Java Man Pages

The Java Man Pages

javac
The Java compiler

java
The Java interpreter, which you use to run Java programs.

javah
Used for creating the header files and stub files that allow you to implement native methods.

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

javaprof
A profiling tool.

hotjava
A World Wide Web browser that allows you to have interactive content.

java@java.sun.com

This documentation was ported to *MS Window's Help* by Bill Bercik.
Bill may be reached at: bill@dippybird.com

Stop by his web site and get the latest update to the *Information Portafilter* for Java at:
http://www.dippybird.com/java.html

HotJava - The JAVA World Wide Web Browser

HotJava - The JAVA World Wide Web Browser

HotJava allows browsing of the World Wide Web and provides the ability to present interactive content across platforms. Interactive content is in the form of applets. Applets are programs written in the JAVA language to run within HotJava.

SYNOPSIS

hotjava [url]

DESCRIPTION

The **hotjava** command starts the HotJava browser and connects to the document or resource designated by *url*. If *url* is not supplied then HotJava goes to either:

- the page designated by the value of the environment variable WWW_HOME; or
- file://~/demo if WWW_HOME is unset.

ENVIRONMENT VARIABLES

HOTJAVA_HOME
: The directory where HotJava looks for resources that it needs to run itself. By default this is set to the installation directory when the system is first installed.

WWW_HOME
: The default home page.

HOTJAVA_READ_PATH
: Used by HotJava to determine whether any applet has permission read a file. The value of HOTJAVA_READ_PATH is a semicolon-separated list of files or directories. By default HOTJAVA_READ_PATH is set to:

    <;hotjava-install-dir>;:$HOME/public_html/

  Each component of the path is used as a prefix to compare against the file that the applet wants to open for reading. Applets are allowed to read from

  - files in matching directories
  - files in subdirectories of matching directories
  - files that match a component exactly

  For example, let's say you have a directory:

    $HOME/Images

As subdirectories under "Images" you have a "Private" subdirectory and a "Public" subdirectory. If you set HOTJAVA_READ_PATH to:

<;hotjava-install-dir>;:$HOME/public_html/:$HOME/Images/

an applet would be able to read the "Private" subdirectory. If you wanted to prevent applets from reading the "Private" subdirectory but allow them to read the "Public" one then you would set HOTJAVA_READ_PATH to:

<;hotjava-install-dir>;:$HOME/public_html/:$HOME/Images/Public/

In this way the directory "$HOME/Images/Private" won't match any component of HOTJAVA_READ_PATH and applets can't read files in it.

Setting HOTJAVA_READ_PATH to "*" disables file checking for reads. The * should be quoted so the shell doesn't expand it.

HOTJAVA_WRITE_PATH

Used by HotJava to determine whether an applet has permission to write to a file. The value of HOTJAVA_WRITE_PATH is a semicolon-separated list of files or directories. By default HOTJAVA_WRITE_PATH is set to:

/tmp/:/devices/:/dev/:~/.hotjava/

The components of the path are used as a prefix to compare against the file that the applet wants to open for writing.

See HOTJAVA_READ_PATH for an example of how matching components works. The * should be quoted so the shell doesn't expand it.

Setting HOTJAVA_WRITE_PATH to "*" disables file checking for writes.

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 contained in the files specified by *filename.java...* into class files containing Java bytecodes. Filenames must end in the .java extension. Java classes are interpreted by the java command.

Every class defined in the files passed to **javac** has its resulting bytecodes stored in a file named classname.class. The *.class* file is stored in the same directory as the corresponding .java file, unless the *-d* option is used. If a class in one of the files passed to **javac** references a class not in any of the files passed to **javac** then **javac** searches for the referenced class using the class path.

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

```
.;C:\users\lindholm\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 dbx or gdb.

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 semicolons. Thus the general format for *path* is:

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

For example:

.;C:\users\lindholm\classes;D:\java\classes

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

javac -d <;my_dir>;\editor\gui\TextWindow.java

causes the class file for the class called editor.gui.Textwindow to be saved as:

<;my_dir>;\editor\gui\TextWindow.class

-g
Enables generation of debugging tables. Debugging tables contain information about line numbers and local variables - information used by debugging tools. By default, this option is on. To turn debugging table generation off, use the -ng option.

-ng
Disables generation of debugging tables. This makes the Java bytecode files smaller, but makes it impossible for debugging tools to access local variables or print out line numbers.

-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 semicolons, for example,

.;C:\users\lindholm\classes;D:\java\classes

SEE ALSO

java, javah, javaprof, javap

javah

javah

javah produces C header files and C stub files from an Java class file.

SYNOPSIS

javah [options] classname. . .

javah_g [options] classname. . .

DESCRIPTION

javah generates headers and stubs that are needed to implement native methods. The header (.h) files created are used by C programs to reference class instance variables. 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 that is declared 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 subdirectory, called "CClassHeaders", where it saves the header files it creates. If the -stubs option is used, **javah** creates a directory named "stubs" and saves the stub file there.

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

OPTIONS

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

-td directory
Overrides the default directory where **javah** stores temporary files. The default is /tmp.

-stubs
Causes **javah** to generate stub 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 semicolons. Thus the general format for *path* is:

.;<;your_path>;

For example:

```
.;C:\users\lindholm\classes;D:\java\classes
```

ENVIRONMENT VARIABLES

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

```
.;C:\users\lindholm\classes;D:\java\classes
```

SEE ALSO

java, javac, javaprof, javap

javaprof - the Java profiling data pretty printer

javaprof - the Java profiling data prettyprinter

javaprof prettyprints the profiling data produced by **java**.

SYNOPSIS

javaprof [options] java.prof

DESCRIPTION

javaprof is a program for prettyprinting the profiling data produced by java -prof. *java.prof* is the name of the file created by using the *-prof* option to **java**. *java.prof* contains the profiling information that **javaprof** uses to compute various statistics.

NOTE: Profiling information is not accurate for multi-threaded applications. Profiling should only be used on single-threaded benchmark programs.

The profiling information is written to stdout and consists of three sections:

1. time/#calls per method
2. time/#calls per class
3. memory usage per data type

OPTIONS

-v

Causes **javaprof** to produce additional profiling information, which is considered less interesting than that listed above.

SEE ALSO

javac, javah, java, javap

JAVAP - Java class file disassembler

javap - 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, output like the following results:

```
Compiled from C:\users\lindholm\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

-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 output like the following:

```
Compiled from C:\users\lindholm/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 semicolons. Thus the general format for *path* is:

.;<;your_path>;

For example:

.;C:\users\lindholm\classes;D:\java\classes

ENVIRONMENT VARIABLES

CLASSPATH

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

.;C:\users\lindholm\classes;D:\java\classes

SEE ALSO

java, javac, javaprof, javah

java-The Java Interpreter

java-The Java Interpreter

java interprets (executes) Java bytecodes.

SYNOPSIS

java [options] classname <;args>;

java_g [options] classname <;args>;

DESCRIPTION

java is the Java interpreter, which executes Java programs.

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

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

Note that any arguments that are after *classname* in the call to java are passed to the class.

The bytecodes for the class are in a file called *classname.class* generated by compiling the corresponding class source file with **javac**. All Java bytecode files have .class as their extension, which the compiler automatically adds when the class is compiled. *classname* must contain a main() method, which **java** executes 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. The main() method is defined as follows:

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

   }
}
```

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

.;C:\users\lindholm\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, source files are compiled using **javac** then the program is run 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 byte codes being interpreted do not violate any language constraints.

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

OPTIONS

-cs, -checksource
When a compiled class is loaded, 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 semicolons. Thus the general format for *path* is:

.;<;your_path>;

For example:

.;C:\users\lindholm\classes;D:\java\classes

-ms *x*
Sets the size of the memory allocation pool (the garbage collected heap) to *x* . 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 is ";-ms3m", for 3 megabytes of memory.

-noasyncgc
Turns off asynchronous garbage collection. When passed as an option 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.

-prof
Causes **java** to produce profiling data. When used, **java** creates a file called "java.prof" in the current directory, when it exits. This file can be passed to **javaprof** for prettyprinting.

-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 64 kilobytes ("-ss 64k").

-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 ("-ss400k").

This option is not supported in HotJava alpha 2 release for NT.

-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.

ENVIRONMENT VARIABLES

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

```
.;C:\users\lindholm\classes;D:\java\classes
```

SEE ALSO

javac, javah, javaprof, javap

URL Not Available

http://java.sun.com/