

SimpleMetrics Package

Version 0.5
December 2015

Gray Watson

This manual is licensed by Gray Watson under the Creative Commons Attribution-Share Alike 3.0 License.

Permission is granted to make and distribute verbatim copies of this manual provided this license notice and this permission notice are preserved on all copies.

Table of Contents

SimpleMetrics	1
1 Start Using Quickly	2
2 Using SimpleMetrics	3
2.1 Downloading Jar	3
2.2 Creating a MetricsManager Instance	3
2.3 Using a Metrics Persister	3
2.4 Creating and Registering Metrics	3
2.5 Updating Metric Values	4
2.6 Using the Built-In Utilities	5
2.7 Publishing Metrics Via JMX	5
2.8 Using With Maven	5
3 Open Source License	6
Index of Concepts	7

SimpleMetrics

Version 0.5 – December 2015

This package provides some simple metrics and associated operations that allow for the recording of application metrics and persisting them to various different local or cloud storage/metric systems. Your code registers metrics and then doesn't have to worry about how they are managed or persisted.

To get started quickly using SimpleMetrics, see [Chapter 1 \[Quick Start\]](#), page 2. There is also a [HTML version of this documentation](#).

Gray Watson <http://256.com/gray/>

1 Start Using Quickly

To use SimpleMetrics you need to do the following steps. For more information, see [Chapter 2 \[Using\], page 3](#).

1. Download SimpleMetrics from the [SimpleMetrics release page](#). See [Section 2.1 \[Downloading\], page 3](#).
2. Create an instance of the `MetricsManager` class which manages the metrics in our application.

```
MetricsManager metricsManager = new MetricsManager();
```

3. Create a persister such as the `LoggingMetricsPersister` which logs the metrics and values to `java.util.Logger`.

```
LoggingMetricsPersister persister =  
    new LoggingMetricsPersister();  
metricsManager.registerMetric(hitCounter);
```

4. Create at least one metric which monitors a particular application value, and register it with the `LoggingMetricsPersister`.

```
ControlledMetricAccum hitCounter =  
    new ControlledMetricAccum("example", null, "hits",  
        "number of hits to the cache", null);  
metricsManager.registerMetric(hitCounter);
```

5. Possibly use the `MetricsPersisterJob` to start a background thread that calls `persist()` on the `MetricsManager` every so often automatically. Otherwise you will need to do so using some other mechanism.

```
MetricsPersisterJob persisterThread =  
    new MetricsPersisterJob(manager, 1000, 1000, true);
```

For somewhat more extensive instructions, see [Chapter 2 \[Using\], page 3](#).

2 Using SimpleMetrics

2.1 Downloading Jar

To get started with SimpleMetrics, you will need to download the jar file. The [SimpleMetrics release page](#) is the default repository but the jars are also available from the [central maven repository](#).

The code works with Java 6 or later.

2.2 Creating a MetricsManager Instance

The `MetricsManager` is the class which manages the metrics in the application, updates the values of the metrics when necessary, and persisting the metrics to disk or network when requested to do so. You need to set at least one metrics persister on the manager and then register metrics from various places in your application. `MetricsManager` also supports `SimpleJmx` annotations which allow you to publish the metrics and view them via JMX.

```
MetricsManager metricsManager = new MetricsManager();
```

2.3 Using a Metrics Persister

Once you have created your `MetricsManager` you should set metrics persister classes on it. Metrics persisters are the classes which are responsible for saving the metrics information to disk, cloud-service, or other archive so they can be reported on and stored for later use.

Persisters implement either the `MetricValuesPersister` or `MetricDetailsPersister` interfaces. The value persister gets the metric values as a simple `Number` class. The details persister provides more extensive information on the metrics such as number of samples, average, minimum, and maximum values through the `MetricValueDetails` class.

There are a couple simple persister implementations that come with the library although they may only be useful as implementation examples;

- `LoggingMetricsPersister` - Logs metrics and their values to `java.util.Logger`. This can be used as an implementation example so you can log metrics to your application's primary logging class such as `log4j`.
- `SystemOutMetricsPersister` - Prints metrics and their values to `System.out`.
- `TextFileMetricsPersister` - Writes metrics and their values to a text-file on the file-system. This text file can then be imported into some reporting system. It is able to cleanup old metrics files with the `cleanMetricFilesOlderThanMillis(...)` method.

Persisters are set on the `MetricsManager`:

```
// persisters that persist a number per metric
metricsManager.setMetricValuesPersisters(
    new MetricDetailsPersister[] { ... });
// persisters that persist metric details
metricsManager.setMetricDetailsPersisters(
    new MetricDetailsPersister[] { ... });
```

2.4 Creating and Registering Metrics

Once you have created your `MetricsManager` and created your persisters, you are ready to start creating and registering metrics with the manager. Metrics are the classes which keep track of the name of the metric as well as its associated value(s). For example, let's say we wanted to count the number of web-requests made to our web-server so we can graph it over time. We might create a metric like the following and register it on the `MetricsManager`:

```
ControlledMetricAccum webRequestMetric =  
    new ControlledMetricAccum("web", "server", "requests",  
        "number of requests handled by our web-server", null);  
metricsManager.registerMetric(webRequestMetric);
```

Whenever a request comes in, you just have to increment the metric:

```
// count a web-request  
webRequestMetric.increment();
```

The `MetricsManager` takes care of persisting the value to disk or network and it also resets the value after it is persisted so the counts per minute (or whatever your persist frequency is) will be accurate.

There are a couple of different types of metrics that are built into the library.

- **ControlledMetricAccum** - A metric that accumulates in value. This is used when we are counting some sort of event like a web-server request or a thrown exception. It supports an `increment()` method although the adjust will add any value to the metric.
- **ControlledMetricValue** - A metric whose value can go up or down. This is used when we are monitoring how much memory the JVM is using or a cache-hit percentage. We often use a `MetricsUpdater` when dealing with values. See [Section 2.5 \[Using MetricsUpdater\]](#), page 4.
- **ControlledMetricTimer** - For metrics that are tracking how long a particular operation takes, this class is useful for doing so. It has a `start()` and `stop()` method which easily records the elapsed time in milliseconds given that it extends `ControlledMetricValue`.
- **ControlledMetricRatio** - This metric separates the numerator from the denominator to keep good precision when recording ratios. You could track cache hit/miss ratios or other information with this.

If these metric types don't fully meet your needs, you can define others that implement the `ControlledMetric` interface and probably extend the `BaseControlledMetric` class.

2.5 Updating Metric Values

In many situations, you may poll a value from another object and update a metric at that time. The `MetricsManager` has support for classes that implement the `MetricsUpdater` interface that can be registered on the manager. Whenever values are to be persisted, the `MetricsManager` will call the configured updaters beforehand so they can calculate or poll the values for their metrics.

For example, let's say you were tracking how much memory your were using in your system. You would register your memory metric with the `MetricsManager` and also register yourself with the `MetricsManager` as an updater. The `MetricsManager` will call your `updateMetrics()` method which gives you an opportunity to calculate how much memory your code is using and update the metric with the information.

2.6 Using the Built-In Utilities

There are a couple of built-in utility classes which are useful for most applications to utilize.

- **SystemMetricsPublisher** - Publishes a number of useful bits of information from the JVM: number of threads, total memory used, maximum memory used, free memory, current heap size, number of loaded classes, total process CPU time, thread load average percentage, old gen memory presentage, process load average percentage.
- **FileMetricsPublisher** - Reads values from files on the file system that are then published via metrics. This is used to read numbers from files in thre `/proc` file-system on linux. A common file metric that you might want to publish is the number of open file-descriptors being used by the JVM.

2.7 Publishing Metrics Via JMX

The library uses the SimpleJMX library to allow for easy publishing of metric values via JMX. It is optional to do so but you can set the `JmxServer` on the `MetricsManager` and metrics will be registered to the `JmxServer` and publishd into JMX folders. For more information about SimpleJMX, see the [SimpleJMX home page](#).

2.8 Using With Maven

To use SimpleMetrics with maven, include the following dependency in your `'pom.xml'` file:

```
<dependency>
<groupId>com.j256.simplemetrics</groupId>
<artifactId>simplemetrics</artifactId>
<version>0.5</version>
</dependency>
```


3 Open Source License

This document is part of the SimpleMetrics project.

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

The author may be contacted via <http://256.com/sources/simplemetrics/>

Index of Concepts

A

accumulator metric	4
author	1
average value	3

B

BaseControlledMetric	4
----------------------------	---

C

cloud service	3
ControlledMetric interface	4
ControlledMetricAccum	4
ControlledMetricRatio	4
ControlledMetricTimer	4
ControlledMetricValue	4
creating metrics	4

D

downloading the jars	3
----------------------------	---

G

getting started	2
-----------------------	---

H

how to download the jars	3
how to get started	2
how to use	3

I

increment, metric	4
introduction	1

J

JMX usage	5
JmxServer	5

L

license	6
logging metrics	3

M

managing the metrics	3
Maven, use with	5
maximum value	3
MetricDetailsPersister	3
metrics	4
metrics base class	4
metrics interface	4

metrics persisters	3
MetricsManager	3
MetricValueDetails	3
MetricValuesPersister	3
minimum value	3

N

number of samples	3
-------------------------	---

O

open source license	6
---------------------------	---

P

persisting metrics	3
pom.xml dependency	5
publishing metrics using JMX	5

Q

quick start	2
-------------------	---

R

ratio metric	4
registering metrics	4

S

samples	3
save metrics to disk	3
simple metrics	1
SimpleJMX	3, 5
storing metrics	3

T

time tracking	4
---------------------	---

U

using SimpleMetrics	3
---------------------------	---

V

value metric	4
--------------------	---

W

where to get new jars	3
-----------------------------	---