Tools Home Page

NonStop Server for Java Tools Reference Pages

Abstract

This document consists of this title page, a table of contents, and the Tools Reference Pages for the HP NonStop™ Server for Java™, based on Java 2 Platform, Standard Edition, SDK 1.4.2.

Product Version

For TNS/R, HP NonStop Server for Java 4.2, based on Java 2 Platform, Standard Edition 4.2
For TNS/E, HP NonStop Server for Java 4.2, based on Java 2 Platform, Standard Edition 4.2

Supported Hardware

All HP Integrity NonStop NS-series (TNS/E) servers and all HP NonStop S-series (TNS/R) servers except S700 and S70000.

Supported Release Version Updates (RVUs)

This manual supports J06.06 and all subsequent J-series RVUs, H06.06 and all subsequent H-series RVUs, and G06.20 and all subsequent G-series RVUs, until otherwise indicated by its replacement publications.

Part Number	Published
529775-002	August 2010

Document History

Part Number	Product Version	Published
425948-001	NonStop Server for Java 2.0	January 2001
425948-002	NonStop Server for Java 2.1	March 2002
425948-003	NonStop Server for Java 3.0	June 2002

425948-004	NonStop Server for Java 3.1	September 2002
425948-005	NonStop Server for Java 3.1 Update 1	December 2002
526239-001	NonStop Server for Java based on Java 2 Platform, Standard Edition, SDK 1.4.1) 1.0	October 2003
526239-002	NonStop Server for Java based on Java 2 Platform, Standard Edition, SDK 1.4.2), Version 2.0	December 2004
529775-001	For TNS/R, HP NonStop Server for Java, based on Java 2 Platform, Standard Edition, SDK 1.4.2, Version 2.0 For TNS/E, HP NonStop Server for Java, based on Java 2 Platform, Standard Edition, SDK 1.4.2, Version 1.0	May 2005
529775-002	For TNS/R, HP NonStop Server for Java 4.2, based on Java 2 Platform, Standard Edition 4.2 For TNS/E, HP NonStop Server for Java 4.2, based on Java 2 Platform, Standard Edition 4.2	August 2010

Ordering Information

For manual ordering information: domestic U.S. customers, call 1-800-243-6886; international customers, contact your local sales representative.

Legal Notices

Hewlett-Packard Development Company L.P.
10600 Ridgeview Court
Cupertino, CA 95014 USA

Tools Home Page

NonStop Server for Java Tools Reference Pages

Command	Tool Name	Function
extcheck	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	Generates Java bindings from a specified IDL file.
jar	Java Archive Tool	Combines multiple files into a single JAR file and retrieves files from a JAR file.
jarsigner	JAR Signing and Verification Tool	Generates signatures for JAR files and verifies signatures of signed JAR files.
java	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	Compiles Java source code into bytecode.
javadoc	Java API Documentation Generator	Generates API documentation in HTML or MIF format from Java source code.
javah	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	Disassembles compiled Java files.
jdb	Java Debugger	Helps you find and fix bugs in Java programs.
keytool	Key and Certificate Management Tool	Manages a database of private keys and their associated certificate chains authenticating the corresponding public keys.
kinit	Kerberos Tool	Obtains and caches Kerberos ticket-granting tickets.

`klist`	Kerberos Tool	Allows the viewing of entries in the local credentials cache and key table.
`ktab`	Kerberos Key Table Manager	Allows for managing the principal names and service keys stored in a local key table.
`native2ascii`	Native-to-ASCII Converter	Converts a file with native-encoded characters to one with Unicode-encoded characters.
`nsjps`	NonStop Java Virtual Machine Process Status Tool	Lists and monitors the Java processes running on a system.
`orbd`	Object Request Broker Daemon	Enables clients to transparently locate and invoke persistent objects on servers in the CORBA environment.
`rmic`	Java RMI Compiler	Generates stubs and skeletons for remote objects that use either the JRMP or IIOP. Also, generates OMG Interface Definition Language (IDL).
`rmid`	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	Starts a remote object registry on the specified port on the current host.
`serialver`	Serial Version Command	Returns the `serialVersionUID` of one or more classes.
`servertool`	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	Provides access to the Naming Service.

Title Page

`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 a complete description of the tool and its use, see the Sun Microsystems documentation for `extcheck`.

Synopsis

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

See Also:

`jar`

`idlj`: IDL-to-Java Compiler

The `idlj` tool generates Java bindings from a specified IDL (Interface Definition Language) file. For a complete description of the tool and its use, see the Sun Microsystems documentation for `idlj`.

Synopsis

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

`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 a complete description of the tool and its use, see the Sun Microsystems documentation for `jar`.

Synopsis

```
jar [ options ] [ jar-file ] [ manifest-file ] [ file |
directory ] ...
```

`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 a complete description of the tool and its use, see the Sun Microsystems documentation for `jarsigner`.

Synopsis

For signing:

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

For verifying:

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

See Also:

- `jar`
- `keytool`
- the Security trail of the Java Tutorial for examples of the use of the `jarsigner` tool

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 a complete description of the tool and its use, see the Sun Microsystems documentation for `java`. Note that you can use the `-Xrunhprof` option, which allows you to obtain profile information about a running program. Through the use of a third-party tools and the Hewlett-Packard HPjmeter tool, you can view this profile information graphically.

Synopsis

```
java [ options ] classname [ arguments ]
```

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

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 4.

`-server`

Selects the Java HotSpot Server virtual machine (VM).

Note: `-server` the default option for NonStop Server for Java 4; therefore, specifying `-server` is optional.

`-d32`
`-d64`

Selects whether the program is to be run in a 32-bit or 64-bit environment.

Note: These options are not valid with NonStop Server for Java 4, which uses only the 32-bit environment to run Java programs.

Nonstandard Java Options

The following nonstandard options are shown for your convenience.

`-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 either 128 kilobytes (`-Xss128k`) for TNS/R or 512 kilobytes (`-Xss512k`) for TNS/E. 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.

Deviations from Nonstandard -XX Java Options

`-XX:+UseParallelGC`

Specifies a parallel garbage collector. This option is disabled. If you specify this option, the Java VM exits with the error: `-XX:+UseParallelGC` option is not supported on this platform.

`-XX:+UseParNewGC`

Specifies a parallel garbage collection. This option is disabled. If you specify this option, the Java VM exits with the error: `-XX:+UseParNewGC` option is not supported on this platform.

`-XX:+UseAdaptiveSizePolicy`

Specifies an adaptive size policy. This option applies only for a parallel collector and, therefore, is disabled. If you specify this option, the Java VM exits with the error: `-XX:+UseAdaptiveSizePolicy` option is not supported on this platform.

`-XX:+AggressiveHeap`
: Obtains the platform resources and configures the heap layout accordingly, uses parallel collector, and enables `AdaptiveSizePolicy` option. This option applies only for a parallel collector and, therefore, is disabled. If you specify this option, the Java VM exits with the error: `-XX:+AggressiveHeap` option is not supported on this platform.

`-XX:+UseConcMarkSweepGC`
: Specifies using the concurrent mark-sweep garbage collector. This option is disabled. If you specify this option, the Java VM exits with the error: `-XX:+UseConcMarkSweepGC` option is not supported on this platform.

`-Xconcgc`
: Enables a concurrent mark-sweep garbage collector. This option is disabled. If you specify this option, the Java VM exits with the error: `-Xconcgc` option is not supported on this platform.

HP Extensions to Standard Java Options

`-nsjversion`
: Prints the NonStop Server for Java 4 build version.

`-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 4 application is run as a Pathway server, you can enable this option to alert Pathmon to restart the server after the NonStop Server for Java 4 application dies.

See Also:

- `javac`
- `jdb`
- `javah`
- `jar`
- The Java Extensions Framework
- Security Features
- Garbage Collection (GC) in the NonStop Server for Java Programmer's Reference for more implementation-specific information on options

---

---

javac: Java Programming Language Compiler

The javac tool compiles Java source code into bytecode. For a complete description of the tool and its use, see the Sun Microsystems 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:

- rmic
- java
- jdb
- javah
- javap
- javadoc
- The Java Extensions Framework

javadoc: Java API Documentation Generator

The `javadoc` tool generates API documentation in HTML format from Java source code. For a complete description of the tool and its use, see the Sun Microsystems documentation for `javadoc`.

Synopsis

```
javadoc [ options ] { package | class.java } ...
```

See Also:

- `javac`
- `rmic`
- `java`
- `jdb`
- `javah`
- `javap`

NonStop Server for Java Tools Reference Pages (529775-002)

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 a complete description of the tool and its use, see the Sun Microsystems documentation for javah.

Synopsis

For files that are needed to implement native methods:

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

Deviations from Standard Java Options

Note: The non-optimized version of javah, javah_g, suitable for use with debuggers is not supported by NonStop Server for Java 4.

See Also:

- javac
- java
- jdb
- javap
- javadoc

javap: Java Class File Disassembler

The `javap` tool disassembles compiled Java files. For a complete description of the tool and its use, see the Sun Microsystems documentation for `javap`.

Synopsis

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

See Also:

- `javac`
- `rmic`
- `java`
- `jdb`
- `javah`
- `javadoc`

jdb: Java Debugger

The `jdb` tool helps you find and fix errors in Java programs. For a complete description of the tool and its use, see the Sun Microsystems documentation for `jdb`.

Synopsis

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

`options`
: See Command-Line Options.

`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. You do 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.
`-Xrunjdwp: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 -Xrunjdwp: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.

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()` (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.

`threads`

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:

```
4. (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 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`.

`-J`*`option`*

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 4.

`-tserv`

Runs the application in the Java HotSpot server VM.

Note: `-tserv` is the default option for NonStop Server for Java 4; 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

i. **The Debugger launches the target Java VM.**

`-launch`

```
jdb -launch ClassName
```

ii. **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>`

To start the server, use the following command :

```
java -Xnoagent -Xdebug -Djava.complier=NONE \
-Xrunjdwp:transport=dt_socket,\
address=hostname:portnum,server=y ClassName
```

Note: 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.

---

iii. **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 \
-Xrunjdwp:transport=dt_socket,\
address=hostname:portnum ClassName
```

Note: You must give the address parameter because the debugger listens at this address.

---

iv. **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 \
-Xrunjdwp: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 4 provides a socket transport that uses the standard TCP/IP sockets to communicate between debugger and the debuggee.

NonStop Server for Java 4 defaults to socket transport. NonStop Server for Java 4 does not support shared memory transport.

See Also:

- `javac`
- `java`
- `javah`
- `javap`
- `javadoc`

`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 a complete description of the tool and its use, see the Sun Microsystems documentation for `keytool`.

Synopsis

```
keytool [ commands ]
```

See Also:

- `jar`
- `jarsigner`
- the Security trail of the Java Tutorial for examples of the use of `keytool`

kinit: Kerberos Tool

The kinit tool obtains and caches Kerberos ticket-granting tickets. Kerberos is an authentication system to enable two parties to exchange private information across an otherwise open network.

For a complete description of the tool and its use, see the Sun Microsystems documentation for kinit.

Synopsis

```
kinit [ commands ] principal-name
```

`klist`: Kerberos Display Entries in Credentials Cache and keytab

The `klist` tool allows you to view entries in the local credentials cache and key table. For a complete description of the tool and its use, see the Sun Microsystems documentation for `klist`.

Synopsis

```
klist [ commands ]
```

ktab: Kerberos Key Table Manager

The ktab tool allows you to manage the principal names and service keys stored in a local key table. For a complete description of the tool and its use, see the Sun Microsystems documentation for ktab.

Synopsis

```
ktab [ commands ]
```

`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 a complete description of the tool and its use, see the Sun Microsystems documentation for `native2ascii`.

Synopsis

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

`nsjps`: NonStop Java Virtual Machine Process Status Tool (NSJPS)

The NSJPS tool is a process status tool that lists and monitors the Java processes running on a system.

Synopsis

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

The following options are supported:

`-cpu <cpuNumber>`
: Lists the Java processes running on the CPU.

`-user <userId>`
: Lists the Java processes owned by a user.

`-heap { < | = | > } <size>`
: Lists the Java processes running with the heap specified in the argument.

`-parent <pid>`
: Lists the children of the specified Java process.

`-l`
: Displays the complete path of all the Java processes running in the system.

`-v`
: Displays the arguments passed to the Java process running on the system.

`-p`
: Displays the Guardian PIN of the Java process.

`-u`
: Includes the owner in the output.

`-t`
: Displays the process and the elapsed time.

`-help`
: Prints the nsjps help text.

`-count <cnt>`
: Repeats the listing for the specified number of times.

`-delay <time>`

Specifies the time to sleep before the next sample.

`orbd`: Object Request Broker Daemon

The Server Manager included with the `ordb` tool enables clients to transparently locate and invoke persistent objects on servers in the CORBA environment. For a complete description of the tool and its use, see the Sun Microsystems documentation for `orbd`.

Synopsis

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

See Also:

- Naming Service
- `servertool`

`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 a complete description of the tool and its use, see the Sun Microsystems documentation for `rmic`.

Synopsis

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

See Also:

- java
- `javac`
- CLASSPATH in the NonStop Server for Java Programmer's Reference

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 a complete description of the tool and its use, see the Sun Microsystems documentation for rmid.

Synopsis

```
rmid [ options ]
```

See Also:

- java
- CLASSPATH in the NonStop Server for Java Programmer's Reference
- rmic

`rmiregistry`: Java Remote Object Registry

The `rmiregistry` tool starts a remote object registry on the specified port on the current host. For a complete description of the tool and its use, see the Sun Microsystems documentation for `rmiregistry`.

Synopsis

```
rmiregistry [ port ]
```

See Also:

- `java`
- `java.rmi.registry.LocateRegistry`
- `java.rmi.Naming`

NonStop Server for Java Tools Reference Pages (529775-002)

`serialver`: Serial Version Command

The `serialver` tool returns the `serialVersionUID` of one or more classes. For a complete description of the tool and its use, see the Sun Microsystems documentation for `serialver`.

Synopsis

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

See Also:

- java.io.ObjectStreamClass

NonStop Server for Java Tools Reference Pages (529775-002)

`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 a complete description of the tool and its use, see the Sun Microsystems documentation for `servertool`.

Synopsis

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

See Also:

- `orbd`

`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 a complete description of the tool and its use, see the Sun Microsystems documentation for Naming Service.

Synopsis

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