Windows Java Tools

Java Developers Kit

Version 1.0

The Java(tm) Tools Reference Pages (Windows)

javac
: The Java compiler that you use to compile Java language programs into bytecodes.

java
: The Java interpreter that you use to run Java programs.

jdb
: The Java language debugger that helps you find and fix bugs in Java programs.

javah
: Creates C header files and C stub files from a Java class. These files provide the connective glue that allow your Java and C code to interact.

javap
: Disassembles compiled Java files and prints out a representation of the Java bytecodes.

javadoc
: Generates API documentation in HTML format from Java source code.

appletviewer
: Allows you to run applets without a World-Wide Web browser.

appletviewer - The Java Applet Viewer

appletviewer - The Java Applet Viewer

The **appletviewer** command allows you to run applets outside of the context of a World-Wide Web browser.

SYNOPSIS

appletviewer [options] urls ...

DESCRIPTION

The **appletviewer** command connects to the document(s) or resource(s) designated by *urls* and displays each applet referenced by that document in its own window. Note: if the document(s) referred to by *urls* does not reference any applets with the *APPLET* tag, **appletviewer** does nothing.

OPTIONS

-debug
Starts the applet viewer in the Java debugger - **jdb** - thus allowing you to debug the applets in the document.

java - The Java Interpreter

java - The Java Interpreter

java interprets (executes) Java bytecodes.

SYNOPSIS

```
java [ options ] classname <;args>;
java_g [ options ] classname <;args>;
```

DESCRIPTION

The **java** command executes Java bytecodes created by the Java compiler - **javac**.

The *classname* argument is the name of the class to be executed. *classname* must be fully qualified by including its package in the name, for example:

```
java java.lang.String
```

Note that any arguments that appear after *classname* on the command line are passed to the class's `main()` method.

java expects the bytecodes for the class to be in a file called *classname.class* which is generated by compiling the corresponding source file with **javac** . All Java bytecode files end with the filename extension *.class* which the compiler automatically adds when the class is compiled. *classname* must contain a `main()` method defined as follows:

```
class Aclass {
    public static void main(String argv[]){
        . . .
    }
}
```

java executes the `main()` method and then exits unless `main()` creates one or more threads. If any threads are created by `main()` then **java** doesn't exit until the last thread exits.

When you define your own classes you need to specify their location. Use CLASSPATH to do this. CLASSPATH consists of a semi-colon separated list of directories that specifies the path. For example:

```
.;C:\users\dac\classes
```

Note that the system always appends the location of the system classes onto the end of the class path unless you use the *-classpath* option to specify a path.

Ordinarily, you compile source files with **javac** then run the program using **java** . However, **java** can be used to compile and run programs when the *-cs* option is used. As each class is loaded its modification date is compared to the modification date of the class source file. If the source has been modified more recently, it is recompiled and the new bytecode file is loaded. **java** repeats this procedure until all the classes are correctly compiled and loaded.

The interpreter can determine whether a class is legitimate through the mechanism of verification. Verification ensures that the bytecodes being interpreted do not violate any language constraints.

java_g is a non-optimized version of **java** suitable for use with debuggers like **jdb**.

OPTIONS

-debug

Allows the Java debugger - **jdb** to attach itself to this **java** session. When *-debug* is specified on the command line **java** displays a password which must be used when starting the debugging session.

-cs, -checksource

When a compiled class is loaded, this option causes the modification time of the class bytecode file to be compared to that of the class source file. If the source has been modified more recently, it is recompiled and the new bytecode file is loaded.

-classpath path

Specifies the path **java** uses to look up classes. Overrides the default or the CLASSPATH environment variable if it is set. Directories are separated by colons. Thus the general format for *path* is:

```
.;<;your_path>;
```

For example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

-mx *x*

Sets the maximum size of the memory allocation pool (the garbage collected heap) to *x* . The default is 16 megabytes of memory. *x* must be >; 1000 bytes.

By default, *x* is measured in bytes. You can specify *x* in either kilobytes or megabytes by appending the letter ";k"; for kilobytes or the letter ";m"; for megabytes.

-ms *x*

Sets the startup size of the memory allocation pool (the garbage collected heap) to *x* . The default is 1 megabyte of memory. *x* must be >; 1000 bytes.

By default, *x* is measured in bytes. You can specify *x* in either kilobytes or megabytes by appending the letter ";k"; for kilobytes or the letter ";m"; for megabytes.

-noasyncgc

Turns off asynchronous garbage collection. When activated no garbage collection takes place unless it is explicitly called or the program runs out of memory. Normally garbage collection runs as an asynchronous thread in parallel with other threads.

-ss *x*

Each Java thread has two stacks: one for Java code and one for C code. The *-ss* option sets the maximum stack size that can be used by C code in a thread to *x* . Every thread that is spawned during the execution of the program passed to **java** has *x* as its C stack size. The default units for *x* are bytes. *x* must be >; 1000 bytes.

You can modify the meaning of *x* by appending either the letter ";k"; for kilobytes or the letter ";m"; for megabytes. The default stack size is 128 kilobytes ("-ss 128k").

-oss *x*

Each Java thread has two stacks: one for Java code and one for C code. The *-oss* option sets the maximum stack size that can be used by Java code in a thread to *x* . Every thread that is spawned during the execution of the program passed to **java** has *x* as its Java stack size. The default units for *x* are bytes. *x* must be >; 1000 bytes.

You can modify the meaning of *x* by appending either the letter ";k"; for kilobytes or the letter ";m"; for megabytes. The default stack size is 400 kilobytes ("-oss 400k").

-t

Prints a trace of the instructions executed (**java_g** only).

-v, -verbose

Causes **java** to print a message to stdout each time a class file is loaded.

-verify

Runs the verifier on all code.

-verifyremote

Runs the verifier on all code that is loaded into the system via a classloader. *verifyremote* is the default for the interpreter.

-noverify

Turns verification off.

-verbosegc

Causes the garbage collector to print out messages whenever it frees memory.

-D*propertyName=newValue*

Redefines a property value. *propertyName* is the name of the property whose value you want to change and *newValue* is the value to change it to. For example, this command line

```
java -Dawt.button.color=green ...
```

sets the value of the property *awt.button.color* to *green*. **java** accepts any number of *-D* options on the command line.

ENVIRONMENT VARIABLES

CLASSPATH
: Used to provide the system a path to user-defined classes. Directories are separated by semi-colons, for example,

```
.;C:\users\dac\classes;C:\tools\java\classes
```

SEE ALSO

javac, jdb, javah, javap, javadoc

javac - The Java Compiler

javac - The Java Compiler

javac compiles Java programs.

SYNOPSIS

```
javac [ options ] filename.java ...
javac_g [ options ] filename.java ...
```

DESCRIPTION

The **javac** command compiles Java source code into Java bytecodes. You then use the Java interpreter - the **java** command - to interprete the Java bytecodes.

Java source code must be contained in files whose filenames end with the *.java* extension. For every class defined in the source files passed to **javac** , the compiler stores the resulting bytecodes in a file named *classname.class* . The compiler places the resulting *.class* files in the same directory as the corresponding *.java* file (unless you specify the *-d* option).

When you define your own classes you need to specify their location. Use CLASSPATH to do this. CLASSPATH consists of a semi-colon separated list of directories that specifies the path. If the source files passed to **javac** reference a class not defined in any of the other files passed to **javac** then **javac** searches for the referenced class using the class path. For example:

```
.;C:\users\dac\classes
```

Note that the system always appends the location of the system classes onto the end of the class path unless you use the *-classpath* option to specify a path.

javac_g is a non-optimized version of **javac** suitable for use with debuggers like **jdb**.

OPTIONS

-classpath path

Specifies the path **javac** uses to look up classes. Overrides the default or the CLASSPATH environment variable if it is set. Directories are separated by semi-colons. Thus the general format for *path* is:

```
.;<;your_path>;
```

For example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

-d directory
Specifies the root directory of the class hierarchy. Thus doing:

```
javac -d <;my_dir>; MyProgram.java
```

causes the .class files for the classes in the MyProgram.java source file to be saved in the directory *my_dir*.

-g
Enables generation of debugging tables. Debugging tables contain information about line numbers and local variables - information used by Java debugging tools. By default, only line numbers are generated, unless optimization (*-O*) is turned on.

-nowarn
Turns off warnings. If used the compiler does not print out any warnings.

-O
Optimizes compiled code by inlining static, final and private methods. Note that your classes may get larger in size.

-verbose
Causes the compiler and linker to print out messages about what source files are being compiled and what class files are being loaded.

ENVIRONMENT VARIABLES

CLASSPATH
Used to provide the system a path to user-defined classes. Directories are separated by semi-colons, for example,

```
.;C:\users\dac\classes;C:\tools\java\classes
```

SEE ALSO

java, jdb, javah, javap, javadoc

javadoc-The Java API Documentation Generator

javadoc - The Java API Documentation Generator

Generates API documentation from source files.

SYNOPSIS

```
javadoc [ options ] package | filename.java...
```

DESCRIPTION

javadoc parses the declarations and doc comments in Java source files and formats the public API into a set of HTML pages. As an argument to **javadoc** you can pass in either a package name or a list of Java source files.

Within doc comments, **javadoc** supports the use of special doc tags to augment the API documentation. **javadoc** also supports standard HTML within doc comments. This is useful for code samples and for formatting text.

The **package** specified on the command line must be in your CLASSPATH. Note that **javadoc** uses *.java* files not *.class* files.

javadoc reformats and displays all public and protected declarations for,

- Classes and Interfaces
- Methods
- Variables

Doc Comments

Java source files can include doc comments. Doc comments begin with /** and indicate text to be included automatically in generated documentation.

Standard HTML

You can embed standard html tags within a doc comment. However,don't use tags heading tags like <;h1>; or <;hr>;. Because **javadoc** creates an entire structured document and these structural tags interfere with the formatting of the generated document.

javadoc Tags

javadoc parses special tags that are recognized when they are embedded within an Java doc comment. These doc tags enable you to autogenerate a complete, well-formatted API from your source code. The tags start with an @.

Tags must start at the beginning of a line. Keep tags with the same name together within a doc comment. For example, put all your @author tags together so **javadoc** can tell where the list ends.

Class Documentation Tags

`@see classname`
Adds a hyperlinked "See Also" entry to the class.

`@see fully-qualified-classname`
Adds a hyperlinked "See Also" entry to the class.

`@see fully-qualified-classname#method-name`
Adds a hyperlinked "See Also" entry to the method in the specified class.

`@version version-text`
Adds a "Version" entry.

`@author your-name`
Creates an "Author" entry. There can be multiple author tags.

An example of a class comment:

```
/**
 * A class representing a window on the screen.
 * For example:
 * <;pre>;
 *    Window win = new Window(parent);
 *    win.show();
 * <;/pre>;
 *
 * @see            awt.BaseWindow
 * @see            awt.Button
 * @version        1.2 12 Dec 1994
 * @author         Sami Shaio
 */
class Window extends BaseWindow {
        ...
}
```

Variable Documentation Tags

In addition to HTML text, variable comments can contain only the `@see` tag (see above).

An example of a variable comment:

```
/**
 * The X-coordinate of the window
 * @see window#1
 */
int x = 1263732;
```

Method Documentation Tags

Can contain `@see` tags, as well as:

`@param parameter-name description...`

Adds a parameter to the "Parameters" section. The description may be continued on the next line.

`@return description...`

Adds a "Returns" section, which contains the description of the return value.

`@exception fully-qualified-class-name description...`

Adds a "Throws" entry, which contains the name of the exception that may be thrown by the method. The exception is linked to its class documentation.

An example of a method comment:

```
/**
 * Return the character at the specified index. An index ranges
 * from <;tt>;0<;/tt>; to <;tt>;length() - 1<;/tt>;.
 * @param index      The index of the desired character
 * @return           The desired character
 * @exception        StringIndexOutOfRangeException When the index
 *                   is not in the range <;tt>;0<;/tt>;>; to <;tt>;length()
- 1<;/tt>;.
 */
public char charAt(int index) {
    ...
}
```

OPTIONS

-classpath *path*
Specifies the path javadoc uses to look up the .java files. Overrides the default or the CLASSPATH environment variable, if it is set. Directories are separated by semi-colons, for example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

-d directory
Specifies the directory where **javadoc** stores the generated HTML files. For example:

```
javadoc -d C:\usrs\dac\public_html\doc java.lang
```

-verbose
Causes the compiler and linker to print out messages about what source files are being compiled and what object files are being loaded.

ENVIRONMENT VARIABLES

CLASSPATH
Used to provide the system a path to user-defined classes. Directories are separated by semi-colons, for example,

```
.;C:\users\dac\classes;C:\tools\java\classes
```

SEE ALSO

javac, java, jdb, javah, javap ,

javah - C Header and Stub File Generator

javah - C Header and Stub File Generator

javah produces C header files and C source files from a Java class. These files provide the connective glue that allow your Java and C code to interact.

SYNOPSIS

```
javah [ options ] classname. . .
javah_g [ options ] classname. . .
```

DESCRIPTION

javah generates C header and source files that are needed to implement native methods. The generated header and source files are used by C programs to reference an object's instance variables from native source code. The .h file contains a struct definition whose layout parallels the layout of the corresponding class. The fields in the struct correspond to instance variables in the class.

The name of the header file and the structure declared within it are derived from the name of the class. If the class passed to **javah** is inside a package, the package name is prepended to both the header file name and the structure name. Underscores (_) are used as name delimiters.

By default **javah** creates a header file for each class listed on the command line and puts the files in the current directory. Use the *-stubs* option to create source files. Use the *-o* option to concatenate the results for all listed classes into a single file.

javah_g is a non-optimized version of **javah** suitable for use with debuggers like **jdb**.

OPTIONS

-o *outputfile*
Concatenates the resulting header or source files for all the classes listed on the command line into *outputfile*.

-d *directory*
Sets the directory where **javah** saves the header files or the stub files.

-td *directory*
Sets the directory where **javah** stores temporary files. By default, **javah** stores temporary files in the

directory specified by the %TEMP% environment variable. If %TEMP% is unspecified, then **javah** checks for a %TMP% environment variable. And finally, if %TMP% is unspecified, **javah** creates the directory C:\tmp and stores the files there.

-stubs
Causes **javah** to generate C declarations from the Java object file.

-verbose
Causes **javah** to print a message to stdout concerning the status of the generated files.

-classpath *path*
Specifies the path **javah** uses to look up classes. Overrides the default or the CLASSPATH environment variable if it is set. Directories are separated by semi-colons. Thus the general format for *path* is:

```
.;<;your_path>;
```

For example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

ENVIRONMENT VARIABLES

CLASSPATH
Used to provide the system a path to user-defined classes. Directories are separated by semi-colons, for example,

```
.;C:\users\dac\classes;C:\tools\java\classes
```

SEE ALSO

javac, java, jdb, javap, javadoc

javap - The Java Class File Disassembler

javap - The Java Class File Disassembler

Disassembles class files.

SYNOPSIS

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

DESCRIPTION

The **javap** command disassembles a class file. Its output depends on the options used. If no options are used, **javap** prints out the public fields and methods of the classes passed to it. **javap** prints its output to stdout. For example, compile the following class declaration:

```
class C {
    static int a = 1;
    static int b = 2;
    static {
        System.out.println(a);
    }
    static {
        a++;
        b = 7;
        System.out.println(a);
        System.out.println(b);
    }
    static {
        System.out.println(b);
    }
    public static void main(String args[]) {
        C c = new C();
    }
}
```

When the resulting class C is passed to **javap** using no options the following output results:

```
Compiled from C:\users\dac\C.java
```

```
private class C extends java\lang\Object {
    static int a;
    static int b;
    public static void main(java\lang\String []);
    public C();
    static void ();
}
```

OPTIONS

-l

Prints out line and local variable tables.

-p

Prints out the private and protected methods and fields of the class in addition to the public ones.

-c

Prints out disassembled code, i.e., the instructions that comprise the Java bytecodes, for each of the methods in the class. For example, passing class C to **javap** using the *-c* flag results in the following output:

```
Compiled from C:\users\dac\C.java
private class C extends java\lang\Object {
    static int a;
    static int b;
    public static void main(java\lang\String []);
    public C();
    static void ();

Method void main(java\lang\String [])
   0 new #4
   3 invokenonvirtual #9 ()V>
   6 return

Method C()
   0 aload_0 0
   1 invokenonvirtual #10 ()V>
   4 return

Method void ()
   0 iconst_1
   1 putstatic #7
   4 getstatic #6
   7 getstatic #7
  10 invokevirtual #8
  13 getstatic #7
  16 iconst_1
  17 iadd
  18 putstatic #7
```

```
  21 bipush 7
  23 putstatic #5
  26 getstatic #6
  29 getstatic #7
  32 invokevirtual #8
  35 getstatic #6
  38 getstatic #5
  41 invokevirtual #8
  44 iconst_2
  45 putstatic #5
  48 getstatic #6
  51 getstatic #5
  54 invokevirtual #8
  57 return

}
```

-classpath *path*

Specifies the path **javap** uses to look up classes. Overrides the default or the CLASSPATH environment variable if it is set. Directories are separated by semi-colons. Thus the general format for *path* is:

```
.;<;your_path>;
```

For example:

```
.;C:\users\dac\classes;C:\tools\java\classes
```

ENVIRONMENT VARIABLES

CLASSPATH

Used to provide the system a path to user-defined classes. Directories are separated by semi-colons, for example,

```
.;C:\users\dac\classes;C:\tools\java\classes
```

SEE ALSO

javac, java, jdb, javah, javadoc ,

jdb - The Java Debugger

jdb - The Java Debugger

jdb helps you find and fix bugs in Java language programs.

SYNOPSIS

```
jdb [ options ]
```

DESCRIPTION

The Java Debugger, **jdb** , is a dbx-like command-line debugger for Java classes. It uses the *Java Debugger API* to provide inspection and debugging of a local or remote Java interpreter.

Starting a jdb Session

Like dbx, there are two ways **jdb** can be used for debugging. The most frequently used way is to have **jdb** start the Java interpreter with the class to be debugged. This is done by substituting the command **jdb** for **java** in the command line. For example, to start HotJava under **jdb**, you use the following:

```
C:\> jdb browser.hotjava
```

or

```
C:\> jdb -classpath %INSTALL_DIR%\classes -ms4m browser.hotjava
```

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

The second way to use **jdb** is by attaching it to a Java interpreter that is already running. For security reasons, Java interpreters can only be debugged if they have been started with the *-debug* option. When started with the *-debug* option, the Java interpreter prints out a password for **jdb**'s use.

To attach **jdb** to a running Java interpreter (once the session password is known), invoke it as follows:

```
C:\> jdb -host <;hostname>; -password <;password>;
```

Basic jdb Commands

The following is a list of the basic **jdb** commands. The Java debugger supports other commands which you can list using **jdb**'s help command.

NOTE: To browse local (stack) variables, the class must have been compiled with the -g option.

help, or **?**

The most important **jdb** command, help displays the list of recognized commands with a brief description.

print

Browses Java objects. The print command calls the object's toString() method, so it will be formatted differently depending on its class.

Classes are specified by either their object ID or by name. If a class is already loaded, a substring can be used, such as Thread for java.lang.Thread. If a class isn't loaded, its full name must be specified, and the class will be loaded as a side effect. This is needed to set breakpoints in referenced classes before an applet runs.

print supports Java expressions, such as print MyClass.clsVar. Method invocation will *not* be supported in the 1.0 release, however, as the compiler needs to be enhanced first.

dump

Dumps an object's instance variables. Objects are specified by their object ID (a hexadecimal integer).

Classes are specified by either their object ID or by name. If a class is already loaded, a substring can be used, such as Thread for java.lang.Thread. If a class isn't loaded, its full name must be specified, and the class will be loaded as a side effect. This is needed to set breakpoints in referenced classes before an applet runs.

The dump command supports Java expressions such as dump 0x12345678.myCache[3].foo. Method invocation will *not* be supported in the 1.0 release, however, as the compiler needs to be enhanced first.

threads

Lists the current threads. This lists all threads in the default threadgroup, which is normally the first non-system group. (The threadgroups command lists all threadgroups.) Threads are referenced by their object ID, or if they are in the default thread group, with the form t@<;index>;, such as t@3.

where

Dumps the stack of either a specified thread, or the current thread (which is set with the thread command). If that thread is suspended (either because it's at a breakpoint or via the suspend command), local (stack) and instance variables can be browsed with the print and dump commands. The up and down commands select which stack frame is current.

Breakpoints

Breakpoints are set in **jdb** in classes, such as " `stop at MyClass:45`". The source file line number must be specified, or the name of the method (the breakpoint will then be set at the first instruction of that method). The `clear` command removes breakpoints using a similar syntax, while the `cont` command continues execution.

Single-stepping is not currently implemented, but is hoped to be available for version 1.0.

Exceptions

When an exception occurs for which there isn't a catch statement anywhere up a Java program's stack, the Java runtime normally dumps an exception trace and exits. When running under **jdb**, however, that exception is treated as a non-recoverable breakpoint, and **jdb** stops at the offending instruction. If that class was compiled with the *-g* option, instance and local variables can be printed to determine the cause of the exception.

Specific exceptions may be optionally debugged with the `catch` command, for example: "`catch FileNotFoundException`" or "`catch mypackage.BigTroubleException`. The Java debugging facility keeps a list of these exceptions, and when one is thrown, it is treated as if a breakpoint was set on the instruction which caused the exception. The `ignore` command removes exception classes from this list.

NOTE: The `ignore` command does not cause the Java interpreter to ignore specific exceptions, only the debugger.

OPTIONS

When you use **jdb** in place of the Java interpreter on the command line **jdb** accepts the same options as the java command.

When you use **jdb** to attach to a running Java interpreter session, **jdb** accepts these options:

-host *<;hostname>;*
Sets the name of the host machine on which the interpreter session to attach to is running.

-password *<;password>;*
"Logs in" to the active interpreter session. This is the password printed by the Java interpreter prints out when invoked with the *-debug* option.

ENVIRONMENT VARIABLES

CLASSPATH
Used to provide the system a path to user-defined classes. Directories are separated by semi-colons, for example,

```
.;C:\users\dac\classes;C:\tools\java\classes
```

SEE ALSO

javac, java, javah, javap javadoc