Java Platform, Standard Edition

Java Usage Tracker Guide

Release 1

E50948-10

April 2024

Java Usage Tracker

Java Usage Tracker tracks how Java Runtime Environments (JREs) are being used in your systems. The output of Java Usage Tracker is a plain text, comma-separated record that contains the JRE version, the application being run, and other details. This record is appended to a file or sent over the network in a User Datagram Protocol (UDP) packet.

Java Usage Tracker System Requirements

Java Usage Tracker is supported in all current versions of the JDK.

Note:

Java Usage Tracker was included in JDK 7, JDK 6u25 and later, JDK 5u38 and later, and JDK 1.4.2_35 and later. These versions of the JDK are no longer supported.

Java Usage Tracker Output

The following is an example of output from Java Usage Tracker; it is a record of one invocation of a Java application. Line breaks were added for clarity; each record appears as one line of text:

```
"javaws application",
"Mon Sep 26 13:10:14 EDT 2022",
"MY-COMPUTER/192.0.2.0",
"https://docs.oracle.com/javase/tutorialJWS/samples/uiswing/
TreeDemoProject/:
  sourceURL=https://docs.oracle.com/javase/tutorialJWS/samples/uiswing/
TreeDemoProject/TreeDemo.jnlp
  app_model=<Base64 encoded data>",
"C:\Java\jre1.8.0_351",
"1.8.0_351",
"25.351-b10",
"Oracle Corporation",
"Oracle Corporation",
"Windows 10",
"amd64",
"10.0",
"-Xbootclasspath/
a:C:\Java\jre1.8.0_351\lib\deploy.jar;C:\Java\jre1.8.0_351\lib\javaws.j
ar;
  C:\Java\jre1.8.0_351\lib\plugin.jar -Xverify:remote -
Djava.security.manager
  -
Djava.security.policy=file:C:\Java\jre1.8.0_351\lib\security\javaws.pol
icy
  -DtrustProxy=true -Djnlpx.home=C:\Java\jre1.8.0_351\bin
  -Djnlpx.origFilenameArg=TreeDemo.jnlp -Djnlpx.remove=false -
Djnlpx.splashport=60867
  -Djnlpx.jvm=C:\Java\jre1.8.0_351\bin\javaw.exe ",
"C:\Java\jre1.8.0_351\lib\deploy.jar",
"user.home=C:\Users\MYUSER my.custom.property=null "
```

The following table describes each value of this comma-separated line:

Example	Description
`javaws application`	Type of start; it has one of the following values: • `VM start`: JVM start-up (either Java application or JNI invocation) • `plugin2`: Applet • `javaws application` or `javaws applet`: Java Web Start The value about the type of start is followed by one of the following values if there is a Deployment Rule Set (DRS) that specifically allows the application to run or block the app. • `permitted` • `denied` • `denied [required version` *`ruleVersion`* `not available]`, where *`ruleVersion`* is the minimum required version of the Deployment Rule Set specification The value about the type of start is followed by one of the following values if there is no DRS run or block rule: • `denied [by user]` • `denied [by security settings]` • No value (empty string): The application is allowed to run
`Mon Sep 26 13:10:14 EDT 2022`	Date and time

Example	Description
`MY-COMPUTER/192.0.2.0`	Host name and IP address in the form *`<hostname>`*`/`*`<literalIPaddress>`*

`https://docs.oracle.com/javase/tutorialJWS/`
`  samples/uiswing/TreeDemoProject/:`
`  sourceURL=https://docs.oracle.com/javase/`
`    tutorialJWS/samples/uiswing/`
`    TreeDemoProject/TreeDemo.jnlp`
`  app_model=`*`<Base64 encoded data>`*
	Java command (name of main class or jar file) and list of arguments, if any (space-separated list)
	Java Web Start applications have the following form:
	`<Document base>``:`
	`  `*`<main class> <arguments>`*
	`  app_model=`*`<serialized classes for use by AMC>`*
	`  app_customer=`*`<contents of the <customer> block in a rule>`*
	Applet invocations have the following form:
	`<Document base>``:`
	`  `*`<parameters>`*
	`  app_model=`*`<serialized classes for use by AMC>`*
	`  app_customer=`*`<contents of the <customer> block>`*
	The values for `app_model` and `app_customer` are encoded in Base64.
	The value of `app_model` contains basic information about the application, including location, codebase, and main class. Its value is the same regardless of whether the application is permitted or denied to run.
	The parameter `app_customer` appears only if the application is governed by a specific DRS rule that DRS run rule contains a `<customer>` block.

`C:\Java\jre1.8.0_351`	Directory that contains the JRE (`java.home` system property value)
`1.8.0_351`	Java version (`java.version` system property value)
`25.351-b10`	JVM version (`java.vm.version` system property value)
`Oracle Corporation`	Java vendor (`java.vendor` system property value)
`Oracle Corporation`	JVM vendor (`java.vm.vendor` system property value)

Example	Description
`Windows 10`	Operating system name (`os.name` system property value)
`amd64`	Operating system architecture (`os.arch` system property value)
`10.0`	Operating system version (`os.version` system property value)
`-Xbootclasspath/a:C:\Java\jre1.8.0_351\lib\deploy.jar; C:\Java\jre1.8.0_351\lib\javaws.jar; C:\Java\jre1.8.0_351\lib\plugin.jar -Xverify:remote -Djava.security.manager -Djava.security.policy=file:C:\Java\jre1.8.0_351\lib\security\javaws.policy -DtrustProxy=true -Djnlpx.home=C:\Java\jre1.8.0_351\bin -Djnlpx.origFilenameArg=TreeDemo.jnlp -Djnlpx.remove=false -Djnlpx.splashport=60867 -Djnlpx.jvm=C:\Java\jre1.8.0_351\bin\javaw.exe`	JVM arguments (space-separated list); empty if there are no JVM arguments; `n/a` if this information is not available (for example, in Java SE 1.4.2, this information is not available)
`C:\Java\jre1.8.0_351\lib\deploy.jar`	Class path (`java.class.path` system property value)
`user.home=C:\Users\CURRENT-USER my.custom.property=null`	`Name=value` pairs of any additional system properties specified in the Java Usage Tracker properties file. Multiple pairs are space-separated; empty if no additional property names are specified (default).

Items that Contain Spaces in Java Usage Tracker Output

In the fields that are space-separated lists, a different quote character (by default, the single quotation mark, `'`) is used to quote an item that contains a space. Any existing quote characters are printed twice.

For example, consider the following command:

```
/jdk1.8.0_20/bin/java
  -Dfoo1="a b"
  -Dfoo=\"
  -jar c:\\Program\ Files\\Java/jdk1.6.0_25/demo/jfc/Java2D/
Java2Demo.jar
```

Java Usage Tracker prints these system properties as follows (line breaks were added for clarity):

```
"'-Dfoo1=a b' -Dfoo="" ",
"-jar c:\Program Files\Java/jdk1.6.0_25/demo/jfc/Java2D/Java2Demo.jar",
""
```

For JRE versions prior to 8u20, Java Usage Tracker does not surround fields with quotation marks.

Enabling and Configuring Java Usage Tracker

Java Usage Tracker is disabled by default. Enable and configure it by creating a properties file named `usagetracker.properties`. See Example Java Usage Tracker properties File.

For JRE 8u152 and later, if you want Java Usage Tracker to track all JREs on your system, then put the `usagetracker.properties` file in the central file system location, which differs depending on your operating system:

- Windows: `%ProgramFiles%\Java\conf\` (Windows x64) or `%ProgramFiles(x86)%\java\conf\` (Windows x86)
- Linux and Solaris: `/etc/oracle/java/`
- macOS: `/Library/Application Support/Oracle/Java/`

If you want Java Usage Tracker to track a specific JRE, then ensure that the `usagetracker.properties` file doesn't exist in the central file system location, and put the `usagetracker.properties` file in the directory `<JRE directory>/conf/management/` (`<JRE directory>/lib/management/` for JRE releases prior to 9). Note that the path name is different, depending on whether you are configuring Java Usage Tracker for a JDK or for a JRE.

For additional flexibility, if you want to use a different properties file, then you can specify it with the system property `-Dcom.oracle.usagetracker.config.file` on the command line. In the following example, Java Usage Tracker uses the properties file `/path/usagetracker.properties`:

```
java -Dcom.oracle.usagetracker.config.file=/path/
usagetracker.properties MyApplication
```

The JVM searches the following locations, in order, for a `usagetracker.properties` file. It uses the first one it finds to enable and configure Java Usage Tracker.

1. Path specified by the system property `-Dcom.oracle.usagetracker.config.file`
2. Central file system location (for JRE 8u152 and later)
3. `<JRE directory>/conf/management/` (`<JRE directory>/lib/management/` for JRE releases prior to 9)

Note:

To enable Java Usage Tracker, the `usagetracker.properties` file that you create must have a valid value for at least one of the following properties:

- `com.oracle.usagetracker.logToFile`
- `com.oracle.usagetracker.logToUDP`

Java Usage Tracker Properties

This section describes the properties you can specify in the Java Usage Tracker properties file.

These properties are set only in the Java Usage Tracker properties file; they are not set at the command line. This is intended so that Java Usage Tracker has no impact on or interaction with the JRE user or existing applications.

Note:

The backslash (`\`) is an escape character in a properties file. Consequently, when specifying file paths that include directories or drive letters, use a forward slash (`/`) or an escaped backslash (`\\`) as a directory separator.

Property	Description
`com.oracle.usagetracker.additionalProperties`	Use this property to record values of additional Java properties and their values. The value of this property is a comma-separated list of properties. For example (ignore line break): `com.oracle.usagetracker.additional Properties = user.home,my.custom.property`
`com.oracle.usagetracker.innerQuote`	The character or string used to quote items that contain a space in the JVM argument field list and the additional properties field. The default value is the single quotation mark (`'`). This property is available in JRE 8u20 and later.
`com.oracle.usagetracker.logFileMaxSize`	The log file size limit, in bytes. If the file size equals or exceeds the given value when logging is attempted, that attempt will be canceled. If this property is not set, then there is no log file limit.
`com.oracle.usagetracker.logToFile`	If this property is specified, the fully qualified path name of the file to which Usage Tracker writes records. You can specify `${user.home}` in the path name. The property will expand to the user's home directory. For example (ignore line break): `com.oracle.usagetracker.logToFile = ${user.home}/.java_usagetracker`
`com.oracle.usagetracker.logToUDP`	If this property is specified, Java Usage Tracker logs to the specified remote host in a UDP packet. For example (ignore line break): `com.oracle.usagetracker.logToUDP = loggingmachine.domainname:32139` Specifying an IP address may be faster in some cases; although, this resolution does not delay the startup of the JVM or the application. See Java Usage Tracker Sample: Receiver for UDP Packets for an example application that can receive UDP packets.

Property	Description
`com.oracle.usagetracker.maxFieldSize`	Any single field limit, in bytes. The default is no limit. Java Usage Tracker truncates a field to this limit, without breaking the record format, if the `com.oracle.usagetracker.sendTruncatedRecords` property is true. This property is available in JRE 8u152 and later.
`com.oracle.usagetracker.maxSize`	Overall record limit, in bytes. The default is no limit. Java Usage Tracker truncates records to this limit if the `com.oracle.usagetracker.sendTruncatedRecords` property is true. This property is available in JRE 8u152 and later.
`com.oracle.usagetracker.quote`	The character or string used to quote fields. The default value is the double quotation mark (""). This property is available in JRE 8u20 and later.
`com.oracle.usagetracker.sendTruncatedRecords`	Truncates records and individual fields if they exceed the sizes specified by the `com.oracle.usagetracker.maxSize` and `com.oracle.usagetracker.maxFieldSize` properties, respectively. The default value is true. This property is available in JRE 8u152 and later.
`com.oracle.usagetracker.separator`	The character or string that separates entries in the log file. The default is the comma (`,`).
`com.oracle.usagetracker.verbose`	If this property is set to true, error information may be reported to the standard error stream; this is only recommended for diagnostic purposes.

Example Java Usage Tracker properties File

To create a Java Usage Tracker properties file, you can use the following example as a template. Lines that begin with the pound sign (#) are comments.

```
# UsageTracker template properties file.
# Copy to <JRE directory>/conf/management/usagetracker.properties
# (or <JRE directory>/lib/management/usagetracker.properties for
# JRE releases prior to 9) and edit, uncommenting required settings,
to enable.

# Settings for logging to a file:
```

```
# Use forward slashes (/) because backslash is an escape character in a
# properties file.
# com.oracle.usagetracker.logToFile = ${user.home}/.java_usagetracker

# Settings for logging to a UDP socket:
# com.oracle.usagetracker.logToUDP = hostname.domain:32139

# (Optional) Specify a file size limit in bytes:
# com.oracle.usagetracker.logFileMaxSize = 10000000

# If the record should include additional Java properties,
# this can be a comma-separated list:
# com.oracle.usagetracker.additionalProperties =

# Additional options:
# com.oracle.usagetracker.verbose = true
com.oracle.usagetracker.separator = ,
com.oracle.usagetracker.quote = "
com.oracle.usagetracker.innerquote = '
```

Java Usage Tracker Sample: Receiver for UDP Packets

The following sample, `UsageServerTracker.java`, is a simple application that listens for Java Usage Tracker data:

```
/*
 * Copyright (c) 2012, 2015, Oracle and/or its affiliates. All rights
reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
are met:
 *
 * -Redistribution of source code must retain the above copyright
notice, this
 *  list of conditions and the following disclaimer.
 *
 * -Redistribution in binary form must reproduce the above copyright
notice,
 *  this list of conditions and the following disclaimer in the
documentation
 *  and/or other materials provided with the distribution.
 *
 * Neither the name of Oracle or the names of contributors may
 * be used to endorse or promote products derived from this software
without
 * specific prior written permission.
 *
 * This software is provided "AS IS," without a warranty of any kind.
ALL
```

```
 * EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES,
INCLUDING
 * ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
PURPOSE
 * OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC.
("SUN")
 * AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY
LICENSEE
 * AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS
 * DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR
ANY LOST
 * REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL,
CONSEQUENTIAL,
 * INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF
THE THEORY
 * OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS
SOFTWARE,
 * EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
 *
 * You acknowledge that this software is not designed, licensed or
intended
 * for use in the design, construction, operation or maintenance of any
 * nuclear facility.
 */

import java.net.InetAddress;
import java.net.DatagramSocket;
import java.net.DatagramPacket;
import java.io.IOException;
import java.net.SocketException;

import java.io.File;
import java.io.OutputStream;
import java.io.FileOutputStream;
import java.io.OutputStreamWriter;

/**
  * A daemon that listens for and logs UsageTracker information.
  */
public class UsageTrackerServer {

    static boolean verbose = false;
    boolean initialized = false;
    String logFileName = null;
    File logFile;
    OutputStreamWriter writer;
    int port = 32139;
    InetAddress address = null;
    DatagramSocket socket;
    private static final int BUFFERSIZE = 65536;
    long received = 0;
    boolean running = true;
    /**
      * Main entry point for starting this daemon.
      */
    public static void main(String [] args) {
        try {
            UsageTrackerServer uts = new UsageTrackerServer(args);
            uts.run();
        } catch (Exception e) {
            System.out.println("UsageTrackerServer: " +
e.getMessage());
            if (verbose) {
                e.printStackTrace();
            }
            System.exit(1);
        }
    }

    public static void usage() {
        System.out.println("UsageTrackerServer [-v] [-o filename]
[host]:port\n" +
            "e.g. UsageTrackerServer -o usagetracker.out :32139\n");
    }

    /**
      * Initialize a UsageTrackerServer given some arguments.
      */
    UsageTrackerServer(String [] args) throws Exception {

        boolean usage = false;
        for (int i=0; i<args.length; i++) {

            if (args[i].equals("-?") || args[i].equals("--h")) {
                usage = true;
                break;
            } else if (args[i].equals("-v")) {
               verbose = true;
            } else if (args[i].equals("-o")) {
                logFileName = args[i+1];
                i++;
            } else if (args[i].contains(":")) {
                // parse [address]:port
                int colon = args[i].indexOf(':');
                if (colon > 0) {
                    try {
                        address = InetAddress.getByName(
                            args[i].substring(0, colon));
                    } catch (Exception ae) {
                        System.out.println("UsageTrackerServer: " +
                            "problem setting listen address: " + ae);
                        usage = true;
                    }
                }
                try {
                    port =
Integer.parseInt(args[i].substring(colon+1));
                } catch (NumberFormatException nfe) {
                    System.out.println("UsageTrackerServer: cannot set
port: " +
                        args[i].substring(colon+1));
                    usage = true;
                }
            } else {
                usage = true;
            }
        }
        // Argument failure or request for usage gets the usage
message only:
        if (usage) {
            usage();
            return;
        }
        // Otherwise, continue to proper initialization:
        socket = null;
        try {
        if (address != null) {
            socket = new DatagramSocket(port, address);
        } else {
            socket = new DatagramSocket(port);
        }
        } catch (SocketException se) {
            throw new Exception("problem creating socket: " + se);
        }
        if (logFileName != null) {
            try {
                File logFile = new File(logFileName);
                if (verbose) {
                    System.out.println("Using logfile: " +
logFileName);
                    if (logFile.exists()) {
                        System.out.println("File exists, will
append.");
                    }
                }
                FileOutputStream fos = new FileOutputStream(logFile,
true);
                writer = new OutputStreamWriter(fos, "UTF-8");
            } catch (IOException ioe) {
                throw new Exception("problem using file " +
logFileName + ": " +
                    ioe);
            }
        }
        initialized = true;
```

```
    }

   public void run() throws Exception {
        if (!initialized) {
            return;
        }
        Runnable r = null;
        if (writer == null) {
            r = new UsageTrackerServerRunnable(socket);
        } else {
            r = new UsageTrackerServerRunnable(socket, writer);
        }
        Thread t = new Thread(r, "UsageTrackerServerRunnable");
        t.start();
        t.join();
    }

    /**
      * Runnable that listens and logs.
      */
    private class UsageTrackerServerRunnable implements Runnable {
        DatagramSocket listenSocket;
        OutputStreamWriter writer = null;

        UsageTrackerServerRunnable(DatagramSocket socket) {
            listenSocket = socket;
        }
        UsageTrackerServerRunnable(DatagramSocket socket,
            OutputStreamWriter writer) {
            this(socket);
            this.writer = writer;
        }

        public void run() {
            byte [] buf = new byte[BUFFERSIZE];
            DatagramPacket packet = new DatagramPacket(buf,
buf.length);

            // Ready to receive data
            if (verbose) {
                String addr =
listenSocket.getLocalAddress().getHostAddress();
                if (addr.equals("0.0.0.0")) {
                    addr = "localhost";
                }
                System.out.println("UsageTrackerServer: ready to
receive on " +
                    addr + ":" + listenSocket.getLocalPort());
            }
            while (running) {
                try {
                    listenSocket.receive(packet);
```

```
                        String dataReceived = new String(packet.getData(),
0,
                            packet.getLength());

                        // The format of a UsageTracker record contains a
newline at
                        // the end; if that is missing, we have a
truncated/corrupt
                        // packet.
                        if (!dataReceived.endsWith("\n")) {
                            System.out.println("Incomplete message
received: " +
                              "size = " + packet.getLength() + ", data =
" +
                              dataReceived);
                            dataReceived = dataReceived + "\n";
                        }
                        received++;
                        if (verbose) {
                            System.out.println("Received message size: " +
                                dataReceived.length());
                        }
                        if (writer != null) {
                            writer.write(dataReceived, 0,
dataReceived.length());
                            writer.flush();
                        } else {
                            System.out.print(dataReceived);
                        }
                    } catch (IOException ioe) {
                        ioe.printStackTrace();
                    }
                }
            }
        }
    }
}
```

The following is an example of running this sample:

```
java UsageTrackerServer -v -o usagetracker.out :32139
```

- The `-v` option is verbose; if you specify this option, the sample displays additional information.
- The `-o` option enables you to specify the name of a log file; if you do not specify this option, the sample prints messages to standard output.
- In this example, the UDP receiver listens on the localhost address on port 32139. When a JRE (with an enabled and configured Java Usage Tracker) sends data, the receiver will send the data to the file `usagetracker.out`. The port number is arbitrary but must be available and must match the one configured in the JRE. If

multiple interfaces exist, it may be necessary to specify the port using the form hostname:port or ipaddress:port.

Java Usage Tracker Errors and Exceptions

If Java Usage Tracker encounters an error or exception during the logging of a record, it does not interrupt the application currently running.

Java Usage Tracker does not report errors unless the property `oracle.usagetracker.verbose` is specified in the properties file.

Managing Disk Space Used by Java Usage Tracker Log File

Although the size of the Java Usage Tracker log file is small, consider periodically truncate, compress, archive, or delete the log file. When Java Usage Tracker incrementally adds records to the log file, it does not check for available disk space or perform administrative tasks such as truncating, deleting, or compressing the log file in order to be minimally intrusive.

In addition, you can specify the maximum size of the log file, in bytes, with the `oracle.usagetracker.logFileMaxSize` property in the properties file.

Java Usage Tracker Limitations

Java Usage Tracker cannot log Java command line options that are processed by the Java launcher before the JVM is started. For example, Java Usage Tracker does not record the command line options `-client` and `-server` that select the Java HotSpot client and server VM, respectively. In addition, Java Usage Tracker may not log an application if it terminates immediately because it will not stop a process from exiting.

Java Usage Tracker Frequently Asked Questions

Here are answers to some frequently asked questions:

Does Java Usage Tracker affect the private JRE within a JDK, or does it only affect the standalone JRE?

If you have a JDK installed in a computer, there is a JRE in the jre subdirectory; this is the private JRE referred to in the question. Yes, Java Usage Tracker logs the usage of both the private JRE and the standalone JRE, but note that they are configured separately through their own individual `conf/management/usagetracker.properties` files.

Can Java Usage Tracker log the usage of JDK tools?

If Java Usage Tracker is enabled, it logs the usage of tools that come with the JDK such as jmap and jstack.

Does Java Usage Tracker log the usage of JVMs created by native Java applications?

Yes. When a native application creates a JVM with the Java Native Interface (JNI), Java Usage Tracker logs this invocation with a blank Java command.

Will an invocation similar to java -jar file.jar be tracked by Java Usage Tracker?

Yes.

Does Oracle capture any of the data logged by Java Usage Tracker?

No. As the administrator of the JRE installation, usage data obtained from Java Usage Tracker is stored in the file of your choice or sent to the UDP host and port that you specify. There is no facility for this data to leave your own network. (Theoretically, if your firewall permits it, the port your UDP host listens on could be configured as remote, but this is not expected or recommended usage.)

What does the log record look like for native applications, applets, and denied applications?

The following is an example of a log record for a native application (line breaks were added for clarity):

```
"VM start",
"Mon Sep 26 13:08:33 EDT 2022",
"MY-COMPUTER/192.0.2.0",
"Main",
"C:\Java\jre1.8.0_351",
"1.8.0_351",
"25.351-b10",
"Oracle Corporation",
"Oracle Corporation",
"Windows 10",
"amd64",
"10.0",
"-Dmy.custom.property=myvalue ",
".",
"user.home=C:\Users\RGALLARD my.custom.property=myvalue "
```

The following is an example of a log record for an applet:

```
"plugin2",
"Mon Sep 26 14:00:48 EDT 2022",
"MY-COMPUTER/192.0.2.0",
"https://docs.oracle.com/javase/tutorial/deployment/applet/
deployingApplet.html:
  jnlp_href=examples/dist/applet_ComponentArch_DynamicTreeDemo/
dynamictree_applet.jnlp
  launchjnlp= codebase_lookup=false
code=appletComponentArch.DynamicTreeApplet.class
  codebase=https://docs.oracle.com/javase/tutorial/deployment/applet/
width=375
  archive=examples/dist/applet_ComponentArch_DynamicTreeDemo/
DynamicTreeDemo.jar height=375
  app_model=<Base 64 encoded data>",
"C:\Java\jre1.8.0_341",
"1.8.0_341",
"25.341-b10",
"Oracle Corporation",
"Oracle Corporation",
"Windows 10",
"x86",
"10.0",
"-Xbootclasspath/
a:C:\Java\jre1.8.0_341\lib\deploy.jar;C:\Java\jre1.8.0_341\lib\javaws.j
ar;
  C:\Java\jre1.8.0_341\lib\plugin.jar -Djava.security.manager -
D__jvm_launched=16873846471
  -D__applet_launched=16873824704 ",
"C:\Java\jre1.8.0_341\lib\deploy.jar",
"user.home=C:\Users\CURRENT-USER my.custom.property=null "
```

The following is an example of a log record of a Java Web Start application that was denied by security settings:

```
"javaws application denied [by user]",
"Mon Sep 26 13:42:49 EDT 2022",
"MY-COMPUTER/192.0.2.0",
"https://docs.oracle.com/javase/tutorialJWS/samples/uiswing/
TreeDemoProject/:
  sourceURL=https://docs.oracle.com/javase/tutorialJWS/samples/uiswing/
TreeDemoProject/TreeDemo.jnlp
  app_model=<Base 64 encoded data>",
"C:\Java\jre1.8.0_351","1.8.0_351",
"25.351-b10",
"Oracle Corporation",
"Oracle Corporation",
"Windows 10",
"amd64",
"10.0",
"-Xbootclasspath/
a:C:\Java\jre1.8.0_351\lib\deploy.jar;C:\Java\jre1.8.0_351\lib\javaws.j
ar;
  C:\Java\jre1.8.0_351\lib\plugin.jar -Xverify:remote -
Djava.security.manager
    -
Djava.security.policy=file:C:\Java\jre1.8.0_351\lib\security\javaws.pol
icy
    -DtrustProxy=true -Djnlpx.home=C:\Java\jre1.8.0_351\bin
    -Djnlpx.origFilenameArg=TreeDemo.jnlp -Djnlpx.remove=false -
Djnlpx.splashport=61230
    -Djnlpx.jvm=C:\Java\jre1.8.0_351\bin\javaw.exe ",
"C:\Java\jre1.8.0_351\lib\deploy.jar",
"user.home=C:\Users\CURRENT-USER my.custom.property=null "
```

How can I remove the quoting behavior of JRE 8u20 and later for records with the previous formatting?

In the Java Usage Tracker properties file, set blank quote characters with the following two lines:

```
com.oracle.usagetracker.quote=
com.oracle.usagetracker.innerQuote=
```

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Java Platform, Standard Edition Java Usage Tracker Guide, Release 1
E50948-10

Access to Oracle Support

Oracle customer access to and use of Oracle support services will be pursuant to the terms and conditions specified in their Oracle order for the applicable services.