read required patches/tunables information
Reading patch list from system
Read patch list from system
List of required patches:
PHCO_30476 supports HPjmeter profiling of unbound (MxN) threads.
PHKL_30192 solves kernel panic with thousands of MxN threads.
PHSS_30015 solves problem with Aries signal handling that overlaps Java signal handling.
PHSS_34201 solves problem emulating floating point conversion when running PA2.0 Java on an IPF system.
   solves problem with Aries signal handling that overlaps Java signal handling.
   solves problem emulating floating point conversion when running PA2.0 Java on an IPF system.

1.6 HPjmeter

HPjmeter is a useful tool for identifying and diagnosing performance problems in Java applications. It can be used to analyze output from -Xrunhprof:heap=dump, -Xeprof, and -XX:+HeapDump. Additionally, HPjmeter can be used to monitor live Java applications and analyze profile data.

HPjmeter features include:
- Automatic problem detection and alerts
- Dynamic real-time display of application behavior
• Drill down into application profile metrics
• Detection of lock contention
• Ability to attach to a running process (version 2 only)

The user’s guide for this tool may be found at:

More information and the HPjmeter download may be found at:

Following are two examples of how to invoke HPjmeter 1.6.x. The first example is for csh and the second example is for ksh.

```bash
$ setenv DISPLAY <your display>:0.0
$ setenv HPJMETERLOC <directory in which HPjmeter.jar is installed>
$ java -mx128m -jar $HPJMETERLOC/HPjmeter.jar

$ export DISPLAY=<your display>:0.0
$ export HPJMETERLOC=<directory in which HPjmeter.jar is installed>
$ java -mx128 -jar $HPJMETERLOC/HPjmeter.jar
```

The following screen snapshot shows the HPjmeter welcome screen.

![Figure 1-5 HPjmeter - Welcome Screen](image)

HPjmeter 2.0 includes an agent for dynamic monitoring of a Java application, in addition to the eprof data visualization console provided in previous versions. The following steps show how to start the monitoring agent when launching the HPjmeter console.

1. Set the SHLIB_PATH environment variable to include the location of the HPjmeter agent library as appropriate for 32 or 64-bit Java VM. The following examples show how to set this variable in both the csh and the ksh for the different libraries.

   To select the PA-RISC 32-bit library:
   ```bash
   (csh) setenv SHLIB_PATH /opt/hpjmeter/lib/PA_RISC2.0
   (ksh) export SHLIB_PATH /opt/hpjmeter/lib/PA_RISC2.0
   
   To select the PA-RISC 64-bit library:
   (csh) setenv SHLIB_PATH /opt/hpjmeter/lib/PA_RISC2.0W
   (ksh) export SHLIB_PATH=/opt/hpjmeter/lib/PA_RISC2.0W
   ```
To select the Integrity 32-bit library:
(csh) setenv SHLIB_PATH /opt/hpjmeter/lib/IA64N
(ksh) export SHLIB_PATH=/opt/hpjmeter/lib/IA64N

To select the Integrity 64-bit library:
(csh) setenv SHLIB_PATH /opt/hpjmeter/lib/IA64W
(ksh) export SHLIB_PATH=/opt/hpjmeter/lib/IA64W

2. Set AGENT to the path of the HPjmeter agent:
(csh) setenv AGENT /opt/hpjmeter/lib/agent.jar
(ksh) export AGENT=/opt/hpjmeter/lib/agent.jar

3. Start the Java monitoring agent. For example, to start the application myapp on Java 1.5, enter:
```
/opt/java1.5/bin/java -Xms256m -Xmx512m -agentlib:jmeter myapp
```

For previous Java versions, enter:
```
/opt/java1.4/bin/java -Xms256m -Xmx512m \
-Xbootclasspath/a:$AGENT -Xrunjmeter myapp
```

This enables the myapp process to be dynamically monitored with the HPjmeter console.

4. Start the HPjmeter console by entering:
```
/opt/hpjmeter/bin/hpjmeter
```

Following are some output screens from HPjmeter:
1.7 HPjtune

HPjtune is a garbage collection visualization tool for analyzing garbage collection activity in a Java program. Data files for HPjtune can be generated using -Xverbosegc or -verbose:gc. HPjtune lets you view this data in the following ways:

- Predefined graphs, which show the utilization of garbage collector resources and the impact of the garbage collector on application performance.
- User-configurable graphs, which access selected GC metrics.
- Other predefined graphs, which show GC behavior pertaining to threads.

HPjtune also includes a unique feature which allows you to use the data collected with the -Xverbosegc option to predict the effect of new garbage collector parameters on future application runs.

For more information about HPjtune and to download the tool, go to:


Following is an example of running Java with the -Xverbosegc option to generate a data file to be used by HPjtune.

$ /opt/java1.5/bin/java -Xverbosegc:file=java2d_gc.out -jar Java2Demo.jar

The -Xverbosegc option causes a data file containing garbage collection data to be generated into file java2d_gc.out.<pid>. This is how to invoke HPjtune on that file:
where `<HPjtune_insdir>` is the location of the HPjtune installation.

Following is an example screen shot to illustrate HPjtune’s output:

**Figure 1-9 HPjtune Screen**

![HPjtune Screen](image)

1.8 *hat*

The *hat* tool is a third-party tool that can be used for heap analysis. It starts a web server on a binary-format heap dump file produced by one of the heap dump options such as `-XX:+HeapDumpOnCtrlBreak` or `-Xrunhprof:heap=dump,format=b`.

Following in an example. The first command generates a binary heap dump file. The second command invokes *hat* on the binary heap profile.

```
$ java -Xrunhprof:heap=dump,format=b MyApp
$ hat -port=7002 java.hprof
```

The *hat* tool sets up an http server on the specified port. It can then be accessed by bringing up the default page in a web browser, for example, `http://<hostname.domain>:7002`. If you run *hat* on the same system as the browser, the server can be accessed by navigating to the URL `http://<hostname.domain>:7002`.

For more information on *hat*, refer to the following website:

https://hat.dev.java.net

For invocation details, refer to:

https://hat.dev.java.net/doc/README.html

**NOTE:** Beginning with Java SE 6, *hat* will be replaced with *jhat*, which will be included with the standard 6.0 distribution. For information on *jhat*, refer to the following website:

http://java.sun.com/javase/6/docs/technotes/tools/share/jhat.html
1.9 hprof

hprof is a simple profiler used for heap and CPU profiling. For more information, refer to:
http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi/jvmpi.html#hprof

1.10 java.security.debug System Property

The java.security.debug system property controls whether the security checks in the JRE (Java Runtime Environment) print trace messages during execution. This option can be useful when trying to determine why a SecurityException is thrown by a security manager. This system property can be set to one of the following values:

- access — print all checkPermission results
- jar — print jar verification information
- policy — print policy information
- scl — print permissions assigned by the SecureClassLoader

The access option has the following sub-options:

- stack — include stack trace
- domain — dump all domains in context
- failure — dump the stack and domain that did not have permission before throwing the exception

For example, to print all checkPermission results and trace all domains in context, set java.security.debug to access,stack. To trace access failures, set it to access,failure.

Following is an example showing the output of a checkPermission failure:

$ java -Djava.security.debug="access,failure" Application
access denied (java.net.SocketPermission server.foobar.com resolve)
java.lang.Exception: Stack trace
at java.lang.Thread.dumpStack(Thread.java:1158)
at java.security.AccessControlContext.checkPermission(AccessControlContext.java:253)
at java.security.AccessController.checkPermission(AccessController.java:427)
at java.lang.SecurityManager.checkPermission(SecurityManager.java:532)
at java.lang.SecurityManager.checkConnect(SecurityManager.java:1031)
at java.net.InetAddress.getAllByName0(InetAddress.java:1117)
at java.net.InetAddress.getAllByName0(InetAddress.java:1098)
at java.net.InetAddress.getAllByName(InetAddress.java:1061)
at java.net.InetAddress.getByName(InetAddress.java:958)
at java.net.InetSocketAddress.<init>(InetSocketAddress.java:124)
at java.net.Socket.<init>(Socket.java:178)
at Test.main(Test.java:7)

1.11 JAVA_TOOL_OPTIONS Environment Variable

The command line used to start an application is not always readily accessible in many environments. This is especially true with applications that use embedded Java VMs or ones where the startup is deeply nested in scripts. In these environments, the JAVA_TOOL_OPTIONS environment variable may be useful to add options to the command line when the application is run. This environment variable is primarily intended to support the initialization of tools, specifically the launching of native or Java agents using the -agentlib or -javaagent options.

The JAVA_TOOL_OPTIONS environment variable is processed at the time of the invocation of the Java VM. When this environment variable is set, the JNI_CreateJavaVM() function prepends the value of the environment variable to the options supplied in its JavaVMInitArgs argument. For security reasons this option is disabled in setuid processes; that is, processes where the effective user or group ID differs from the real user or group ID.

In the following example, the environment variable is set to launch the hprof profiler when the application is started:
export JAVA_TOOL_OPTIONS="-agentlib:hprof"

Although this environment variable is intended to support the initialization of tools, it is also useful for augmenting the command line with options for diagnostics purposes. For example, you could use it to add the -XX:OnError option to the command line when it would be helpful for a script or command to be executed when a fatal error occurred.

Since this environment variable is processed when JNI_CreateJavaVM() is called, it cannot be used to augment the Java launcher options. Some examples of these launcher options are the following VM selection options:

- java -d64
- java -client
- java -server

To pass arguments to the Java launcher, set the JAVA_LAUNCHER_OPTIONS environment variable to a string containing the desired arguments.

This environment variable is fully described in the JVMTI specification at:
http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/jvmti.html#tooloptions

1.12 jconsole (1.5 only)

The jconsole command launches a graphical console tool that enables you to monitor and manage Java applications on a local or remote machine.

jconsole can attach to any application that is started with the Java Management Extensions (JMX) agent. A system property defined on the command line enables the JMX agent. Once attached, jconsole can be used to display useful information such as thread usage, memory consumption, and details about class loading, runtime compilation, and the operating system.

In addition to monitoring, jconsole can be used to dynamically change several parameters in the running system. For example, the setting of the -verbose:gc option can be changed so that garbage collection trace output can be dynamically enabled or disabled for a running application.

To use jconsole:

1. Start the application with the -Dcom.sun.management.jmxremote option. This option sets the com.sun.management.jmxremote system property, which enables the JMX agent.
2. Start jconsole with the jconsole command.
3. When jconsole starts, it shows a window listing the managed Java VMs on the machine. The process id (pid) and command line arguments for each Java VM are displayed. Select one of the Java VMs, and jconsole attaches to it.

Following is an example invocation of jconsole. First the Java application must be started with the JMX agent enabled:

```
$ java -Dcom.sun.management.jmxremote -jar Java2Demo.jar &
[1] 13028
```

Now the jconsole tool can be started on the managed Java VM:

```
$ /opt/java1.5/bin/jconsole 13028
```

The following figure shows a screen shot.
To learn more about jconsole, refer to the following website:
http://java.sun.com/j2se/1.5.0/docs/guide/management/jconsole.html

1.13 jdb

The SDK includes a command-line debugger, jdb, to help you find and fix bugs in Java programs running on a local or remote Java machine. Refer to the following website for more information:
http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/jdb.html
A jdb tutorial may be found at:

1.14 jps (1.5 only)

The jps tool lists the Java VMs on the target system. The tool is limited to reporting information on Java VMs that the user has access rights to, as determined by HP-UX specific access control mechanisms. For example, if a non-root user executes the jps command, a listing of all virtual machines started with that user's uid is given by the operating system.

Following is the usage information for the jps command:

Usage:  jps [-help]
        jps [-q] [-mlvV] [<hostname>[:<port>]]

Description of options:
-q  Suppress the output of the class name, JAR file name, and arguments passed to the main method, producing only a list of local JVM pids
-m  Show the arguments passed to the main method. This output may be null for embedded JVMs.
-l  Show the full package name for the application's main class or the full path name of the application's JAR file.
-v  Show the arguments passed to the JVM.
-V  Show the arguments passed to the JVM through the flags file (the .hotspotrc file or the file specified by -XX:Flags=<filename>).

Note: These options are subject to change or removal in the future.
Following is an example using jps:

```
$ /opt/java1.5/bin/jps -lmv
16666 sun.tools.jps.Jps -lmv
  -Denv.class.path=/opt/java1.5/lib/classes.zip -Dapplication.home=/opt/java1.5 -Xms8m
16665 MyObjectWaiterApp -Xverbosegc
16641 spec.jbb.JBBmain -propfile S.pr.8 -Xmx1600m -Xms1600m -Xmn1500m
```

For more information about jps, refer to the following document:

http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jps.html

### 1.15 jstat (1.5 only)

The jstat utility is a statistics monitoring tool. It attaches to a Java VM and collects and logs performance statistics as specified by the command line options. The target Java VM is identified by its virtual machine identifier.

The jstat utility does not require the Java VM to be started with any special options. This utility is included in the JDK download.

The following table lists the jstat command options.

<table>
<thead>
<tr>
<th>Table 1-11 Options to the jstat Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>option</td>
</tr>
<tr>
<td>-class</td>
</tr>
<tr>
<td>-compiler</td>
</tr>
<tr>
<td>-gc</td>
</tr>
<tr>
<td>-gccapacity</td>
</tr>
<tr>
<td>-gccause</td>
</tr>
<tr>
<td>-gcnew</td>
</tr>
<tr>
<td>-gcnewcapacity</td>
</tr>
<tr>
<td>-gcold</td>
</tr>
<tr>
<td>-gcoldcapacity</td>
</tr>
<tr>
<td>-gcpermcapacity</td>
</tr>
<tr>
<td>-gcutil</td>
</tr>
<tr>
<td>-printcompilation</td>
</tr>
</tbody>
</table>

A complete description of the jstat tool, including examples, can be found at:

http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstat.html

Following is an example jstat command which attaches to pid 27395 and takes five samples at 250 millisecond intervals. The -gcnew option specifies that statistics of the behavior of the new generation is output.

```
$ jstat -gcnew 27395 250 5
```

```
S0C  S1C  S0U  S1U  TT  MTT  DSS  EC  EU  YGC  YGCT
64.0 64.0  0.0  31.7  31  31  32.0  512.0  178.6  249  0.203
64.0 64.0  0.0  31.7  31  31  32.0  512.0  355.5  249  0.203
64.0 64.0  35.4  0.0  2  31  32.0  512.0  21.9  250  0.204
64.0 64.0  35.4  0.0  2  31  32.0  512.0  245.9  250  0.204
64.0 64.0  35.4  0.0  2  31  32.0  512.0  421.1  250  0.204
```
1.16  **jstatd (1.5 only)**

The jstatd tool launches an RMI (remote method invocation) server that monitors the creation and termination of Java VMs and provides an interface to allow remote monitoring tools to attach to Java VMs running on the local host.

For more information, refer to the following website:

http://java.sun.com/j2se/1.5.0/docs/tooldocs/share/jstatd.html

1.17  **-verbose:class**

The **-verbose:class** option displays information about each loaded class. It enables logging of class loading and unloading.

1.18  **-verbose:gc**

The **-verbose:gc** option enables logging of garbage collection (GC) information. It can be combined with other Java VM specific options such as **-XX:+PrintGCDetails** and **-XX:+PrintGCTimeStamps** to get more information about the GC. The information output includes the size of the generations before and after each GC, total size of the heap, the size of objects promoted, and the time taken.

These options along with detailed information about GC analysis and tuning, are described at Sun’s GC portal site:

http://java.sun.com/developer/technicalArticles/Programming/GCPortal

The **-verbose:gc** option can be dynamically enabled at runtime using the management API or JVMTI. The jconsole monitoring and management tool can also enable or disable this option when attached to a management Java VM.

For other GC logging options, see **-Xverbosegc**.

1.19  **-verbose:jni**

The **-verbose:jni** option enables logging of Java Native Interface (JNI). Specifically, when a JNI native method is resolved, the Java VM prints a trace message to the application console (standard output). It also prints a trace message when a native method is registered using the JNI RegisterNative() function. The **-verbose:jni** option may be useful when trying to diagnose issues with applications that use native libraries.

1.20  **-Xcheck:jni**

The **-Xcheck:jni** option is useful when trying to diagnose problems with applications that use the Java Native Interface (JNI). Sometimes there are bugs in the native code that cause the Java VM to crash or behave incorrectly. Add the **-Xcheck:jni** option to the command line when starting the application.

For example:

```java
java -Xcheck:jni MyApplication
```

The **-Xcheck:jni** tells the Java VM to do additional validation on the arguments passed to JNI functions. This option may not find all invalid arguments or diagnose logic bugs in the application code; however, it can help diagnose these types of problems.

When an invalid argument is detected, the Java VM prints a message to the application console (standard output), prints the stack trace of the offending thread, and aborts the Java VM. Following is an example where a NULL is incorrectly passed to a JNI function that does not allow NULL:

```
FATAL ERROR in native method: Null object passed to JNI
   at java.net.PlainSocketImpl.socketAccept(Native Method)
   at java.net.PlainSocketImpl.accept(PlainSocketImpl.java:343)
- locked <0x450b9f70> (a java.net.PlainSocketImpl)
   at java.net.ServerSocket.implAccept(ServerSocket.java:439)
   at java.net.ServerSocket.accept(ServerSocket.java:410)
   at org.apache.tomcat.service.PoolTcpEndpoint.acceptSocket(PoolTcpEndpoint.java:286)
```
Following is another example of output that is displayed when something other than a jfieldID is provided to a JNI function that expects a jfieldID:

FATAL ERROR in native method: Instance field not found in JNI get/set field operations

Following are some types of problems that -Xcheck:jni can help diagnose:
1. the JNI environment for the wrong thread is used
2. an invalid JNI reference is used
3. a reference to a non-array type is provided to a function that requires an array type
4. a non-static field ID is provided to a function that expects a static field ID
5. a JNI call is made with an exception pending

In general, all errors detected by -Xcheck:jni are fatal; the error is printed and the Java VM is aborted. One exception to this is a non-fatal warning that is printed when a JNI call is made within a JNI critical region. This is the warning that is displayed when this happens:

Warning: Calling other JNI functions in the scope of
Get/ReleasePrimitiveArrayCritical or Get/ReleaseStringCritical

A JNI critical region arises when native code uses the JNI GetPrimitiveArrayCritical() or GetStringCritical() functions to obtain a reference to an array or string in the Java heap. The reference is held until the native code calls the corresponding release function. The time between the get and release is called a JNI critical section, and during that time the Java VM cannot reach a state that allows garbage collection to occur. The general recommendation is that other JNI functions should not be used when in a JNI critical section, and in particular any JNI function that blocks could potentially cause a deadlock. The warning printed by -Xcheck:jni is an indication of a potential issue; it does not always indicate an application bug.

1.21 -Xverbosegc

The -Xverbosegc option prints out detailed information about the Java heap before and after garbage collection. The syntax is:

-Xverbosegc [:help] | [0 | 1] [:file = [stdout | stderr | <filename>]]

The “:help” option prints a description of the verbosegc output format.

The “0 | 1” option controls the printing of help information. Specifying value “0” will cause the heap information to be printed after every Old Generation GC or Full GC. Specifying value “1” (the default) will cause the heap information to be printed after every GC.

The “file = [stdout | stderr | <filename>]” option specifies the output file. The default is stderr, which directs the output to the standard error stream. Alternative choices for the output file are stdout and a user-specified filename.

At every garbage collection, 20 fields are printed as follows:

GC: %1 %2 %3 %4 %5 %6 %7 %8 %9 %10 %11 %12 %13 %14 %15 %16 %17 %18 %19 %20
The following table contains brief descriptions of these 20 fields.

**Table 1-12 Garbage Collection Field Information**

<table>
<thead>
<tr>
<th>Field</th>
<th>Information in Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Type of GC:</td>
</tr>
<tr>
<td></td>
<td>• 1: Scavenge (GC of New Generation only)</td>
</tr>
<tr>
<td></td>
<td>• 2: Old Generation GC or a Full GC</td>
</tr>
<tr>
<td></td>
<td>• 3: Complete background CMS GC</td>
</tr>
<tr>
<td></td>
<td>• 4: Incomplete background CMS GC</td>
</tr>
<tr>
<td></td>
<td>• 11: Ongoing CMS GC</td>
</tr>
<tr>
<td>2</td>
<td>Additional information based on GC type in field 1.</td>
</tr>
<tr>
<td>3</td>
<td>Program time at the beginning of the GC, in seconds.</td>
</tr>
<tr>
<td>4</td>
<td>GC invocation. Counts of background CMS GCs and other GCs are maintained separately.</td>
</tr>
<tr>
<td>5</td>
<td>Size of the object allocation request that forced the GC, in bytes.</td>
</tr>
<tr>
<td>6</td>
<td>Tenuring threshold — determines how long the newborn object remains in the New Generation.</td>
</tr>
<tr>
<td>7</td>
<td>Eden Sub-space (within the New Generation) occupied before GC.</td>
</tr>
<tr>
<td>8</td>
<td>Eden Sub-space (within the New Generation) occupied after GC.</td>
</tr>
<tr>
<td>9</td>
<td>Eden Sub-space (within the New Generation) current capacity.</td>
</tr>
<tr>
<td>10</td>
<td>Survivor Sub-space (within the New Generation) occupied before GC.</td>
</tr>
<tr>
<td>11</td>
<td>Survivor Sub-space (within the New Generation) occupied after GC.</td>
</tr>
<tr>
<td>12</td>
<td>Survivor Sub-space (within the New Generation) current capacity.</td>
</tr>
<tr>
<td>13</td>
<td>Old Generation occupied before GC.</td>
</tr>
<tr>
<td>14</td>
<td>Old Generation occupied after GC.</td>
</tr>
<tr>
<td>15</td>
<td>Old Generation current capacity.</td>
</tr>
<tr>
<td>16</td>
<td>Permanent Generation (storage of Reflective Objects) occupied before GC.</td>
</tr>
<tr>
<td>17</td>
<td>Permanent Generation (storage of Reflective Objects) occupied after GC.</td>
</tr>
<tr>
<td>18</td>
<td>Permanent Generation (storage of Reflective Objects) current capacity.</td>
</tr>
<tr>
<td>19</td>
<td>The total stop-the-world duration, in seconds.</td>
</tr>
<tr>
<td>20</td>
<td>The total time used in collection, in seconds.</td>
</tr>
</tbody>
</table>

For more details about these fields, use the `:help` option or refer to the Java Programmers Guide at the following website:


To better understand how garbage collection works in the Java VM, read the article "Improving Java Application Performance and Scalability by Reducing Garbage Collection Times and Sizing Memory Using JDK 1.4.1" (November 2002) by Nagendra Nagarajayya and J. Steven Mayer at the following website:

http://developers.sun.com/techtopics/mobility/midp/articles/garbagecollection2/#17.1

Additionally, HP recommends using the HPjte influence tool, which can display graphically the information contained in a `-Xverbosegc` log. Refer to the HPjte command for more information.

### 1.22 -XX:+HeapDump and `_JAVA_HEAPDUMP` Environment Variable

The `-XX:+HeapDump` option can be used to observe memory allocation in a running Java application by taking snapshots of the heap over time. Another way to get heap dumps is to use the `_JAVA_HEAPDUMP` environment variable; setting this environment variable allows memory snapshots to be taken without making any modifications to the Java command line. In order to enable this functionality, either use the command-line option or set the environment variable (for example, `export _JAVA_HEAPDUMP=1`) before starting the Java application. This option is available beginning with SDK 1.4.2.10 and JDK 1.5.0.03.
The output is similar to that produced by the \texttt{-Xrunhprof:heap=dump} option except that the thread and trace information is not printed to the output file.

With the \texttt{-XX:+HeapDump} option enabled, each time the process is sent a SIGQUIT signal, the Java VM produces a snapshot of the Java heap in hprof ASCII format. The name of the file has the following format: \texttt{java\_<pid>\_<date>\_<time>\_heapDump.hprof.txt}.

If \_JAVA\_HEAPDUMP\_ONLY is set, then heap dumps are triggered by SIGVTALRM instead of SIGQUIT for this option. Only the heap dump is produced; that is, the thread and trace dump of the application to stdout is suppressed. Setting the \_JAVA\_BINARY\_HEAPDUMP environment variable along with \_JAVA\_HEAPDUMP\_ONLY produces a binary format heap dump when the SIGVTALRM is sent to the process instead of an ASCII one.

\textbf{NOTE:} A full GC is executed prior to taking the heap snapshot.

\subsection{Other HeapDump Options}

In addition to \texttt{-XX:+HeapDump}, there are three other HeapDump options available: \texttt{-XX:+HeapDumpOnCtrlBreak}, \texttt{-XX:+HeapDumpOnOutOfMemory}, and \texttt{-XX:+HeapDumpOnly}. Following is a table describing the four heap dump options. Additional information on these three heap dump options is provided following the table.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|}
\hline
Option & Trigger & hprof Format & Filename \\
\hline
\texttt{-XX:+HeapDump} & SIGQUIT & ASCII; set the \_JAVA\_BINARY\_HEAPDUMP environment variable to get binary & \texttt{java\_<pid>\_<date>\_<time>\_heapDump.hprof.txt} \\
\hline
\texttt{-XX:+HeapDumpOnCtrlBreak} & SIGQUIT & binary & \texttt{java\_<pid>.hprof.<millitime>} \\
\hline
\texttt{-XX:+HeapDumpOnOutOfMemory} & Out of Memory & binary & \texttt{java\_<pid>.hprof.<millitime>} or the file specified by \texttt{-XX:HeapDumpPath=file} \\
\hline
\texttt{-XX:+HeapDumpOnly} & SIGVTALRM & ASCII; set the \_JAVA\_BINARY\_HEAPDUMP environment variable to get binary & \texttt{java\_<pid>\_<date>\_<time>\_heapDump.hprof.txt} \\
\hline
\end{tabular}
\caption{Overview of HeapDump Options}
\end{table}

\subsection{\texttt{-XX:+HeapDumpOnCtrlBreak}}

The \texttt{-XX:+HeapDumpOnCtrlBreak} option is available beginning with SDK 1.4.2.11 and JDK 1.5.0.05. It enables the ability to take snapshots of the Java heap when a SIGQUIT signal is sent to the Java process without using the JVMTI-based \texttt{-Xrunhprof:heap=dump} option. This option is similar to \texttt{-XX:+HeapDump} except the output format is in binary hprof format and the output is placed into a filename with the following naming convention: \texttt{java\_<pid>.hprof.<millitime>}.

If the HP environment variable \_JAVA\_HEAPDUMP is set and this option is specified, then both hprof ASCII and binary dump files are created when a SIGQUIT is sent to the process. For example, the following file names are created: \texttt{java\_27298\_hprof.1152743593943} and \texttt{java\_27298\_060712\_153313\_heapDump.hprof.txt}.

If \texttt{JAVA\_BINARY\_HEAPDUMP} is set and the \texttt{-Xrunhprof:heap=dump} command is given, then both hprof ASCII and binary files are produced for this option.

\subsection{\texttt{-XX:+HeapDumpOnOutOfMemory}}

The \texttt{-XX:+HeapDumpOnOutOfMemory} option is available beginning with SDK 1.4.2.11 and JDK 1.5.0.04. This option enables dumping of the Java heap when an “Out Of Memory” error condition occurs in the Java VM. The heap dump file name defaults to \texttt{java\_pid\_hprof} in the current working directory. The option \texttt{-XX:HeapDumpPath=file} may be used to specify the heap dump file name or a directory where the heap dump file should be created. The only heap dump format generated by the \texttt{-XX:+HeapDumpOnOutOfMemory} option is the hprof binary format.
One known issue exists: the -XX:+HeapDumpOnOutOfMemory option does not work with the low-pause collector (option -XX:+UseConcMarkSweepGC).

1.22.4 -XX:+HeapDumpOnly

Starting with SDK 1.4.2.11 and JDK 1.5.0.05, the -XX:+HeapDumpOnly option or the _JAVA_HEAPDUMP_ONLY environment variable can be used to enable heap dumps using the SIGVTALRM signal (signal 20). This interface is provided to separate the generation of thread and trace information triggered via SIGQUIT from the heap dump information. If the -XX:+HeapDumpOnly option is specified or the _JAVA_HEAPDUMP_ONLY environment variable is set, then the heap dump functionality is triggered by sending SIGVTALRM to the process. The printing of thread and trace information to stdout is suppressed.

The heap dump is written to a file with the following filename format:
java_<pid>_<date>_<time>_heapDump.hprof.txt.

The default output format is ASCII. The output format can be changed to hprof binary format by setting the _JAVA_BINARY_HEAPDUMP environment variable. This environment variable can also be used with the -XX:+HeapDump option to generate hprof binary format with the SIGQUIT signal.

1.22.5 Using Heap Dumps to Monitor Memory Usage

By creating a series of heap dump snapshots, you can see how the number and size of objects varies over time. It is a good idea to collect at least three snapshots. The first one serves as a baseline. It should be taken after the application has finished initializing and has been running for a short time. The second snapshot should be taken after the residual heap size has grown significantly. Monitor this using -Xverbosegc and HPjtune. Try to take the last snapshot just before the heap has grown to a point where it causes problems resulting in the application spending the majority of its time doing full GCs. If you take other snapshots, spread them out evenly based on residual heap size throughout the running of the application.

Once you have collected the snapshots, read them into HPjmeter (run with -Xverbosegc to monitor memory usage). Use small heap sizes so that the analysis with HPjmeter requires less memory. Read two files in and compare them using the File->Compare option. You should be able to find out the types of objects that are accumulating in the Java heap. Select a type using the Mark to Find option and go back to a view of one of the snapshots. Go to the Metric->Call Graph Tree option and do a Find. You should be able to see the context of the object retention.

1.23 -XX:OnError

When a fatal error occurs, the Java VM can optionally execute a user-supplied script or command. The script or command is specified using the -XX:OnError:<string> command line option, where <string> is a single command or a list of commands each separated by a semicolon. Within <string> all occurrences of “%p” are replaced with the current process id (pid), and all occurrences of “%%%” are replaced by a single “%”.

Following is an example showing how the fatal error report can be mailed to a support alias when a fatal error is encountered:
java -XX:OnError="cat hs_err_pid%p.log|mail support@acme.com" MyApplication

Following is an example that launches gdb when an unexpected error is encountered. Once launched, gdb attaches to the Java VM process:
java -XX:OnError="gdb - %p" MyApplication

1.24 -XX:+ShowMessageBoxOnError

In addition to the -XX:OnError option, the Java VM can also be provided with the option -XX:+ShowMessageBoxOnError. When this option is set and a fatal error is encountered, the Java VM outputs information about the fatal error and asks the user if the debugger should be launched. The output and prompt are sent to the application console (standard input and standard output). Following is an example:
Unexpected Error -----------------------------------------------------------------------------
SIGSEGV (0xb) at pc=0x2000000001164db1, pid=10791, tid=1026

Do you want to debug the problem?

To debug, run 'gdb /proc/10791/exe 10791'; then switch to thread 1026
Enter 'yes' to launch gdb automatically (PATH must include gdb)
Otherwise, press RETURN to abort...

==============================================================================

In this case, a SIGSEGV has occurred and the user is prompted whether to launch the debugger to attach to the process. If the user enters “y” or “yes” then gdb is launched.

In the previous example, the output includes the process id (10791) and also the thread id (1026). If the debugger is launched then one of the initial steps taken in the debugger should be to select the thread and obtain its stack trace.

While waiting for a response from the process, it is possible to use other tools to obtain a crash dump or query the state of the process.

Generally, -XX:+ShowMessageOnError option is more useful in a development environment where debugger tools are available. The -XX:OnError option is more suitable for production environments where a fixed sequence of commands or scripts are executed when a fatal error is encountered.
2 Useful System Tools for Java Troubleshooting

This chapter contains information about some system tools available on HP-UX that are useful when troubleshooting Java application problems. The tools discussed include: GlancePlus, tusc, Prospect, HP Caliper, sar, vmstat, iostat, swapinfo, top, netstat, and others.

2.1 GlancePlus

GlancePlus is a system performance monitoring and diagnostic tool. It lets you easily examine system activities, identify and resolve performance bottlenecks, and tune your system for more efficient operation. For more information on GlancePlus, refer to the following website:


2.2 tusc

tusc gives you another view into the system activity, in addition to Java stack traces, GlancePlus, and HPjmeter. It has many options, which you can display by entering the command tusc -help. For more information on tusc, refer to the following website:


2.3 Prospect

Prospect is a performance analysis tool. Beginning with Prospect revision 2.2.0, you can use Prospect to get a profile of the compiled Java methods that the Java VM compiler creates in data space. In order to activate this functionality, you must have SDK 1.3.1.02 or following releases. For more information on the Prospect performance analysis tool, refer to the following website:

http://h21007.www2.hp.com/dspp/tech/tech_TechSoftwareDetailPage_ID/1,1703,3282,00.html

2.4 HP Caliper

HP Caliper is a general-purpose performance analysis tool for applications running on Integrity systems. It helps you understand the execution of your applications and identify ways to improve their performance. For more information on the HP Caliper tool, refer to the following website:


2.5 sar

The sar command is a tool to report various system activities, such as CPU, I/O, context switches, interrupts, page faults, and other kernel actions. For more information on this command, refer to the following website:

http://docs.hp.com/en/B2355-60127/sar.1M.html

2.6 vmstat

The vmstat command reports statistics about the process, virtual memory, trap, and CPU activity. For more information on this command, refer to the following website:


2.7 iostat

The iostat command iteratively reports I/O statistics for each active disk on the system. For more information on this command, refer to the following website:


2.8 swapinfo

The swapinfo command displays information about device and file system paging space. For more information on this command, refer to the following website:
2.9 top

The top command displays the top processes on the system, periodically updating the information; raw CPU percentage is used to rank the processes. For more information on this command, refer to the following website:


2.10 netstat

The netstat command displays statistics for network interfaces and protocols as well as the contents of various network-related data structures. It can show packet traffic, connections, error rates, and more. For more information on this command, refer to the following website:


2.11 Other Tools

The Developer and Solution Partner Program's (DSPP) technical information web page contains links to debugging information. There are links from this page to other websites containing technical papers, tips, tutorials, and more. To review this information, refer to the following website:

http://h21007.www2.hp.com/dspp/topic/topic_DetailSubHeadPage_IDX/1,4946,0-10301-TECHDOCUMENT,00.html
3 Getting Help from Hewlett-Packard

Sometimes you need help troubleshooting your Java application problems. Before opening a support call, search for information that may help you by referring to the Go Java! website:

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

This site contains much information about Java, including known issues, release notes, patches, downloads, documentation, and more. If you still need troubleshooting help after looking at this website and you have a support contract with Hewlett-Packard (HP), follow the instructions outlined in this chapter to collect the necessary information before opening a support call.

3.1 Problem Report Checklist

Use this checklist to collect information before you request support. Providing more information when you initiate your support call reduces the time it takes for support engineers to start working on your problem.

1. Problem Description
   a. Did this Java application ever work?
   b. What is the problem (abort, hang, performance, and so on)?
   c. What messages are written to stdout or stderr relating to the problem?
   d. Does the problem occur every time the application is run or intermittently?
   e. What are the application details? Include the following:
      • Name of the application.
      • What the application does.
      • The command line and options used to start the application.
      • Description of the expected behavior.
      • Description of the actual behavior.
      • The application stack that you are running—for example, the webserver name or the application server name.
   g. Do you have a workaround for the problem? If so, describe it.

2. Problem Data.
   a. Core file
   b. Fatal error log (hs_err_pid<pid>.log)
   c. Stack trace

3. System Information
   a. What version of HP-UX is on the system? Provide the output from the `uname -a` command.
   b. What patches are installed on the system?
   c. What window manager is being used? For example, Reflections X or X Windows. Or is the application running inside a browser? If so, which one?

4. Java Environment
   a. What is the version of the Java VM that is having the problem? Run the command `java -version` to get this information.
   b. What are the values of the environment variables used by Java?
   c. What libraries are being loaded?

5. Contact Information
   a. Contact name
   b. Company name
   c. Phone number
   d. E-mail address
The following subsections provide instructions for collecting the necessary problem, system, and Java environment information. The final subsection contains instructions for packaging the files you need to send to Hewlett-Packard.

### 3.2 Collecting Problem Data

Three pieces of information are essential for analyzing most problems—the core file, the fatal error log, and the stack trace. Following are instructions for how to collect this information.

#### 3.2.1 Collecting Core File Information

This section begins with a checklist to follow to make sure you can collect useful core files. It then reviews how you can generate a core file if one is not generated for you. Finally, there is a discussion about how to verify that your core file is valid.

##### 3.2.1.1 Core File Checklist

Core files contain useful information, if they are complete. Sometimes you need to configure your system to make sure you can save complete core files. Consider the following items to ensure you can create complete core files.

1. Estimate the core file size.
2. Ensure your process can write large core files.
3. Verify you have enough free disk space.
4. Make sure the directory where the core file will reside supports a large file system. If not, write the core file to a directory that does.
5. Make sure you have the correct permissions to write core files.

Following are additional details on each of these steps.

##### 3.2.1.1.1 Estimate Core File Size

The size of the \(-Xmx\) option affects the core file size. Use these rules to estimate the size of the Java core file:

- \(-Xmx\) is less than 1,500 MB. The core file will be less than or equal to 2 GB.
- \(-Xmx\) is between 1,500 and 2,400 MB. The core file will be less than or equal to 3 GB.
- \(-Xmx\) is greater than 2,400 MB. The core file will be less than or equal to 4 GB.

##### 3.2.1.1.2 Ensure Process Can Write Large Core Files

Check your coredump block size to make sure it is set to unlimited using the `ulimit -a` command:

```bash
$ ulimit -a
```

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (seconds)</td>
<td>unlimited</td>
</tr>
<tr>
<td>file (blocks)</td>
<td>unlimited</td>
</tr>
<tr>
<td>data (kbytes)</td>
<td>4292870144</td>
</tr>
<tr>
<td>stack (kbytes)</td>
<td>8192</td>
</tr>
<tr>
<td>memory (kbytes)</td>
<td>unlimited</td>
</tr>
<tr>
<td>coredump (blocks)</td>
<td>4194303</td>
</tr>
</tbody>
</table>

If coredump is not set to unlimited, set it to unlimited using the `ulimit -c` command:

```bash
$ ulimit -c unlimited
```

```bash
$ ulimit -a
```

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>time (seconds)</td>
<td>unlimited</td>
</tr>
<tr>
<td>file (blocks)</td>
<td>unlimited</td>
</tr>
<tr>
<td>data (kbytes)</td>
<td>4292870144</td>
</tr>
<tr>
<td>stack (kbytes)</td>
<td>8192</td>
</tr>
<tr>
<td>memory (kbytes)</td>
<td>unlimited</td>
</tr>
<tr>
<td>coredump (blocks)</td>
<td>unlimited</td>
</tr>
</tbody>
</table>
3.2.1.1.3 Verify Amount of Disk Space

Check the amount of disk space available in the current working directory using the `df -kP` command.

```bash
$ df -kP /home/mycurrentdir
Filesystem 1024-blocks Used Available Capacity Mounted on
/dev/vg00/lvol5 1022152 563712 458440 56% /home
```

3.2.1.1.4 Check If Directory Supports Large File Systems

Use the `fsadm` command as root to check if your directory supports large file systems. If you do not execute this command as root, you may not get meaningful results. Following is an example:

```bash
<root>$ /usr/sbin/fsadm <mount_point>
```

Following is example output when the file system is set up to support large files and when it is not set up to support large files:

```bash
<root>$ /usr/sbin/fsadm /extra
fsadm: /etc/default/fs is used for determining the file system type
largefiles
<root>$ /usr/sbin/fsadm /stand
fsadm: /etc/default/fs is used for determining the file system type
nolargefiles
```

If the directory does not support large file systems, you need to write the core file to an alternate directory. Do this by setting the `JAVA_CORE_DESTINATION` environment variable (available starting with SDK 1.4.2) to the name of the directory and create the directory. For example:

```bash
$ export JAVA_CORE_DESTINATION=<alt_dir>
$ mkdir $JAVA_CORE_DESTINATION
```

Java creates a directory named `core` under the `JAVA_CORE_DESTINATION` directory where the core and `hs_err_pid<pid>.log` files are written. For example:

```bash
$ cd $JAVA_CORE_DESTINATION
$ ls
core.29757
```

Use a symbolic link to link to this directory. For example,

```bash
$ ln -s <alt_dir>/core <cwd_of_app>/core
```

3.2.1.1.5 Ensure Permissions Allow Core Files

Some Java processes run setuid; that is, a process where the effective uid or gid differs from the real uid or gid. On HP-UX 11.11 and later versions a kernel security feature prevents core file creation for these processes. Use the following command when you are logged in as the root user to enable core dumps of setuid Java processes:

```bash
$ echo "dump_all/W 1" | adb -w /stand/vmunix /dev/kmem
```

This capability is turned on only for the current boot.
3.2.1.2 Generating a Core File

Analyzing the core file is essential for troubleshooting problems. Core files are automatically generated for application aborts. For hung processes and performance issues, you need to generate them using gdb’s dumpcore command.

The gdb dumpcore command forces the generation of a core file without killing a running process. This command causes a core file named core.<pid> to be created. The current process state is not modified when this command is issued.

Following is an example for a Java application running on an Integrity system:

```
$ echo "dumpcore\nq" > gdb_cmds
$ ps -u myuser | grep java
12290 pts/6  12:58 java
$ gdb --command=gdb_cmds -batch /opt/java1.4/bin/IA64N/java 12290
```

This generates a core file in the current directory with the name core.12290.

3.2.1.3 Verifying a Core File

Once you have successfully collected your core file, you should verify that it is complete and valid. To do this, open the core file in gdb and check the error and warning messages. If you get the message “<corefilename> is not a core dump: File format not recognized” when you open the file, your core file is invalid. Following is example output for a corrupt core file:

```
$ gdb lib/java core1
HP gdb 3.1.5 for PA-RISC 1.1 or 2.0 (narrow), HP-UX 11.00.
Copyright 1986 - 2001 Free Software Foundation, Inc.
Hewlett-Packard Wildebeest 3.1.5 (based on GDB) is covered by the
GNU General Public License. Type "show copying" to see the conditions to
change it and/or distribute copies. Type "show warranty" for warranty/support.
.. 
"/home/sample/pics/4000069294/core1" is not a core dump: File format not recognized
```

Sometimes core files get truncated. Check for this by issuing the “what core” command. If you do not see the dld.s1 version at the bottom of the what output, then the core file is truncated and is not usable. In the following example, the dld.s1 version exists at the bottom of the what output so you know the core file is not truncated.

```
$ what core
core:
some other library names and version information ...
92453-07 dld dld dld.s1 B.11.48 EXP 051121
```

3.2.2 Collecting Fatal Error Log Information

When a Java application aborts, the fatal error log file (hs_err_pid<pid>.log) is generated. The contents of this file vary depending on the architecture and the Java version (for example, early Java versions generate less information in the fatal error log). Following is a summary of the type of information contained in this file:

1. The error causing the Java VM to abort, including the pc, process id, and thread id at which the error occurred. For example:

   ```
   # An unexpected error has been detected by HotSpot Virtual Machine:
   #
   # SIGSEGV (11) at pc=7541df20, pid=25675, tid=1
   ```

2. The Java version and problematic frame. For example:

   ```
   # Java VM: Java HotSpot(TM) Server VM (1.4.2
   # 1.4.2.10-060112-19:42-IA64N IA64 mixed mode)
   ```
3. Information about the current thread, including:
   a. the executing thread
   b. siginfo at the point of failure
   c. stack pointer and hex dump of the top of memory stack
   d. hex dump at the location of the current pc
   e. stack range and stack free space

4. Process information, including:
   a. a dump of all active threads at the time of the abort (SDK 1.4.2.04+)
   b. Java VM state (whether at safepoint or not) (SDK 1.4.2.10+)
   c. mutex state (SDK 1.4.2.10+)
   d. a summary of heap status; for example:

```
Heap
  def new generation   total 5632K, used 144K [6d400000, 6da10000, 6e950000)
  eden space 5056K,  2% used [6d400000, 6d424040, 6d8f0000)
  from space 576K,  0% used [6d8f0000, 6d8f0000, 6d980000)
  to space 576K,  0% used [6d980000, 6d980000, 6da10000)
  tenured generation total 12480K, used 0K [6e950000, 6f580000, 71400000)
  the space 12480K,  0% used [6e950000, 6e950000, 6e950200, 6f580000)
  compacting perm gen total 16384K, used 1118K [71400000, 72400000, 75400000)
  the space 16384K,  6% used [71400000, 71517860, 71517a00, 72400000)
```

   e. dynamic libraries loaded by the process (SDK 1.4.2.04+)
   f. Java VM arguments (SDK 1.4.2.04+)
   g. Java-related environment variables

5. System Information. This includes operating system name, version, CPU, memory, and system load. For example:

```
OS: HPUX
uname:HP-UX B.11.23 U ia64
rlimit: STACK 98252k, CORE 2097151k, NOFILE 4096, AS infinity
load average:0.12 0.19 0.22

CPU:total 8 Processor          = McKinley
Processor features = branchlong
Memory: 4k page, physical 16743644k

vm_info: Java HotSpot(TM) Server VM (1.4.2.10-060112-19:42-IA64N)
for hp-ux-ia64 built on Jan 12 2006 20:09:37 by jnteg with aCC
```

3.2.3 Collecting Stack Trace Information

On PA-RISC systems, a stack trace is printed to stderr when the application aborts. On Integrity systems, branch and general register contents are printed to stderr when an application aborts. The stack trace (PA-RISC systems) and register contents (Integrity systems) are not printed to the hs_err_pid<pid>.log file; therefore, the contents of stderr should be captured into a file and sent to HP along with the hs_err_pid<pid>.log, core file, and libraries.

3.3 Collecting System Information

Along with HP-UX version information and information about which window manager is being used, it is also useful to know which patches are installed on the system. Use the swlist command to get this list. For example:
3.4 Collecting Java Environment Information

In order to perform core file analysis, you need to collect information about some environment variables and libraries used by the failed application. The following subsections describe how to do this.

3.4.1 Environment Variables

To facilitate troubleshooting, it is important to know the value of the environment variables that can affect the behavior of Java applications (for example, CLASSPATH). To collect these application runtime environment variable values, run the following command under the same environment (that is, the same user) that the Java application was executed:

(ksh)$ env > app_environment.txt
(csh)$ getenv > app_environment.txt

Include the app_environment.txt file when you send in your collected data files to Hewlett-Packard.

3.4.2 Libraries

In order to perform core file analysis, you must have access to libraries used by the failed application. The method used for determining which libraries were used depends on whether or not gdb is available on the system.

If gdb is not available, then locate files by either examining the stdout of the failed application or the hs_err_pid<pid>.log file. Either of these should list all the libraries used. Using this list, manually copy the files.
If `gdb` is available on the system where the failure occurred, issue `gdb`'s `packcore` command:

```
(gdb) packcore
```

This command creates a directory called `packcore` under the current directory, and in this directory a file called `modules.tar` is created. This tar file contains all libraries used by the application.

In some situations, only a core file can be obtained. In this case limited troubleshooting can take place since some crucial pieces of information are missing.

There is one additional library that should be collected: `libjunwind`. This library is used by `gdb` to unwind Java bytecode frames; its routines help make stack traces more readable and understandable. Since this library is only used during debugging, it is not included in the tar file generated by `getcore`.

The following table shows the location of the `libjunwind` library for PA-RISC applications:

<table>
<thead>
<tr>
<th>Application Type</th>
<th><code>libjunwind</code> Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>PA1.1 applications (java -pa11)</td>
<td><code>/opt/&lt;java_vers&gt;/jre/lib/PA_RISC/server/libjunwind.sl</code></td>
</tr>
<tr>
<td>PA2.0 32-bit applications (default PA-RISC)</td>
<td><code>/opt/&lt;java_vers&gt;/jre/lib/PA_RISC2.0/server/libjunwind.sl</code></td>
</tr>
<tr>
<td>PA2.0 64-bit applications (java -d64)</td>
<td><code>/opt/&lt;java_vers&gt;/jre/lib/PA_RISC2.0W/server/libjunwind.sl</code></td>
</tr>
</tbody>
</table>

On Integrity systems, beginning with SDK 1.4.0.10 and JDK 1.5.0.03, there are two `libjunwind` libraries for each Java VM, `libjunwind64.so` and `libunwind.so`. The following table shows the location of these libraries for both 32-bit and 64-bit applications.

<table>
<thead>
<tr>
<th>Application Type</th>
<th><code>libjunwind</code> Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>32-bit applications</td>
<td><code>/opt/&lt;java_vers&gt;/jre/lib/IA64N/server/libjunwind*.so</code></td>
</tr>
<tr>
<td>64-bit applications</td>
<td><code>/opt/&lt;java_vers&gt;/jre/lib/IA64W/server/libjunwind*.so</code></td>
</tr>
</tbody>
</table>

### 3.5 Packaging Files

The core file, `modules.tar` file, and `libjunwind` library are all large, so they should be compressed to save time and disk space during their transmission. One method for compressing files is to use the Java archive tool, `jar`. This tool is included with all Java installations.

For example, to compress the `core.7145`, `modules.tar`, and `libjunwind.sl` files into file `debug.jar`, issue the following command:

```
jar cvf debug.jar core.7145 modules.tar libjunwind.sl
```

Alternately, you can use `compress` or `gzip` to compress the files, and then use `tar` to bundle them together.
## Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>GC</td>
<td>Garbage collection.</td>
</tr>
<tr>
<td>gid</td>
<td>Group id.</td>
</tr>
<tr>
<td>HotSpot VM</td>
<td>The JDK comes with a virtual machine implementation called the Java HotSpot VM.</td>
</tr>
<tr>
<td>Java VM</td>
<td>On HP implementations this is the same as the HotSpot VM.</td>
</tr>
<tr>
<td>JDK</td>
<td>The Java Developer's Kit is the set of Java development tools consisting of the API classes, a Java compiler, and the Java virtual machine.</td>
</tr>
<tr>
<td>JMX</td>
<td>Java Management Extensions technology provides the tools for building distributed, web-based, modular and dynamic solutions for managing and monitoring devices, applications, and service-driven networks.</td>
</tr>
<tr>
<td>JNI</td>
<td>The JNI is the native programming interface for Java that is part of the JDK. It allows Java code to operate with applications and libraries written in other languages, such as C, C++, and assembly.</td>
</tr>
<tr>
<td>JRE</td>
<td>The Java Runtime Environment provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language.</td>
</tr>
<tr>
<td>JVMTI</td>
<td>The Java Virtual Machine Tool Interface provides both a way to inspect the state and to control the execution of applications running in the Java VM.</td>
</tr>
<tr>
<td>RMI</td>
<td>Java Remote Invocation lets Java applications communicate across a network.</td>
</tr>
<tr>
<td>SDK</td>
<td>The Java Software Developer's Kit is the set of Java development tools consisting of the API classes, a Java compiler, and the Java virtual machine.</td>
</tr>
<tr>
<td>setuid process</td>
<td>A process where the effective uid or gid differs from the real uid or gid.</td>
</tr>
<tr>
<td>uid</td>
<td>User id.</td>
</tr>
</tbody>
</table>
## Index

### Symbols
- `-verbose:class`, 31
- `-verbose:gc`, 31
- `-verbose:jni`, 31
- `-Xcheck:jni`, 31
- `-Xverbose:gc`, 32
- `-XX:+HeapDump`, 33
- `-XX:+HeapDumpOnCtrlBreak`, 34
- `-XX:+HeapDumpOnly`, 35
- `-XX:+HeapDumpOutOfMemory`, 34
- `-XX:+ShowMessageBoxOnError`, 35
- `-XX:+OnError`, 35
- `_JAVA_HEAPDUMP` environment variable, 33

### C
- core file checklist, 40
- crash analysis tools, 11
- ctrl-break handler, 13
  - example output, 14

### D
- deadlocked process
  - tools and options for debugging, 11
- Developer and Solution Partner Program (DSPP), 38
- dumpcore, 42

### F
- fatal error handling
  - options, 12
- fatal error log, 15
  - information contained in, 42

### G
- gdb
  - dumpcore, 42
  - invoking on a core file, 18
  - invoking on a hung process, 18
  - Java stack unwind features, 16
  - packcore, 45
  - subcommands for Java VM debugging, 17
  - support for Java, 16
- `GDB_JAVA_UNWINDLIB` environment variable, 16
- generating core files, 42
- GlancePlus, 37
- GoJava! website, 39

### H
- hat, 26
- heap dump
  - monitoring memory usage, 35
  - options, 34
- HP Caliper, 37
- HPjconfig, 19
  - GUI mode, 20
  - non-GUI mode, 21

### J
- jar, 45
- Java archive tool, 45
- `java.security.debug` system property, 27
- `JAVA_CORE_DESTINATION` environment variable, 41
- `JAVA_LAUNCHER_OPTIONS` environment variable, 28
- `JAVA_TOOL_OPTIONS` environment variable, 27
- jconsole, 28
- jdb, 29
- jhat, 26
- jinfo, 13
- jmap, 13
- jps, 29
  - example, 30
  - usage, 29
- jstack, 13
- jstat, 30
- jstatd, 31

### L
- libjunwind, 45
  - location on Integrity systems, 45
  - location on PA-RISC systems, 45

### M
- memory monitoring
  - tools and options, 12
- miscellaneous troubleshooting options, 13

### N
- netstat, 38

### P
- packcore, 45
- performance monitoring tools, 13
- problem report checklist, 39
- Prospect, 37

### S
- sar, 37
- Serviceability Agent, 13
- stack trace information, 43
- swapinfo, 37
system information, 43
system tools, 37

T
top, 38
tusc, 37

V
vmstat, 37