Platform Statistics Recorder Guide

About Terracotta Documentation

This documentation is about Terracotta DSO, an advanced distributed-computing technology aimed at meeting special clustering requirements.

Terracotta products without the overhead and complexity of DSO meet the needs of almost all use cases and clustering requirements. To learn how to migrate from Terracotta DSO to standard Terracotta products, see Migrating From Terracotta DSO. To find documentation on non-DSO (standard) Terracotta products, see Terracotta Documentation. Terracotta release information, such as release notes and platform compatibility, is found in Product Information.

The Platform Statistics Recorder

Release: 3.6
Publish Date: November, 2011 **Documentation Archive »**

- Troubleshooting

Introduction

The Terracotta Platform Statistics Recorder allows you to configure and manage the recording of statistics for your entire cluster. This guide describes the Statistics Recorder in detail and shows you how to work with it.

With the Statistics Recorder, you can obtain detailed information about every node in a running Terracotta cluster, and correlate the results. Initial use of the Statistics Recorder is non-intrusive because no application restart is required to begin capturing statistics. However, note that live collection of statistics could degrade the performance of your system, and therefore the Statistics Recorder may not be an appropriate tool in a critical or load-sensitive production environment.

The Statistics Recorder comes ready to use. However, in certain environments, it may be necessary to change its configuration. See Configuration for more information.

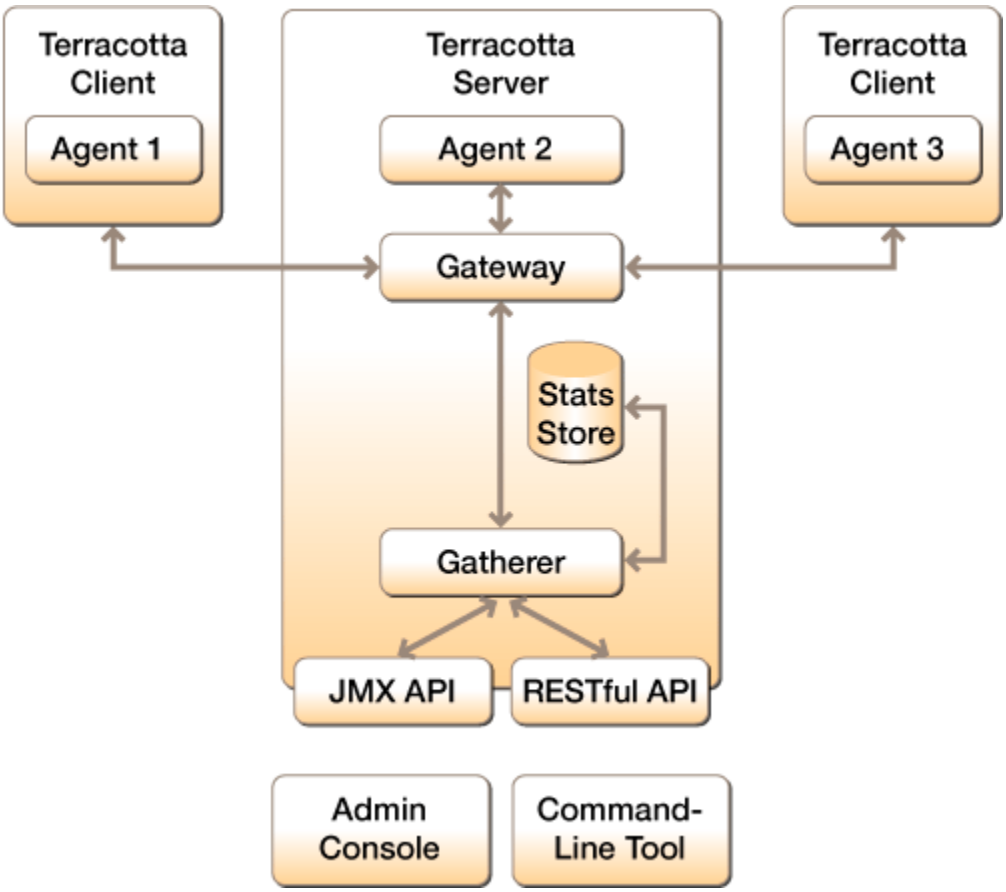
If you are ready to use the Statistics Recorder, you can do so in any one of the following ways:

- By launching the Terracotta Developer Console
- By opening a terminal window and using the command-line interface
- Through the RESTful interface

To learn about the architecture of the statistics recorder, begin with the Architecture section below.

Architecture

The architecture of the statistics recorder is illustrated by the following schematic.



The recorder's major components are defined below.

Agents

Each node in a Terracotta cluster is able to record statistics about its running system using an internal statistics capture agent. Agents run in parallel with the rest of the functionalities of your application. They are able to retrieve and to emit the data of the statistics that have been enabled in a capture session. Since the capturing and the emitting of the data is done independently in the agents, the statistics data is spooled to a disk buffer before being sent.

Agents are identified in two ways: their IP address, and a differentiator. The differentiator is generated by the environment in which the agent runs to distinguish among several agents that might be running in different JMVs on the same machine.

Gateway

Since the nodes in the cluster can come and go, the actual topology is dynamic. It would be quite tedious to have to keep track of each note individually before being able to perform the statistics recording. This function is therefore performed by the gateway, which sits in front of all the nodes and groups together the functionalities into a single API. When interacting with the gateway, all the actions are always performed on all the nodes that are part of the cluster. Also, when the statistics capturing session is ongoing and a new node joins, the gateway will automatically configure that node and starts recording statistics on it also. Basically, the gateway makes the whole cluster look like a single machine.

Capture sessions

Before being able to start capturing, you need to create a capturing session with a unique identifier. This session allows you to setup which statistics are enabled and also to configure options that should be taken into account when the data capturing is performed. The unique identifier makes it possible to later group the data from all the individual nodes into a same capturing session.

Gatherer

The Terracotta server embeds a statistics gatherer that connects to the gateway. This gatherer will receive all the statistics data that will be sent by the gateway and stores this locally in embedded database. The data of the capturing sessions will thus end up in the gatherer, ready to be used by any tool that might need it. Note that this allows the statistics gathering process to be running without any client application being connected. This makes it possible to connect with the Terracotta Developer Console, set up a capturing session, start capturing, disconnect the console, wait for a certain amount of time, connect again and finally retrieve the data of the captured statistics.

Statistics store

The gatherer stores statistics data in an embedded database, which is the statistics store. The same statistics store is used by the Snapshot Visualization Tool to peruse the statistics data and make it possible to perform meaningful queries on what is available. It's also able to connect through JDBC to statistics stores and use SQL to perform queries.

Statistics

Statistics have names to identify them and each agent is aware of the statistics that it supports. When statistics data is captured, the entries that are entered in the store don't necessarily have the same name. For example, the '`memory`' statistic will each time store three entries: '`memory free`', '`memory max`' and '`memory used`'. Also, each name might have several data elements. For instance, the '`cpu`' statistic will create separate entries for each CPU in the machine called for example '`cpu 0`' and '`cpu 1`'. The names and the elements of each statistic data capture can thus create a lot of entries that are stored at the same time.

Supported statistics

The statistics that are supported by the agents are dependent on the system that the application is running on and also whether it's a Terracotta client or server. So, for example, the operating system, the JDK version, and possibly later the frameworks that are being used by an application, all contribute to determining the set of statistics that are supported by individual agents. Here again, the gateway takes away the burden of having to interact with the individual agents to handle their set of statistics. The gateway dynamically queries the agents in the cluster to provide an aggregated and comprehensive list of all the statistics that are supported by the cluster as a whole. When the statistics are enabled in a capturing session, the gateway will enable the statistics on the appropriate agents.

Statistic types

The statistics recorder supports three different types of statistics, each supported statistic has one particular type that is appropriate to the data that is being collected.

- **startup**: statistics of the `startup` type will be automatically captured at the beginning of a capturing session.
- **snapshot**: statistics of the `snapshot` type will be automatically captured during the capturing session, based on the frequency that has been set up in the configuration.
- **triggered**: statistics of the `triggered` type will not be captured automatically, they are either captured through API calls or when certain conditions in the running system occur.

Generally you shouldn't worry about these types at all since each statistic is aware of its own type and the statistics recorder behaves appropriately during the capture process.

Configuration

The default behavior of the configuration should suffice unless you are running several JVMs on the same machine, require specific locations where statistics data need to be buffered and stored, or want to protect the access to the restful interface.

Paths in `tc-config.xml`

The statistics recorder needs to store data on disc, both for the agent buffers and the gatherer store.

Terracotta server agent

By default, the directory that will be used both for the buffer and the store will be '`terracotta`' in the current directory.

If you want to change the directory, you can add a <statistics> element inside a <server> element of the tc-config.xml file.

For example, below, the terracotta/server-stats will be used in the user's home directory instead:

```
<server>
  <!-- other config -->
  <statistics>%(user.home)/terracotta/server-stats</statistics>
  <!-- other config -->
</server>
```

Terracotta client agent

By default, the directory that will be used both for the buffer will be 'terracotta-10.0.2.1' in the current directory, where 10.0.2.1 is the IP address of the client machine.

If you want to change the directory, you can add a <statistics> element inside the <clients> element of the tc-config.xml file.

For example, below, the terracotta/client-stats will be used in the user's home directory instead:

```
<clients>
  <!-- other config -->
  <statistics>%(user.home)/terracotta/client-stats</statistics>
  <!-- other config -->
</clients>
```

Authentication

You'll read below that one of the ways to work with the statistics recorder is a RESTful interface. This opens up functionalities over regular HTTP that you only might want certain people to have access to. To protect this interface you can setup a file with usernames and passwords. The path to this file can be added through a <http-authentication> element inside a <server> element of the tc-config.xml file.

For example, below the terracotta/realm.properties file inside the user's home directory will be used to get credentials from for the HTTP authentication:

```
<server>
  <!-- other config -->
  <http-authentication>
    <user-realm-file>%(user.home)/terracotta/realm.properties</user-realm-file>
  </http-authentication>
  <!-- other config -->
</server>
```

The format of this file is as follows:

```
username1: password1, statistics
username2: password2, statistics
...
```

You should fill in the appropriate values for username1, password1, ... and each user must be listed on its own line. Make sure to end each line with 'statistics' since that indicates that the user has access to the statistics RESTful interface.

Passwords may be clear text, obfuscated or checksummed. The class org.mortbay.jetty.security.Password from the Jetty webserver should be used to generate obfuscated passwords or password checksums. Please refer the documentation there for more information about how to generate the other password versions.

Non-dynamic statistics

Even though we strive for making the whole statistics recorder infrastructure totally dynamic, certain statistics create such a high performance impacts, that they need to be explicitly enabled before your application is started up. This is done by setting up the appropriate tc.properties file.

Below is the list of non-dynamic statistics and the properties that need to be set to true before they can be used:

statistic name	property name
`l2 faults from disk`	`l2.objectmanager.fault.logging.enabled`
`message monitor`	`tcm.monitor.enabled`

Running the Platform Statistics Recorder

The Platform Statistics Recorder provides multiple control interfaces.

Graphical User Interface (GUI)

The Platform Statistics Recorder has a visual interface that's part of Terracotta Developer Console. This GUI allows you to perform most statistics-gathering and processing operations without having to learn any commands. It also provides access to the Snapshot Visualization Tool for graphing captured statistics.

See the Developer Console documentation for more information.

Command-line interface (CLI)

The CLI interface allows you to perform all the operations that are available through the GUI in a text console. This makes it possible to integrate the statistics recorder into scripts, to easily launch capture sessions on headless displays, and more ...

Running the CLI

Windows

```
> %TERRACOTTA_HOME%\platform\bin\tc-stats.bat
```

UNIX/Linux

```
> ${TERRACOTTA_HOME}/platform/bin/tc-stats.sh
```

tc-stats help output

```
usage: java com.tc.statistics.cli.CVT [OPTION]... [COMMAND [ARGUMENTS]]...
Options:
 -H,--host <hostname|ip>   host name or address of the gatherer (defaults
                           to localhost).
 -f,--file <filename>      file name of the script with commands to play.
 -h,--help                 shows this text.
 -p,--port <number>        JMX port of the gatherer (defaults to 9520).
 -u,--username <arg>       username
 -w,--password <arg>       password

Commands:
 startup
 shutdown
 createSession <sessionId>
 closeSession
 getActiveSessionId
 getAvailableSessionIds
 getSupportedStatistics
 enableStatistics <comma-separated list of names>
 captureStatistic <name>
 startCapturing
 stopCapturing
 retrieveStatistics <filename>
 setGlobalParam <key> <value>
 getGlobalParam <key>
 setSessionParam <key> <value>
 getSessionParam <key>
 clearStatistics <sessionId>
 clearAllStatistics
 reinitialize
```

The username and password, which are passed to the script using the `-u` and `-w` flags, may be required on the command-line at the time of execution. Set the username, password, and associated roles in `jmxremote.password` and `jmxremote.access` in your JDK installation.

CLI usage and commands

The commands that are provided to `tc-stats` are separated by spaces and you can use as many as you want on the same command-line entry. It works as a series of commands that are executed in the same order as they've been listed.

Instead of listing all the command on the command-line, you can put them in a dedicated file and use `tc-stats` with the `--file` option to retrieve the commands from that file instead.

For instance, the example below will startup the local gatherer and the gateway, retrieve the list of supported statistics of the cluster, and shut the gatherer and the gateway down again.

```
bin\tc-stats.bat startup getSupportedStatistics shutdown
```

The output could be:

```
> Started up.
cpu
distributed gc
l2 l1 fault
memory
stage queue depth
thread dump
vm garbage collector
```

Starting a capture session from the CLI

To start a capture session from the CLI, you have to use the following commands in order:

- `startup`
- `createSession`
- `enableStatistics`
- `startCapturing`

For example:

```
bin\tc-stats.bat startup createSession "test session" \
  enableStatistics "cpu,l2 l1 fault,memory" startCapturing
```

Whose output could be:

```
> Already started up.
> Session 'test session' created.
> Statistics 'cpu,l2 l1 fault,memory' enabled.
> Capturing started.
```

Stopping a capture session and retrieving the data

To stop a capture session from the CLI, you have to use the `stopCapturing` command:

```
bin\tc-stats.bat stopCapturing
```

Afterwards, to retrieve the statistics data, you can do:

```
bin\tc-stats.bat retrieveStatistics statistics.zip
```

This zip file contains a single entry with all the statistics that are stored in the gatherer in the CSV format. See below for more information about the structure of the CSV file.

Capturing triggered statistics

Some statistics will never be captured automatically since they are of the '`triggered`' type. The '`thread dump`' statistics is one of those. To capture the data for such statistics, you have to use the '`captureStatistic`' command.

For example:

```
bin\tc-stats.bat captureStatistic "thread dump"
```

You will receive the data of the captured statistic as text in the standard output, but the same data has also been recorded in the gatherer. When you retrieve the statistics afterwards, the data of the triggered statistics will be present together with the data that is captured automatically.

RESTful interface

All the commands that are available through the CLI and console GUI are also available through a RESTful interface. This means that you can use any HTTP client to perform these operations, for example a curl or a web browser.

Accessing the list of supported RESTful commands (methods)

The RESTful interface is available on your Terracotta server and runs on the same port as the DSO server (see the config reference guide for more information). By default this will be port 9510. The RESTful interface is located at the `statistics-gatherer` path.

So if you're working on the same machine as the one you're running a server on, you'd access the URL below with a web browser to get a list of all the methods that are supported through the RESTful interface. If you're running to server on another machine or with another port number, you'd have to change those in the URL of course.

```
http://localhost:9510/statistics-gatherer/
```

The commands you see will look very familiar since most of them are the same as what you've seen above for the CLI.

Starting a capture session from the RESTful interface

To start a capture session from the RESTful interface, you have to use the same commands as for the CLI. You'll have to visit each one independently with a HTTP client.

```
http://localhost:9510/statistics-gatherer/startup
http://localhost:9510/statistics-gatherer/createSession?sessionId=test session
http://localhost:9510/statistics-gatherer/enableStatistics?names=cpu&names=l2%20l1%20fault&names=memory
http://localhost:9510/statistics-gatherer/startCapturing
```

Note that the arguments to the commands are separated through a question mark (?). Each argument has the format '`key=value`' and they are separated through ampersands (&). If an argument has multiple values, like the '`names`' argument above, you have to repeat the key/value pair several times. If there are any spaces in the argument values, you need to replace them with '`%20`', otherwise the URL will not be correct.

Live Retrieval of Captured Data as Text

The RESTful interface allows you to use its '`retrieveStatistics`' command to see the captured statistics data at any moment in time. By providing it with the '`format=txt`' argument, you can view the data directly in the browser.

This is the full command:

```
http://localhost:9510/statistics-gatherer/retrieveStatistics?format=txt
```

RESTful methods and their arguments

Below is a list of all the supported RESTful methods and the arguments they support. Note that an asterisk (*) in front of argument names indicates that these arguments are mandatory.

Method	Argument(s)
`startup` starts up the gatherer	*none*
`shutdown` shuts down the gatherer	*none*
`reinitialize` reinitializes the entire statistics recorder cluster	*none*
`createSession` creates a new capture session and actives it	`*sessionId`: the unique session ID
`closeSession` closes the currently active session	*none*
`getActiveSessionId` returns the ID of the active session	*none*
`getAvailableSessionIds` returns the list of session IDs in the statistics store	*none*
`getAvailableAgentDifferentiators` returns the list of agent differentiators	*none*
`getSupportedStatistics` returns the list of all the supported statistics in the cluster	*none*
`enableStatistics` sets the enabled statistics for the active session	`*names`: the names of the statistics
`captureStatistic` captures and returns the data for a triggered statistic	`*name`: the name of the statistic

`startCapturing` starts the capturing for the active session	*none*
`stopCapturing` stops the capturing for the active session	*none*
`setGlobalParam` sets the value of a global configuration parameter	`*key`: the config parameter key `*value`: the config parameter value
`getGlobalParam` retrieves the value of a global configuration parameter	`*key`: the config parameter key
setSessionParam sets the value of a configuration parameter for the active session	`*key`: the config parameter key `*value`: the config parameter
getSessionParam retrieves the value of a configuration parameter for the active session	`*key`: the config parameter key
`clearStatistics` clears the stores statistics for a particular session	`*sessionId`: the unique session ID
`clearAllStatistics` clears all the stored statistics	*none*
`retrieveStatistics` retrieves the data from the statistics store	`format`: `txt` or `zip` to indicate the output format `sessionId`: the session ID to retrieve data for `agentIp`: the agent IP to retrieve data for `agentDifferentiator`: the agent differentiator to retrieve data for `names`: the statistic names to retrieve data for `elements`: the statistic data elements to retrieve data for
`aggregateStatisticsData` Aggregates statistics data for use in time-series.	`interval`: the interval in milliseconds, this can be used to create a fixed-size sliding window that will only show the capture data from now until a fixed point in the past `format`: `csv` to generate CSV data or `xml` to generate XML data `*sessionId`: see above `*agentIp`: see above `*agentDifferentiator`: see above `*names`: see above `elements`: see above

Data retrieval

After the recording of the statistics, you're able to retrieve the data as a dump of all the recorded entries or in an aggregated fashion.

CSV 'dump' format

When using the '`retrieveStatistics`' statistics command through one of the interfaces, you'll receive the data from the statistics store in an unprocessed fashion. The format that is used is CSV and even if you receive the data as a zip file, the only entry in the archive will be that CSV file.

The CSV format is very handy for transferring the recorded data to other tools and applications. You can for instance import this file into Excel and generate graphs yourself if you want.

These are the columns of the CSV file:

- `Session ID` : the unique identifier of the session
- `IP` : the IP address of the agent
- `Differentiator` : the string that can be used to differentiate the agent
- `Moment` : the moment when the data was captured, expressed as milliseconds since epoch (unix time)
- `Name` : the name of the statistic
- `Element` : the name of the data element of the statistic (can be empty)
- `Data Number` : the statistic data as an integer number
- `Data Text` : the statistic data as text
- `Data Date` : the statistic data as a timestamp, expressed as milliseconds since epoch
- `Data Decimal` : the statistic data as a decimal number

Note that only one of the four data columns can be filled in, the one that's used is up to internal implementation of the statistic.

For example:

```
Session ID,IP,Differentiator,Moment,Name,Element,Data Number,Data Text,Data Date,Data Decimal
"mysession","192.168.1.33","L2/9510","1208722084118","startup timestamp",,,,"1208722084118",
"mysession","192.168.1.33","L2/9510","1208722084120","l2 l1 fault",,"0",,,
"mysession","192.168.1.33","L2/9510","1208722084120","memory free",,"244609968",,,
"mysession","192.168.1.33","L2/9510","1208722084120","memory max",,"265486336",,,
"mysession","192.168.1.33","L2/9510","1208722084120","memory used",,"20876368",,,
"mysession","192.168.1.33","L1/0","1208722084122","startup timestamp",,,,"1208722084122",
"mysession","192.168.1.33","L1/0","1208722084123","l1 transaction size",,"0",,,
"mysession","192.168.1.33","L1/0","1208722084123","memory free",,"44666792",,,
"mysession","192.168.1.33","L1/0","1208722084123","memory max",,"66650112",,,
"mysession","192.168.1.33","L1/0","1208722084123","memory used",,"21983320",,,
"mysession","192.168.1.33","L1/1","1208722084127","startup timestamp",,,,"1208722084127",
"mysession","192.168.1.33","L1/1","1208722084128","l1 transaction size",,"0",,,
"mysession","192.168.1.33","L1/1","1208722084128","memory free",,"43802560",,,
"mysession","192.168.1.33","L1/1","1208722084128","memory max",,"66650112",,,
"mysession","192.168.1.33","L1/1","1208722084128","memory used",,"22847552",,,
"mysession","192.168.1.33","L2/9510","1208722089118","l2 l1 fault",,"30",,,
"mysession","192.168.1.33","L2/9510","1208722089118","memory free",,"232671448",,,
"mysession","192.168.1.33","L2/9510","1208722089118","memory max",,"265486336",,,
"mysession","192.168.1.33","L2/9510","1208722089118","memory used",,"32814888",,,
"mysession","192.168.1.33","L2/9510","1208722089119","shutdown timestamp",,,,"1208722089119",
"mysession","192.168.1.33","L1/0","1208722089123","l1 transaction size",,"0",,,
"mysession","192.168.1.33","L1/0","1208722089123","memory free",,"43161296",,,
"mysession","192.168.1.33","L1/0","1208722089123","memory max",,"66650112",,,
"mysession","192.168.1.33","L1/0","1208722089123","memory used",,"23488816",,,
"mysession","192.168.1.33","L1/0","1208722089124","shutdown timestamp",,,,"1208722089124",
"mysession","192.168.1.33","L1/1","1208722089128","l1 transaction size",,"10",,,
"mysession","192.168.1.33","L1/1","1208722089128","memory free",,"40551680",,,
"mysession","192.168.1.33","L1/1","1208722089128","memory max",,"66650112",,,
"mysession","192.168.1.33","L1/1","1208722089128","memory used",,"26098432",,,
"mysession","192.168.1.33","L1/1","1208722089129","shutdown timestamp",,,,"1208722089129",
```

Aggregated formats

The CSV dump format is handy when you want to work with all the data after the statistics recording has finished. However, often you also want a real-time view into the statistics data while it's being captured. For that you typically only want to know use the values of certain statistics, aggregated according to the moment on which they were captured. This makes it easy to create a timeline on which graphs are created that correspond to the values of the statistics.

You can receive this information through the '`aggregateStatisticsData`' command of the RESTful interface.

For example, let's say that you want to get all the data of the memory statistics for the Terracotta server, over the last 10 seconds. This URL will give you exactly that (on one line):

```
http://localhost:9510/statistics-gatherer/aggregateStatisticsData?sessionId=mysession
    &format=xml&agentDifferentiator=L2/9510
    &names=memory%20used&names=memory%20free&names=memory%20max
    &interval=10000
```

The result could be:

```
<?xml version="1.0" encoding="UTF-8"?>
<data>
<d><m>1208762443629</m><v1>247576464</v1><v2>265486336</v2><v3>17909872</v3></d>
<d><m>1208762444629</m><v1>247525704</v1><v2>265486336</v2><v3>17960632</v3></d>
<d><m>1208762445629</m><v1>246510872</v1><v2>265486336</v2><v3>18975464</v3></d>
<d><m>1208762446629</m><v1>246460096</v1><v2>265486336</v2><v3>19026240</v3></d>
<d><m>1208762447629</m><v1>246408976</v1><v2>265486336</v2><v3>19077360</v3></d>
<d><m>1208762448629</m><v1>245382232</v1><v2>265486336</v2><v3>20104104</v3></d>
<d><m>1208762449629</m><v1>245330712</v1><v2>265486336</v2><v3>20155624</v3></d>
<d><m>1208762450629</m><v1>245279976</v1><v2>265486336</v2><v3>20206360</v3></d>
</data>
```

If you would change the '`format`' argument to '`csv`', the result could be:

```
1208762443629,247576464,265486336,17909872
1208762444629,247525704,265486336,17960632
1208762445629,246510872,265486336,18975464
1208762446629,246460096,265486336,19026240
1208762447629,246408976,265486336,19077360
1208762448629,245382232,265486336,20104104
1208762449629,245330712,265486336,20155624
1208762450629,245279976,265486336,20206360
```

You can see that the first column or the XML element '`m`' always contains the unix timestamp. The other columns or XML elements are dynamic and will correspond to the statistics names that you've requested. In this case, the first column is '`memory used`', the second one is '`memory free`' and the third one is '`memory max`'. The names of the XML elements have been compressed deliberately to reduce the content size as much as possible.

With this aggregated output, you can use many tools directly with the statistics recording to create real-time charts of the statistics that interest you (for instance YUI or Flex).

Troubleshooting

In the unlikely event that something goes wrong with the Platform Statistics Recorder and it is not able to repair the agent buffers and stores itself, the recording will not be functional but the rest of Terracotta's features will all be operational. This gives you the opportunity to determine the best approach towards resolving the issue without impacting the rest of your application.

Reinitializing all the nodes

Usually, the recorded statistics data in not very valuable and the sessions that you need to keep around have already been exported for use in the Snapshot Visualization Tool or something else.

The easiest way to reset the Platform Statistics Recorded to a clean state, is to use the '`reinitialize`' command through the CLI or through the RESTful interface. This will stop any ongoing recordings, remove all existing sessions and recreate all the buffers and stores in the cluster. After the reinitialize call, the statistics recorder will have a clean slate that is immediately ready to be used, without having to restart your application.

On the CLI you can perform this re-initialization as follows:

```
bin\tc-stats.bat reinitialize
```

Through the RESTful interface it would be like this:

```
http://localhost:9510/statistics-gatherer/reinitialize
```