SolarWinds Technical Reference

Java Application Server (SNMP) Template Pack

This document describes the template included in the Orion APM Java Application Server (SNMP) Template Pack.

Document Revised: 05/16/2011

The Orion APM Java application Server (SNMP) template allows you to monitor vital statistics from Java Virtual Machines (JVM). Before assigning the template to a node with a JVM you want to monitor, you must configure the target JVM to accept SNMP queries.

Configuring Java Virtual Machines for SNMP

The sections below discuss how to configure JVM servers for SNMP monitoring including those that are embedded with Apace Tomcat, JBoss, Glassfish, IBM Websphere, and Oracle Weblogic. Configuration instructions are given for both Linux and Microsoft Windows operating systems.

- Configuring a Standalone Java Virtual Machine
- Configuring Apache Tomcat (tested on version 7.0)
- Configuring JBoss (tested on versions 5.0.1, 5.1, and 6.0)
- Configuring GlassFish (tested on version 3.1)
- Configuring IBM WebSphere (tested on version 3.1)
- Configuring Oracle WebLogic (tested on version 10.3.4.0)

Configuring a Standalone Java Virtual Machine

Windows

1. Download the JDK from the Oracle website: http://www.oracle.com/technetwork/java/javase/downloads/index.html (tested on JDK SE 6, update version 24).
2. Install the JDK.
3. On the license agreement page, read and accept the license and the click **Next**.
4. Click **Change** to change the installation directory to `C:\Program Files\Java\jdk1.6.0_24`, and then click **OK**.
5. Click **Next**.
6. Click **Finish** to exit from the installer.
7. Add a variable to the system with its path pointing to the JDK installation folder. For example:

   Variable name: `JAVA_HOME`
   Variable value: `C:\Program Files\Java\jdk1.6.0_24`

   Note: If you want to run Java commands from the command line, you should add the JDK installation path to the `PATH` variable.
8. Navigate to the `%JAVA_HOME%\jre\lib\management` folder.

9. Rename the `snmp.acl.template` file to `snmp.acl`. Edit `snmp.acl` by replacing its content with the following:

```
acl = {
 {
   communities = public, private
   access = read-only
   managers = localhost, apm-server
 }
}
```

   Where `apm-server` is the hostname of your APM server.

10. Grant access to the `snmp.acl` file only for the Administrator. To grant access only for the Administrator, refer to this topic at the following location: http://download.oracle.com/javase/6/docs/technotes/guides/management/security-windows.html

Linux

1. Download the JDK from the Oracle website: http://www.oracle.com/technetwork/java/javase/downloads/index.html (tested on JDK SE 6, update version 24).
2. Unpack and run the JDK. In this case, the JDK was installed into the `/usr/java/jdk1.6.0_24` folder.
3. Move this folder to `/usr/local`.
4. In the `/usr/local/` directory, create a symbolic link to jdk1.6.0_24 named *Java.* Now the JDK is installed in `/usr/local/jdk1.6.0_24` and linked to `/usr/local/java`.
5. Add the `/usr/local/java/bin` folder to the system path. (If it is only for your account, add it in `.bash_profile` file in your home directory). In Slackware, it should be in the `.profile` file. To make it a system wide environment, add it in the `/etc/profile`.
6. Edit a line in `.bash_profile` to be similar to: `PATH=$PATH:$HOME/bin:/usr/local/java/bin.` The path will be automatically set at boot time. To set the path immediately, use the command:

```
$ export PATH=$PATH:/usr/local/java/bin
```

7. You can call java directly in your shell:

```
$ java -version
java version "1.6.0_24"
Java(TM) SE Runtime Environment (build 1.6.0_24-b04)
Java HotSpot(TM) 64-Bit Server VM (build 14.3-b01, mixed mode)
```

8. Add a variable to the system with the path pointing to the JDK installation folder.
9. Add the following lines to the `/etc/profile` file. Syntax dependencies may be different in various Linux distributions. This example is shown for the CentOS system:

```
export JAVA_HOME=/usr/java/jdk1.6.0_24

export PATH=$PATH:/usr/java/jdk1.6.0_24
```

   To set this immediately, run the export commands in your shell. Or, logout then log back in to your system.

10. Go to the `$JAVA_HOME/jre/lib/management` folder and rename the `snmp.acl.template` file to `snmp.acl`.

11. Edit `snmp.acl` by removing all lines and writing the following:

```
acl = {
 {
   communities = public, private
   access = read-only
   managers = localhost, apm-server
 }
}
```

    Where `apm-server` is the hostname of your APM server.

12. Grant access to the `snmp.acl` file only for the root user. Run the following commands as the root user in your shell:

```
# chown root.root $JAVA_HOME/jre/lib/management/snmp.acl
# chmod 600 $JAVA_HOME/jre/lib/management/snmp.acl
```

Testing a Standalone JVM in Linux

You can test that your JVM can respond to SNMP queries by adding `-Dcom.sun.management.snmp.port=1161 -Dcom.sun.management.snmp.interface=0.0.0.0` to the java command line.

Example:

```
java -Dcom.sun.management.snmp.port=1161 -Dcom.sun.management.snmp.interface=0.0.0.0
some_java_applet
```

In another window/shell, run the following command to test SNMP and the JDK configurations:

```
snmpwalk -v 2c -c public 127.0.0.1:1161 1.3.6.1.4.1.42.2.145.3.163.1.1.2.11
```

The command should return a value similar to this:

```
SNMPv2-SMI::enterprises.42.2.145.3.163.1.1.2.11.0 = Counter64: 4803352
```

Configuring Apache Tomcat (tested on version 7.0)

Windows Service

1. Open Tomcat configuration**: Start > All Programs > Apache Tomcat > Configure Tomcat.**
2. Open the **Java** tab, and then add the following lines to the *Java Options* box:
   `-Dcom.sun.management.snmp.port=1161`
   `-Dcom.sun.management.snmp.interface=0.0.0.0`
3. Click **Apply**.
4. Go to the **Windows Services** console**.**
5. Right-click the **Apache Tomcat** service.
6. Click **Properties**.
7. Click the **Log On tab**, and then select **Log on as this account.**
8. Click **Browse**, find the user “Administrator” and type the Administrator password twice.
9. Click **OK**.
10. In the **Tomcat Configuration** window, return to the **General** tab, and then start the service.

Windows Command Line

1. Open the file `%TOMCAT_HOME%\bin\catalina.bat` and add the following lines into the **Debug**, **Run** and **Start** sections (where `%TOMCAT_HOME%` is the path to your Tomcat installation):

```
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.port=1161"
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.interface=0.0.0.0"

…
:doDebug
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.port=1161"
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.interface=0.0.0.0"
shift
…
:doRun
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.port=1161"
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.interface=0.0.0.0"
shift
…
:doStart
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.port=1161"
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.interface=0.0.0.0"
shift
…
```

2. Run `%TOMCAT_HOME%\bin\startup.bat` to start Tomcat.

Linux

1. Open `$TOMCAT_HOME/bin/catalina.sh` and then add the following lines into **Debug**, **Run** and **Start** sections (where `%TOMCAT_HOME%` is the path to your Tomcat installation):

```
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.port=1161"
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.interface=0.0.0.0"
…
if [ "$1" = "debug" ] ; then
  JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.port=1161"
  JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.interface=0.0.0.0"
  if $os400; then
…
elif [ "$1" = "run" ]; then
  JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.port=1161"
  JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.interface=0.0.0.0"
  shift
…
elif [ "$1" = "start" ] ; then
  JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.port=1161"
  JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.interface=0.0.0.0"
  if [ ! -z "$CATALINA_PID" ]; then
…
```

3. Run `$TOMCAT_HOME/bin/startup.sh` command to start Tomcat.

Configuring JBoss (tested on versions 5.0.1, 5.1, and 6.0)

Windows Service

1. Edit `%JBOSS_HOME%\bin\run.bat` by adding the following lines (where `%JBOSS_HOME%` is the path to your JBoss installation):

```
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.port=1161"
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.interface=0.0.0.0"

…
rem Setup JBoss specific properties
set JAVA_OPTS=-Dprogram.name=%PROGNAME% %JAVA_OPTS%
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.port=1161"
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.interface=0.0.0.0"
…
```

4. Register JBoss as a service by running: `%JBOSS_HOME%\bin\service.bat -install`
5. Go to the **Windows Services** console
6. Right-click *JBoss Application Server* service.
7. Click **Properties**.
8. Click the **Log On** tab and then select **Log on as this account**.
9. Click **Browse**, find the user “Administrator,” and then type the Administrator password twice.
10. Click **OK**.
11. Start the JBoss service.

Windows Command Line

1. Edit `%JBOSS_HOME%\bin\run.bat` by adding the following lines (where `%JBOSS_HOME%` is the path to your JBoss installation):

```
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.port=1161"set
"JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.interface=0.0.0.0"

…
rem Setup JBoss specific properties
set JAVA_OPTS=-Dprogram.name=%PROGNAME% %JAVA_OPTS%
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.port=1161"
set "JAVA_OPTS=%JAVA_OPTS% -Dcom.sun.management.snmp.interface=0.0.0.0"
…
```

2. Start JBoss by running `%JBOSS_HOME%\bin\run.bat`.

Linux

1. Edit `$JBOSS_HOME/bin/run.sh` by adding the following lines (where `$JBOSS_HOME$` is the path to your JBoss installation):

```
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.port=1161"
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.interface=0.0.0.0"
…
# Setup JBoss specific properties
JAVA_OPTS="${JAVA_OPTS:+$JAVA_OPTS -Dprogram.name=$PROGNAME}"
JAVA_OPTS="${JAVA_OPTS:--Dprogram.name=$PROGNAME}"
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.port=1161"
JAVA_OPTS="$JAVA_OPTS -Dcom.sun.management.snmp.interface=0.0.0.0"
…
```

2. Run JBoss by running `$JBOSS_HOME/bin/run.sh`.

Configuring GlassFish (tested on version 3.1)

1. Run the GlassFish Application Server.
2. Open a web browser and then navigate to: http://hostname:4848 where *hostname* is the name of your GlassFish server.
3. In the left panel, click **Configurations: server-config**.
4. In the main window, click **JVM settings**.
5. Click the **JVM Options** tab.
6. Click **Add JVM Option** and then type `-Dcom.sun.management.snmp.port=1161` in the blank field.
7. Click **Add JVM Option** and then type `-Dcom.sun.management.snmp.interface=0.0.0.0` in the blank field.
8. Click **Save**.
9. Restart the GlassFish server.

Configuring IBM WebSphere (tested on version 3.1)

IBM WebSphere uses its own JDK, installed at `%WEBSHERE_HOME%\java` (where `%WEBSHERE_HOME%` is the path to your WebSphere installation).Configure the IBM JDK in the following manner:

1. Go to the `$JAVA_HOME/jre/lib/management` folder.
2. Rename the `snmp.acl.template` file to `snmp.acl`.
3. Edit `snmp.acl` by replacing its content with the following:

```
acl = {
 {
   communities = public, private
   access = read-only
   managers = localhost, apm-server
 }
}
```

Where `apm-server` is the hostname of your APM server.

4. Grant access to the `snmp.acl` file only for the root user. Run the following commands as the root user in your shell:
```
# chown root.root $JAVA_HOME/jre/lib/management/snmp.acl
# chmod 600 $JAVA_HOME/jre/lib/management/snmp.acl
```
5. Run the **IBM WebSphere Application Server**.
6. Open a web browser and then navigate to: https://hostname:9043 where *hostname* is the name of the **IBM WebSphere** server.
7. In the left panel, click **Expand Servers and Server types.**
8. Click **WebSphere Application Servers**.
9. In the main window, click your server.
10. In the **Server Infrastructure** section, expand **Java and Process Management.**
11. Click **Process Definition**.
12. In the **Additional Properties** section, click **Java Virtual Machine.**
13. In **Generic JVM Arguments,** add the following:
```
-Dcom.sun.management.snmp.port=1161 -Dcom.sun.management.snmp.interface=0.0.0.0
```
14. Click **OK.**
15. Click, **Save** and then click **OK.**
16. Click **Save**.
17. Restart the **IBM WebSphere Application Server**.

Configuring Oracle WebLogic (tested on version 10.3.4.0)

Windows

Oracle WebLogic uses its own JDK installed in `%MIDDLEWARE_HOME%\jdk*` (where `%MIDDLEWARE_HOME%` is the path to your WebLogic installation).

1. Go to the `$JAVA_HOME/jre/lib/management` folder
2. Rename the `snmp.acl.template` file to snmp.acl.
3. Edit `snmp.acl` by replacing its content with the following:
```
acl = {
 {
   communities = public, private
   access = read-only
   managers = localhost, apm-server
 }
}
```
Where `apm-server` is the hostname of your APM server.
4. Grant access to the snmp.acl file only for the Administrator. To grant access only for the Administrator,see: http://download.oracle.com/javase/6/docs/technotes/guides/management/security-windows.html

5. Edit the following file:
   `%MIDDLEWARE_HOME%\C:\Oracle\Middleware\user_projects\domains\<your_domain>\bin\startWebLogic.cmd`

   by adding the lines:

   ```
   set "JAVA_OPTIONS=%JAVA_OPTIONS% -Dcom.sun.management.snmp.port=1161"
   set "JAVA_OPTIONS=%JAVA_OPTIONS% -Dcom.sun.management.snmp.interface=0.0.0.0"
   ```

   The result should resemble:

   ```
   …
   call "%DOMAIN_HOME%\bin\setDomainEnv.cmd" %*
   set "JAVA_OPTIONS=%JAVA_OPTIONS% -Dcom.sun.management.snmp.port=1161"
   set "JAVA_OPTIONS=%JAVA_OPTIONS% -Dcom.sun.management.snmp.interface=0.0.0.0"
   set SAVE_JAVA_OPTIONS=%JAVA_OPTIONS%
   …
   ```

   WARNING: This file is created by the Configuration Wizard. Your changes to this script will be lost the next time you use the configuration wizard.

6. Restart WebLogic Server.

Linux

Oracle WebLogic uses its own JDK which in `$MIDDLEWARE_HOME/jdk*` (where `$MIDDLEWARE_HOME` is the path to your WebLogic installation).

1. Go to the `$JAVA_HOME/jre/lib/management` folder.
2. Rename the `snmp.acl.template` file to snmp.acl.
3. Edit `snmp.acl` by replacing its content with the following:

   ```
   acl = {
    {
      communities = public, private
      access = read-only
      managers = localhost, apm-server
    }
   }
   ```

   Where `apm-server` is the hostname of your APM server.
4. Grant access to the `snmp.acl` file only for the root user. Run the following commands as the root user in your shell:

   ```
   # chown root.root $JAVA_HOME/jre/lib/management/snmp.acl
   # chmod 600 $JAVA_HOME/jre/lib/management/snmp.acl
   ```

5. Edit the following file:
   `$MIDDLEWARE_HOME/user_projects/domains/<your_domain>/bin/startWebLogic.sh`

   by adding the lines:

   ```
   JAVA_OPTIONS="$JAVA_OPTIONS -Dcom.sun.management.snmp.port=1161"
   JAVA_OPTIONS="$JAVA_OPTIONS -Dcom.sun.management.snmp.interface=0.0.0.0"
   ```

   The result should resemble:

   ```
   …
   . ${DOMAIN_HOME}/bin/setDomainEnv.sh $*
   JAVA_OPTIONS="$JAVA_OPTIONS -Dcom.sun.management.snmp.port=1161"
   JAVA_OPTIONS="$JAVA_OPTIONS -Dcom.sun.management.snmp.interface=0.0.0.0"
   SAVE_JAVA_OPTIONS="${JAVA_OPTIONS}"
   …
   ```

   WARNING: This file is created by the Configuration Wizard. Your changes to this script will be lost the next time you use the configuration wizard.

6. Restart the WebLogic Server.

Java Application Server (SNMP) Template

This template assesses the overall performance of Java Application Servers by using SNMP protocol. The following application servers are supported: *Apache Tomcat, JBoss, GlassFish, IBM WebSphere* and *Oracle WebLogic.*

Prerequisites: SNMP enabled on the operating system. Target JVM configured to allow SNMP queries. For more information, see "Configuring Java Virtual Machines for SNMP" on page 1.

Note: This template is configured to send SNMP requests on port 1161.

Credentials: None (uses the SNMP public string assigned to the node).

Monitored Components

Some components may not have preset warning or critical threshold values. You can add your own threshold limits as necessary. For more information, see http://knowledgebase.solarwinds.com/kb/questions/2415.

Classes Loaded Count

Indicates the number of classes currently loaded in the JVM.

For more information, reference the following Java method:
`java.lang.management.ClassLoadingMXBean.getLoadedClassCount()`

Classes Total Loaded Count

Indicates the number of classes that have been loaded since the JVM was started.

For more information, reference the following Java methods:
`java.lang.management.ClassLoadingMXBean`

`getTotalLoadedClassCount()`

Classes Unloaded Count

Indicates the number of classes that have been unloaded since the JVM was started.

For more information, reference the following Java method:
`java.lang.management.ClassLoadingMXBean.getUnloadedClassCount()`

Memory Pending Final Count

The approximate number of objects that are pending finalization. This should be as low as possible.

For more information, reference the following Java methods:
`java.lang.management.MemoryMXBean.`

`getObjectPendingFinalizationCount()`

Memory Heap Init Size (B)

The amount of memory (in bytes) that the JVM initially requests from the operating system for memory management used for heap memory pools.

For more information, reference the following Java method:
`java.lang.management.MemoryMXBean.getHeapMemoryUsage()`

`getInit()`

Memory Heap Used (B)

The amount of used memory (in bytes) from heap memory pools. This should be as low as possible.

For more information, reference the following Java method:
`java.lang.management.MemoryMXBean.getHeapMemoryUsage()`

`getUsed()`

Memory Heap Committed (B)

The amount of memory (in bytes) committed by heap memory pools.

For more information, reference the following Java methods:
`java.lang.management.MemoryMXBean.getHeapMemoryUsage().`

`getCommitted()`

Memory Heap Max Size (B)

The maximum amount of memory (in bytes) for all heap memory pools.

For more information, reference the following Java method:
`java.lang.management.MemoryMXBean.getHeapMemoryUsage()`

`getMax()`

Memory Non-heap Init Size (B)

The amount of memory (in bytes) that the JVM initially requests from the operating system for memory management for non-heap memory pools.

For more information, reference the following Java method:
`java.lang.management.MemoryMXBean.getNonHeapMemoryUsage()`

`getInit()`

Memory Non-heap Used (B)

The amount of used memory (in bytes) from non-heap memory pools. This should be as low as possible.

For more information, reference the following Java method:
`java.lang.management.MemoryMXBean.getNonHeapMemoryUsage()`

`getUsed()`

Memory Non-heap Committed (B)

The amount of memory (in bytes) committed by non-heap memory pools.

For more information, reference the following Java methods:
`"java.lang.management.MemoryMXBean."`

`getNonHeapMemoryUsage()`

`getCommitted()`

Memory Non-heap Max Size (B)

The maximum size of memory (in bytes) for all non-heap memory pools.

For more information, reference the following Java method:
`java.lang.management.MemoryMXBean.getNonHeapMemoryUsage()`

`getMax()`

Memory Garbage Collections Count

The number of collections that have occurred, as returned by
`GarbageCollectorMXBean.getCollectionCount()`

If garbage collection statistics are not available, this object is set to 0.

For more information, reference the following Java method:
`java.lang.management.GarbageCollectorMXBean.getCollectionCount()`

Memory Garbage Collection Time (Ms)

The approximate accumulated collection elapsed time (in milliseconds) since the Java virtual machine has started. This object is set to 0 if the collection elapsed time is undefined for this collector.

For more information, reference the following Java method:
`java.lang.management.GarbageCollectorMXBean.getCollectionTime()`

Memory Pool Init Size (B)

The initial size of the memory pool. This counter shows statistics for the first memory pool. If another pool needs monitoring, change the last digit of the OID.

For more information, reference the following Java method:
`java.lang.management.MemoryPoolMXBean.getUsage()`

`getInit()`

Memory Pool Used (B)

The amount of used memory in the memory pool. This counter shows statistics for the first memory pool. If another pool needs monitoring, change the last digit of the OID. This should be as low as possible.

For more information, reference the following Java method:
`java.lang.management.MemoryPoolMXBean.getUsage()`

`getUsed()`

Memory Pool Committed (B)

The amount of committed memory in the memory pool. This counter shows statistics for the first memory pool. If another pool needs monitoring, change the last digit of the OID.

For more information, reference the following Java method:
`java.lang.management.MemoryPoolMXBean.getUsage()`

`getCommitted()`

Memory Pool Max Size (B)

The maximum size of the memory pool. This counter shows statistics for the first memory pool. If another pool needs monitoring, change the last digit of the OID.

For more information, reference the following Java method:
`java.lang.management.MemoryPoolMXBean.getUsage()`

`getMax()`

Memory Pool Peak Used (B)

The amount of used memory in the memory pool at the peak usage point. This counter shows statistics for the first memory pool. If another pool needs monitoring, change the last digit of the OID.

For more information, reference the following Java method:
`java.lang.management.MemoryPoolMXBean.getPeakUsage()`

`getUsed()`

Memory Pool Peak Committed (B)

The amount of used memory in the memory pool at the peak usage point. This counter shows statistics for the first memory pool. If another pool needs monitoring, change the last digit of the OID.

For more information, reference the following Java method:
`java.lang.management.MemoryPoolMXBean.getPeakUsage()`

`getCommitted()`

Memory Pool Peak Max Size (B)

The maximum size of the memory pool at the peak usage point. This counter shows statistics for the first memory pool. If another pool needs monitoring, change the last digit of the OID.

For more information, reference the following Java method:
`java.lang.management.MemoryPoolMXBean.getPeakUsage()`

`getMax()`

Threads Total Started Count

The number of threads created and started since the Java Virtual Machine started. This counter shows statistics for the first thread. If another thread needs monitoring, change the last digit of the OID.

For more information, reference the following Java method:
`java.lang.management.ThreadMXBean.getTotalStartedThreadCount()`

Thread Instant Blocked Count

The number of times that this thread has blocked the “to enter” or “re-enter” monitor. This counter shows statistics for the first thread. If another thread needs monitoring, change the last digit of the OID.

For more information, reference the following Java methods:
`java.lang.management.ThreadMXBean.getThreadInfo(long,boolean)`

`getBlockedCount()`

Thread Instant Blocked Time (Ms)

The approximate accumulated elapsed time (in milliseconds) that a thread has blocked the “to enter” or “re-enter” monitor since it has started, or since thread contention monitoring was enabled.

This object is always set to 0 if thread contention monitoring is disabled or not supported. This counter shows statistics for the first thread. If another thread needs monitoring, change the last digit of the OID. This should be as low as possible.

For more information, reference the following Java method:
`java.lang.management.ThreadMXBean.getThreadInfo(long,boolean)`

`getBlockedTime()`

Thread Instant Wait Count

This counter shows the number of times that this thread waited for notification. It also shows statistics for the first thread. If another thread needs monitoring, change the last digit of the OID.

For more information, reference the following Java methods:
`java.lang.management.ThreadMXBean.getThreadInfo(long,boolean).`

`getWaitedCount()`

Thread Instant Wait Time (Ms)

The approximate accumulated elapsed time (in milliseconds) that a thread has waited for a monitor through a “java.lang.Object.wait” method since it has started, or since thread contention monitoring was enabled.

This object is always set to 0 if thread contention monitoring is disabled or not supported. It also shows statistics for first thread. If another thread needs monitoring, change the last digit of the OID. This should be as low as possible.

For more information, reference the following Java methods:
`java.lang.management.ThreadMXBean.getThreadInfo(long,boolean)`

`getWaitedTime()`

Runtime Input Arguments Count

This is the number of input arguments passed to the Java Virtual Machine.

For more information, reference the following Java method:
`java.lang.management.RuntimeMXBean.getInputArguments()`

Runtime Uptime (Ms)

This is the uptime of the Java virtual machine (in milliseconds). This is equivalent to (System.currentTimeMillis() - jvmStartTimeMs).

For more information, reference the following Java methods:
`jvmRTStartTimeMs.:`

`java.lang.management.RuntimeMXBean.getUptime()`

Compiler Time (Ms)

This gets the approximate accumulated elapsed time (in milliseconds) spent in compilation since the Java virtual machine has started. If multiple threads are used for compilation, this value is the summation of the approximate time that each thread spent in compilation. If compiler time monitoring is not supported, then this object remains set at 0.

For more information, reference the following Java method:
`java.lang.management.CompilationMXBean.getTotalCompilationTime()`