NonStop Server for Java 7.0 Tools Reference Pages

HP Part Number: 693950-002
Published: March 2014
Edition: J06.26 and all subsequent J-series RVUs and H06.15 and all subsequent H-series RVUs

Printed in the US

Contents

About this Manual

This document contains the *Tools Reference Pages* for the HP NonStop™ Server for Java™, based on the Java Platform Standard Edition 7.

Product Version

HP NonStop Server for Java, based on Java Platform Standard Edition 7

Supported Hardware

All HP Integrity NonStop NS-series servers

Supported Release Version Updates (RVUs)

This manual supports J06.15 and all subsequent J-series RVUs and H06.26 and all subsequent H-series RVUs, until otherwise indicated by its replacement publications.

Document history

Part number	Product version	Published
693950–001	HP NonStop Server for Java, based on Java Platform Standard Edition 7	February 2013
693950–002	HP NonStop Server for Java, based on Java Platform Standard Edition 7	March 2014

New and Changed Information for March 2014 (693950–002)

Changes to 693950–002 manual are as follows:

- Updated the section and added a note in "Connecting for Remote Debugging" (page 33).
- Replaced "–Xrunjdwp:" with "–agentlib:jdwp=" in "Starting a `jdb` Session" (page 29) and "Connecting for Remote Debugging" (page 33).

New and Changed Information for February 2013 (693950–001)

Changes to 693950–001 manual are as follows:

- Updated the section "nsjps: NonStop Java Virtual Machine Process Status Tool" (page 46).

Notation Conventions

General Syntax Notation

This list summarizes the notation conventions for syntax presentation in this manual.

UPPERCASE LETTERS

Uppercase letters indicate keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
MAXATTACH
```

lowercase italic letters

Lowercase italic letters indicate variable items that you supply. Items not enclosed in brackets are required. For example:

```
file-name
```

computer type

`Computer type` letters within text indicate C and Open System Services (OSS) keywords and reserved words. Type these items exactly as shown. Items not enclosed in brackets are required. For example:

```
myfile.c
```

italic computer type

`Italic computer type` letters within text indicate C and Open System Services (OSS) variable items that you supply. Items not enclosed in brackets are required. For example:

```
pathname
```

[ ] Brackets

Brackets enclose optional syntax items. For example:

```
TERM [\.system-name]$terminal-name
```

```
INT[ERRUPTS]
```

A group of items enclosed in brackets is a list from which you can choose one item or none. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
FC [ num ]
   [ -num ]
   [ text ]
```

```
K [ X | D ] address
```

{ } Braces

A group of items enclosed in braces is a list from which you are required to choose one item. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
LISTOPENS PROCESS { $appl-mgr-name }
                  { $process-name }
```

```
ALLOWSU { ON | OFF }
```

| Vertical Line

A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
INSPECT { OFF | ON | SAVEABEND }
```

… Ellipsis

An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed sequence of syntax items any number of times. For example:

```
M address [ , new-value ]…
```

```
[ - ] {0|1|2|3|4|5|6|7|8|9}…
```

An ellipsis immediately following a single syntax item indicates that you can repeat that syntax item any number of times. For example:

```
"s-char"
```

Punctuation

Parentheses, commas, semicolons, and other symbols not previously described must be typed as shown. For example:

```
error := NEXTFILENAME ( file-name ) ;
```

```
LISTOPENS SU $process-name.#su-name
```

Quotation marks around a symbol such as a bracket or brace indicate the symbol is a required character that you must type as shown. For example:

```
"[" repetition-constant-list "]"
```

Item Spacing

Spaces shown between items are required unless one of the items is a punctuation symbol such as a parenthesis or a comma. For example:

```
CALL STEPMOM ( process-id ) ;
```

If there is no space between two items, spaces are not permitted. In this example, no spaces are permitted between the period and any other items:

```
$process-name.#su-name
```

Line Spacing

If the syntax of a command is too long to fit on a single line, each continuation line is indented three spaces and is separated from the preceding line by a blank line. This spacing distinguishes items in a continuation line from items in a vertical list of selections. For example:

```
ALTER [ / OUT file-spec / ] LINE
   [ , attribute-spec ]…
```

!i and !o

In procedure calls, the !i notation follows an input parameter (one that passes data to the called procedure); the !o notation follows an output parameter (one that returns data to the calling program). For example:

```
CALL CHECKRESIZESEGMENT ( segment-id                                    !i
              , error ) ;                                  !o
```

!i,o

In procedure calls, the !i,o notation follows an input/output parameter (one that both passes data to the called procedure and returns data to the calling program). For example:

```
error := COMPRESSEDIT ( filenum ) ;                                !i,o
```

!i:I

In procedure calls, the !i:i notation follows an input string parameter that has a corresponding parameter specifying the length of the string in bytes. For example:

```
error := FILENAME_COMPARE_ ( filename1:length                          !i:i
                           , filename2:length ) ;                       !i:i
```

!o:I

In procedure calls, the !o:i notation follows an output buffer parameter that has a corresponding input parameter specifying the maximum length of the output buffer in bytes. For example:

```
error := FILE_GETINFO_ ( filenum                                        !i
            , [ filename:maxlen ]) ;                          !o:i
```

Notation for Messages

This list summarizes the notation conventions for the presentation of displayed messages in this manual.

Bold Text

Bold text in an example indicates user input typed at the terminal. For example:

```
ENTER RUN CODE
?123
CODE RECEIVED: 123.00
```

The user must press the Return key after typing the input.

Nonitalic text

Nonitalic letters, numbers, and punctuation indicate text that is displayed or returned exactly as shown. For example:

```
Backup Up.
```

lowercase italic letters

Lowercase italic letters indicate variable items whose values are displayed or returned. For example:

```
p-register
process-name
```

[ ] Brackets

Brackets enclose items that are sometimes, but not always, displayed. For example:

```
Event number = number [ Subject = first-subject-value
```

A group of items enclosed in brackets is a list of all possible items that can be displayed, of which one or none might actually be displayed. The items in the list can be arranged either vertically, with aligned brackets on each side of the list, or horizontally, enclosed in a pair of brackets and separated by vertical lines. For example:

```
proc-name trapped [ in SQL | in SQL file system ]
```

{ } Braces

A group of items enclosed in braces is a list of all possible items that can be displayed, of which one is actually displayed. The items in the list can be arranged either vertically, with aligned braces on each side of the list, or horizontally, enclosed in a pair of braces and separated by vertical lines. For example:

```
obj-type obj-name state changed to state caused by
{ Object | Operator | Service }
```

```
process-name State changed from old-objstate to objstate
{ Operator Request. }
{ Unknown. }
```

| Vertical Line

A vertical line separates alternatives in a horizontal list that is enclosed in brackets or braces. For example:

```
Transfer status: { OK | Failed }
```

% Percent Sign

A percent sign precedes a number that is not in decimal notation. The % notation precedes an octal number. The %B notation precedes a binary number. The %H notation precedes a hexadecimal number. For example:

```
%005400
```

```
%B101111
```

```
%H2F
```

```
P=% p-register E=% e-register
```

Notation for Management Programming Interfaces

This list summarizes the notation conventions used in the boxed descriptions of programmatic commands, event messages, and error lists in this manual.

UPPERCASE LETTERS

Uppercase letters indicate names from definition files. Type these names exactly as shown. For example:

```
ZCOM-TKN-SUBJ-SERV
```

lowercase letters

Words in lowercase letters are words that are part of the notation, including Data Definition Language (DDL) keywords. For example:

```
token-type
```

!r

The !r notation following a token or field name indicates that the token or field is required. For example:

```
ZCOM-TKN-OBJNAME   token-type ZSPI-TYP-STRING.     !r
```

!o. The !o notation following a token or field name indicates that the token or field is optional. For example:

```
ZSPI-TKN-MANAGER   token-type ZSPI-TYP-FNAME32.     !o
```

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 **docsfeedback@hp.com**.

Include the document title, part number, and any comment, error found, or suggestion for improvement concerning this document.

NonStop Server for Java 7.0 Tools Reference Pages

This section provides information about the tools used in NonStop Server for Java 7.0. Table 1 (page 11), illustrates the commands, tools, and its relative functionality.

Table 1 NonStop Server for Java 7.0 Tools Reference Pages

Command	Tool Name	Function
"apt: Annotation Processing Tool" (page 13)	Annotation Process Tool	Processes program annotations through a set of reflective APIs and a supporting infrastructure.
"extcheck: JAR Conflict Detection Tool" (page 14)	JAR Conflict Detection Tool	Detects version conflicts between a target Java Archive (JAR) file and currently installed extension JAR files.
"idlj: IDL-to-Java Compiler" (page 15)	IDL-to-Java Compiler	Generates Java bindings from a specified IDL file.
"jar: Java Archive Tool" (page 16)	Java Archive Tool	Combines multiple files into a single JAR file and retrieves files from a JAR file.
"jarsigner: JAR Signing and Verification Tool" (page 17)	JAR Signing and Verification Tool	Generates signatures for JAR files and verifies signatures of signed JAR files.
"java: Java Application Launcher" (page 18)	Java Application Launcher	Launches a Java application by starting a Java run time environment, loading a specified class, and invoking that class's main method.
"javac: Java Programming Language Compiler" (page 25)	Java Programming Language Compiler	Compiles Java source code into bytecode.
"javadoc: Java API Documentation Generator" (page 26)	Java API Documentation Generator	Generates API documentation in HTML or MIF format from Java source code.
"javah: C Header and Stub File Generator" (page 27)	C Header and Stub File Generator	Generates C header files and C stub files from a Java class. These files provide the connections that allow your Java and C code to interact.
"javap: Java Class File Disassembler" (page 28)	Java Class File Disassembler	Disassembles compiled Java files.
"jdb: Java Debugger" (page 29)	Java Debugger	Helps you find and fix bugs in Java programs.
"jrunscript: Command Line Script Shell" (page 35)	Command Line Script Shell	Supports both an interactive (read-eval-print) mode and a batch (-f option) mode of script execution.
"keytool: Key and Certificate Management Tool" (page 36)	Key and Certificate Management Tool	Manages a database of private keys and their associated certificate chains authenticating the corresponding public keys.
"native2ascii: Native-to-ASCII Converter" (page 37)	Native-to-ASCII Converter	Converts a file with Native-encoded characters to one with Unicode-encoded characters.
"orbd: Object Request Broker Daemon" (page 38)	Object Request Broker Daemon	Enables clients to transparently locate and invoke persistent objects on servers in the CORBA environment.
"rmic: Java RMI Compiler" (page 39)	Java RMI Compiler	Generates stubs and skeletons for remote objects that use either the JRMP

Table 1 NonStop Server for Java 7.0 Tools Reference Pages *(continued)*

Command	Tool Name	Function
		or IIOP. Also, generates OMG Interface Definition Language (IDL).
"rmid: Java RMI Activation System Daemon" (page 40)	Java RMI Activation System Daemon	Starts the activation system daemon that allows objects to be registered and activated in a Java virtual machine (VM).
"rmiregistry: Java Remote Object Registry" (page 41)	Java Remote Object Registry	Starts a remote object registry on the specified port on the current host.
"schemagen: Java Architecture for XML Binding Schema Generator" (page 42)	Java Architecture for XML Binding Schema Generator	Creates a schema file for each namespace referenced in your Java classes generator for Java Architecture for XML Binding.
"serialver: Serial Version Command" (page 43)	Serial Version Command	Returns the `serialVersionUID` of one or more classes.
"servertool: Java IDL Server Tool" (page 44)	Java IDL Server Tool	Provides a command line interface for application programmers to register, unregister, start up, and shut down a persistent server.
"tnameserv: Naming Service Access" (page 45)	Naming Service Access	Provides access to the naming service.
"nsjps: NonStop Java Virtual Machine Process Status Tool" (page 46)	NonStop Java Virtual Machine Process Status Tool	Lists and monitors the Java processes running on a system.
"wsgen : Java API for XML Web Services (JAX-WS) 2.0" (page 51)	Java API for XML Web Services (JAX-WS) 2.0	Generates JAX-WS portable artifacts used in JAX-WS web services.
"wsimport: JAX-WS 2.0" (page 52)	JAX-WS 2.0	Generates JAX-WS portable artifacts.
"xjc: Java Architecture for XML Binding Compiler" (page 53)	Java Architecture for XML Binding	Compiles schemas.

NOTE: The 64–bit version of NSJ7 tools are available at the location `$JAVA_HOME/bin/oss64`. The 32–bit version of the tools are available at the location `$JAVA_HOME/bin`[1].

1. `$JAVA_HOME` is the installation location of Java in the NonStop system. For example, `/usr/tandem/nssjava/jdk170_h70`.

1 apt: Annotation Processing Tool

The apt tool processes program annotations through a set of reflective APIs and a supporting infrastructure. For more information on apt tool, see the Oracle Java documentation for apt.

Synopsis

```
apt [-classpath classpath] [-sourcepath sourcepath] [-d directory] [-s
directory] [-factorypath path] [-factory class] [-print] [-nocompile]
[-Akey[=val] ...] [javac option] sourcefiles [@files]
```

See Also:

- "java: Java Application Launcher" (page 18)
- "javac: Java Programming Language Compiler" (page 25)

2 extcheck: JAR Conflict Detection Tool

The extcheck tool detects version conflicts between a target Java Archive (JAR) file and currently installed extension JAR files. For more information on extcheck tool, see the Oracle Java documentation for extcheck.

Synopsis

```
extcheck [ -verbose ] targetfile.jar
```

See Also:

"jar: Java Archive Tool" (page 16)

3 idlj: IDL-to-Java Compiler

The idlj tool generates Java bindings from a specified IDL (Interface Definition Language) file. For more information on idlj tool, see the Oracle Java documentation for idlj.

Synopsis

```
idlj [ options ] idl-file
```

4 jar: Java Archive Tool

The jar tool combines multiple files into a single Java Archive (JAR) file or retrieves files from a JAR file. For more information on jar tool, see the Oracle Java documentation for jar.

Synopsis

Create jar file

```
jar c[v0M]f jarfile [-C dir] inputfiles [-Joption]
jar c[v0]mf manifest jarfile [-C dir] inputfiles [-Joption]
jar c[v0M] [-C dir] inputfiles [-Joption]
jar c[v0]m manifest [-C dir] inputfiles [-Joption]
```

Update jar file

```
jar u[v0M]f jarfile [-C dir] inputfiles [-Joption]
jar u[v0]mf manifest jarfile [-C dir] inputfiles [-Joption]
jar u[v0M] [-C dir] inputfiles [-Joption]
jar u[v0]m manifest [-C dir] inputfiles [-Joption]
```

Extract jar file

```
jar x[v]f jarfile [inputfiles] [-Joption]
jar x[v] [inputfiles] [-Joption]
```

List table of contents of jar file

```
jar t[v]f jarfile [inputfiles] [-Joption]
jar t[v] [inputfiles] [-Joption]
```

Add index to jar file

```
jar i jarfile [-Joption]
```

5 jarsigner: JAR Signing and Verification Tool

The jarsigner tool generates signatures for Java Archive (JAR) files and verifies the signatures of signed JAR files. For more information on jarsigner tool, see the Oracle Java documentation for jarsigner.

Synopsis

For signing:

```
jarsigner [ options ] jar-file alias
```

For verifying:

```
jarsigner -verify [ options ] jar-file
```

See Also:

- "jar: Java Archive Tool" (page 16)
- "keytool: Key and Certificate Management Tool" (page 36)
- The Security trail of the Java Tutorial for examples of the use of the jarsigner tool

6 java: Java Application Launcher

The java tool launches a Java application by starting a Java run time environment, loading a specified class, and invoking that class' main method. For more information on java tool, see the Oracle Java documentation for java.

Synopsis

```
java [ options ] classname [ arguments ]
java [ options ] -jar file.jar [ arguments ]
```

`classname`

Name of the class to be invoked.

`file.jar`

Name of the jar file to be invoked. Used only with -jar.

`arguments`

Arguments passed to the main function.

Deviations from Standard Java Options

`-client`

Selects the Java HotSpot Client virtual machine (VM).

NOTE: The `-client` option is not valid with NonStop Server for Java 7.0.

If `-client` option is used, then the following error message is displayed:

Unrecognized option: *`-client`*

Error: Could not create the Java virtual machine.

Error: A fatal exception has occurred. Program will exit.

–d64

After 64–bit NSJ7 is installed on the system, the following standard Java command is used to launch the 64–bit NSJ7 from 32–bit NSJ7:

```
$ /usr/tandem/nssjava/jdk170_h70/bin/java -d64 <java application>
```

-Xeprof

The –`Xeprof` option generates profile data for HPjmeter. The –`Xeprof` option controls profiling of Java applications running on JRE for NonStop operating system for the Java 2 Platform and collects method clock and CPU times, method call count, and call graph. (For more information on HPjmeter, see www.hp.com/go/hpjmeter.)

NOTE: Zero preparation profiling is started from the command line by sending a signal to the JVM to start eprof. Engaging zero preparation profiling might have a short term impact on application performance as the JVM adjusts to the demands of performing dynamic measurements.

Synopsis

To profile your application, use:

```
java -Xeprof:options ApplicationClassName
```

where

`options`

is a list of `key[=value]` arguments separated by commas.

The following options are used in most of the cases:

- For CPU time metrics with minimal intrusion:

  ```
  -Xeprof
  ```

- Exact call count information and object creation profiling:

  ```
  -Xeprof:inlining=disable
  ```

- To see the complete list of available options, use:

  ```
  java -Xeprof:helps
  ```

After the profiled application terminates execution, the Java Virtual Machine writes the profile data to a file in the current directory. Use HPjmeter to analyze the file.

`-Xeprof` options

Generally, expect a two-fold decrease of your Java application performance when profiling with `-Xeprof`. Depending on your JRE version or platform, dynamically enabling and disabling profiling (see below) might introduce much higher overhead for several minutes. It is advised that the data collection period is no shorter than 30 minutes.

The `-Xeprof` option needs to allocate memory to store the profile data. Therefore, it uses the same memory pool as the JVM. If you specify a very large heap size (generally greater than 1GB), the JVM may run out of address space before it runs out of memory.

off

Turn off `-Xeprof` profiling entirely. This is the default behavior for all SDKs up to 5.0.02. For SDK 5.0.03 and later, if no `-Xeprof` option is specified, it is equivalent to:

```
-Xeprof:time_on=sigusr2,time_slice=sigusr2
```

There is no degradation of performance unless you actually start profiling by the specified signal.

times=quick|thorough

The quick value instructs the profiler to use the hardware Interval Timer register for time measurement. This value results in faster profiling runs, but in extremely rare circumstances, it can produce incorrect data. This is the default value. If you ever suspect that the profile data generated using the quick value is incorrect, rerun the quick command and verify whether the results can be replicated.

The thorough or strict value is the opposite of quick, and disables the use of the Interval Timer. The profiling runs are longer, and provides timing data with the same (relatively poor) quality as the system calls used to measure the time. The profiling intrusion and overhead also increases. This is the default value.

Do not specify this option unless you know what you are doing. The collected profile data will almost certainly have less accuracy than when collected using the quick value.

time_on=<integer>

Specifies the time in seconds between the application start and the time when the profile data collection starts. If no time_on option is present, the data collection begins at the VM initialization.

time_on=sigusr1|sigusr2

Specifies which signal causes profiling to begin (profile data collection). Ensure that the application or the VM may already be using the `sigusr` signals for their own purposes; see the documentation. Specifying a signal and a timeout at the same time is possible by repeating the `time_on` option. Only one of the two signals can be declared to use as the signal to start profiling. During the application's run, the specified signal can be delivered to the Java process multiple times.

time_slice=<integer>

Specifies the time in seconds between the profiling start and the time when profiling will be terminated. When the profiling is terminated, the profile data is written to a file. The application will continue running. If `time_slice` is not specified, or if the application terminates before the specified time elapses but the profiling has started, the profile data is written after the termination of the application.

time_slice=sigusr1 | sigusr2

Specifies which signal causes profiling termination and the profile data output. The signals for profiling start and profiling termination can be the same. Specifying a signal and a timeout at the same time is possible by repeating the `time_slice` option - termination of profiling occurs when the first qualifying event takes place. The application continues running. Only one of the two signals can be declared to use as the signal to terminate profiling. During the application's run, the signal to terminate profiling can be delivered to the Java process multiple times. However, profiling is terminated and a result file generated, if profiling is active when the termination signal is delivered.

file=<filename>

The profile data is written to the named file. If `time_on=sig...` has not been specified, the default is `java<pid> .eprof`, where `<pid>` is the integer number representing the process ID of the Java VM process. If a signal has been specified to start profiling, several data files are created, with names `java<pid>_<t>.eprof`, where `<t>` is the time in seconds between the application start and the profiling start.

inlining=disable | enable

The compiler in the HotSpot VM optimizes Java applications by inlining frequently called methods. Execution of inlined methods is not reported as "calls", since the actual calls have been eliminated. Instead, the time spent in an inlined method is attributed to its "caller".

The default value for this option is enable. The other way of disabling inlining is to collect the profile data while running the VM in interpreted mode `(-Xint)`, although this is usually much slower.

The consequences of inlining for the profiling are:

- The obtained profile data does not reflect faithfully all the calls within the Java code as written by the programmer, but rather as it is actually executed by the VM. For most performance analysis cases, this is a desired feature.
- As the calls within the Java application are eliminated, the corresponding calls to the profiler are eliminated too, resulting in lower profiling overhead.
- The count of created objects cannot be reliably estimated from the call graph in the presence of inlining, because the calls to the constructors may have been inlined.

ie=yes | no

Enables or disables the profiling intrusion estimation.

`ie=yes`, the default value, specifies that the profiler estimates the profiling intrusion and writes the estimated values to the profile data file. HPjmeter uses this data to compensate for intrusion, which means that the estimated intrusion is subtracted from the CPU times before they are presented to the user. Disabling intrusion estimation slightly reduces the size of the data files, but also disables the intrusion compensation feature. This option has no impact on the actual profiling overhead.

–Xverbosegc

The `-Xverbosegc` option prints out a detailed information about the spaces within the Java Heap before and after garbage collection.

The process ID is automatically appended to the `verbosegc` filename you specify. This helps you to associate a `verbosegc` output with the corresponding Java process, especially in cases where an application executes several Java processes.

Synopsis

The syntax of the option is:

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

-Xverbosegc:help for Java 7.0

NOTE: Parallel and Concurrent GC are available in NSJ 7. Therefore, the `-Xverbosegc` options pertaining to them are applicable to NSJ 7.

```
-Xverbosegc options
:help prints this message
0|1 controls the printing of heap information:
0 Print after every Old Generation GC or Full GC
1 (default) Print after every Scavenge and Old Generation GC or Full
GC
:file=[stdout|stderr|<filename>] specifies output file
stderr (default) directs output to standard error stream
stdout directs output to standard output stream
<filename> file to which the output will be written
n – prevents appending pid to gclog filename
h – appends hostname to gclog filename
u – appends username to gclog filename
d – appends date to gclog filename
t – appends time to gclog filename
At every garbage collection, the following 20 fields are printed:
<GC: %1 %2 %3 %4 %5 %6 %7 %8 %9 %10 %11 %12 %13 %14 %15 %16 %17 %18 %19
%20>
%1 Indicates the type of the garbage collection.
1: represents a Scavenge (GC of New Generation only)
%2: indicates if this is a parallel scavenge.
0: non-parallel scavenge
n(>0): parallel scavenge, n represents the number of parallel GC threads
2: represents an Old Generation GC or a Full GC
%2: indicates the GC reason:
1: Allocation failure, followed by a failed scavenge, leading to a Full
GC
2: Call to System.gc
3: Tenured Generation full
4: Permanent Generation full
5: Scavenge followed by a Train collection
6: Concurrent-Mark-Sweep (CMS)eneration full
```

7: Old generation expanded on last scavenge

8: Old generation too full to scavenge

9: FullGCAlot

10: Allocation profiler triggered

11: JVMTI force GC

12: Adaptive Size Policy

13: Last ditch collection

3: represents a complete background CMS GC

- %2: indicates the GC reason:
  - 1: Occupancy > initiatingOccupancy
  - 2: Expanded recently
  - 3: Incremental collection will fail
  - 4: Linear allocation will fail
  - 5: Anticipated promotion
  - 6: Incremental CMS
- m indicates the background CMS state when yielding:
  - 0: Resetting
  - 1: Idling
  - 2: InitialMarking
  - 3: Marking
  - 4: FinalMarking
  - 5: Precleaning
  - 6: Sweeping
  - 7: AbortablePreclean

  (exited after yielding to foreground GC)

%3: Program time at the beginning of the collection, in seconds

%4: Garbage collection invocation. Counts of background CMS GCs and other GCs are maintained separately

%5: Size of the object allocation request that forced the GC, in bytes

%6: Tenuring threshold - determines how long the new born object remains in the New Generation

The report includes the size of each space:

- Occupied before garbage collection (Before)
- Occupied after garbage collection (After)
- Current capacity (Capacity)

All values are in bytes

```
Eden Sub-space (within the New Generation)
```

- `%7: Before`
- `%8: After`
- `%9: Capacity`

```
Survivor Sub-space (within the New Generation)
```

- `%10: Before`
- `%11: After`
- `%12: Capacity`

```
Old Generation
```

- `%13: Before`
- `%14: After`
- `%15: Capacity`

```
Permanent Generation (Storage of Reflective Objects)
```

- `%16: Before`
- `%17: After`
- `%18: Capacity`
- `%19: The total stop-the-world duration, in seconds.`
- `%20: The total time used in collection, in seconds.`

```
Could not create the Java virtual machine.
```

Nonstandard Java Options

The nonstandard options are:

`-Xssn`

Sets the thread stack size.

Every thread spawned while a Java program runs has its own stack. This thread stack is shared by Java program code, any native (JNI) code, and the Java VM code. The default stack size is 512 kilobytes `(-Xss512k)`. You can use this option to increase the stack size if you experience stack overflow exceptions. The default units for `n` are bytes; `n` must be greater than 1000 bytes. To modify the meaning of `n`, append either the letter `k` (or `K`) to indicate kilobytes, or the letter `m` (or `M`) to indicate megabytes. For example, `–Xss10240` and `–Xss10k` are equal.

`-Xincgc`

Specifies using the incremental low-pause garbage collector. This option is supported but using it can lead to about a 10% decrease in garbage collection performance.

Deviations from Nonstandard Java Options

`-Xmsn`

Specifies the initial size, in bytes, of the memory allocation pool. The default value is implementation specific; the value is about 3.6 megabytes.

HP Extensions to Standard Java Options

`-nsjversion`

Prints the NonStop Server for Java 7.0 build version.

HP Extensions to NonStandard Java Options

`-Xabend`

Turns on the abend option to abort the process instead of exiting with a non-zero exit code. If the NonStop Server for Java 7.0 application runs as a Pathway server, you can enable this option to alert Pathmon to restart the server after NSJ7 application shuts down.

`-XX:GuaranteeFreeHeapSizeAfterGC=<number>`

Prevents excessive garbage collection (GC) activity in Java applications that have not been tuned well with respect to GC. If the application uses most of the memory heap and the Java virtual machine is unable to collect much garbage, excessive GC activity might occur to satisfy the demand for allocation of new objects. This activity may result in a busy CPU in which the Java application is executing. To prevent this occurrence, use the `-XX:GuaranteeFreeHeapSizeAfterGC=<number>` option, where `<number>` is specified in bytes, kilobytes (k), or megabytes (m). After a GC, if the Java virtual machine cannot free the heap size more than the value specified with the `GuaranteeFreeHeapSizeAfterGC` option, the Java virtual machine throws an OutOfMemoryError.

`-XX:+ForceStopableMode`

Forces the Java process stop mode to 1 (stoppable) when a Java thread transitions to native state. Under certain rare conditions, the Java Virtual Machine (JVM) can incorrectly be running in unstoppable mode, which can result in a processor halt if the JVM process hits a trap or exception. Normally, the process stop mode is set to unstoppable by privileged code when the code is run, and set back to stoppable mode on exit to the calling routine. While Java itself does not run in privileged mode when executing Java byte codes, Java application code as well as the JVM can make calls to native code or system APIs that run privileged code. If any of these privileged routines do not reset the stop mode on exit, it can result in a processor halt if a trap occurs. Enable this flag to avoid a processor halt in case the stop mode has been set incorrectly by privileged code. Some applications may see performance degradation when this flag is enabled.

See Also:

- "javac: Java Programming Language Compiler" (page 25)
- "jdb: Java Debugger" (page 29)
- "javah: C Header and Stub File Generator" (page 27)
- "jar: Java Archive Tool" (page 16)
- The Java Extensions Framework
- Security Features
- Garbage Collection (GC) in the *NonStop Server for Java 7.0 Programmer's Reference* for more implementation-specific information on options

7 javac: Java Programming Language Compiler

The javac tool compiles Java source code into bytecode. For more information on javac tool, see the Oracle Java documentation for javac.

Synopsis

```
javac [ options ] [ sourcefiles ] [ @argfiles ]
```

Arguments may be in any order.

`options`

Command line options.

`sourcefiles`

One or more source files to be compiled (such as MyClass.java).

`@argfiles`

One or more files that list options and source files. The `-J` options are not allowed in these files.

See Also:

- "java: Java Application Launcher" (page 18)
- "jdb: Java Debugger" (page 29)
- "javah: C Header and Stub File Generator" (page 27)
- "javap: Java Class File Disassembler" (page 28)
- "javadoc: Java API Documentation Generator" (page 26)
- "jar: Java Archive Tool" (page 16)
- The Java Extensions Framework

8 javadoc: Java API Documentation Generator

The javadoc tool generates API documentation in HTML format from Java source code. For more information on javadoc tool, see the Oracle Java documentation for javadoc.

Synopsis

```
javadoc [ options ] { packagenames ] [ sourcefilenames ] [ - subpackages
pkg1:pkg2:... ] [@argfiles ]
```

See Also:

- "javac: Java Programming Language Compiler" (page 25)
- "java: Java Application Launcher" (page 18)
- "jdb: Java Debugger" (page 29)
- "javah: C Header and Stub File Generator" (page 27)
- "javap: Java Class File Disassembler" (page 28)
- Javadoc Home Page
- How to Write Doc Comments for Javadoc

9 javah: C Header and Stub File Generator

The javah tool generates C header files and stub C source files from a Java class. These files provide the connections that allow your Java code and C code to interact. For more information on javah tool, see the Oracle Java documentation for javah.

Synopsis

For files that are needed to implement native methods:

```
javah [ options ] fully-qualified-classname ...
```

See Also:

- “javac: Java Programming Language Compiler” (page 25)
- “java: Java Application Launcher” (page 18)
- “jdb: Java Debugger” (page 29)
- “javadoc: Java API Documentation Generator” (page 26)

10 javap: Java Class File Disassembler

The javap tool disassembles compiled Java files. For more information on javap tool, see the Oracle Java documentation for javap.

Synopsis

```
javap [ options ] class ...
```

See Also:

- “javac: Java Programming Language Compiler” (page 25)
- “rmic: Java RMI Compiler” (page 39)
- “java: Java Application Launcher” (page 18)
- “jdb: Java Debugger” (page 29)
- “javah: C Header and Stub File Generator” (page 27)
- “javadoc: Java API Documentation Generator” (page 26)

11 jdb: Java Debugger

The jdb tool helps you to find and fix errors in Java programs. For more information on jdb tool, see the Oracle Java documentation for jdb.

Synopsis

```
jdb [ options ] [ class ] [ arguments ]
```

`options`

See "Command Line Options" (page 31).

`class`

Name of the class to begin debugging.

`arguments`

Arguments passed to the main() method of class.

Description

The Java Debugger, `jdb`, is a simple command line debugger for Java classes. It is an example of the use of the Java Platform Debugger Architecture that provides inspection and debugging of a local or remote Java virtual machine (VM).

Starting a `jdb` Session

There are many ways to start a `jdb` session. The most frequent way is to have jdb launch a new Java VM with the main class of application to be debugged. Perform this by substituting the command `jdb` for the command `java` in the command line. For example, if your application's main class is named `MyClass`, you use the following command to debug it under JDB:

```
jdb MyClass
```

When started this way, jdb invokes a second Java VM with any specified parameters, loads the specified class, and stops the Java VM before executing the first instruction of that class.

Another way to use `jdb` is by attaching it to a Java VM that is already running. A Java VM that is to be debugged with `jdb` must be started with the following `java` options:

Option	Purpose
`-Xdebug`	Enables debugging support in the Java VM.
`-agentlib:jdwp=transport=dt_socket, server=y,suspend=n`	Loads in-process debugging libraries and specifies the kind of connection to be made.

For example, the following command runs the `MyClass` application and allows `jdb` to connect to the application at a later time.

```
java -Xdebug –agentlib:jdwp=transport=dt_socket,\
address=8000,server=y,suspend=n MyClass
```

You can then attach `jdb` to the Java VM with the following command:

```
jdb -attach 8000
```

NOTE: `MyClass` is not specified in the `jdb` command line in this case because `jdb` connects to an existing Java VM instead of launching a new one.

There are many other ways to connect the debugger to a Java VM, and all of them are supported by jdb, as specified in "Connecting for Remote Debugging" (page 33).

Basic `jdb` Commands

The following is a list of the basic `jdb` commands. The Java debugger supports other commands, which you can list by using the `jdb help` command.

`{help | ?}`

Displays the list of recognized commands with a brief description.

`run`

After starting jdb and setting any necessary breakpoints, you can use this command to start the execution of the debugged application. This command is available only when jdb launches the debugged application (as opposed to attaching to an existing Java VM).

`cont`

Continues execution of the debugged application after a breakpoint, exception, or step.

`print`

Displays Java objects and primitive values. For variables or fields of primitive types, the actual value is printed. For objects, a short description is printed. See the dump command below for getting more information about an object.

NOTE: To display local variables, the containing class must have been compiled with the `javac -g` option.

`print` supports many simple Java expressions including those with method invocations, for example:

- `print MyClass.myStaticField`
- `print myObj.myInstanceField`
- `print i + j + k (i, j, k are primitives and either fields or local variables)`
- `print myObj.myMethod() (to print the value if myMethod() returns a non-null)`
- `print new java.lang.String("Hello").length()`

`dump`

For primitive values, this command is identical to print. For objects, it prints the current value of each field defined in the object. Static and instance fields are included.

The dump command supports the same set of expressions as the print command.

`thread`

List the threads that are currently running. For each thread, its name and current status are printed, as well as an index that can be used for other commands, for example:

`(java.lang.Thread)0x1 main running`

In this example, the thread index is 4, the thread is an instance of `java.lang.Thread`, the thread name is main, and it is currently running.

`thread`

Select a thread to be the current thread. Many jdb commands are based on the setting of the current thread. The thread is specified with the thread index described in the threads command.

`where`

`where` with no arguments dumps the stack of the current thread. `where all` dumps the stack of all threads in the current thread group. `where` *`threadindex`* dumps the stack of the specified thread.

If the current thread is suspended (either through an event such as a breakpoint or through the `suspend` command), local variables and fields can be displayed with the`print` and`dump` commands. The up and down commands select which stack frame is current.

Breakpoints

Breakpoints can be set in jdb at line numbers or at the first instruction of a method, for example:

- stop at `MyClass:22` (sets a breakpoint at the first instruction for line 22 of the source file containing MyClass)
- stop in `java.lang.String.length` (sets a breakpoint at the beginning of the method `java.lang.String.length`)
- stop in `MyClass.init` (init identifies the MyClass constructor)
- stop in `MyClass.clinit`(clinit identifies the static initialization code for MyClass)

If a method is overloaded, you must also specify its argument types so that the proper method can be selected for a breakpoint. For example, `MyClass.myMethod(int,java.lang.String)`, or `MyClass.myMethod()`.

The clear command removes breakpoints by using a syntax as in clear `MyClass:45`. Using the clear command with no argument displays a list of all breakpoints currently set. The `cont` command continues execution.

Stepping

The `step` command advances execution to the next line whether it is in the current stack frame or a called method. The `next` command advances execution to the next line in the current stack frame.

Exceptions

When an exception occurs for which there is not a `catch` statement anywhere in the throwing thread's call stack, the Java VM normally prints an exception trace and exits. When running under `jdb`, however, control returns to `jdb` at the offending throw. You can then use `jdb` to diagnose the cause of the exception.

Use the `catch` command to cause the debugged application to stop at other thrown exceptions, for example: `catch java.io.FileNotFoundException` or `catch mypackage.BigTroubleException`. Any exception that is an instance of the specified class (or of a subclass) stops the application at the point where it is thrown.

The `ignore` command negates the effect of a previous catch command.

NOTE: The `ignore` does not cause the debugged VM to ignore specific exceptions, only the debugger.

Command Line Options

When you use `jdb` in place of the Java application launcher on the command line, `jdb` accepts many of the same options as the "java: Java Application Launcher" (page 18) command, including `-D`, `-classpath`, and `-Xoption`.

The following additional options are accepted by jdb:

`-help`

Displays a help message.

`-sourcepath directory1 [:directory2]...`

Uses the given path in searching for source files in the specified path. If this option is not specified, the default path of "." is used.

`-attach address`

Attaches the debugger to the previously running Java VM by using the default connection mechanism.

`-listen address`

Waits for a running VM to connect to the specified address through a standard connector.

`-listenany`

Waits for a running VM to connect to any available address through a standard connector.

`-launch`

Launches the debugged application immediately upon startup of jdb. This option removes the need for using the `run` command. The debugged application is launched and then stopped just before the initial application class is loaded. At that point you can set any necessary breakpoints and use the `cont` to continue execution.

`-connect connector-name:name1=value1,...`

Connects to the target VM through a named connector that uses the listed argument values.

`-dbgtrace [flags]`

Prints information for debugging jdb.

`-Joption`

Pass `option` to the Java virtual machine, where `option` is one of the options described on the reference page for the java application launcher. For example, `-J-Xms48m` sets the startup memory to 48 megabytes.

Other `options` are supported for alternate mechanisms for connecting the debugger and the Java VM it is to debug. The Java Platform Debugger Architecture has additional documentation on these connection alternatives.

Deviations from Standard Java

`-tclient`

Runs the application in the Java HotSpot client VM.

NOTE: The `-tclient` option is not valid with NonStop Server for Java 7.0.

`-tserv`

Runs the application in the Java HotSpot server VM.

NOTE: `-tserv` is the default option for NonStop Server for Java 7.0; therefore, specifying `-tserv` is optional.

Options Forwarded to the Process Being Debugged

`-v -verbose[:class|gc|nji]`

Turns on verbose mode.

`-D name=value`

Sets a system property.

`-classpath directory1 [:directory2]...`

Lists directories in which to look for classes.

`-X option`

Sets a nonstandard target VM option.

Connecting for Remote Debugging

1. The Debugger launches the target Java VM.

   ```
   -launch
   jdb -launch ClassName
   ```

2. The Debugger attaches to a previously running Java VM.

   ```
   -attach
   jdb -attach hostname:portnum
   ```

   For this command, the JVM must already be running as a server at `[<hostname>:]<portnum>|<start port>-<end port>`

   To start the server, use the following command :

   ```
   java -Xnoagent -Xdebug -Djava.complier=NONE \
   -agentlib:jdwp=transport=dt_socket,\
   address=[<hostname>:]<portnum>|<start port>-<end port>,server=y \
   ClassName
   If address option is not given, the server will start on any
   available port on the local host and print portnum. This portnum
   should be used by the jdb to attach.
   ```

   NOTE: In NonStop, there is an additional option to specify the port range, where, <start port> and <end port> are the starting and ending port numbers for a range of ports.

3. The target JVM attaches to previously running debugger.

   ```
   -listen
   jdb -listen hostname:portnum
   ```

   To attach a target JVM, use the following command:

   ```
   java -Xnoagent -Xdebug -Djava.complier=NONE \
   -agentlib:jdwp=transport=dt_socket, address=hostname:portnum \
   ClassName
   ```

4. The Debugger selects a connector.

   ```
   -connect
   jdb -connect option
   ```

   NOTE: Only the `com.sun.jdi.SocketListen` option is supported.

   The target Java VM can then attach as:

   ```
   java -Xnoagent -Xdebug -Djava.compiler=NONE \
   -agentlib:jdwp=transport=dt_socket, address=hostname:portnum \
   ClassName
   ```

Transports

A Java Platform Debugger Architecture (JPDA) transport is a form of inter-process communication used by a debugger application and the debuggee. NonStop Server for Java 7.0 provides a socket transport that uses the standard TCP/IP sockets to communicate between debugger and the debuggee.

NonStop Server for Java 7.0 defaults to socket transport. NonStop Server for Java 7.0 does not support shared memory transport.

See Also:

"javac: Java Programming Language Compiler" (page 25)
"java: Java Application Launcher" (page 18)
"javah: C Header and Stub File Generator" (page 27)
"javap: Java Class File Disassembler" (page 28)
"javadoc: Java API Documentation Generator" (page 26)

12 jrunscript: Command Line Script Shell

`jrunscript` is a command line script shell that supports an interactive (read-eval-print) mode and a batch (`-f` option) mode of script execution. This is a scripting language-independent shell. By default, JavaScript is the language used, but the `-l` option can be used to specify a different language. Through Java to scripting language communication,`jrunscript` supports "exploratory programming" style. For more information on jrunscript tool, see the Oracle Java documentation for jrunscript.

Synopsis

```
jrunscript [ options ] [ arguments... ]
```

13 keytool: Key and Certificate Management Tool

The `keytool` tool manages a keystore (database) of private keys and their associated X.509 certificate chains authenticating the corresponding public keys. For more information keytool on tool, see the Oracle Java documentation for keytool.

Synopsis

```
keytool [ commands ]
```

See Also:

- "jar: Java Archive Tool" (page 16)
- "jarsigner: JAR Signing and Verification Tool" (page 17)
- The Security trail of the Java Tutorial for examples of the use of keytool

14 native2ascii: Native-to-ASCII Converter

The `native2ascii` tool converts a file that has native-encoded characters (characters that are not Latin-1 and not Unicode) to a file with Unicode-encoded characters. For more information on `native2ascii` tool, see the Oracle Java documentation for native2ascii.

Synopsis

```
native2ascii [ options ] [ inputfile [ outputfile ] ]
```

15 orbd: Object Request Broker Daemon

The server manager included with the orbd tool enables clients to transparently locate and invoke persistent objects on servers in the CORBA environment. For more information on orbd tool, see the Oracle Java documentation for orbd.

Synopsis

```
orbd -ORBInitialPort nameserverport [ options ]
```

See Also:

- Naming Service
- servertool

16 rmic: Java RMI Compiler

The `rmic` tool generates stubs and skeletons for remote objects that use either the JRMP or Internet Inter-ORB Protocol (IIOP). The `rmic` tool also generates Object Management Group (OMG) Interface Definition Language (IDL). For more information on `rmic` tool, see the Oracle Java documentation for rmic.

Synopsis

```
rmic [ options ] package-qualified-classname ...
```

See Also:

- "java: Java Application Launcher" (page 18)
- "javac: Java Programming Language Compiler" (page 25)
- CLASSPATH in the *NonStop Server for Java 7.0 Programmer's Reference*

17 rmid: Java RMI Activation System Daemon

The rmid tool starts the activation system daemon that allows objects to be registered and activated in a Java virtual machine (VM). For more information on rmid tool, see the Oracle Java documentation for rmid.

Synopsis

```
rmid [ options ]
```

See Also:

- "java: Java Application Launcher" (page 18)
- CLASSPATH in the *NonStop Server for Java 7.0 Programmer's Reference*
- "rmic: Java RMI Compiler" (page 39)

18 rmiregistry: Java Remote Object Registry

The `rmiregistry` tool starts a remote object registry on the specified port on the current host. For more information on `rmiregistry` tool, see the Oracle Java documentation for rmiregistry.

Synopsis

```
rmiregistry [ port ]
```

See Also:

- "java: Java Application Launcher" (page 18)
- java.rmi.registry.LocateRegistry
- java.rmi.Naming

19 schemagen: Java Architecture for XML Binding Schema Generator

The `schemagen` tool creates a schema file for each namespace referenced in your Java class generator for Java Architecture for XML Binding. For more information on schemagen tool, see the Oracle Java documentation for schemagen.

Synopsis

```
schemagen [-options ...] <java files>
```

The following table lists the `schemagen` options.

Option	Description
`-d <path>`	Specifies where to place the processor and javac generated class files.
`-cp <path>`	Specifies where to find user-specified files.
`-classpath <path>`	Specifies where to find user-specified files.
`-help`	Displays this usage message.

20 serialver: Serial Version Command

The serialver tool returns the `serialVersionUID` of one or more classes. For more information on serialver command, see the Oracle Java documentation for serialver.

Synopsis

```
serialver [ option ] [ classname ... ]
```

See Also:

- java.io.ObjectStreamClass

21 servertool: Java IDL Server Tool

The `servertool` tool provides a command line interface for application programmers to register, unregister, start up, and shut down a persistent server. For more information in `servertool` tool, see the Oracle Java documentation for servertool.

Synopsis

```
servertool -ORBInitialPort nameserverport options [ commands ]
```

See Also:

- "orbd: Object Request Broker Daemon" (page 38)

22 tnameserv: Naming Service Access

The `tnameserv` tool starts the Java Interface Definition Language (IDL) name server to provide access to the CORBA Common Object Services (COS) Naming Service. For more information on `tnameserv` tool, see the Oracle Java documentation for Naming Service.

Synopsis

```
tnameserv [ -ORBInitialPort n ]
```

23 nsjps: NonStop Java Virtual Machine Process Status Tool

The nsjps tool is a process status tool that lists and monitors the Java processes running on a system. The input and output options of nsjps tool are enhanced in the 64–bit version of NSJ7.

NOTE: The nsjps tool can list or monitor NSJ7 Java applications. To list or monitor previous Java applications, use previous version of nsjps tool.

Synopsis

```
nsjps [ <options> ]
```

The following options are supported:

Option	Purpose
`-gc`	Lists the Java and GC processes separately.
`-gcpid`	Lists the Java process corresponding to the GC process.
`-cpu <cpuNumber>`	Lists the Java processes running on the specified CPU.
`-user <userId|userName>`	Lists the Java processes owned by specified user.
`-heap { < | = | > } <size {K|M|G|k|m|g} >`	Lists the Java processes with matching heap.
`-parent <pid>`	Lists the children of the specified Java process.
`-h`	Displays the current heap size.
`-l`	Displays Java or GC processes in `long` format.
`-v`	Includes argument in the display.
`-p`	Includes the Guardian pin in the output.
`-u`	Includes the owner in the output.
`-t`	Displays the process and elapsed time.
`-x`	Provides cross reference of parent and child Java processes.
`-help`	Prints the `nsjps` help text.
`-count <cnt>`	Repeats the listing of Java or GC process for the specified number of times.
`-delay <time>`	Specifies the time to sleep before the next sample.

Input Options

The input option is used to invoke the nsjps tool with various options to select the Java process based on the selection criteria.

`-gc`

In NSJ7, parallel and concurrent mark sweep GCs (CMS GC) are enabled for J-series systems. To utilize the multi-core capabilities of Integrity platform, the parallel and CMS GCs are run as a separate process in the same processor where the Java process is running. Because of this, in NSJ7, a Java application consists of a Java process, and one or more GC processes (if parallel or CMS GC is enabled).

By default, nsjps lists only the Java processes. NSJ7 provides the `-gc` input option to obtain details about GC processes. If `-gc` option is specified, GC process details are printed with the corresponding Java process. If `-gc` is not specified, the heap

size and process time are consolidated values for the Java and corresponding GC processes.

NOTE: This option is applicable only for J-series systems.

The following example shows the usage and sample output for `-gc` option:

```
nsjps –gc

PID       Cmd
45676     java
67894     javagc
```

-gcpid <pid>

To obtain details about the Java process corresponding to a GC process, `-gcpid <pid>` is provided, where, `<pid>` refers to the `<pid>` of the GC process.

NOTE: This option is applicable only for J-series systems.

The following example shows the usage and sample output for `-gcpid` option:

```
nsjps –gcpid 67894

PID      Cmd
45676    java
```

-cpu <cpuNumber>

This option lists the Java process running on the CPU specified by the `cpuNumber`. The following example shows the usage and output for `-cpu` option:

```
nsjps –cpu 1

PID         Cmd
50331674    java
```

-user <UserID | username>

This option lists the Java process run by the user specified by `userID` or `userName`. The following example shows the usage and output for `-user` option:

```
nsjps -user nsjava.guest

PID         Cmd
50331674    java
```

Specifying Heap Size

The `-heap` option lists the Java processes. This option accepts heap sizes in kilobyte and megabyte by suffixing the value with `K` or `M`(the suffix character is not case sensitive). In a 64–bit JVM, the heap sizes are large, therefore, the heap size can be suffixed with `G` (indicates that value is in Gigabyte). The following example shows the usage and output for `-heap` option:

```
nsjps –heap \> 1G

PID         Cmd
79289      java
45678      java
```

-parent <pid>

This option lists the child process of the given process ID of the Java process. The following example shows the usage and output for `-parent` option:

```
nsjps -parent 16777243

PID           Cmd
754974740     java
```

Cross Reference Option

NSJPS provides `-x` option that prints the parent-child Java process relationships. This helps to identify the child Java processes started by a particular parent process. In the cross reference listing, only the Java processes are considered and not the GC processes. The following example shows the usage and output for `-x` option.

```
nsjps –x

PID            ParentPID       Cmd
743899154      911671351       java
89587743       1062666314      java

Child processes for parent 911671351
743899154

Child processes for parent 1062666314
89587743
```

Output Options

The output options are used to print details such as, PIN, User ID, and process ID on the selected Java processes.

Heap Sizes

NSJPS prints the heap size only in bytes. Therefore, even if the heap size is large, for example 24 GB, NSJPS displays it as 25769803776 bytes. Note that with NSJ7, the heap space required for JVM and Java heap are allocated from flat segments and the size output of a Java or GC process includes the size of these segments in addition to the native C-heap. The following example shows the usage and output for `-h` option:

```
/home/daya/ACC/nssjava/jdk170_h70/bin/nsjps -h
PID                 Cmd    Heap
16777243            java   377126912
```

`-h` option

If `–h` option is used with `–gc` option, then the heap size for each process in the Java process group is listed separately. The following example shows the usage and output for `-h` option:

```
nsjps –h –gc
PID                 Cmd      Heap
251658247           java     1963311104
67108872            javagc   278528
33554441            javagc   278528
16777226            javagc   278528
```

`-l` option

This option lists the Java or GC process command in long format. The following example shows the usage and output for `-l` option:

```
nsjps –l
PID                  Full Path
1040187405           /usr/tandem/nssjava/jdk170_h70/bin/java
```

-v option

This option lists the argument passed to the Java process. The following example shows the usage and output for -v option:

```
nsjps –v
PID                 Cmd     arguments
989855920           java    -XX:+UseParallelGC HelloWorld
```

-p option

This option lists the Guardian pin of the Java or GC process. The following example shows the usage and output for -p option:

```
nsjps –p
PID                 Cmd     PIN
989855920           java    1,1173
922747051           java    1,390
```

-u option

This option lists the Java process and the owner of this Java or GC process. The following example shows the usage and output for -u option:

```
nsjps –u
PID                 Cmd     User
989855920           java    SUPER.SUPER
922747051           javagc  SUPER.SUPER
```

-t option

This option displays the process and elapsed time of a Java Process. If –gc option is specified then the time specified is separate for Java and GC process. Otherwise, the time is a consolidated output time of Java and GC process. The following example shows the usage and output for -t option:

```
nsjps –t
PID                 Cmd     CpuTime         ElapseTime
1895825444          java    0:00:23.346     0:33:10.325

nsjps –t –gc
PID                 Cmd     CpuTime         ElapseTime
989855920           java    0:26:28.543     2:01:17.923
922747051           javagc  0:00:01.085     2:01:05.844
1090519214          javagc  0:00:00.799     2:01:05.594
```

-count <cnt> option

This option repeats the listing of Java process specified by number in <cnt>. The following example shows the usage and output for -count option:

```
nsjps -count 2 -delay 1

PID                  Cmd
1895825444           java
1040187405           java

1895825444           java
1040187405           java
```

-delay <time> option

This option lists the Java process listing after specified delay <time>. The following example shows the usage and output for -delay option:

```
nsjps -count 2 -delay 1
```

```
PID                 Cmd
1895825444          java
1040187405          java

1895825444          java
1040187405          java
```

Help Command Output

The help command reflects the command line options and they are listed in “Synopsis” (page 46).

24 wsgen : Java API for XML Web Services (JAX-WS) 2.0

The wsgen tool generates the JAX-WS portable artifacts used in JAX-WS web services. The tool reads a web service endpoint implementation class (SEI) and generates all the required artifacts for web service deployment, and invocation. For more information on wsgen tool, see the Oracle Java documentation for wsgen.

Synopsis

```
wsgen.sh <options> <SEI>
```

The following table lists the `wsgen` options.

Option	Description
`-classpath <path>`	Specifies where to find input class files.
`-cp <path>`	Specifies where to find input class files.
`-d <directory>`	Specifies where to place generated output files.
`-extension`	Allows vendor extensions (functionality not specified by the specification). Use of extensions may result in applications that are not portable or may not interoperate with other implementations.
`-help`	Displays help.
`-keep`	Keeps generated files.
`-r <directory>`	Used only in conjunction with the `-wsdl` option. Specifies where to place generated resource files such as WSDLs.
`-s <directory>`	Specifies where to place generated source files.
`-verbose`	Displays output messages about what the compiler is doing.
`-version`	Prints version information. Use of this option will ONLY print version information. Normal processing will not occur.
`-wsdl[:protocol]`	By default, `wsgen` does not generate a `WSDL` file. This flag is optional and will cause `wsgen` to generate a WSDL file and is usually used so that the user can look at the WSDL before the endpoint is deployed. The `protocol` is optional and is used to specify what protocol should be used in the `wsdl:binding`. Valid protocols include: soap1.1 and Xsoap1.2. The default is soap1.1. Xsoap1.2 is not standard and can only be used in conjunction with the `-extension` option.
`-servicename <name>`	Used only in conjunction with the `-wsdl` option. Specifies a particular `wsdl:service` name to be generated in the WSDL. For example, `-servicename "{http://mynamespace/}MyService"`.
`-portname <name>`	Used only in conjunction with the `-wsdl` option. Specifies a particular `wsdl:port` name to be generated in the WSDL. For example, `-portname "{http://mynamespace/}MyPort"`.

25 wsimport: JAX-WS 2.0

The wsimport tool generates JAX-WS portable artifacts, such as:

- Service Endpoint Interface (SEI)
- Service
- Exception class mapped from wsdl:fault (if any)
- Async Reponse Bean derived from response wsdl:message (if any)
- JAXB generated value types (mapped Java classes from schema types)

For more information on wsimport tool, see the Oracle Java documentation for wsimport.

Synopsis

```
wsimport [options] <wsdl>
```

26 xjc: Java Architecture for XML Binding Compiler

The `xjc` tool is the XML binding compiler. The binding compiler can be launched using the appropriate `xjc` shell script in the `bin` directory for your platform. For more information on xjc tool, see the Oracle Java documentation for xjc.

Synopsis

```
xjc [-options ...] <schema_file/URL/dir> ... [-b <bindinfo>] ...
```

The following table lists the `xjc` options.

Option	Description
`-nv`	Does not perform strict validation of the input schema(s).
`-extension`	Allows vendor extensions—does not strictly follow the Compatibility Rules and App E.2 from the JAXB Spec.
`-b <file/dir>`	Specifies external bindings files (each `<file>` must have its own `-b`). If a directory is given, `**/*.xjb` is searched.
`-d <dir>`	Stores the generated files.
`-p <pkg>`	Specifies the target package.
`-httpproxy <proxy>`	Sets HTTP/HTTPS proxy. Format is `[user[:password]@]proxyHost:proxyPort`.
`-httpproxyfile <file>`	Sets the proxy string. Format is `[user[:password]@]proxyHost:proxyPort`.
`-classpath <arg>`	Specifies where to find user class files.
`-catalog <file>`	Specifies catalog files to resolve external entity references support TR9401, XCatalog, and OASIS XML Catalog format.
`-readOnly`	Enables the XJC binding compiler to mark the generated files read-only.
`-npa`	Suppresses generation of package-level annotations (`**/package-info.java`).
`-no-header`	Suppresses generation of a file header with timestamp.
`-target 2.0`	Behaves like XJC 2.0 and generates code that does not use any 2.1 features.
`-xmlschema`	Treats input as W3C XML Schema (default).
`-relaxng`	Treats input as RELAX NG (experimental, unsupported).
`-relaxng-compact`	Treats input as RELAX NG compact syntax (experimental, unsupported).
`-dtd`	Treats input as XML DTD (experimental, unsupported).
`-wsdl`	Treats input as WSDL and compile schemas inside it (experimental, unsupported).
`-verbose`	Be extra verbose, such as printing informational messages or displaying stack traces upon some errors.
`-quiet`	Suppresses compiler output.
`-help`	Displays this help message.
`-version`	Displays version information.