Perfmon4j Sample Configurations

Configuration Guide

Purpose

This document contains samples of XML configuration for Perfmon4j. This document is a work in progress. As specific configurations come up they are appended here.

Please help keep this document relevant. If you have problems implementing any of these examples please send feedback to perfmon4j@fsc.follett.com. Feel free to send comments, suggestions and questions – we appreciate any and all feedback.

Example 1 – Monitor duration for all methods on a class

This example shows how you can use Perfmon4j to capture duration/throughput measurement for all methods on a given class. For the purpose of this example we will capture information regarding the `org.apache.catalina.connector.Request` class. Typically you will instrument your own java classes, however any class can be instrumented (Access to source code is NOT required). Note the class used in this example requires an Apache/Tomcat or JBoss application server.

Configuration

The first step is to ensure that the class is instrumented by the Perfmon4j javaagent. Basic configuration of the javaagent is documented in the Perfmon4j Apache/Tomcat or JBoss configuration guides. To instrument all of the methods of a class, the name of the class OR any parent package of the class must be indicated via the repeatable **–e** option. For this example **append** the value specified in bold below to your javaagent string.

Javaagent declaration

```
-javaagent:..\lib\endorsed\perfmon4j.jar=-f..\bin\perfmonconfig.xml,-eorg.apache.catalina.connector
```

A monitor on the class must be configured and attached to an appender. In this example every method invoked on the `org.apache.catalina.connector.Request` method will be monitored (Based on the child only pattern `'/*'`). The result of any method invoked will be written to the server log every minute. Note: the method must execute at least 1 time for the monitor to be initialized.

Perfmonconfig.xml

```
<Perfmon4JConfig enabled='true'>
     <appender name='high-res' className='org.perfmon4j.TextAppender' interval='1 minutes'/>

     <monitor name='org.apache.catalina.connector.Request'>
            <appender name='hi-res'  pattern='/*'/>
     </monitor>
</Perfmon4JConfig>
```

Sample Output

Access the server via a browser through a servlet request. Within 1 minute output similar to the following should be written to the server log.


```
2009-09-21 14:18:13,791 INFO  [org.perfmon4j.TextAppender]
*************************************************************************
org.apache.catalina.connector.Request.getAttribute
14:17:11:531 -> 14:18:11:531
 Max Active Threads. 1 (2009-09-21 14:18:08:764)
 Throughput..... 254.00 per minute
 Average Duration... 0.00
 Standard Deviation. 0.06
 Max Duration..... 1 (2009-09-21 14:17:38:002)
 Min Duration..... 0 (2009-09-21 14:17:38:002)
 Total Hits..... 254
 Total Completions.. 254
Lifetime (2009-09-21 14:17:11):
 Max Active Threads. 1 (2009-09-21 14:18:08:764)
 Max Throughput..... 254.00 (2009-09-21 14:17:11 -> 2009-09-21 14:18:11)
 Average Duration... 0.00
 Standard Deviation. 0.06
 Max Duration..... 1 (2009-09-21 14:17:38:002)
 Min Duration..... 0 (2009-09-21 14:18:08:764)
*************************************************************************
2009-09-21 14:18:13,833 INFO  [org.perfmon4j.TextAppender]
*************************************************************************
org.apache.catalina.connector.Request.removeAttribute
14:17:11:540 -> 14:18:11:540
 Max Active Threads. 1 (2009-09-21 14:18:08:764)
 Throughput..... 72.00 per minute
 Average Duration... 0.00
 Standard Deviation. 0.00
 Max Duration..... 0 (2009-09-21 14:18:08:764)
 Min Duration..... 0 (2009-09-21 14:18:08:764)
 Total Hits..... 72
 Total Completions.. 72
Lifetime (2009-09-21 14:17:11):
 Max Active Threads. 1 (2009-09-21 14:18:08:764)
 Max Throughput..... 72.00 (2009-09-21 14:17:11 -> 2009-09-21 14:18:11)
 Average Duration... 0.00
 Standard Deviation. 0.00
 Max Duration..... 0 (2009-09-21 14:18:08:764)
 Min Duration..... 0 (2009-09-21 14:18:08:764)
*************************************************************************
```

Example 2 – Monitor composite duration for a class

This example builds on the above example “Monitor duration for all methods on a class”. In this example we will monitor the composite duration/throughput metrics for all methods on a given class. This will show us how much time our application code is spending in a given class.

Configuration

Follow the configuration example specified above in the example “Monitor duration for all methods on a class”. The only change required for this example is to modify the pattern attribute. We will change the pattern from `'/*'` to `'.'`. A period indicates “parent monitor only”, in our example metrics associated with the class.

If you are making this change after performing the previous example you will NOT need to restart your application server. Changes to the perfmonconfig.xml file will be dynamically reloaded every 60 seconds.

Perfmonconfig.xml

```
<Perfmon4JConfig enabled='true'>
     <appender name='high-res' className='org.perfmon4j.TextAppender' interval='1 minutes'/>

     <monitor name='org.apache.catalina.connector.Request'>
            <appender name='hi-res'  pattern='.'/>
     </monitor>
</Perfmon4JConfig>
```

Sample Output

Access the server via a browser through a servlet request. Within 1 minute (up to 2 minutes if you did not restart the application server) output similar to the following should be written to the server log.


```
2009-09-21 14:44:25,414 INFO  [org.perfmon4j.TextAppender]
**************************************************************************
org.apache.catalina.connector.Request
14:43:25:343 -> 14:44:25:353
 Max Active Threads. 1 (2009-09-21 14:44:24:529)
 Throughput......... 280.95 per minute
 Average Duration... 0.00
 Standard Deviation. 1.98
 Max Duration....... 33 (2009-09-21 14:44:04:209)
 Min Duration....... 0 (2009-09-21 14:44:04:209)
 Total Hits......... 281
 Total Completions.. 281
Lifetime (2009-09-21 14:43:25):
 Max Active Threads. 1 (2009-09-21 14:44:24:529)
 Max Throughput..... 280.95 (2009-09-21 14:43:25 -> 2009-09-21 14:44:25)
 Average Duration... 0.00
 Standard Deviation. 1.98
 Max Duration....... 33 (2009-09-21 14:44:04:209)
 Min Duration....... 0 (2009-09-21 14:44:24:529)
**************************************************************************
```

Example 3 – JVMSnapShot monitor

This example will take a snapshot of the following Java management objects (`ThreadMXBean`, `ClassLoadingMXBean`, `CompilationMXBean`, `OperatingSystemMXBean`, `and` `MemoryMXBean`) and write the output to the system log every minute.

Configuration – Perfmonconfig.xml

```
<Perfmon4JConfig enabled='true'>
     <appender name='high-res' className='org.perfmon4j.TextAppender' interval='1 minutes'/>

     <snapShotMonitor name='JVMSnapShot' className='org.perfmon4j.java.management.JVMSnapShot'>
            <appender name='high-res'/>
     </snapShotMonitor>
</Perfmon4JConfig>
```

Sample Output

```
**************************************************************************
JVMSnapShot
17:02:28:311 -> 17:03:28:311
 daemonThreadCount........ 29
 threadCount.............. 46
 totalLoadedClassCount.... 409.0/per minute
 unloadedClassCount....... 0.0/per minute
 systemLoadAverage........ -1.0
 classesLoaded............ 15851
 compilationTime.......... 1607.0/per minute
 compilationTimeActive.... true
 heapMemUsed.............. 405.631 MB
 heapMemCommitted......... 506.125 MB
 nonHeapMemUsed........... 200.857 MB
 nonHeapMemCommitted...... 201.875 MB
 pendingFinalization...... 0
 threadsStarted........... 5.0/per minute
**************************************************************************
```

Example 4 – GarbageCollectorSnapShot monitor

The garbage collector snapshot monitor provides a view of the Java management object (`GarbageCollectorMXBean`). The JVM contains one or more garbage collectors based on the active configuration. The GarbageCollectorSnapShot monitor can be configured to display composite data from all active garbage collectors or view an individual collector. This example contains a monitor for all collectors and the Old gen collector.

Configuration – Perfmonconfig.xml


```
<Perfmon4JConfig enabled='true'>
      <appender name='high-res' className='org.perfmon4j.TextAppender'
                interval='1 minutes'/>
      <snapShotMonitor name='Composite Garbage Collector'
className='org.perfmon4j.java.management.GarbageCollectorSnapShot'>
               <appender name='high-res'/>
      </snapShotMonitor>


      <snapShotMonitor name=' MarkSweep Garbage Collector'
className='org.perfmon4j.java.management.GarbageCollectorSnapShot'>
               <appender name='high-res'/>
               <attribute name='instanceName'>PS Scavenge</attribute>
      </snapShotMonitor>
</Perfmon4JConfig>
```

Sample Output

The composite collector contains composite information for all collectors. Note the `'monitorName'` attribute displays all of the active collectors. Each active collector can be monitored individually by specifying the collector name as the attribute `instanceName` in the `perfmonconfig.xml` file.

```
********************************************************************************
Composite Garbage Collector
15:23:43:863 -> 15:24:44:446
 monitorName.............. Composite("PS Scavenge", "PS MarkSweep")
 collectionCount.......... 3.9615073535480247/per minute
 collectionTime........... 7836.851922156381/per minute
********************************************************************************
```

The `PS MarkSweep` collector shows the individual information for the mark and sweep collector.

```
********************************************************************************
MarkSweep Garbage Collector
15:23:43:865 -> 15:24:44:446
 monitorName.............. PS MarkSweep
 collectionCount.......... 0.9904095343424506/per minute
 collectionTime........... 7229.98960069989/per minute
********************************************************************************
```

Example 5 – MemoryPoolSnapShot monitor

The memory pool snapshot monitor provides a view of the Java management object (`MemoryPoolMXBean`). The JVM contains one or more memory pools based on the active configuration. The MemoryPoolSnapShot monitor can be configured to display composite data from all active memory pools or view an individual pool. This example contains a monitor for all pools and the Old Gen pool.

Configuration – Perfmonconfig.xml

```
<Perfmon4JConfig enabled='true'>
      <appender name='high-res' className='org.perfmon4j.TextAppender'
                interval='1 minutes'/>
      <snapShotMonitor name='Composite Memory Pool'
               className='org.perfmon4j.java.management.MemoryPoolSnapShot'>
               <appender name='high-res'/>
      </snapShotMonitor>

      <snapShotMonitor name='Old Gen Memory Pool'
               className='org.perfmon4j.java.management.MemoryPoolSnapShot'>
               <attribute name='instanceName'>PS Old Gen</attribute>
               <appender name='high-res'/>
      </snapShotMonitor>
</Perfmon4JConfig>
```

Sample Output

The composite memory pool contains composite information for all pools. Note the `'monitorName'` attribute displays all of the active pools. Each active pool can be monitored individually by specifying the pool name as the attribute `instanceName` in the `perfmonconfig.xml` file.


```
*************************************************************************
Composite Memory Pool
16:52:31:680 -> 16:53:31:681
 type..................... (N/A)
 init..................... 513.875 MB
 used..................... 600.293 MB
 committed................ 753.938 MB
 max...................... 789.000 MB
 monitorName.............. Composite("Code Cache", "PS Eden Space", "PS Survivor Space", "PS Old Gen", "PS Perm
Gen")
 usedCommittedRatio....... 79.620995%
 usedMaxRatio............. 76.0827%
*************************************************************************
```

The `PS Old Gen` collector shows the individual information for the old gen memory pool.

```
*************************************************************************
Old Gen Memory Pool
16:52:31:770 -> 16:53:31:789
 type..................... Heap memory
 init..................... 341.375 MB
 used..................... 336.254 MB
 committed................ 341.375 MB
 max...................... 341.375 MB
 monitorName.............. PS Old Gen
 usedCommittedRatio....... 98.499825%
 usedMaxRatio............. 98.499825%
*************************************************************************
```

Example 6 – Perfmon4j Instrumentation monitor

This monitor exposes the internals of the Perfmon4j instrumentation agent. It displays raw counters detailing the number of classes instrumented.

Configuration – Perfmonconfig.xml

```
<Perfmon4JConfig enabled='true'>
     <appender name='high-res' className='org.perfmon4j.TextAppender'
               interval='1 minutes'/>
     <snapShotMonitor name='PerfmonInstrumentation' className='org.perfmon4j.instrument.InstrumentationMonitor'>
                    <appender name='high-res'/>
     </snapShotMonitor>
</Perfmon4JConfig>
```

Sample Output

```
*************************************************************************
PerfmonInstrumentation
21:44:30:926 -> 21:45:30:926
 numClassesInst........... 0.0/per minute
 totalClassesInst......... 1941
 numMethodsInst........... 0.0/per minute
 totalMethodsInst......... 15412
 numClassInstFailures..... 0.0/per minute
 totalClassInstFailures... 0
 numBootstrapClassesInst.. 0.0/per minute
 totalBootstrapClassesIns. 1
 instrumentationMillis.... 0.0/per minute
 totalInstrumentationMill. 57169
*************************************************************************
```

Example 7 – Configure JBoss Server Application Code for Extreme logging.

In rare instances some classes are not compatible with Perfmon4j instrumentation. In most cases Perfmon4j will simply skip these classes. The `org.jboss.security` package contains 1 or more classes that must be explicitly excluded from instrumentation. The following configuration will instrument all JBoss classes except the security classes.

Notes:

- See the Perfmon4j-JBossConfigurationGuide for details on how to install the java agent.
- `On` JBoss 5.x/64bit the default `MaxPermSize` is not sufficient for instrumentation. The `MaxPermSize` parameter increases the default – this parameter is most likely not required under a 32bit JVM.

Configuration – Javaagent declaration

```
SET JAVA_OPTS=-javaagent:..\lib\endorsed\perfmon4j.jar=-eorg.jboss,-iorg.jboss.security,-f..\bin\perfmonconfig.xml
-XX:MaxPermSize=256m
```

Example 8 – Output Interval data to a SQL Database (1.0.1.GA+)

The `JDBCSQLAppender` and `PooledSQLAppender` appenders, introduced in perfmon4j version 1.0.1.GA, allow interval data to be written to a SQL database. The `JDBCSQLAppender` allows configuration through a JDBC Driver. The `PooledSQLAppender` can be configured to connect to a JNDI based connection pool.

Configuration

Step 1 – Create Database Tables

The first step in configuration is to create the Perfmon4j tables within an existing database. The `MSSQL-CreateTable.sql` and `MySQL-CreateTable.sql` file are SQL scripts to create these tables in Microsoft SQL and MySQL respectively. With minimal modification one of these scripts could be ported to most SQL compatible databases.

Step 2) Perfmonconfig.xml

There are two options for configuring the SQL based appender. Option 1 will create a connection to the database through the specified JDBC Driver. Option 2 will use a JNDI lookup to acquire a connection pool.

Perfmonconfig.xml (Option 1 – JDBCDriver)

```
<Perfmon4JConfig enabled='true'>
<appender name='jdbcAppender' className='org.perfmon4j.JDBCSQLAppender' interval='5 minutes'>
            <attribute name='driverClass'>net.sourceforge.jtds.jdbc.Driver</attribute>

            <!-- driverPath is optional… If this parameter is not specified the driver
                    Must exist in the classpath of the java process -->
            <attribute name='driverPath'>c:/drivers/jtds.jar</attribute>

            <attribute name='jdbcURL'>jdbc:jtds:sqlserver:/localhost/perfmon4j</attribute>

            <!-- dbSchema is optional… If this parameter is not specified
                    the tables must exist in the users default schema -->
            <attribute name='dbSchema'>myschema</attribute>

            <attribute name='userName'>myusername</attribute>
            <attribute name='password'>mypassword</attribute>

            <attribute name='medianCalculator'>factor=10</attribute>
            <attribute name='thresholdCalculator'>2 seconds, 5 seconds, 10 seconds</attribute>
    </appender>

    <monitor name='WebRequest'>
            <appender name='jdbcAppender' pattern='.'/>
    </monitor>
</Perfmon4JConfig>
```

Perfmonconfig.xml (Option 2 –JNDI Based connection pool)

```
<Perfmon4JConfig enabled='true'>

    <-- IMPORTANT – This appender will NOT output and should not be attached to SnapShot Monitors -->
    <appender name='pooledAppender' className='org.perfmon4j.PooledSQLAppender' interval='5 Minutes'>
            <attribute name='poolName'>java:/MSSQLDS</attribute>
            <!-- dbSchema is optional… If this parameter is not specified
                    the tables must exist in the users default schema -->
            <attribute name='dbSchema'>myschema</attribute>

            <attribute name='medianCalculator'>factor=10</attribute>
            <attribute name='thresholdCalculator'>2 seconds, 5 seconds, 10 seconds</attribute>
    </appender>
    <monitor name='WebRequest'>
            <appender name='pooledAppender' pattern='.'/>
    </monitor>
</Perfmon4JConfig>
```

Example 9 – Output UserAgent (Browser Summary) data to a SQL Database (1.0.2.GA+)

Perfmon4j can be configured to capture browser user agent data (Browser type/version, Client OS type/version) for each web request.

This example shows how to use the `UserAgentSnapShotMonitor` and the `SQLAppender` to capture client information and store the results in a SQL database. See (Example 8 in this document) for information on configuring a SQL database for logging.

Configuration

**** IMPORTANT!! ****

To capture user agent information you MUST install the Perfmon4j servlet filter or the Tomcat Valve (Instructions for installing the tomcat valve can be found in the JBoss and Tomcat setup guides).

Perfmonconfig.xml

```
<Perfmon4JConfig enabled='true'>

<appender name='userAgentAppender' className='org.perfmon4j.JDBCSQLAppender' interval='10 minutes'>
              <attribute name='driverClass'>net.sourceforge.jtds.jdbc.Driver</attribute>
              <attribute name='driverPath'>c:/drivers/jtds.jar</attribute>
              <attribute name='jdbcURL'>jdbc:jtds:sqlserver:/localhost/perfmon4j</attribute>
              <attribute name='dbSchema'>myschema</attribute>
              <attribute name='userName'>myusername</attribute>
              <attribute name='password'>mypassword</attribute>
      </appender>

      <snapShotMonitor name='UserAgent' className='org.perfmon4j.UserAgentSnapShotMonitor'>
              <appender name='userAgentAppender'/>
      </snapShotMonitor>

</Perfmon4JConfig>
```

Output

You can view the output of this by running a query against the P4JUserAgentView. This view groups the total number of hits per browser configuration by day.

```
SQL-> SELECT * FROM P4JUserAgentView
CollectionDate       BrowserName     BrowserVersion  OSName          OSVersion       RequestCount
-------------------------------------------------------------------------------------------------
2010-3-20            Firefox         3.6             Windows NT      6.0             567
2010-3-20            MSIE            8.0             Windows NT      6.0             443
…
```

Example 10 – Setup a ThreadTrace for detailed peformance logging

One of perfmon4j's most powerful features is Thread Trace logging. Thread trace logging allows you to collect a detailed thread stack detailing the performance of a single thread. This can be extremely helpful in trouble shooting bottlenecks in a web application. For example if a particular web request that is performing poorly you can profile a single execution of that thread and discover the bottleneck.

This example was implemented on Apache Tomcat 6.0 and uses the "Hello World" servlet example. To reproduce simply follow the steps detailed in the "Perfmon4j-Tomcat-ConfigGuide" and add the configuration settings shown here.

Javaagent declaration

The first step is to enable extreme logging on the application packages you are interested in timing.

```
-javaagent:../lib/endorsed/perfmon4j.jar=-f../conf/perfmonconfig.xml,-eorg.apache
```

Perfmonconfig.xml

The next step is to configure a thread trace on the "examples/servlets" web request URL. Note: Perfmon4j servlet filter or the Tomcat Valve must be installed.


```
<Perfmon4JConfig enabled='true'>
      <appender name='myappender' className='org.perfmon4j.TextAppender'
              interval='1 minutes'/>

      <threadTrace monitorName="WebRequest.examples.servlets"
              randomSamplingFactor="5"
              maxDepth="10"
              minDurationToCapture=”5 ms”>
              <appender name='myappender'/>
      </threadTrace>
</Perfmon4JConfig>
```

Two optional attributes are included on the threadTrace configuration:

- `randomSamplingFactor=”5”` indicates that a detailed thread trace snapshot should be taken approximately once over every 5 occurrences.
- `maxDepth=”10”` indicates the maximum desired stack depth to capture.
- `minDurationToCapture=’5 ms’` reduces the stack trace to branches with a total duration of 5 milliseconds or greater.

Output

If you invoke the “Hello World” servlet example 5-10 times you will see output similar to the following:

Notice the numbers in parenthesis. This is the time in milliseconds taken for the invocation of the code segment.

```
+-10:57:36:509 (25) WebRequest.examples.servlets
|     +-10:57:36:509 (15) org.apache.catalina.core.StandardEngineValve.invoke
|     |       +-10:57:36:509 (11) org.apache.catalina.valves.ErrorReportValve.invoke
|     |       |       +-10:57:36:509 (5) org.apache.catalina.core.StandardHostValve.invoke
|     |       |       |
... Output truncated ...
|     |       |       +-10:57:36:509 org.apache.catalina.connector.Request.getAttribute
|     |       |       +-10:57:36:509 (5) org.apache.catalina.valves.ErrorReportValve.report
|     |       |       +-10:57:36:509 org.apache.catalina.valves.ErrorReportValve.report
|     |       +-10:57:36:509 org.apache.catalina.valves.ErrorReportValve.invoke
|     +-10:57:36:509 org.apache.catalina.core.StandardEngineValve.invoke
+-10:57:36:509 WebRequest.examples.servlets
```

Example 11 –Trigger ThreadTrace by request parameter (1.0.2.GA+)

This example extends on the configuration in Example 10. The previous example used a random sampling factor to limit the number of request that produce a detailed stack trace. This example will only produced a stack trace when a request is received containing a specific request parameter/value.

This example was implemented on Apache Tomcat 6.0 and uses the “Request Parameters” servlet example. To reproduce simply follow the steps detailed in the “Perfmon4j-Tomcat-ConfigGuide” and add the configuration settings shown here.

Javaagent declaration

The first step is to enable extreme logging on the application packages you are interested in timing.

```
-javaagent:../lib/endorsed/perfmon4j.jar=-f../conf/perfmonconfig.xml,-eorg.apache
```

Perfmonconfig.xml

The next step is to configure a thread trace on the “examples/servlets” web request URL. Note: Perfmon4j servlet filter or the Tomcat Valve must be installed.


```
<Perfmon4JConfig enabled='true'>
      <appender name='myappender' className='org.perfmon4j.TextAppender'
              interval='1 minutes'/>

      <threadTrace monitorName="WebRequest.examples.servlets" maxDepth="10">
              <appender name='myappender'/>
              <Triggers>
                      <HTTPRequestTrigger name="firstname" value="Dave"/>
              </Triggers>
      </threadTrace>
</Perfmon4JConfig>
```

Output

If you invoke the “Request Parameters” servlet with “First Name” set to a value of “Dave” you will get the following output.

Notice the numbers in parenthesis. This is the time in milliseconds taken for the invocation of the code segment.

```
+-10:57:36:509 (25) WebRequest.examples.servlets
|     +-10:57:36:509 (15) org.apache.catalina.core.StandardEngineValve.invoke
|     |       +-10:57:36:509 (5) org.apache.catalina.valves.ErrorReportValve.invoke
|     |       |       +-10:57:36:509 (0) org.apache.catalina.core.StandardHostValve.invoke
|     |       |       |
... Output truncated ...
|     |       |       +-10:57:36:509 org.apache.catalina.connector.Request.getAttribute
|     |       |       +-10:57:36:509 (0) org.apache.catalina.valves.ErrorReportValve.report
|     |       |       +-10:57:36:509 org.apache.catalina.valves.ErrorReportValve.report
|     |       +-10:57:36:509 org.apache.catalina.valves.ErrorReportValve.invoke
|     +-10:57:36:509 org.apache.catalina.core.StandardEngineValve.invoke
+-10:57:36:509 WebRequest.examples.servlets
```

Example 12 –Trigger ThreadTrace by cookie (1.0.2.GA+)

This example extends on the configuration in Example 10. The previous example used a random sampling factor to limit the number of request that produce a detailed stack trace. This example will only produced a stack trace when a request is received containing a specific cookie/value.

This example was implemented on Apache Tomcat 6.0 and uses the “Cookies” servlet example. To reproduce simply follow the steps detailed in the “Perfmon4j-Tomcat-ConfigGuide” and add the configuration settings shown here.

Javaagent declaration

The first step is to enable extreme logging on the application packages you are interested in timing.

```
-javaagent:../lib/endorsed/perfmon4j.jar=-f../conf/perfmonconfig.xml,-eorg.apache
```

Perfmonconfig.xml

The next step is to configure a thread trace on the “examples/servlets” web request URL. Note: Perfmon4j servlet filter or the Tomcat Valve must be installed.

```
<Perfmon4JConfig enabled='true'>
      <appender name='myappender' className='org.perfmon4j.TextAppender'
              interval='1 minutes'/>

      <threadTrace monitorName="WebRequest.examples.servlets" maxDepth="10">
              <appender name='myappender'/>
              <Triggers>
                      <HTTPCookieTrigger name="MyCookie" value="1"/>
              </Triggers>
      </threadTrace>
</Perfmon4JConfig>
```

Output

If you invoke the “Cookies” servlet with “Name” set to a value of “MyCookie” and “Value” set to “1” you will get the following output.

Notice the numbers in parenthesis. This is the time in milliseconds taken for the invocation of the code segment.

```
+-10:57:36:509 (25) WebRequest.examples.servlets
|    +-10:57:36:509 (15) org.apache.catalina.core.StandardEngineValve.invoke
|    |       +-10:57:36:509 (5) org.apache.catalina.valves.ErrorReportValve.invoke
|    |       |       +-10:57:36:509 (0) org.apache.catalina.core.StandardHostValve.invoke
|    |       |       |
... Output truncated ...
|    |       |       +-10:57:36:509 org.apache.catalina.connector.Request.getAttribute
|    |       |       +-10:57:36:509 (0) org.apache.catalina.valves.ErrorReportValve.report
|    |       |       +-10:57:36:509 org.apache.catalina.valves.ErrorReportValve.report
|    |       +-10:57:36:509 org.apache.catalina.valves.ErrorReportValve.invoke
|    +-10:57:36:509 org.apache.catalina.core.StandardEngineValve.invoke
+-10:57:36:509 WebRequest.examples.servlets
```

Example 12 – Monitor Arbitrary JMX Attributes.

The JMXSnapShotProxyFactory class provides the ability to monitor any JMX accessible attributes. The following example demonstrates monitoring the attributes associated with the jboss.web:name=httpThreadPool,type=Executor JMX object. For more examples of the JMXWrapper configuration see the unit tests in the JMXSnapShotProxyFactoryTest.java test file.

Configuration – Perfmonconfig.xml

```
<Perfmon4JConfig enabled='true'>
     <snapShotMonitor name='httpsThreadPool.Executor'
             className='org.perfmon4j.instrument.jmx.JMXSnapShotProxyFactory'>
             <attribute name='jmxXML'><![CDATA[
                     <JMXWrapper defaultObjectName='jboss.web:name=httpThreadPool,type=Executor'>
                             <attribute name='largestPoolSize'/>
                             <attribute name='activeCount'/>
                             <attribute name='TotalTaskCount' jmxName='completedTaskCount'/>
                             <attribute name='completedTasks'  jmxName='completedTaskCount'>
                                     <snapShotCounter formatter='org.perfmon4j.util.NumberFormatter'
                                             display='DELTA_PER_MIN'/>
                             </attribute>
                             <attribute name='poolSize'/>
                     </JMXWrapper>
             ]]></attribute>
             <appender name='snapshot-appender'/>
     </snapShotMonitor>
</Perfmon4JConfig>
```

Sample Output

```
2010-09-28 17:47:28   INFO    TextAppender    ()
********************************************************************************
httpThreadPool.Executor
17:46:24:550 -> 17:47:24:550
 largestPoolSize.......... 30
 activeCount.............. 0
 totalTaskCount........... 44
 completedTasks........... 2.340/per minute
 poolSize................. 30
********************************************************************************
```

Example 13 – Log Request Duration and Parameters

Either the Perfmon4j valve or Perfmon4j servlet filter can be used to log out each web request and the duration associated with the request.

Configuration - (Option 1) PerfMonValve

The PerfMonValve is available for JBoss or Tomcat installations. See the Perfmon4j-JBossConfigGuide or Perfmon4j-Tomcat-ConfigGuide for details on how to install the valve. The valve accepts the following parameter to turn on request logging.

```
<Valve className="org.perfmon4j.extras.tomcat55.PerfMonValve"
     outputRequestAndDuration='true'/>
```

Sample Output

```
Oct 4, 2010 9:10:04 PM org.perfmon4j.util.JavaLoggingLogger logInfo
INFO: 22 /examples/servlets/servlet/RequestParamExample?lastname=Smith&firstname=John
```

Example 14 – Evaluate SQL/JDBC Duration (Version 1.1.0+)

This example demonstrates how to configure SQL/JDBC monitoring. SQL Monitoring allows you to track the percentage of processing time spent within the database layer

Configuration

The first step is to configure the Perfmon4j java agent to enable instrumentation of the JDBC classes. This is accomplished via the -eSQL command line option.

```
-javaagent:../lib/perfmon4j.jar=-ecom.follett.fsc,-eSQL
```

Restarting the application will now include SQL durations in each interval monitor. For example following perfmonconfig.xml

```
<Perfmon4JConfig enabled='true'>
      <appender name='default' className='org.perfmon4j.TextAppender' interval='1 minutes'>
      </appender>

      <monitor name='WebRequest'>
              <appender name='default' pattern="."/>
      </monitor>

      <threadTrace monitorName='WebRequest'
              randomSamplingFactor='5'
              maxDepth='10'
              minDurationToCapture='15'>
              <appender name='default'/>
      </threadTrace>
</Perfmon4JConfig>
```

will produce the following output. Note: the breakout of the SQL duration on both the interval data and the thread trace.

Sample Output

```
2010-10-06 16:20:50   INFO    TextAppender    ()
********************************************************************************
WebRequest
16:19:49:998 -> 16:20:50:031
 Max Active Threads. 1 (2010-10-06 16:20:05:183)
 Throughput..... 26.99 per minute
 Average Duration... 45.00
 Standard Deviation. 17.33
 Max Duration..... 83 (2010-10-06 16:19:52:776)
 Min Duration..... 26 (2010-10-06 16:19:52:776)
 Total Hits..... 27
 Total Completions.. 27
 (SQL)Avg. Duration. 18.00
 (SQL)Std. Dev..... 3.75
 (SQL)Max Duration.. 30 (2010-10-06 16:19:58:569)
 (SQL)Min Duration.. 12 (2010-10-06 16:19:51:622)
********************************************************************************
2010-10-06 16:19:57   INFO    TextAppender    ()
********************************************************************************
+-16:19:57:351 (54)(SQL:21) WebRequest
|     +-16:19:57:352 (53)(SQL:21) com.follett.fsc.destiny.client.filters.DestinyFilter.doFilter
|     |       +-16:19:57:352 (53)(SQL:21) com.follett.fsc.destiny.client.filters.LogFilter.doFilter
. . . Output truncated . . .
|     |       +-16:19:57:405 com.follett.fsc.destiny.client.filters.LogFilter.doFilter
|     +-16:19:57:405 com.follett.fsc.destiny.client.filters.DestinyFilter.doFilter
+-16:19:57:405 WebRequest
********************************************************************************
```


