PerfCake 7.x

User Guide

Pavel Macík
Martin Večeřa

PerfCake 7.x: User Guide

by Pavel Macík and Martin Večeřa

Table of Contents

Acknowledgements xi
1. Introduction 1
  1.1. Obtaining and installing PerfCake 1
    1.1.1. Downloading distribution 1
    1.1.2. Building distribution 1
    1.1.3. Minimal requirements 1
    1.1.4. Installing PerfCake 2
    1.1.5. Running PerfCake 2
  1.2. Your first performance test 4
    1.2.1. First out-of-the box demo with httpbin.org 4
    1.2.2. Your own quickstart 5
      Preparing PerfCake 5
      Configure and run 6
      Evaluate results 8
2. PerfCake Features 11
  2.1. PerfCake Architecture Overview 11
  2.2. Performance scenario definition 12
    2.2.1. XML scenario 12
      Scenario structure 12
      Sections of the scenario 13
    2.2.2. DSL scenario 15
    2.2.3. Scenario specified through API 16
    2.2.4. IDE plugins 18
    2.2.5. Filtering properties 18
  2.3. Running PerfCake 21
    2.3.1. Command line parameters 21
    2.3.2. Running scenarios from Maven 24
    2.3.3. Logging 25
    2.3.4. Results Replay 25
    2.3.5. Debug Agent 27
  2.4. What to take care of during execution 28
  2.5. Exit codes 29
  2.6. Migrating scenarios to latest version 30
    2.6.1. From v1.0 to v2.x 30
      Migration steps 30
    2.6.2. From v2.x to v3.x 30
      Migration steps 31
      Scenario conversion using XSLT 33
    2.6.3. From v3.x to v4.x 33
      Migration steps 34
    2.6.4. From v4.x to v5.x 34
      Migration steps 34
      Scenario conversion using XSLT 35
    2.6.5. From v5.x to v6.x 35
      Migration steps 35
    2.6.6. From v6.x to v7.x 35
      Migration steps 36
3. General Usage 38
  3.1. PerfCake: Performance Testing Scenarios 38
  3.2. Performance Engineering 38
  3.3. YouTube Channels 38

4. Reference Guide 39
4.1. How - Generating load 39
4.1.1. DefaultMessageGenerator 39
4.1.2. ConstantSpeedMessageGenerator 40
4.1.3. CustomProfileGenerator 41
CsvProfile 42
4.1.4. RampUpDownGenerator 43
4.2. Where - Sending messages 45
4.2.1. CamelSender 45
4.2.2. CoapSender 46
4.2.3. CommandSender 46
4.2.4. DummySender 47
4.2.5. GroovySender 47
4.2.6. HttpSender 48
4.2.7. HttpsSender 49
4.2.8. ChannelDatagramSender 49
4.2.9. ChannelFileSender 50
4.2.10. ChannelSocketSender 50
4.2.11. JdbcSender 51
4.2.12. Jms[11]Sender 51
4.2.13. LdapSender 53
4.2.14. MqttSender 53
4.2.15. OauthHttpSender 54
4.2.16. PlainSocketSender 56
4.2.17. RequestResponseJms[11]Sender 56
4.2.18. ScriptSender 57
4.2.19. SslSocketSender 58
4.2.20. WebSocketSender 59
4.3. Receiving messages 59
4.3.1. HttpReceiver 60
4.4. Correlating messages 61
4.4.1. GenerateHeaderCorrelator 61
4.4.2. PrefixCorrelator 61
4.5. What - Messages 62
4.5.1. Filtering and templates 63
4.6. Sequences 63
4.6.1. PrimitiveNumberSequence 64
4.6.2. NumberSequence 64
4.6.3. RandomSequence 65
4.6.4. RandomUuidSequence 65
4.6.5. ThreadIdSequence 65
4.6.6. TimeStampSequence 66
4.6.7. FileLinesSequence 66
4.6.8. FilesContentSequence 66
4.7. Reporting 67
4.7.1. Reporters 68
ClassifyingReporter 68
GeolocationReporter 69
IterationsPerSecondReporter 70
MemoryUsageReporter 70
ResponseTimeHistogramReporter 73
ResponseTimeStatsReporter 76
ThroughputStatsReporter 79
WarmUpReporter 81

RawReporter 83
4.7.2. Destinations 83
ChartDestination 84
ConsoleDestination 87
CsvDestination 88
ElasticsearchDestination 89
InfluxDbDestination 91
Log4jDestination 92
4.8. Validation 93
4.8.1. Validators 94
DictionaryValidator 94
PrintingValidator 94
RegExpValidator 95
RulesValidator 95
ScriptValidator 95
5. Result repository 97
6. Extending PerfCake 98
6.1. Client libraries 98
6.2. Custom components - Plugins 98
7. Troubleshooting PerfCake 99
7.1. Running in Virtual Environment 99
7.2. Too many open HTTP connections 99
7.3. "java.net.BindException: Address already in use: connect” issue on Windows 99
8. Changelog 100

List of Figures

1.1. Your first performance test chart report ... 10
2.1. Architecture Overview ... 11
2.2. JMX tree with PerfCake debug information in `jvisualvm` ... 28
4.1. RampUpDownGenerator time chart ... 43
4.2. ChartDestination example chart ... 86

List of Tables

2.1. Examples of templates and how they are rendered ... 19
2.2. Available PerfCake internal properties ... 20
2.3. PerfCake CLI arguments ... 22
2.4. PerfCake recognized properties ... 23
2.5. PerfCake Exit Codes ... 29
2.6. Renamed classes in PerfCake v3.x ... 31
4.1. Run options ... 39
4.2. DefaultMessageGenerator properties ... 40
4.3. ConstantSpeedMessageGenerator properties ... 40
4.4. CustomProfileGenerator properties ... 42
4.5. RampUpDownGenerator properties ... 44
4.6. CoapSender properties ... 46
4.7. CommandSender properties ... 46
4.8. DummySender properties ... 47
4.9. GroovySender properties ... 48
4.10. HttpSender properties ... 48
4.11. HttpsSender additional properties ... 49
4.12. ChannelDatagramSender properties ... 50
4.13. ChannelFileSender properties ... 50
4.14. ChannelSocketSender properties ... 50
4.15. JdbcSender properties ... 51
4.16. JmsSender properties ... 51
4.17. HttpSender properties ... 53
4.18. MqttSender properties ... 53
4.19. HttpsSender additional properties ... 55
4.20. RequestResponseJms[11]Sender properties ... 56
4.21. ScriptSender properties ... 58
4.22. SslSocketSender additional properties ... 58
4.23. WebSocketSender properties ... 59
4.24. HttpReceiver properties ... 60
4.25. PrefixCorrelator properties ... 62
4.26. NumberSequence properties ... 64
4.27. RandomSequence properties ... 65
4.28. FileLinesSequence properties ... 66
4.29. FilesContentSequence properties ... 66
4.30. ClassifyingReporter properties ... 68
4.31. GeolocationReporter properties ... 69
4.32. GeolocationReporter result names ... 69
4.33. IterationsPerSecondReporter result names ... 70
4.34. PerfCake agent properties ... 70
4.35. MemoryUsageReporter properties ... 72
4.36. MemoryUsageReporter result names ... 73
4.37. ResponseTimeHistogramReporter properties ... 74
4.38. ResponseTimeStatsReporter properties ... 76
4.39. ResponseTimeStatsReporter result names ... 77
4.40. ThroughputStatsReporter properties ... 79
4.41. ThroughputStatsReporter result names ... 80
4.42. WarmUpReporter properties ... 82
4.43. RawReporter properties ... 83
4.44. Destination `period` options ... 83
4.45. ChartDestination properties ... 85

4.46. ConsoleDestination properties ... 87
4.47. CsvDestination properties ... 88
4.48. ElasticsearchDestination properties ... 90
4.49. InfluxDbDestination properties ... 91
4.50. Log4jDestination properties ... 92
4.51. DictionaryValidator properties ... 94
4.52. ScriptValidator properties ... 96

List of Examples

2.1. Loading of a scenario definition and its execution from API 16
2.2. Executing a scenario with a single method call using PerfCake API 16
2.3. Complete scenario definition and execution using its API 16
2.4. An example of a scenario with properties 18
2.5. An example of a scenario with combined property replacement strategies 20
2.6. PerfCake Maven plugin sample configuration 24
2.7. Sample usage of the results replay mode 26
2.8. Old sequence configuration in XML 36
2.9. New sequence configuration in XML 36
2.10. Old Sequence implementation 37
2.11. New Sequence implementation 37
4.1. An example of the `run` configuration in a scenario: 39
4.2. An example of DefaultMessageGenerator configuration 40
4.3. An example of ConstantSpeedMessageGenerator configuration 41
4.4. An example of CustomProfileGenerator configuration 42
4.5. Sample profile specified in a CSV file 43
4.6. An example of RampUpDownGenerator configuration 44
4.7. An example of a Sender configuration 45
4.8. An example of CamelSender configuration 45
4.9. An example of CoapSender configuration 46
4.10. An example of CommandSender configuration 47
4.11. An example of DummySender configuration 47
4.12. An example of GroovySender configuration 48
4.13. An example of HttpSender configuration 49
4.14. An example of HttpsSender configuration 49
4.15. An example of JdbcSender configuration 51
4.16. An example of JmsSender configuration 52
4.17. An example of LdapSender configuration 53
4.18. An example of MqttSender configuration 54
4.19. An example of OauthHttpSender configuration 55
4.20. An example of PlainSocketSender configuration 56
4.21. An example of RequestResponseJmsSender configuration 57
4.22. An example of ScriptSender configuration 58
4.23. An example of SslSocketSender configuration 58
4.24. An example of WebSocketSender configuration 59
4.25. An example of a Receiver configuration 59
4.26. An example of HttpReceiver configuration 60
4.27. An example of a correlator configuration 61
4.28. An example of GenerateHeaderCorrelator usage 61
4.29. An example of PrefixCorrelator usage 62
4.30. An example of a simple message configuration: 62
4.31. An example of multiple messages configuration: 62
4.32. An example of a Sequence configuration 64
4.33. An example of PrimitiveNumberSequence configuration 64
4.34. An example of NumberSequence configuration 65
4.35. An example of RandomSequence configuration 65
4.36. An example of RandomUuidSequence configuration 65
4.37. An example of ThreadIdSequence configuration 66
4.38. An example of TimeStampSequence configuration 66
4.39. An example of FileLinesSequence configuration 66
4.40. An example of FilesContentSequence configuration 66

4.41. An example reporting configuration: 67
4.42. An example of a disabled reporter: 68
4.43. An example of ClassifyingReporter configuration 68
4.44. An example of the output when ClassifyingReporter is used 69
4.45. An example of GeolocationReporter configuration 70
4.46. An example of IterationsPerSecondReporter configuration 70
4.47. JVM argument to attach PerfCake agent to the tested JVM 71
4.48. PerfCake JVM argument example 71
4.49. Attaching to running JVM 71
4.50. An example of MemoryUsageReporter configuration 73
4.51. An example of ResponseTimeHistogramReporter configuration 75
4.52. An example of ResponseTimeStatsReporter configuration 77
4.53. An example of ResponseTimeStatsReporter configuration with histogram 77
4.54. An example of ThroughputStatsReporter configuration with a sliding window over last 30 iterations 80
4.55. An example of output with the above configuration 81
4.56. An example of WarmUpReporter configuration 82
4.57. An example of RawReporter configuration 83
4.58. An example of the `period` configuration in a destination: 84
4.59. An example of ChartDestination configuration 86
4.60. An example of ConsoleDestination configuration 87
4.61. An example of CsvDestination configuration 89
4.62. Sample output of CsvDestination 89
4.63. An example of a ElasticsearchDestination configuration 91
4.64. An example of a InfluxDbDestination configuration 92
4.65. An example of Log4jDestination configuration 92
4.66. An example of validation configuration: 93
4.67. An example of DictionaryValidator configuration 94
4.68. An example of ScriptValidator configuration 94
4.69. An example of RegExpValidator configuration 95
4.70. RulesValidator rules 95
4.71. An example of RulesValidator configuration 95
4.72. An example of ScriptValidator configuration 96

Acknowledgements

We would like to thank to everybody who helped PerfCake to be born and to grow up by any means. Special thanks goes to our contributors who we love!

Chapter 1. Introduction

Here we would like to smoothly introduce you PerfCake - the lightweight performance testing tool and a load generator with the aim to be minimalistic, easy to use, provide stable results, have minimum influence on the measured system, be platform independent, use component design, allow high throughput.

First, we show you how to obtain PerfCake, run it and create your first performance test.

Then we introduce some core features of PerfCake as an application and how to configure it. This is useful for further integration in your testing environment.

More details and practical test scenarios and use cases will be described in a separate chapter. Its content will be created over time, for now we can suggest you to follow us on Twitter and see recordings of our talks.

The most comprehensive part for now is the Reference Guide where you can find details about configuration of individual components of PerfCake.

1.1. Obtaining and installing PerfCake

So let's get our hands on PerfCake and start creating our first performance test.

1.1.1. Downloading distribution

The best way to obtain a complete distribution of PerfCake is at https://www.perfcake.org/download/. There you can find the latest stable binary release in multiple formats.

We assume that you already have Java installed. For the best performance of PerfCake, we need to ask you to install Java Development Kit (JDK) as we compile some classes dynamically during test execution. This cannot be done just with Java Runtime Environment (JRE). We also do not provide distributions with Java included in them. There are two simple reasons - first, you are likely to already have one, second, we do not have enough time to maintain and test such distributions.

If you are brave enough, you can even try our nightly builds from https://perfcake.ci.cloudbees.com/job/PerfCake-devel/lastSuccessfulBuild/artifact/perfcake/target/.

We try not to commit broken code and the nightly builds contain all the newest features. However, there are no guarantees and the features are usually not yet documented. You would need to have a look in the source code to be able to fully use them.

Once you have downloaded your favourite build file, you can just uncompress (unzip, untar...) it in a directory you like. This is your complete installation and the location is further referenced as $PERFCAKE_HOME.

1.1.2. Building distribution

Another option is to build your own distribution from source code. PerfCake is written in Java and the project lifecycle is managed by Maven. If you are familiar with these technologies, you can find more information on using the Maven goals in the Developers' Guide.

1.1.3. Minimal requirements

As was mentioned in Section 1.1.1, “Downloading distribution” , the main requirement is having Java Development Kit version 8 installed in your environment. In case you have troubles installing Java, please

search some tutorials on installing JDK on your operating system on YouTube. There are plenty of useful videos on this topic.

Other requirements on the hardware depend on what you expect from PerfCake. The memory and CPU power demands depand mainly on your performance scenario.

Up to our knowledge all components of PerfCake work offline without an Internet connection. This is our intent, please report a bug should you have different findings.

1.1.4. Installing PerfCake

There is no extra step required to install PerfCake. As long as you have unpacked the downloaded distribution, the installation is complete.

1.1.5. Running PerfCake

There are multiple ways of running PerfCake. In the case you have just donwloaded and unpacked the binary distribution, you can find a shell script and a Windows bat script in the $PERFCAKE_HOME/bin directory. These are executables that can find your JDK and run PerfCake properly. The scripts were tested on Linux, Windows and MacOS (in this case, some command line tweaking might be necessary).

Note

Windows users should be running perfcake.bat from the old Command shell and avoid using PowerShell. Another way is to run PerfCake with the following command in PowerShell:

```
cmd /c perfcake.bat
```

It might be also necessary to place some parameters in double quotes. Like the following:

```
"-Dperfcake.run.time=1000"
```

Note

Some MacOS installations of Java might have a preconfigured directory with Java libraries that are pre-loaded with each Java application. These libraries can interfere with PerfCake libraries resulting in failure to start PerfCake and are placed under `/Library/Java/Extensions/` directory.

Move the libraries to some other temporary directory to be able to run PerfCake.

In case of building from the source code, please refer to the Developers' Guide for more instructions. Basically, the easiest way is to create the binary distribution from source code and follow the usual steps. It is also possible to run PerfCake directly by invoking the java command but this needs to take care of the classpath configuration.

It is also possible to run PerfCake using a Maven plugin or its API. These ways are described either in the Developers' Guide or will be presented in Chapter 3, *General Usage* .

You can verify your installation by invoking the following command in the $PERFCAKE_HOME directory. This uses a 3rd party HTTP test server, so try not to over-use it. Please note that you need to be online for this to work.

```
$ ./bin/perfcake.sh -s http -Dserver.host=httpbin.org
```

Typical output of this command is:

```
2015-09-22 18:35:51,774 INFO  {org.perfcake.ScenarioExecution} ===
 Welcome to PerfCake 7.5 ===
2015-09-22 18:35:52,109 INFO  {org.perfcake.scenario.ScenarioBuilder}
 Scenario configuration: file:/home/perfcake/perfcake-7.5/resources/
scenarios/http.xml
2015-09-22 18:35:52,175 INFO  {org.perfcake.util.TimerBenchmark}
 Benchmarking system timer resolution...
2015-09-22 18:35:52,176 INFO  {org.perfcake.util.TimerBenchmark} This
 system is able to differentiate up to 296ns. A single thread is now
 able to measure maximum of 3378378 iterations/second.
2015-09-22 18:35:52,254 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Starting to
 generate...
[0:00:01][705 iterations][5%] [520.564841647836 iterations/s] [Threads
 => 100] [warmUp => false]
[0:00:02][1274 iterations][8%] [543.478973435464 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:03][1864 iterations][12%] [554.858934169279 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:05][2702 iterations][17%] [558.7941704931123 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:06][3259 iterations][20%] [556.9051580698836 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:07][4102 iterations][25%] [558.5885486018642 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:08][4640 iterations][28%] [555.9863705792504 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:10][5479 iterations][33%] [555.9884127459794 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:11][6056 iterations][37%] [557.3011260443153 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:12][6844 iterations][42%] [554.6675191815857 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:13][7422 iterations][45%] [555.9091245467328 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:15][8251 iterations][50%] [555.2521148338107 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:16][8794 iterations][53%] [554.5147995503934 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:17][9653 iterations][58%] [556.0123329907502 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:18][10187 iterations][62%] [555.1464631365413 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:20][11067 iterations][67%] [556.8823382463153 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:21][11646 iterations][70%] [558.2889227255424 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:22][12513 iterations][75%] [559.6908864807248 iterations/s]
 [Threads => 100] [warmUp => false]
```

```
[0:00:24][13377 iterations][80%] [560.5196319273848 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:25][13952 iterations][83%] [561.1783986888915 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:26][14795 iterations][88%] [561.2248746087415 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:27][15363 iterations][92%] [561.705065775129 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:29][16243 iterations][97%] [562.9609208077745 iterations/s]
 [Threads => 100] [warmUp => false]
[0:00:30][16812 iterations][100%] [562.8103856281039 iterations/s]
 [Threads => 100] [warmUp => false]
2015-09-22 18:36:22,256 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Reached test
 end. All messages were prepared to be sent.
2015-09-22 18:36:22,257 INFO  {org.perfcake.reporting.ReportManager}
 Reporting final results:
2015-09-22 18:36:22,258 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Shutting
 down execution...
2015-09-22 18:36:22,421 INFO  {org.perfcake.ScenarioExecution} ===
 Goodbye! ===
```

For more details on running perfcake from a command line see Section 2.3.1, “Command line parameters”

1.2. Your first performance test

We have actually executed a performance test while validating our installation. Let's take a look on it in more detail. Then we will walk through a quickstart example taken from our web pages.

1.2.1. First out-of-the box demo with httpbin.org

Performance test execution in PerfCake is driven by a so called scenario. Scenarios are by default placed in $PERFCAKE_HOME/resources/scenarios. You can place them in any location you want, or they can be even online. But just this location is searched automatically. You even do not need to specify the file extension when running a scenario. There are couple of supported formats (see Section 2.1 for a complete list) that are recognized automatically.

The scenario simply specifies how the load should be generated, where the load/requests/messages should be sent to, what the request should look like and what do you want to measure/report. You can also ask PerfCake to validate your messages or use sequences to make each request uniqe. These advanced concepts are described later.

The scenario specification is pretty self-explaining. You just need to know what are the possibilities (see Chapter 4, *Reference Guide* for hints on this).

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <scenario xmlns="urn:perfcake:scenario:7.0">
3   <run type="${perfcake.run.type:time}"
4        value="${perfcake.run.duration:30000}"/>
5   <generator class="DefaultMessageGenerator"
6        threads="${perfcake.thread.count:100}"/>
7   <sender class="HttpSender">
```

```
 8        <target>http://${server.host}/post</target>
 9        <property name="method" value="POST"/>
10     </sender>
11     <reporting>
12        <reporter class="IterationsPerSecondReporter">
13           <destination class="ConsoleDestination">
14              <period type="time" value="1000" />
15           </destination>
16        </reporter>
17     </reporting>
18     <messages>
19        <message uri="plain.txt"/>
20     </messages>
21  </scenario>
```

As you can see, the simplest scenario runs for 30000ms = 30 seconds. It generates messages/requests using 100 threads and sends them via HTTP to the server specified in a property (see below for explanation) using the POST method. Performance test results are reported to the console every 1000ms or 1 second. The content of the messages that are being send is specified in the plain.txt file.

As you can see, there are some strange construts at a few places in the scenario. These are ${property:default}. These are replaced by real values specified at the command line. If they are not specified, they are replaced by the default values (configured following the colon). If there is no default value and you do not pass the value at the command line, an empty string is used instead. These properties can be used to dynamically change the behavior of the scenario without actually changing the file.

To run the scenario, we can simply invoke PerfCake via the shell/bat script as described earlier. The only mandatory command line argument is -s <scenario name>. To provide property values we use -Dproperty=value. That's it. Try it once more.

```
./bin/perfcake.sh -s http -Dserver.host=httpbin.org
```

1.2.2. Your own quickstart

Preparing PerfCake

After getting (see Section 1.1.1, “Downloading distribution”) and unpacking PerfCake you will have your directory with the following structure:

```
$PERFCAKE_HOME
├── bin/
│  ├── perfcake.bat
│  └── perfcake.sh
├── docs/
│  └── perfcake-7.x-javadoc.jar
├── lib/
│  ├──ext/
│  ├──plugins/
│  └── *.jar
├── resources/
│  ├── keystores/
│  ├── messages/
│  ├── scenarios/
│  ├── schemas/
```

```
│  └── xslt/
├── LICENSE.txt
└── log4j2.xml
```

You may try to run PerfCake, you should receive output like this:

```
$PERFCAKE_HOME/bin/perfcake.sh
=== Welcome to PerfCake 7.5 ===
usage: ScenarioExecution -s <SCENARIO> [options] [-D<property=value>]*
 -d,--debug                                  start debug JMX agent for
                                             external monitoring
 -D <property=value>                         system properties
 -dn,--debug-name <AGENT_NAME>               debug agent name in the JMX
                                             tree
 -h,--help                                   prints help/usage
 -log,--log-level <LOG_LEVEL>                logging level
 -md,--messages-dir <MESSAGES_DIR>           directory for messages
 -pd,--plugins-dir <PLUGINS_DIR>             directory for plugins
 -pf,--properties-file <PROPERTIES_FILE>     custom system properties
 file
 -r,--replay <RAW_FILE>                      raw file to be replayed
 -s,--scenario <SCENARIO>                    scenario to be executed
 -sd,--scenarios-dir <SCENARIOS_DIR>         directory for scenarios
 -skip,--skip-timer-benchmark                skip system timer benchmark
```

The script assumes you have JDK installed and available on the system path, minimal version 1.8 is required. Please note that the system being tested is not required to run on Java 8. It might not run on Java at all!

In the `bin` directory you can find executable scripts for running PerfCake on Linux, Windows and Mac.

The `lib` directory contains application libraries. You do not have to take any care of these.

What is more interesting is the `resources` directory. In its subdirectories you can find sample scenarios, messages and all versions of XSD schemas for scenario files. The `keystores` directory is used for specific message sender, but we will not deal with it in this quickstart.

If you feel like going wild, you can download [1] the source distribution and compile it by

```
mvn clean package assembly:assembly
```

Then you can find the binary distributions in the `target` directory and continue with this quickstart guide. You will also see the output of tests so you can be sure the project works fine on your system.

Configure and run

In these days, your only possibility to prepare your first scenario is an XML file. You can use your favourite editor to create this file. The structure is defined by an XSD schema that can be found under `resources/schemas` directory. Some of the editors are able to use the schema file to suggest you valid tags. Our future plans include providing GUI editor for Eclipse and InteliJ Idea that would allow you to create and edit scenarios, stay tuned! If you wanted to contribute, we are happy to welcome you in our community [2].

[1] https://www.perfcake.org/download
[2] https://www.perfcake.org/community

At minimum, simple scenario has to contain definitions for:

- Generator - how the load will be generated (see Section 4.1, “How - Generating load”)
- Sender - where the load will be sent - interface or protocol with address, you can choose from many already implemeted (see Section 4.2, “Where - Sending messages”)

Let’s assume you need to stress your web application that has some function exposed on the following URL: `http://your-app.com/cool-app` , and you need to test how fast the function is. You want to generate maximum load for 10 seconds (10000 miliseconds) with 10 simultaneous clients (working threads).

If you do not have any such application at hand, you can consider using http://httpbin.org/get but be polite and do not overload their server. It is provided for free.

Now the important part comes. Create a file called `http-echo.dsl` (`http-echo.xml` resp.) in the `$PERFCAKE_HOME/resources/scenarios` directory. Now insert the following DSL (XML resp.) snippet in it:

`http-echo.dsl`:

```
  1  scenario "http-echo"
  2     run 10.s with ${thread.count:10}.threads
  3     generator "DefaultMessageGenerator"
  4     sender "HttpSender" target "http://your-app.com/cool-app"
 method "GET"
  5  end
```

`http-echo.xml`:

```
  1 <?xml version="1.0" encoding="utf-8"?>
  2 <scenario xmlns="urn:perfcake:scenario:7.0">
  3    <run type="time" value="10000"/>
  4    <generator class="DefaultMessageGenerator"
  5               threads="${thread.count}"/>
  6    <sender class="HttpSender">
  7       <target>http://your-app.com/cool-app</target>
  8    </sender>
  9 </scenario>
```

You can see `"${thread.count:10}"` in the generator’s definition. That is a system property `"thread.count"` that you may set and the actual value of the property will be used. If the property is not set, the default value (10) will be used.

Now, all you need to do is to execute your new test scenario by running the following command:

```
$PERFCAKE_HOME/bin/perfcake.sh -s http-echo
```

Please note you do not need to specify the DSL (XML) extension. Only if you used both DSL and XML variants.

Now you are running your fisrt stress test. Even if you cannot see what is going on, PerfCake sends requests to your application in many threads. The test should run approximately for 10 seconds. If you want to see some numbers (e.g. how fast your system is), you have to add one more element to your scenario to evaluate the results - the reporting.

Evaluate results

For reporting some results of your measurement, you have to configure a Reporter - an object that is capable of computing results in some way and outputing them wherever you can imagine.

Copy your `http-echo.dsl` (`http-echo.xml` resp.) file to `http-reporting.dsl` (`http-reporting.xml` resp.) and have it look like the listing below.

`http-reporting.dsl`:

```
 1 scenario "http-reporting"
 2    run 10.s with ${thread.count:10}.threads
 3    generator "DefaultMessageGenerator"
 4    sender "HttpSender" target "http://your-app.com/cool-app"
method "GET"
 5    reporter "ResponseTimeStatsReporter" minimumEnabled "false"
maximumEnabled "false"
 6       destination "ChartDestination" every 1.s
name "Response Time" group "rt" yAxis "Response Time [ms]"
attributes "Result,Average"
 7       destination "ConsoleDestination" every 1.s
 8 end
```

`http-reporting.xml`:

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <scenario xmlns="urn:perfcake:scenario:7.0">
 3    <run type="time" value="10000"/>
 4    <generator class="DefaultMessageGenerator"
 5               threads="${thread.count:10}"/>
 6    <sender class="HttpSender">
 7       <target>http://your-app.com/cool-app</target>
 8       <property name="method" value="GET"/>
 9    </sender>
10    <reporting>
11       <reporter class="ResponseTimeStatsReporter">
12          <property name="minimumEnabled" value="false"/>
13          <property name="maximumEnabled" value="false"/>
14          <destination class="ChartDestination">
15             <period type="time" value="1000"/>
16             <property name="name" value="Response Time"/>
17             <property name="group" value="rt"/>
18             <property name="yAxis" value="Response Time [ms]"/>
19             <property name="attributes" value="Result,Average"/>
20          </destination>
21          <destination class="ConsoleDestination">
22             <period type="time" value="1000"/>
23          </destination>
24       </reporter>
25    </reporting>
26 </scenario>
```

Adding the `reporting` section you let your scenario to log results to some destination - in our case to the PerfCake's console. Output will be provided every 2 seconds (2000 miliseconds).

Try to run the scenario again by the following command:

```
2016-06-14 22:01:56,155 INFO  {org.perfcake.ScenarioExecution} ===
 Welcome to PerfCake 7.5 ===
2016-06-14 22:01:56,386 INFO  {org.perfcake.scenario.ScenarioBuilder}
 Scenario configuration: file:/home/perfcake/perfcake-7.5/resources/
scenarios/http-reporting.xml
2016-06-14 22:01:56,455 INFO  {org.perfcake.util.TimerBenchmark}
 Benchmarking system timer resolution...
2016-06-14 22:01:56,456 INFO  {org.perfcake.util.TimerBenchmark} This
 system is able to differentiate up to 334ns. A single thread is now
 able to measure maximum of 2994011 iterations/second.
2016-06-14 22:01:56,468 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Starting to
 generate...
[0:00:01][84 iterations][17%] [141.868302 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [Minimum => 136.702416
 ms] [failures => 0] [RequestSize => 0.00 B] [Average =>
 170.70281994047613 ms]
[0:00:02][155 iterations][27%] [140.435482 ms] [warmUp =>
 false] [Threads => 10] [ResponseSize => 0.00 B] [Minimum =>
 135.436574 ms] [failures => 0] [RequestSize => 0.00 B] [Average =>
 157.30522732903225 ms]
[0:00:03][225 iterations][37%] [136.248079 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [Minimum => 134.629032 ms]
 [failures => 0] [RequestSize => 0.00 B] [Average => 151.3000900355556
 ms]
[0:00:04][292 iterations][47%] [146.018769 ms] [warmUp =>
 false] [Threads => 10] [ResponseSize => 0.00 B] [Minimum =>
 134.629032 ms] [failures => 0] [RequestSize => 0.00 B] [Average =>
 151.79159557191784 ms]
[0:00:05][360 iterations][57%] [140.239096 ms] [warmUp =>
 false] [Threads => 10] [ResponseSize => 0.00 B] [Minimum =>
 134.629032 ms] [failures => 0] [RequestSize => 0.00 B] [Average =>
 150.78039243888892 ms]
[0:00:06][430 iterations][67%] [140.608011 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [Minimum => 134.077543 ms]
 [failures => 0] [RequestSize => 0.00 B] [Average => 149.0632376767442
 ms]
[0:00:07][500 iterations][77%] [150.971978 ms] [warmUp =>
 false] [Threads => 10] [ResponseSize => 0.00 B] [Minimum =>
 134.077543 ms] [failures => 0] [RequestSize => 0.00 B] [Average =>
 148.20699049399997 ms]
[0:00:08][571 iterations][87%] [136.759408 ms] [warmUp =>
 false] [Threads => 10] [ResponseSize => 0.00 B] [Minimum =>
 134.077543 ms] [failures => 0] [RequestSize => 0.00 B] [Average =>
 147.44373894921188 ms]
[0:00:09][641 iterations][97%] [137.870229 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [Minimum => 134.077543 ms]
 [failures => 0] [RequestSize => 0.00 B] [Average => 146.7737520499219
 ms]
2016-06-14 22:02:06,508 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Reached test
 end. All messages were prepared to be sent.
2016-06-14 22:02:06,509 INFO  {org.perfcake.reporting.ReportManager}
 Checking whether there are more results to be reported...
```

```
[0:00:10][662 iterations][100%] [142.601692 ms] [warmUp =>
 false] [Threads => 10] [ResponseSize => 0.00 B] [Minimum =>
 134.077543 ms] [failures => 0] [RequestSize => 0.00 B] [Average =>
 146.55668455286997 ms]
2016-06-14 22:02:06,620 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Shutting
 down execution...
2016-06-14 22:02:06,620 INFO  {org.perfcake.ScenarioExecution} ===
 Goodbye! ===
```

The `warmUp` attribute you can see in the results determines the mode of the test. In our example we do not wait for the server to warm up so the attribute is set to false all the time.

ChartDestination specified in the scenario produces a chart report with the chart similar to the following that can be found under `$PERFCAKE_HOME/perfcake-charts` directory.

Response Time (created: 11.5.2016 15:08:43)

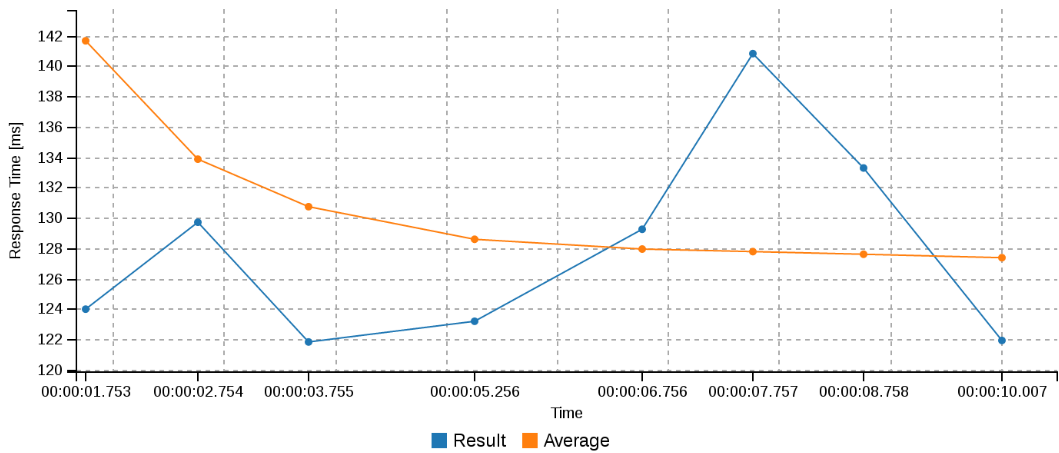

Figure 1.1. Your first performance test chart report

Chapter 2. PerfCake Features

2.1. PerfCake Architecture Overview

Let's start with a Figure that is worth a thousand words.

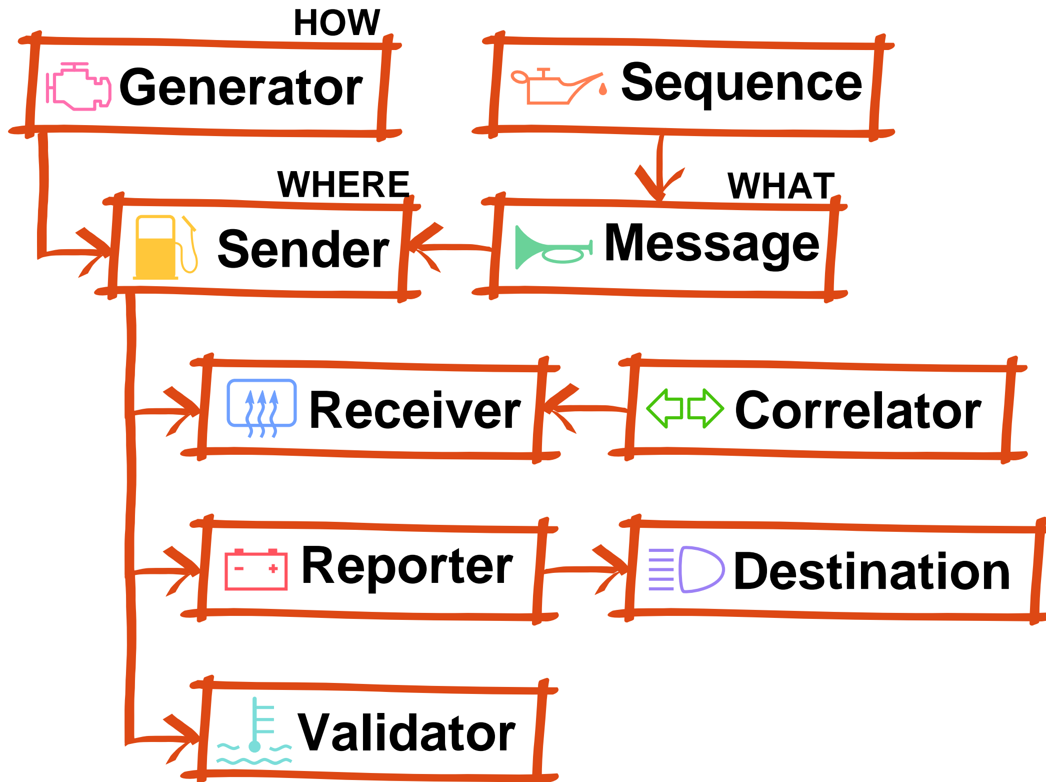

Figure 2.1. Architecture Overview

In the Figure 2.1, "Architecture Overview" , we can see the high level overview of PerfCake's architecture. There is always a single `Generator`that works like an *engine* in the performance test. The main purpose of the `Generator` is to specify how the messages are generated. The easiest case would be to send a message to the target system, wait for the response and measure the response time. However, this would not tell us anything about performance of the target system. What is more interesting is a load generated in many parallel threads as fast as possible.

The default generator implementation (`DefaultMessageGenerator`) has a queue of messages ready to be send by a `Sender` to the target system. More details about `Generator` architecture cen be found in the next section. For now, just remember that `Generator` specifies *HOW* the messages are sent.

A `Sender` represents a protocol specific *"pump"* of message to a target service or system under test. The protocols can be HTTP, REST, JMS, JDBC, SOAP, socket, file etc. The `Sender` is configured with a specific address for the given protocol. So the Sender tells *WHERE TO* send the messages.

A `Message` is the content of the smallest unit of load that is sent to the target system. So basically it says *WHAT* is sent.

For the ability to have different `Message` each time it is sent or to fill in some dynamic values, a sophisticated templating engine is used. Each message can be enriched (*"oiled"*) with dynamic values. To provide the dynamic values, we can use a `Sequence` . This is a simple interface that returns another value in the row each time it is called.

Optionally, a response to the original request can be received through a separate message channel. This is where `Receivers` come at hand. A `Receiver` manages an `"inflow"` of responses to the original requests and passes them to a `Correlator`. The `Correlator` then matches the responses to the original requests based on some specific information (i.e. *correlation ID*) in both of the messages. A complete request-response cycle is then being measured.

The procedure of sending the messages is carefully monitored by the reporting facility of PerfCake. The metrices are implemented using `Reporters`. A `Reporter` accumulates single measurements of individual requests into meaningful values (e.g. throughput, memory usage, response time, service time, response time histogram, etc.) and waits for them to be *"discharged"* at specified intervals into a `Destination`. The `Destination` specifies where the measured results are stored (e.g. chart, CSV file, log file, database, etc.).

Finally, there are validators that prove the responses to be valid. Some system can start responding very fast under a heavy load, but just with a blunt overload warning. To prevent any such failure, a `Validator` can valide its content.

More details about individual parts of the application can be found in the later sections.

2.2. Performance scenario definition

Scenario is a receipt for telling PerfCake what to do. You can specify how PerfCake would generate load by configuring a generator, where and what to send by defining a sender and messages. To get any measured results such as an average throughput or a memory usage you can use reporting capabilities. To check that the responses are correct a validation is available for you to set in the scenario. There is also a possibility to specify scenario meta-data by setting the scenario's properties.

2.2.1. XML scenario

As you can see from the following listing the XML scenario is defined by `urn:perfcake:scenario:7.0` namespace [1]. The scenario is divided into several sections: Properties, Run, Generator, Sequences, Sender, Receiver, Reporting, Messages and Validation.

Scenario structure

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <scenario xmlns="urn:perfcake:scenario:7.0">
 3    <!-- Scenario properties (optional) -->
 4    <properties>
 5       <property name="..." value="..."/>
 6       ...
 7    </properties>
 8
 9    <!-- Run section (required) -->
10    <run ... >
11       ...
```

[1] Schema can be found at http://schema.perfcake.org/perfcake-scenario-7.0.xsd

```
12      </run>
13
14      <!-- Generator section (required) -->
15      <generator ... >
16         ...
17      </generator>
18
19      <!-- Sequences section (optional) -->
20      <sequences>
21         ...
22      </sequences>
23
24      <!-- Sender section (required) -->
25      <sender ... >
26         ...
27      </sender>
28
29      <!-- Receiver section (optional) -->
30      <receiver>
31         ...
32      </receiver>
33
34      <!-- Reporting section (optional) -->
35      <reporting>
36         ...
37      </reporting>
38
39      <!-- Messages section (optional) -->
40      <messages>
41         ...
42      </messages>
43
44      <!-- Validation section (optional) -->
45      <validation>
46         ...
47      </validation>
48   </scenario>
```

Sections of the scenario

Let's take a look at particular sections of the scenario.

Scenario properties

This optional section allows you to add some meta-data about your scenario. It can contain multiple properties.

All the scenario properties are set as Java System properties so can be used further in scenario (See Section 2.2.5, “Filtering properties” for more details.).

Run

The run section specifies the duration for what the scenario will run. It is mandatory since PerfCake needs to know how long to generate load.

The scenario run configuration is described in more details in Section 4.1, “How - Generating load” .

Generator

The generator section specifies the way how the load is generated. It is mandatory since PerfCake needs to know how to generate load.

The generators are described in more details in Section 4.1, “How - Generating load” .

Sender

The sender section is about the transport (e.g. HTTP, MQTT, JMS, ...) and the target where the load is directed. It is required to be specified in the scenario.

More information about the senders can be found in Section 4.2, “Where - Sending messages” .

Receiver and Correlator

The receiver section defines a component used to receive responses from a separate message channel. Because it is possible to send requests to some protocol and use a completely different protocol to receive responses.

A receiver always needs to know which response matches which original request. This is why it needs a correlator to be specified.

More information about the receivers and senders can be found in their corresponding chapters Section 4.3, “Receiving messages” and Section 4.4, “Correlating messages”.

Reporting

Reporting module is responsible for gathering metrics and reporting the results to various places in specified moments. It is not required to configure the reporting in the scenario but without it the PerfCake has no way of measuring and reporting results.

The reporting abilities are described in Section 4.7, “Reporting” .

Messages

The messages represent the payload that is transferred by senders to the tested system. It is optional since there can be situations where there is no actual message being send.

The Section 4.5, “What - Messages” describes the messages in more details.

Sequences

Inside of the messages and scenario configuration properties, you can use sequences of values. The sequences can be configured separately in the scenario and just provide a next value in the sequence each time they are queried. This allows you to have unique messages or even send the message to various targets.

The Section 4.6, “Sequences” describes the sequences in more details.

Validation

Validation module allows to validate the response messages.

The validation capabilities are described in Section 4.8, “Validation” .

2.2.2. DSL scenario

There is a possibility to specify scenarios in the form that resembles a natural language. This is typically stored in a file with the .dsl suffix. It is useful to open these scenarios in an editor that supports Groovy syntax as the DSL language is actually developed in Groovy.

The DSL scenarios can use the same features, properties and constructs as the XML scenario. The language is just different. Following is a sample DSL scenario.

```
 1 scenario "my cool scenario"
 2    qsName "test" propA "hello"
 3    run 10.s with 4.threads
 4    generator "DefaultMessageGenerator" senderTaskQueueSize 3000
 5    sender "TestSender" target "httpbin.org" delay 12.s
 6    reporter "WarmUpReporter"
 7    reporter "ThroughputStatsReporter" minimumEnabled false
 8       destination "CsvDestination" every 3.s
path '${perfcake.scenario}-stats.csv' enabled
 9       destination "ConsoleDestination" every 5.percent disabled
10    reporter "ResponseTimeStatsReporter"
11       destination "ConsoleDestination" every 10.percent
12    message file:"message1.xml" send 10.times
13    message content:"Hello World" values 1,2,3
14    message "file://message2.txt" validate "text1","text2"
15    message "Simple text" propA "kukuk" headers
name:"Franta",count:10 validate "text1, text2"
16    validation fast disabled
17       validator "RegExpValidator" id "text1" pattern "I am a
fish!"
18       validator "RegExpValidator" id "text2" pattern "I was a
fish!"
19 end
```

The language format is typically in the form of `<keyword> <attribute>`. Mandatory keywords used are `scenario`, `run`, `generator` and `sender`.

In general, strings are in quotes or apostrophes, an array and a map are specified by the elements separated by commas. The map elements are specified by the pairs `key:value` separated by a colon. There are some special units defined that can be used with numbers. The format is a number followed by a dot and the unit name. Following is the list of supported units, in the scenario source, they are represented by the abbreviations shown in the parentheses: milliseconds (ms), seconds (s), minutes (m), hours (h), days (d), iterations (iteration, iterations), percents (percent, percents), threads (thread, threads), and times (times).

`Scenario` is followed by the scenario name and mandatory properties and their values.

`Run` is followed by the time specification and the number of threads after the `with` keyword.

`Generator`, `sender`, `reporter`, `destination`, `validator` and `sequence` are followed by the class name implementing the component. Then there are properties and their values for the given component. A single component configuration cannot be split to multiple lines. It must all be present at a single line.

`Message` is followed by the location of the message or its content. Either there is a differentiator in the form `file:` or `content:`, or there is a string with the protocol specification (`file://`, `http://`...), or a string with message content. Following are more configuration properties.

2.2.3. Scenario specified through API

It is also possible to run PerfCake by utilizing its API. The simplest way is to load a scenario specified in an external file (or even URL location) and execute it.

Example 2.1. Loading of a scenario definition and its execution from API

```
import org.perfcake.scenario.Scenario;
import org.perfcake.scenario.ScenarioLoader;
...
      final Scenario scenario = ScenarioLoader.load("/full/path/
http.xml");

      scenario.init();
      scenario.run();
      scenario.close();
```

A call to the `run()` method is blocking and does not return until all messages are sent or a fatal error occurs. It is however possible to stop the scenario execution from another thread by a call to the `stop()` method of the `Scenario` class. The termination is not immediate, rather a graceful shutdown is performed.

In a case when you do not require access to the scenario control object (org.perfcake.scenario.Scenario) and you just want to run a test (for example in TestNG or jUnit), there is a simplified method of the above.

Example 2.2. Executing a scenario with a single method call using PerfCake API

```
import org.perfcake.scenario.ScenarioExecution;
...
      ScenarioExecution.execute("scenario-name", new Properties());
```

As a scenario name, the same value is passed as in the -s parameter on the command line. The properties can configure anything that would normally go into -D parameter. Most other command line parameters can be configured via these properties (see ug.perfcake-features.scenario-definition.filtering-properties).

It is also possible to build the complete scenario from scratch using PerfCake API. Most objects are designed with fluent API so you can easily join multiple setter methods.

Example 2.3. Complete scenario definition and execution using its API

```
import org.perfcake.PerfCakeException;
import org.perfcake.RunInfo;
import org.perfcake.common.BoundPeriod;
import org.perfcake.common.Period;
import org.perfcake.common.PeriodType;
import org.perfcake.message.Message;
import org.perfcake.message.MessageTemplate;
import org.perfcake.message.correlator.Correlator;
import org.perfcake.message.correlator.GenerateHeaderCorrelator;
import org.perfcake.message.generator.DefaultMessageGenerator;
import org.perfcake.message.generator.MessageGenerator;
import org.perfcake.message.receiver.HttpReceiver;
import org.perfcake.message.receiver.Receiver;
import org.perfcake.message.sender.HttpSender;
```

```
import org.perfcake.message.sender.MessageSender;
import org.perfcake.message.sequence.PrimitiveNumberSequence;
import org.perfcake.message.sequence.Sequence;
import org.perfcake.reporting.destination.ConsoleDestination;
import org.perfcake.reporting.destination.Destination;
import org.perfcake.reporting.reporter.IterationsPerSecondReporter;
import org.perfcake.reporting.reporter.Reporter;
import org.perfcake.validation.MessageValidator;
import org.perfcake.validation.RegExpValidator;
...
      final Period period = new Period(PeriodType.TIME, 30_000);
      final RunInfo runInfo = new RunInfo(period);

      final MessageGenerator generator = new
 DefaultMessageGenerator();
      generator.setThreads(10);

      final MessageValidator validator = new
 RegExpValidator().setCaseInsensitive(true).setPattern(".*");

      final Message message = new Message();
      message.setPayload("Hello world no. @{intSeq}!");

      final MessageTemplate messageTemplate = new
 MessageTemplate(message, 1, Collections.singletonList("regExp"));

      final MessageSender sender = new
 HttpSender().setMethod(HttpSender.Method.POST).setTarget("http://
httpbin.org/post");

      final Correlator correlator = new GenerateHeaderCorrelator();

      final Receiver receiver = new
 HttpReceiver().setSource("localhost:8282").setThreads(10);

      final Sequence sequence = new PrimitiveNumberSequence();

      final Destination destination = new ConsoleDestination();

      final Reporter reporter = new IterationsPerSecondReporter();
      reporter.registerDestination(destination, new
 Period(PeriodType.TIME, 1000));

      final ScenarioBuilder builder = new ScenarioBuilder(runInfo,
 generator, sender);
      builder.setReceiver(receiver).setCorrelator(correlator);
      builder.putMessageValidator("regExp",
 validator).addMessage(messageTemplate).
            putSequence("intSeq", sequence).addReporter(reporter);

      final Scenario scenario = builder.build();

      scenario.init();
      scenario.run();
```

```
scenario.close();
```

The API is slightly counter-intuitive because the scenario needs to be composed from the bottom to top. For example, we first need to define a *Message Validator* to validate a message and then use it to create a *Message Template*. Next, we set the *Message Validator* reference ID later when passing it to the `ScenarioBuilder`.

Similar concept works for *Sequences* and *Message Reporters* and others.

2.2.4. IDE plugins

So far we have prepared preliminary versions of plugins for Eclipse and IntelliJ Idea. Some efforts were done for NetBeans as well. None of the plugins is stable enough at the moment. We are in a search of a brave contributor who could consolidate all the plugins and make them stable and user friendly.

If you want to go wild, try to search the default repositories of your IDE. This comes with no guarantees at the moment.

2.2.5. Filtering properties

It is possible to use property placeholders in scenarios (and in other components of the scenario, see later). The placeholders are replaced by the actual value of the particular property or by the default value if specified in a process called property filtering.

This proces is quite complex and allows many useful manipulations with the values provided in the scenario.

At first, when the scenario is loaded from a file (no matter which format is used), all the placeholders of the following format are loaded and replaced by appropriate values if they are found.

```
${<property name>:<default value>}
```

The colon and the default value are optional. Also, any properties specified in the scenario are loaded and put into system properties.

During this phase, PerfCake checks for properties in the system properties and then in environment properties. We can also limit it to check just one of the sources by providing a prefix to the property name. The prefix can be *env.* for environment properties and *props.* for system properties.

The properties found in the scenario can be used throughout the scenario as well and can be also used as in other properties. However, circular definitions do not work.

Example 2.4. An example of a scenario with properties

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <scenario xmlns="urn:perfcake:scenario:7.0">
3    <properties>
4
<property name="composedProperty" value="${defaultProperty}-2"/>
5
<property name="defaultProperty" value="${test.missing.property:default-property-value}"/>
6    </properties>
7    ...
```

```
 8     <messages>
 9        <message uri="unfiltered-message.txt">
10           <header name="testHeader" value="${defaultProperty}"/>
11        </message>
12     </messages>
13 </scenario>
```

The property filtering process is performed in a moment, when a scenario file is loaded by PerfCake, before it is parsed. There are no advanced features, just a simple string replacement for scenario files. The placeholders can be at any place in the scenario file because PerfCake approaches the scenario as a simple string without any syntax meaning.

The filtering tries to find the property by name. If the property is found, the whole placeholder is replaced by property's actual value. If the specified property does not exist, it looks for the default value if it is specified. If so the whole placeholder is replaced by the default value. Otherwise it leaves the placeholder in place intact.

More complex filtering is available for some components of the system. The advanced filtering works for: *Sender*'s *target* attribute, any part of *Message* including content loaded from an external file, and for patterns in *RegExpValidator*.

The following types of placeholders are supported:

```
${<property name>:<default value>}
@{<property name>:<default value>}
```

The following characters can be escaped by a backslash: `\` (backslash itself), `$`, `@`, `{`, and `}`. These characters should not appear unescaped if they are not supposed to act as control characters. However, the default value of a property can contain anything except for `}`. Also, backshlash needs to be escaped. Colon is not allowed in the property name and no characters can be escaped in it.

The following table lists a few examples of strings and how they are rendered in the end. Suppose we have the following system properties set: `ab = 1`, `cd = 2`.

Template	Result
`${ab:cd}`	`1`
`\\${cd}`	`\2`
`\${ab}`	`${ab}`
`${non.existing:\}$@\\}`	`}$@\`
`\\\@{cd}`	`\@{cd}`
`${env.JAVA_HOME}`	System dependant, for example: `/opt/jdk`

Table 2.1. Examples of templates and how they are rendered

Properties with the *dollar* sign ($) are replaced just once, while the properties with the *at* sign (@, so called dynamic properties) are alsways rendered again with every single use and thus containing a fresh value. This is mainly useful for Section 4.6, “Sequences”. Please note that rendering values of the placeholders with the *at* sign can introduce a slight performance impact. PerfCake tries to switch off the placeholder rendering whenever possible (when there are no dynamic properties present in the particular string).

In the case of advanced property filtering, the environment and system properties need to be strictly separated and referenced with the corresponding prefix. The environment properties are accessed by

`${env.<property name>}`. The system properties can be accessed by `${props.<property value>}`, `${props[<property value>]}`, and `${props['<property value>']}`.

Also, these placeholders are considered ultimate and no more replacing is possible. So in the case when there is no property available and there is no default value, they are replaced by the `null` string.

Both of the replacement strategies can be combined to achieve interesting results. For example, to define a property in a scenario the value of which is derived from another property defined there, and to use it in the meesage content, we need to use the following approach.

Example 2.5. An example of a scenario with combined property replacement strategies

```
 1 <?xml version="1.0" encoding="utf-8"?>
 2 <scenario xmlns="urn:perfcake:scenario:7.0">
 3    <properties>
 4
  <property name="composedProperty" value="${props.defaultProperty}-2"/
>
 5
  <property name="defaultProperty" value="${test.missing.property:default-
property-value}"/>
 6    </properties>
 7    ...
 8    <messages>
 9       <message content="${props.composedProperty}"/>
10    </messages>
11 </scenario>
```

As you can see, we had to use the advanced approach to refer to the `composedProperty` and to the `defaultProperty`. This is because PerfCake first reads the scenario, defines the values of both `composedProperty` and `defaultProperty`. But at that time, we did not know the value of the `defaultProperty` and it could not be used to replace the value in the `composedProperty`. So the `composedProperty` remains exactly as seen the example. Now PerfCake parses the scenarion by a syntax parser. The value of the `defaultProperty` is now known but in this stage, only the advanced templating is supported. So the `props.` prefix worked and we got the correct value `default-property-value-2` in the message content.

To provide PerfCake the actual value of the property you can just pass it using an ordinary way:

`-D<property name>=<property value>`

Note

In Windows command shell, the parameter specifying the property name and value might need to be placed in double quotes: `"-D<property name>=<property value>"`

There are several properties that exist in PerfCake and that might be usefull in the scenarios or messages (e.g. a timestamp of the scenario execution start). The following table describes all available internal properties.

Property name	Description
perfcake.encoding	Default encoding

Property name	Description
perfcake.messages.dir	Messages directory
perfcake.plugins.dir	Plugins directory
perfcake.properties.file	Custom properties file
perfcake.run.timestamp	A Unix timestamp of the moment of the scenario execution start
perfcake.run.nice.timestamp	A timestamp of the moment of the scenario execution start in a human readable format (`yyyyMMddHHmmss`).
perfcake.scenario	A name of the scenario
perfcake.scenarios.dir	Scenarios directory
perfcake.logging.level	Logging level of the PerfCake core

Table 2.2. Available PerfCake internal properties

2.3. Running PerfCake

2.3.1. Command line parameters

PerfCake is a command line tool so it provides a CLI (command line interface) for executing a PerfCake scenario (see Section 2.2, “Performance scenario definition”). The main script can be found in `$PERFCAKE_HOME/bin/perfcake.sh` for Linux OS or `%PERFCAKE_HOME%\bin\perfcake.bat` in case of Windows OS.

If the script is run without arguments, from the following help that is printed you can see the arguments that can be used to control the scenario execution.

```
[PERFCAKE_HOME]$ ./bin/perfcake.sh
=== Welcome to PerfCake 7.5 ===
usage: ScenarioExecution -s <SCENARIO> [options] [-D<property=value>]*
 -d,--debug                                start debug JMX agent for
                                           external monitoring
 -D <property=value>                       system properties
 -dn,--debug-name <AGENT_NAME>             debug agent name in the JMX
                                           tree
 -h,--help                                 prints help/usage
 -log,--log-level <LOG_LEVEL>              logging level
 -md,--messages-dir <MESSAGES_DIR>         directory for messages
 -pd,--plugins-dir <PLUGINS_DIR>           directory for plugins
 -pf,--properties-file <PROPERTIES_FILE>   custom system properties
 file
 -r,--replay <RAW_FILE>                    raw file to be replayed
 -s,--scenario <SCENARIO>                  scenario to be executed
 -sd,--scenarios-dir <SCENARIOS_DIR>       directory for scenarios
 -skip,--skip-timer-benchmark              skip system timer benchmark
```

The only mandatory argument is `-s` (or `--scenario`), that specifies the name of the scenario to be executed. PerfCake will look for the scenario definition in the file called `<SCENARIO>.xml` that is placed under the directory that is located at `$PERFCAKE_HOME/resources/scenarios` . The path to the directory with scenarios can be specified by the `-sd` (or `--scenarios-dir`) argument.

If the scenario is configured to send any message (see Section 4.5, “What - Messages”), it will look under the directory that is by placed at $PERFCAKE_HOME/resources/messages . That location can be specified by the -md (or --messages-dir) argument.

If you need to set one or more system properties for scenario (see Section 2.2.5, “Filtering properties”), there are two ways. You can either use -Dproperty=value like arguments or specify a path to the property file using the -pf (or --properties-file) argument.

Following table describes all the PerfCake CLI arguments.

Argument	Description	Required	Default value
-s, --scenario	Name of the scenario to be executed.	Yes	-
-log, --log-level <LOG_LEVEL>	Default PerfCake logging level. Can be trace, debug, info, warn, error.	No	info
-D<property>=<value>	Sets a single system property with a given name and value. Multiple property arguments are allowed.	No	-
-sd, --scenarios-dir	Path to the directory where scenarios are located.	No	$PERFCAKE_HOME/resources/scenarios
-md, --messages-dir	Path to the directory where message files are located.	No	$PERFCAKE_HOME/resources/messages
-pd, --plugins-dir	Path to the directory where plugins are located (see Section 6.2, “Custom components - Plugins”).	No	$PERFCAKE_HOME/resources/plugins
-r, --replay <RAW_FILE>	Replay the previously raw data with the given scenario. No test is executed, just the reporters are used to create reports from the <RAW_FILE> data file.	No	-
-pf, --properties-file	Path to the file containing system properties.	No	-
-skip, --skip-timer-benchmark	Skip system timer benchmark during PerfCake startup. This can speedup the startup a little bit.	No	false
-d, --debug	Starts an internal debug agent that publishes status of PerfCake components via JMX. This is an experimental feature suited for performance scenario development. It has a significant performance drawback and must not be used for real testing.	No	false

Argument	Description	Required	Default value
-dn, --debug-name	Name of the debug agent for the case when there were multiple PerfCake instances in the same JVM sharing the JMX data.	No	perfcake-1
-h, --help	Prints PerfCake help/usage information.	No	-

Table 2.3. PerfCake CLI arguments

Some useful properties can be configured using the `-Dproperty=value parameter`. These are not so common so there is no dedicated short parameter variant. The following table lists the other properties recognized by PerfCake. None of them is required.

Property name	Description	Default value
perfcake.scenario	An alternative way to specify the PerfCake scenario. It is equivalent to the `-s` parameter.	-
perfcake.encoding	The encoding used for all inputs and outputs.	UTF-8
perfcake.scenarios.dir	Equivalent of the `-sd` command line parameter.	`$PERFCAKE_HOME/resources/scenarios`
perfcake.messages.dir	Equivalent of the `-md` command line parameter.	`$PERFCAKE_HOME/resources/messages`
perfcake.plugins.dir	Equivalent of the `-pd` command line parameter.	`$PERFCAKE_HOME/resources/plugins`
perfcake.properties.file	Equivalent of the `-pf` command line parameter.	-
perfcake.logging.level	Equivalent of the `-log` command line parameter.	info
perfcake.fail.fast	Allows preliminary termination of PerfCake in the case there is a failure during sending a message. PerfCake needs to wait for all the threads to finish, so you might see an error to be shown multiple times actually. In case this is set to true, no more requests for sending a messages will be created.	`false`
perfcake.templates.disabled	Completely disables the templating engine, no placeholders will be replaced in scenarios and messages. Normally, it is sufficient just not to use the templates. Having the engine on and not using it does not bring any performance overhead. By setting this to true,	`false`

Property name	Description	Default value
	PerfCake can also run in JRE (the templating is the only part of PerfCake depending on JDK).	
perfcake.skip.timer.benchmark	Equivalent of the `-skip` command line parameter.	`false`
perfcake.replay	Equivalent of the `-r` command line parameter.	-
perfcake.debug	Equivalent of the `-d` command line parameter.	`false`
perfcake.debug.name	Equivalent of the `-dn` command line parameter.	perfcake-1

Table 2.4. PerfCake recognized properties

2.3.2. Running scenarios from Maven

PerfCake is provided with a Maven plugin which allows to run PerfCake scenario within Maven build. This makes perfomance test automation more easy and encourages you to run performance tests on a regular basis (e.g. within your favourite CI server). Running performance test on a regular basis allows you to spot performance drops very soon and thus makes it much easier to identify which commit has caused performance regression.

Currently, the plugin has only one goal `scenario-run`, which runs specified PerfCake scenario. By default, this goal is executed in *integration-test* phase (i.e. assumes, that you deploy/start your application in *pre-integration-test* phase and shut it down in *post-integration-test* phase).

The only mandatory parameter is `<scenario>`, which specifies the name of the scenario to be run. Optionally, you can also specify `<scenarios-dir>`, `<messages-dir>` and `<plugins-dir>`, which specify paths to directories containing scenarios, messages and plugins, respectively. If you don't setup these parameters, plugin assumes, that appropriate directories (`scenarios`, `messages`, `plugins`) are in `src/test/resources/perfcake`. This can be switched to `src/main/resources/perfcake` by configuring `<use-test-resources>` to `false`. If any of these directories does not exists, plugin will use `src/test/resources` (or `src/main/resources`) as a fallback value for missing parameter.

The PerfCake Maven plugin requires log4j2 configuration file for PerfCake to report properly. By default it assumes the file to be located in project root and named `log4j2.xml`. This can be changed by the `<log4j2-config>` tag. The PerfCake logging level can be configured with `<log-level>` tag and a file with additional configuration properties can be specified by `<properties-file>`.

The PerfCake version used with the plugin is always the same as plugin version. Both jar files are released together.

Example 2.6. PerfCake Maven plugin sample configuration

```
<properties>
   <perfcake.version>7.5</perfcake.version>
   ...
</properties>
...
```

```
    <build>
       <plugins>
          <plugin>
             <groupId>org.perfcake.maven</groupId>
             <artifactId>perfcake-maven-plugin</artifactId>
             <version>${perfcake.version}</version>
             <configuration>
                <scenario>my_perfcake_scenario</scenario>
                <log4j2-config>src/test/resources/log4j2.xml</log4j2-
config>
             </configuration>
             <executions>
                <execution>
                   <id>perfcake-scenario-run</id>
                   <goals>
                      <goal>scenario-run</goal>
                   </goals>
                </execution>
             </executions>
             <dependencies>
                <dependency>
                   <groupId>org.perfcake</groupId>
                   <artifactId>perfcake</artifactId>
                   <version>${perfcake.version}</version>
                   <scope>test</scope>
                </dependency>
             </dependencies>
          </plugin>
       ...
       </plugins>
    </build>
```

2.3.3. Logging

By default, PerfCake comes configured for a production environment. Logging level is set to *info* and validation messages are logged in a separate file. You can change the default logging level for PerfCake by a command line parameter `-log` (see Section 2.3.1, “Command line parameters” for more details). Possible logging levels are: *all*, *trace*, *debug*, *info*, *warn*, *error*, *fatal*, *off*.

More tweaking can be done in the `log4j2.xml` file. It is out of the scope of this guide to describe this configuration as this is a standard Log4j2 configuration file and one can find its own documentation.

2.3.4. Results Replay

It is possible to record raw results data from a performance test run and later replay them. This can be useful when we use a remote machine to run the test and we need to process the data on a different location or when we want to fine-tune the reporting facilities and we cannot or don't want to rerun the real test many times.

To record the test data, you can use the section called “RawReporter”. This will create a single output file with the recorded data. The file can be of a significant size depending on the number of test iterations. From our experience, a test with 100,000 iterations results in afile of size approx 4-8MB (depending on test meta-data available, sequences used etc.).

The file can be later replayed by providing the corresponding scenario with a different configuration of reporting section and passing the recorded data file to PerfCake in the `--replay` parameter.

Example 2.7. Sample usage of the results replay mode

```
> $PERFCAKE_HOME/bin/perfcake.sh -s http-reporting -r perfcake-
measurement-1465935794498.raw
2016-06-14 22:37:20,359 INFO  {org.perfcake.ScenarioExecution} ===
 Welcome to PerfCake 7.5 ===
2016-06-14 22:37:20,699 INFO  {org.perfcake.scenario.ScenarioBuilder}
 Scenario configuration: file:/home/perfcake/perfcake-7.5/resources/
scenarios/http-reporting.xml
2016-06-14 22:37:20,660 INFO  {org.perfcake.ScenarioExecution}
 Replaying raw results recorded in perfcake-
measurement-1465935794498.raw.
[0:00:00][1 iterations][3%] [319.848264 ms] [warmUp => false] [Threads
 => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize => 0.00
 B] [Average => 319.848264 ms]
[0:00:01][59 iterations][14%] [179.383135 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 207.6054971694915 ms]
[0:00:02][128 iterations][24%] [148.827863 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 179.01238675000008 ms]
[0:00:03][198 iterations][34%] [138.458243 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 167.52745448989901 ms]
[0:00:04][265 iterations][44%] [172.478225 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 161.4592350792453 ms]
[0:00:05][315 iterations][54%] [232.630756 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 167.65647934285724 ms]
[0:00:06][369 iterations][64%] [170.956762 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 170.2701377127373 ms]
[0:00:07][421 iterations][74%] [205.153517 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 173.20733971496443 ms]
[0:00:08][470 iterations][85%] [201.284832 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 176.6280006106384 ms]
[0:00:09][520 iterations][95%] [232.02807 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 179.5895076057694 ms]
[0:00:10][536 iterations][100%] [196.740767 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 0.00 B] [failures => 0] [RequestSize
 => 0.00 B] [Average => 181.79292323694048 ms]
2016-06-14 22:37:21,058 INFO  {org.perfcake.ScenarioExecution} ===
 Goodbye! ===
```

Please not the time difference between the start and the end of the replay (22:37:20,660 - 22:37:21,058). This means that all the results from a test that originally took 10 seconds was replayed in less than a second.

2.3.5. Debug Agent

The Debug Agent is an experimental feature suited for performance scenario development. It has a significant performance drawback and must not be used for real testing.

This agent publishes status of PerfCake components via JMX and can be read by utilities like `jvisualvm` (with MBean plugin).

The agent can be started by configuring the system property `perfcake.debug` to `true` or by passing the `-d` or `--debug` command line options. For the agent to work properly, the JDK's `tools.jar` must be on the classpath. This necessity will be removed in Java 9. The PerfCake startup scripts try to configure the classpath accordingly but they still might fail. As a workaround, you might copy the `tools.jar` file to `${PERFCAKE_HOME}/lib/ext`.

When there are multiple PerfCake instances running in the same JVM, they can be differentiated by setting `perfcake.debug.name` property (or `-dn`, `--debug-name` command line options). The default agent name is `perfcake-1`.

The main purpose of the Debug Agent is for IDE plugin integration and debugging performance scenarios. The feature state will be claimed stable once there is enough testing with the IDE plugins and probably when we can remove the `tools.jar` dependency.

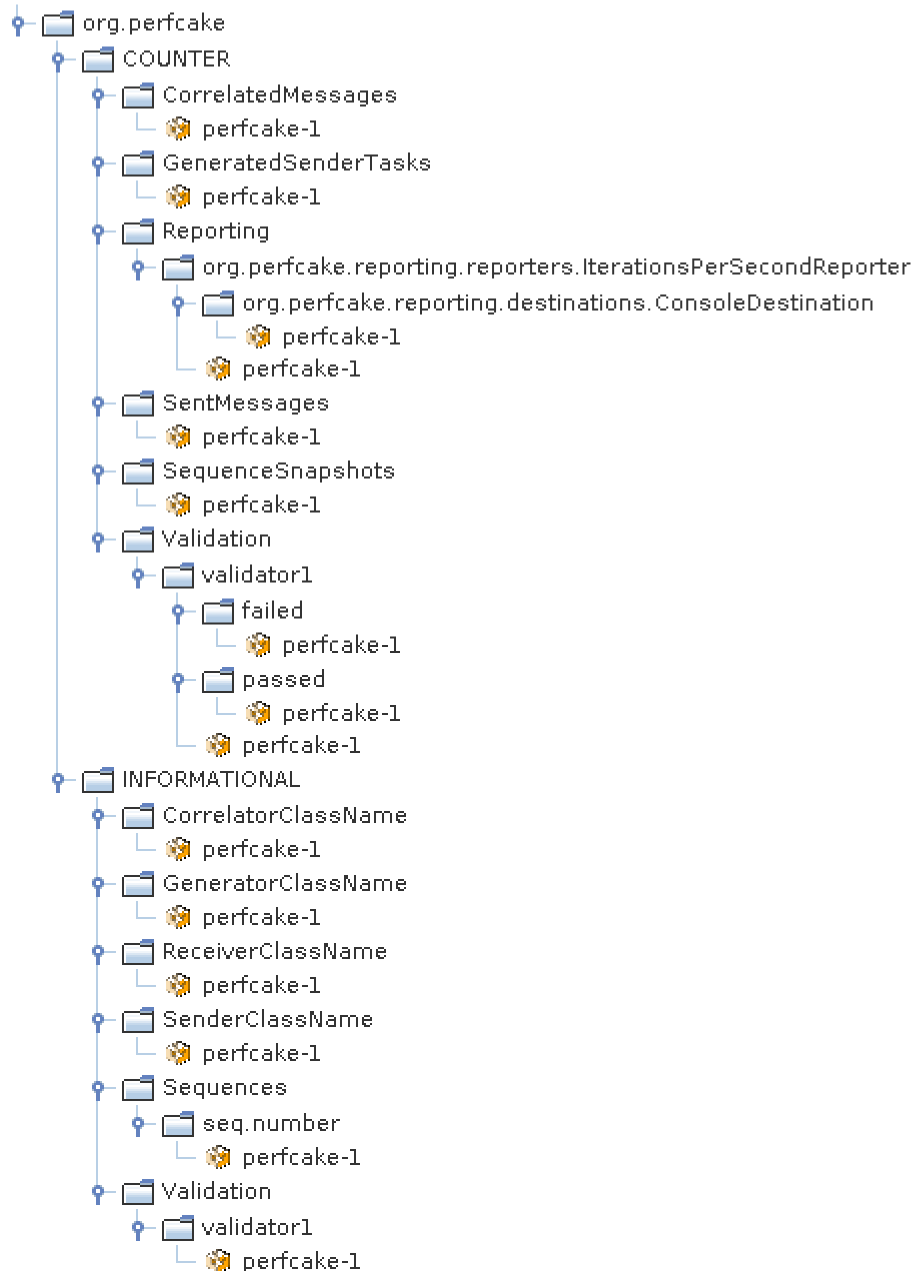

Figure 2.2. JMX tree with PerfCake debug information in `jvisualvm`

2.4. What to take care of during execution

By default PerfCake performs and initial benchmark to figure out the local system timer resolution. This is important to know because some systems (especially virtual machines) might have a very low resolution. If the system being measured by PerfCake is faster than the shortest period between system timer updates, PerfCake is not capable of reporting correct results. The information about the resolution can be found in the log and in the output.

```
2016-05-27 12:53:32,888 INFO  {org.perfcake.util.TimerBenchmark}
 Benchmarking system timer resolution...
```

```
2016-05-27 12:53:32,889 INFO  {org.perfcake.util.TimerBenchmark} This
 system is able to differentiate up to 279ns. A single thread is now
 able to measure maximum of 3584229 iterations/second.
```

There might be some typos or misconfigurations in the scenario definition. For a serious syntax issues with both XML and DSL, appropriate exceptions are thrown. In case of a wrong property name configured for any of the components, a warning similar to the following one is displayed during the initialization.

```
2016-06-01 22:02:10,776 WARN  [main] {org.perfcake.util.ObjectFactory}
 It was not possible to reliably configure property fooBar on class
 org.perfcake.reporting.reporter.ResponseTimeHistogramReporter. You
 may have a mistake in the scenario, or the class does not allow
 reading of the property.
```

PerfCake tries to overcome as many issues as it can. For a 100% validity of your performance test results, always make sure there are no warning or errors in the log. Such issues can be a problem with message template (the template was used without being parsed), a reporter with smaller reporting period than 500ms (the reporter is skipped) etc. More critical issues are reported with a non-zero exit code. See section Section 2.5, “Exit codes”. It is also possible to configure PerfCake to fail immediately when a Message Sender experiences an error (see `perfcake.fail.fast` property in Section 2.3, “Running PerfCake”).

Also pay attention to PerfCake output at the end of the execution. `DefaultMessageGenerator` waits for termination of all threads that were used to send the messages. This is configured by the `shutdownPeriod` property in the scenario. The default setting is to wait for 1000ms. If the threads do not manage to terminate their work, you can see warnings in the log like in the example below.

```
2016-05-27 12:54:06,027 WARN
 {org.perfcake.message.generator.DefaultMessageGenerator} Cannot
 terminate all sender tasks. Set higher shutdownPeriod for the
 generator in your scenario. Remaining tasks/threads active: 8/8
2016-05-27 12:54:06,027 WARN  {org.perfcake.ScenarioExecution} There
 are some blocked threads that were not possible to terminate. The
 test results might be flawed. This is usually caused by deadlocks or
 race conditions in the application under test.
```

2.5. Exit codes

When PerfCake terminates normally, the exit code returned to the operating system is 0 as usual. In some cases and to ease automation, there might be a non zero return code depending on the situation. In the following table you can find the codes and their root causes.[2]

Exit Code	Root Cause
1	No scenario was specified on the command line or via the system property.
2	Wrong parameters on the command line.
3	It was not possible to find, parse or load the scenario.
4	There was a general error during execution of the scenario. This can be caused by incorrect or incompatible configuration parameters.
5	Some responses to messages did not pass validation.

[2]In Linux systems, you can obtain the exit code from the `$?` environment property.

Exit Code	Root Cause
6	There were blocked threads after the test execution that was not possible to terminate cleanly. This can be caused by hanging connections or deadlocks in the application under test.
7	There was an error while replaying a provided test record.
8	The user requested help/usage information with a command line option and it was printed.

Table 2.5. PerfCake Exit Codes

2.6. Migrating scenarios to latest version

This chapter describes all you need to do in order to migrate PerfCake to a newer version.

2.6.1. From v1.0 to v2.x

The following list shows the steps that are needed to migrate from PerfCake version 1.0 to 2.x:

- Rename scenario XML schema namespace
- Rename message number constant name

The following sections describe each step in detail.

Migration steps

Rename XML schema namespace

The namespace of the XML schema for scenarios has been renamed from `urn:perfcake:scenario:1.0` to `urn:perfcake:scenario:2.0` so it must be updated in the scenario XML files.

Rename Message Number Constant Name

If your sender depends on message ordinal number you need to change the name of the particular constant from `PerfCakeConst.MESSAGE_NUMBER_PROPERTY` to `PerfCakeConst.MESSAGE_NUMBER_HEADER`

2.6.2. From v2.x to v3.x

There are several steps that are needed to migrate scenarios from PerfCake version 2.x to 3.x. The following list shows them:

- Rename scenario XML schema namespace
- Rename class names
- Replace reporters
- Update RequestResponseJmsSender

The following sections take a look at each step in detail.

Migration steps

Rename XML schema namespace

The namespace of the XML schema for scenarios has been renamed from `urn:perfcake:scenario:2.0` to `urn:perfcake:scenario:3.0` so it must be updated in the scenario XML files.

Rename class names

Following classes has been renamed to follow the naming conventions (See Developers' Guide).

Class name in version 2.x	Class name in version 3.x
HTTPSender	HttpSender
HTTPSSender	HttpsSender
JDBCSender	JdbcSender
JMSSender	JmsSender
RequestResponseJMSSender	RequestResponseJmsSender
SOAPSocketSender	SoapSocketSender
SSLSocketSender	SslSocketSender
CSVDestination	CsvDestination
TextMessageValidator	RegExpValidator
RulesMessageValidator	RulesValidator

Table 2.6. Renamed classes in PerfCake v3.x

Replace reporters

There are several reporters that has been removed from PerfCake and has been replaced by others. The following sections illustrate that change and how to perform the migration at the same time.

AverageThroughputReporter -> ThroughputStatsReporter

```
1   <reporter class="AverageThroughputReporter">
2      ...
3      (destinations)
4      ...
5   </reporter>
```

is replaced by

```
1   <reporter class="ThroughputStatsReporter">
2      <property name="minimumEnabled" value="false"/>
3      <property name="maximumEnabled" value="false"/>
4         ...
5         (destinations)
6         ...
7   </reporter>
```

ResponseTimeReporter -> ResponseTimeStatsReporter

```
1   <reporter class="ResponseTimeReporter">
2      ...
3      (destinations)
4      ...
5   </reporter>
```

is replaced by

```
1   <reporter class="ResponseTimeStatsReporter">
2      <property name="minimumEnabled" value="false"/>
3      <property name="maximumEnabled" value="false"/>
4         ...
5         (destinations)
6         ...
7   </reporter>
```

WindowResponseTimeReporter -> ResponseTimeStatsReporter

```
1   <reporter class="WindowResponseTimeReporter">
2      <property name="windowSize" value="10"/>
3         ...
4         (destinations)
5         ...
6   </reporter>
```

is replaced by

```
1   <reporter class="ResponseTimeStatsReporter">
2      <property name="windowSize" value="10"/>
3      <property name="minimumEnabled" value="false"/>
4      <property name="maximumEnabled" value="false"/>
5         ...
6         (destinations)
7         ...
8   </reporter>
```

Update RequestResponseJmsSender

The `RequestResponseJmsSender` newly introduces the possibility to specify different connection properties for request and response. But it changes the default behavior. To ensure the original behavior is kept, the security properties for the response has to be added to keep request and response credentials the same.

RequestResponseJMSSender -> RequestResponseJmsSender

```
1   <sender class="RequestResponseJMSSender">
2      ...
3      <property name="connectionFactory"
4                value="${jms.connection.factory}"/>
5
6      <property name="username" value="${username}"/>
7      <property name="password" value="${password}"/>
8      <!-- JNDI -->
```

```
 9          <property name="jndiContextFactory"
10                    value="${jndi.ctx.factory}"/>
11          <property name="jndiUrl" value="${jndi.url}"/>
12          <property name="jndiSecurityPrincipal"
13                    value="${jndi.username}"/>
14          <property name="jndiSecurityCredentials"
15                    value="${jndi.password}"/>
16          ...
17       </sender>
```

is replaced by

```
 1       <sender class="RequestResponseJmsSender">
 2          ...
 3          <property name="connectionFactory"
 4                    value="${jms.connection.factory}"/>
 5          <property name="username" value="${username}"/>
 6          <property name="password" value="${password}"/>
 7          <property name="responseUsername" value="${username}"/>
 8          <property name="responsePassword" value="${password}"/>
 9          <!-- JNDI -->
10          <property name="jndiContextFactory"
11                    value="${jndi.ctx.factory}"/>
12          <property name="jndiUrl" value="${jndi.url}"/>
13          <property name="jndiSecurityPrincipal"
14                    value="${jndi.username}"/>
15          <property name="jndiSecurityCredentials"
16                    value="${jndi.password}"/>
17          <property name="responseJndiSecurityPrincipal"
18                    value="${jndi.username}"/>
19          <property name="responseJndiSecurityCredentials"
20                    value="${jndi.password}"/>
21          ...
22       </sender>
```

Scenario conversion using XSLT

After understanding all the steps needed to migrate PerfCake scenarios from version 2.x to version 3.x we can take a look at an option to make the migration automatically using XSL transformation [3] . The particular XSLT is available as a part of the PerfCake distribution and it can be found at `$PERFCAKE_HOME/resources/xslt/scenario-2.0-to-3.0.xsl`.

There is a way to perform tha automated transformation using PerfCake sources. Please refer to the section *"Transformation of Scenarios"* in the Developers' Guide.

2.6.3. From v3.x to v4.x

The following list shows the steps that are needed to migrate from PerfCake version 3.x to 4.x:

- Rename scenario XML schema namespace

The following sections describe each step in detail.

[3] http://www.w3.org/TR/xslt

Migration steps

Rename XML schema namespace

The namespace of the XML schema for scenarios has been renamed from `urn:perfcake:scenario:3.0` to `urn:perfcake:scenario:4.0` so it must be updated in the scenario XML files.

2.6.4. From v4.x to v5.x

The following list shows the steps that are needed to migrate from PerfCake version 4.x to 5.x:

- Rename scenario XML schema namespace
- Move `run` element out of generator
- Change `target` property of senders to a dedicated `<target>` element.

The following sections describe each step in detail.

Migration steps

Rename XML schema namespace

The namespace of the XML schema for scenarios has been renamed from `urn:perfcake:scenario:4.0` to `urn:perfcake:scenario:5.0` so it must be updated in the scenario XML files.

Move **run** element out of generator

The `run` element of `generator` has been moved out of `generator` to a dedicatd element.

```
1     <generator class="...">
2        <run type="..." value="..."/>
3        ...
4        (properties)
5        ...
6     </reporter>
```

is replaced by

```
1     <run type="..." value="..."/>
2        <generator class="...">
3        ...
4        (properties)
5        ...
6     </reporter>
```

Change **target** property of senders to a dedicated **<target>** element.

The `target` property of senders was ascended from ordinary property to a dedicated element.

```
1     <sender class="...">
2        <property name="target" value="..."/>
3        ...
4        (properties)
```

```
5        ...
6     </sender>
```

is replaced by

```
1     <sender class="...">
2        <target>...</target>
3        ...
4        (properties)
5        ...
6     </sender>
```

Scenario conversion using XSLT

It is possible to automatically migrate scenarios from version 4 to version 5. For instructions on running the migration using PerfCake sources, please refer to *"Transformation of Scenarios"* section in the Developers' Guide.

2.6.5. From v5.x to v6.x

The following list shows the steps that are needed to migrate from PerfCake version 5.x to 6.x:

- Rename scenario XML schema namespace

The following sections describe each step in detail.

Migration steps

Rename XML schema namespace

The namespace of the XML schema for scenarios has been renamed from `urn:perfcake:scenario:5.0` to `urn:perfcake:scenario:6.0` so it must be updated in the scenario XML files.

2.6.6. From v6.x to v7.x

The following list shows the steps that are needed to migrate from PerfCake version 6.x to 7.x:

- Rename scenario XML schema namespace.
- RegExp validator now needs to have the backslashed in the patter specification escaped as well.
- Due to an update in the templating engine, we now longer support arithmetic expressions in templates and the default values in property placeholders are replaced by colon (instead of the *pipe* "|" character).
- There is no sequence defined by default (although the default sequences were not documented).
- The implicit message header storing the *Message Number* was removed, instead the same value is now available in message attributes passed to every sender and available for usage in both message headers and payload. It is now correctly stored under the key `perfcake.iteration.number` (which is defined in the constant `PerfCakeConst.ITERATION_NUMBER_PROPERTY`). If there are more messages or any of the messages has multiplicity higher than one in the scenario definition, all of the messages will have the same iteration number for each iteration.
- We changed the configuration of sequences to be aligned with validators. This means that the sequence `name` attribute has been changed to sequence `id`.

- We unified and corrected component package names. This required changing `org.perfcake.reporting.reporters` and `org.perfcake.reporting.destinations` to singular `org.perfcake.reporting.reporter` and `org.perfcake.reporting.destination`.
- Sequences interface has updated the method to obtain the next value of the sequence.
- Support for JMS API 2.0 in `JmsSender` and `RequestResponseJmsSender`.

The following sections describe some steps in more detail.

Migration steps

Rename XML schema namespace

The namespace of the XML schema for scenarios has been renamed from `urn:perfcake:scenario:6.0` to `urn:perfcake:scenario:7.0` so it must be updated in the scenario XML files.

Update RegExp Validator patterns

In every patter specified in the scenario, add one more backslash to places where a backslash was previously used to escape RegExp character. For example, change `.*"I'm a [Ff]ish"!\.*` to `.*"I'm a [Ff]ish"!\\.*` because the second dot is meant to match a dot character and not any character as it does in a RegExp.

Please refer to the section called “RegExpValidator” for more details.

Update template placeholders

Replace `${property||default}` with `${property:default}`. Replace arithmetic expressions like `@{property + 1}` with custom Section 4.6, “Sequences”.

For more details on the advanced features of proeprties filtering read Section 2.2.5, “Filtering properties”.

Rename the name attribute in sequences

In all scenario definitions change the sequence name attribute to sequence id. For example, in XML scenarios change the following:

Example 2.8. Old sequence configuration in XML

```
<sequence name="..." class="...">
```

To read like this:

Example 2.9. New sequence configuration in XML

```
<sequence id="..." class="...">
```

Rename of packages under org.perfcake.reporting

This would affect you only when you used the canonical package name in scenario configurations, or when you developed custom components and put them in the same pack-

ages as PerfCake components are. Namely under `org.perfcake.reporting.reporters` and `org.perfcake.reporting.destinations`. All the plural package names should be now converted to singular (i.e. `org.perfcake.reporting.reporter` and `org.perfcake.reporting.destination`).

Update Sequence implementations

In your `Sequence` implementations change the following:

Example 2.10. Old Sequence implementation

```
public String getNext() {
   ...
   return nextValue;
}
```

To the new implementation:

Example 2.11. New Sequence implementation

```
public void publishNext(final String sequenceId, final Properties
 values) {
   ...
   values.setProperty(sequenceId, nextValue);
}
```

This update allow a single section to publish more values. All the value keys should start with the sequence specific prefix from the `sequenceId` parameter.

Support for JMS API 2.0

By default, JmsSender and RequestResponseJmsSender now use JMS API 2.0. If you want to continue using JMS API 1.1, change your scenarios to use `Jms11Sender`, or `RequestResponseJms11Sender` respectively.

Chapter 3. General Usage

The general usage of PerfCake is covered in the following books and online resources. They provide useful hints, describe complex performance test scenarios and give guidance in solving real world problems.

3.1. PerfCake: Performance Testing Scenarios

This a cookbook with recipes for creating complex performance testing scenarios to easily bootstrap your automated performance testing.

The book is currently in the phase of preparation and we do not have much content yet. It is published on ??? [https://leanpub.com/perfcake] and we will keep releasing updates regularly. The book is for free with the option to donate a small amount of money to support our further work.

3.2. Performance Engineering

This is another book that is currently a work in progress. It will teach you everything a succesful software engineer needs to know about performance testing from experts in the field.

The book will be be published on ??? [https://leanpub.com/perfengineering] and we will keep this documentation up-to-date with its current state. This time, the book won't be free but not very expensive either.

In this book we examine many possible aspects of performance testing and figure out the best approaches to handle challenges along the way. We always analyze the problems, think about possible solutions and figure out the best within the given constraints.

In the first part of the book, we go through the theory of performance testing and we define useful terms. Next, we will discuss what is important in performance testing, what to take care of, and how to best handle performance testing. Running real experiments is covered in the third part. In the last part we will discuss a design of an ideal performance testing tool and demonstrate it on a real implementation of an open-source performance testing framework PerfCake.

3.3. YouTube Channels

We are getting ready to create two YouTube channels with videos on performance testing. The first channel will cover performance testing in general, and the second channel will be more about creating real scenarios with PerfCake.

Chapter 4. Reference Guide

4.1. How - Generating load

Generator is an object which uses Sender objects to send and receive messages. Each generator represents a method or technique how the load (sending the messages) is handled. It is capable of generating the load for a specified duration which can be an amount of time or a number of iterations using a specified number of concurrent threads. To define the duration and its length you have to configure a `run` of the specified type in the scenario.

Following table shows the `run` options:

Run type	Value description
time	Time duration in milliseconds
iteration	Number of iterations

Table 4.1. Run options

The generating is performed in so called iterations. In a single iteration generator takes a sender from the sender pool, uses it to send all the messages specified in the scenario (See Section 4.5, “What - Messages”) and after that it returns the sender back to the sender pool. The reporting module (See Section 4.7, “Reporting”) treats this as a single iteration.

Example 4.1. An example of the `run` configuration in a scenario:

```
1    <run type="time" value="10000"/>
2    <generator class="..." threads="...">
3        ...
4        (properties)
5        ...
6    </generator>
```

In the example above a scenario is defined to run for 10 seconds.

When specifying the generator class, unless you enter a fully classified class name, the default package `org.perfcake.message.generator` is assumed.

The following sections describes the generators that can be used in PerfCake to generate the load.

4.1.1. DefaultMessageGenerator

The generator is able to generate a load using multiple threads for a long period of time (matter of days, weeks,...). The generator uses a thread pool of a fixed size that stores threads executing performance iterations. The thread pool uses an internal queue where tasks implementing the iterations are buffered and scheduled for execution. The size of the thread queue can be specified by the `senderTaskQueueSize` property. The generator regularly fills the queue with new tasks each period specified by the `monitoringPeriod` property.

This approach guarantees that the maximum number of thread instances (that is equal to the value of the `senderTaskQueueSize` property) exists in the memory during all the time the generator is working.

To configure the generator we need to specify a number of concurrent threads (using the `threads` attribute) and a duration (using the `run` discussed above).

During a shutdown, the thread queue is regularly checked every period specified by the `shutdownPeriod` for the threads finishing their work. If the same amount of threads keeps running for this period, they are forcefully stopped. If the `shutdownPeriod` is set to the value of `-1` it's value is auto-tuned at the end of the scenario execution to the value of 5 times the average response time.

The following table shows the properties of the DefaultMessageGenerator:

Property name	Description	Required	Default value
monitoringPeriod	A period in milliseconds by which the thread queue is filled with new tasks.	No	1000
shutdownPeriod	A period in milliseconds by which the thread queue is checked for the threads finishing their work during a shutdown.	No	1000
senderTaskQueueSize	The size of the task queue.	No	1000

Table 4.2. DefaultMessageGenerator properties

Example 4.2. An example of DefaultMessageGenerator configuration

```
1    <run type="time" value="60000"/>
2    <generator class="DefaultMessageGenerator" threads="100">
3       <property name="senderTaskQueueSize" value="5000"/>
4    </generator>
```

In the example above a `DefaultMessageGenerator` is defined to run for 60 seconds using 100 concurrent threads with the sender task queue with the size of 5000. The main goal of limiting the queue of sender tasks is to limit memory usage and improve performance of PerfCake.

4.1.2. ConstantSpeedMessageGenerator

This generator is based on `DefaultMessageGenerator` (seeSection 4.1.1, “DefaultMessageGenerator”) and inherits all its configuration properties. It tries to achieve given speed of messages per second. Uses the underlying buffer of sender tasks of`DefaultMessageGenerator`. This buffer smoothens the changes in the speed. If you need the generator to change its speed more aggressively, configure `senderTaskQueueSize` property.

The following table shows the properties of the ConstantSpeedMessageGenerator:

Property name	Description	Required	Default value
speed	Desired constant speed in messages per second. Setting the speed to -1 makes the generator run at maximum possible speed like	No	5000

Property name	Description	Required	Default value
	DefaultMessageGenerator.		

Table 4.3. ConstantSpeedMessageGenerator properties

Example 4.3. An example of ConstantSpeedMessageGenerator configuration

```
1    <run type="time" value="60000"/>
2    <generator class="ConstantSpeedMessageGenerator" threads="100">
3       <property name="speed" value="10000"/>
4    </generator>
```

In the example above a `ConstantSpeedMessageGenerator` is defined to run for 60 seconds using 100 concurrent threads with the desired constant speed of 10000 messages per second.

4.1.3. CustomProfileGenerator

This generator allows the users to specify custom profiles to configure message generating. A custom profile is simply a function that for the given time period of the performance test returns the required number of threads and speed (number of generated messages per second). By such a function, the process of generating messages can be freely configured as needed.

The generator is based on `DefaultMessageGenerator` (seeSection 4.1.1, “DefaultMessageGenerator”) and inherits all its properties.

Each generator configuration has to specify the `threads` attribute. In the case of `CustomProfileGenerator`, this specifies the maximum number of threads used during the whole performance test. The profile function must not request more threads than that. Otherwise some tasks for sending message won't have threads available and some requests will be skipped while producing an error message. Please be careful about this.

When the profile function returns higher number of threads or higher speed then for the previous time period, the change is applied immediatelly. However, when it returns lower number of threads, the existing threads first need to finish their tasks before they are removed from the pool. When a slower speed is requested, the generator first finishes the tasks that are already in the task queue. The length of this queue is set by the `senderTaskQueueSize` property. When the queue is too short, it runs out of tasks frequently and this has a negative impact on the performance results. When it is too long, it takes longer time for the change in speed to be reflected.

For example, when sending messages to a system that is capable to response in 10 milliseconds with 20 threads, PerfCake can process `1000 / 10 * 20` tasks per second. If the queue length is set to 2000 (`1000 / 10 * 20 = 2000`), it would take 1 second for the speed to change.

The profile function can have its limits and simply do not provide any more values after a certain boundary. For this purpose, the generator has an `autoReplay` property which instructs it to replay the profile function from the beginning.

The profile function must implement the `Profile` interface and its class name is passed to the generator. By default, the package `org.perfcake.message.generator.profile` is assumed. The most important method is `getProfile()` that returns the profile (number of threads and speed) for the given time period. It must always return the correct value, despite the order of requests to the method. The period passed to the method specifies the type of time information (either an iteration number or milliseconds since test start) and time. The profile function must also properly handle the `autoReplay` property.

For an easier development, there is an `AbstractProfile` class where the only responsibility of a developer is to read all the time points where there is a change in either thread count or speed in the profile function. All these points are registered with the `addRequestEntry()` method. The class then handles all the correct behaviour.

It is important to note that the last value in the profile function either remains until the end of the performance test (when `autoReplay` is set to false), or it is immediatelly overriden by the first value. This means that the values for the last entry should be the same as for the first one when `autoReplay` is turned on.

The first entry in the profile function should always start with 0 to set the conditions at the very beginning of the performance test. When there is no such entry starting with 0, the first entry is taken for the starting conditions. The order of registered entries does not matter and they are automatically sorted by their time property.

Whenever the speed is set to -1, the generator runs at maximal possible speed as `DefaultMessageGenerator`.

The following table shows the properties of the `CustomProfileGenerator`:

Property name	Description	Required	Default value
autoReplay	Instructs the generator to replay the profile function from the beginning when we hit its end.	No	true
profileClass	The class name of the custom profile to use.	Yes	-
profileSource	The source where the profile is specified. This is passed directly to the profile. Some profiles might not use this property.	No	-
senderTaskQueueSize	Determines the length of the queue for sender tasks. Influences how quickly the generator can react to slow downs in speed.	No	1000

Table 4.4. CustomProfileGenerator properties

Example 4.4. An example of CustomProfileGenerator configuration

```
1   <generator class="CustomProfileGenerator" threads="100">
2      <property name="profileClass" value="CsvProfile"/>
3      <property name="profileSource" value="test-profile.csv"/>
4   </generator>
```

The following sections describe the provided profile functions.

CsvProfile

This profile reads the points of profile function from a CSV file. The format of a single line in the file is `<time>;<threads>;<speed>`.

The following example shows a ramp-up function that keeps the maximum values between the last steps and then starts over (supposing `autoReplay` is set to `true`).

Example 4.5. Sample profile specified in a CSV file

```
0;10;100
250;20;100
500;30;100
750;30;200
1000;30;300
9999;10;100
```

4.1.4. RampUpDownGenerator

The generator is based on the `DefaultMessageGenerator` (SeeSection 4.1.1, “DefaultMessageGenerator”) and inherits all its properties. In addition to this functionality, `RampUpDownGenerator` is able to change the number of threads during the execution. The number of threads evolution during execution is illustrated in Figure 4.1, “RampUpDownGenerator time chart” .

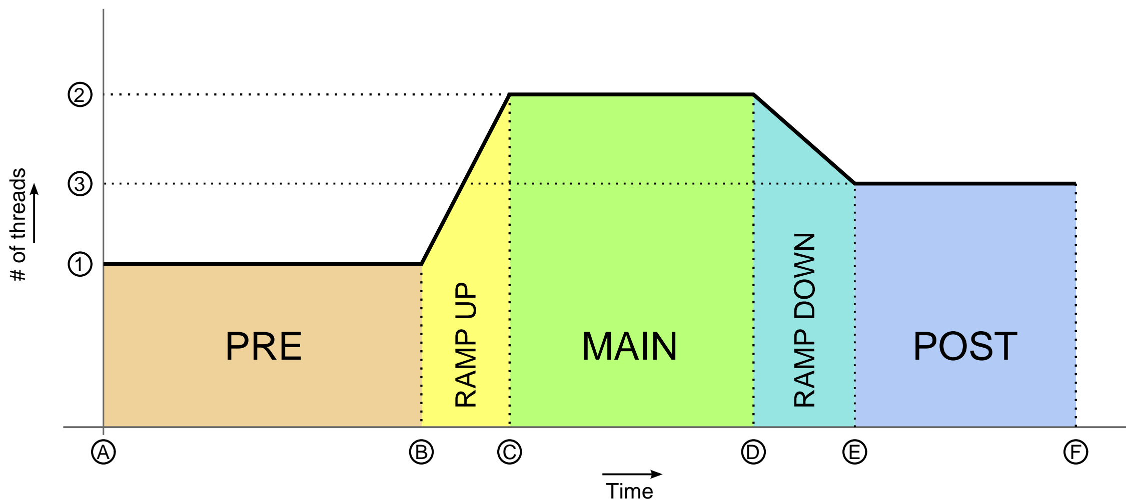

Figure 4.1. RampUpDownGenerator time chart

The scenario starts **(A)** with the number of threads set to the value of the `preThreadCount` property **(1)** . It continues to execute for the duration set by the `preDuration` property **(A)** -> **(B)** , which is called the *PRE* phase. When *PRE* phase ends **(B)** , the *RAMP UP* phase starts.

In the *RAMP UP* phase the number of the threads is changed by the value of the `rampUpStep` property each period set by the `rampUpStepPeriod` until it reaches the number of threads set by the value of the `mainThreadCount` property **(2)** .

In that moment **(C)** *MAIN* phase starts and the execution continues for the duration set by the `mainDuration` property **(C)** -> **(D)** , when the *RAMP DOWN* phase starts.

In the *RAMP DOWN* phase **(D)** -> **(E)** the number of threads is again changed but this time in the opposite direction than in the *RAMP UP* phase. It changes by the value of the `rampDownStep` property each period specified by the `rampDownStepPeriod` property until the final number of threads **(3)** is reached. By that moment **(E)** the final phase called *POST* starts.

The *POST* phase ends by the end of the scenario **(F)** .

The outer borders of the number of threads and the duration is set by the maximum number of threads specified by the `threads` attribute of the generator and by the maximum duration set by the `run` element.

The following table describes all the properties of the RampUpDownGenerator:

Property name	Description	Required	Default value
senderTaskQueueSize	The size of the task queue.	No	1000
preThreadCount	Initial number of threads.	No	Generator's `threads` value.
preDuration	A duration period in the units of `run` type of "PRE" phase.	No	`Long.MAX_VALUE`
rampUpStep	A number by which the number of threads is changed in the "RAMP UP" phase.	No	0
rampUpStepPeriod	A period in the units of `run` type after which the number of threads is changed by `rampUpStep` value.	No	`Long.MAX_VALUE`
mainThreadCount	A number of threads in the main phase.	No	Generator's `threads` value.
mainDuration	A duration in the units of `run` type for which the main phase lasts.	No	0
rampDownStep	A number by which the number of threads is changed in the "RAMP DOWN" phase.	No	0
rampDownStepPeriod	A period in the units of `run` type after which the number of threads is changed by `rampDownStep` value.	No	`Long.MAX_VALUE`
postThreadCount	Final number of threads.	No	Generator's `threads` value.

Table 4.5. RampUpDownGenerator properties

Example 4.6. An example of RampUpDownGenerator configuration

```
1     <run type="time" value="60000"/>
2     <generator class="RampUpDownGenerator" threads="20">
3        <property name="preThreadCount" value="10"/>
4        <property name="preDuration" value="10000"/>
5        <property name="rampUpStep" value="2"/>
6        <property name="rampUpStepPeriod" value="1000"/>
7        <property name="mainThreadCount" value="20"/>
8        <property name="mainDuration" value="20000"/>
9        <property name="rampDownStep" value="1"/>
10       <property name="rampDownStepPeriod" value="1000"/>
```

```
11        <property name="postThreadCount" value="15"/>
12     </generator>
```

In the example above the scenario starts with 10 threads (preThreadCount). After 10s (preDuration) the number of threads starts to increase by number of 2 (rampUpStep) each second (rampUpStepPeriod) until the number of threads reaches 20 (mainThreadCount). Than after another 20 (mainDuration) seconds the number of threads starts to decrease by 1 (rampDownStep) each second (rampDownStepPeriod) until the number reaches 15 (postThreadCount) which remains until the end of the scenario. The whole scenario ends after 60s (run).

4.2. Where - Sending messages

As it was mentioned before, a sender is an object responsible for sending a message to the tested system and receiving the response in a specific way via a specific transport.

To tell the sender where to send the messages, there is a <target> element mandatory for all senders.

All message senders provided in PerfCake distribution are based on the AbstractSender class which cannot be used standalone but provides an important property available in all other senders. The property is keepConnection and instructs the sender to stay initialized for the whole run of the performance test. When set to false, the sender get initialized for each message. This influences the performance results and needs to be considered whether this is a required behavior. The default value of this property is true and it is not required in the scenario definition.

Example 4.7. An example of a Sender configuration

```
1     <sender class="...">
2        <target>...</target>
3        ...
4        sender properties
5        ...
6     </sender>
```

When specifying the sender class, unless you enter a fully classified class name, the default package org.perfcake.message.sender is assumed.

In the following sections you can find a complete description of all senders, that can be used by PerfCake including all of their properties.

4.2.1. CamelSender

The CamelSender can be used to send messages to any Apache Camel [http://camel.apache.org] endpoint. The endpoint is defined in the target attribute in the configuration. Any Camel endpoints are allowed. Just make sure the corresponding Camel component JAR file and its dependencies are on the classpath. In the binary distribution, this means putting the JAR files in the lib/ext directory.

Endpoints can be usually configured by using message headers. To set the headers use the header element of the particular message element in the messages section of the scenario definition.

There are no extra configuration properties for CamelSender.

Example 4.8. An example of CamelSender configuration

```
1     <sender class="CamelSender">
2        <target>http:127.0.0.1:8283/perfcake?param1=value1</target>
```

```
3     </sender>
4     ...
5     <messages>
6        <message content="Hello from Camel">
7           <header name="CamelHttpMethod" value="POST" />
8        </message>
9     </messages>
```

In this example, the `CamelSender` is configured to use the HTTP Camel component to send messages to an HTTP endpoint. The messages will be sent using the POST method as configured in the message header.

4.2.2. CoapSender

The sender can be used to send a single message using CoAP protocol [1] .

The target of the sender is a CoAP endpoint in a form of CoAP URI [2] .

Property name	Description	Required	Default value
method	A name of CoAP method. One of `GET`, `POST`, `PUT` or `DELETE` is supported [a] .	No	POST
requestType	CoAP request type that the sender will use. Either `confirmable` or `nonConfirmable` is supported [b] .	No	nonConfirmable

[a] See https://tools.ietf.org/html/rfc7252#section-5.8 for more detail on request methods of the CoAP protocol.

[b] See https://tools.ietf.org/html/rfc7252#section-2.1 for more detail on (non)confirmable requests of the CoAP protocol.

Table 4.6. CoapSender properties

Example 4.9. An example of CoapSender configuration

```
1     <sender class="CoapSender">
2        <target>coap://${server.host}:5683/resource</target>
3        <property name="method" value="GET"/>
4        <property name="requestType" value="confirmable"/>
5     </sender>
```

4.2.3. CommandSender

The sender is able to invoke an external command (specified by the `target` property) in a separate process to send a message payload. The payload can be passed to the standard input of the process (default behavior) or as the command argument. That is configurable via `messageFrom` proerty.

The target of the sender is a path to the file containing the commands that are to be executed.

Message properties and headers are passed to the process as the environmental variables.

Property name	Description	Required	Default value
messageFrom	Specifies where the message is taken from by the sender.	No	stdin

[1] Costrained Application Protocol http://coap.technology/

[2] See https://tools.ietf.org/html/rfc7252#section-6 for more detail on CoAP endpoint URI

Property name	Description	Required	Default value
	The supported values are `stdin` or `arguments`		

Table 4.7. CommandSender properties

Example 4.10. An example of CommandSender configuration

```
1   <sender class="CommandSender">
2      <target>/tmp/script.sh</target>
3   </sender>
4   ...
5   <messages>
6      <message>
7         <property name="GREETINGS" value="Be well!"/>
8      </message>
9   </messages>
```

In the example above there is a `CommandSender` configured to execute a script located in `/tmp/script.sh` file. The environmental variable `GREETINGS` is set to the value of "`Be well!`".

4.2.4. DummySender

This sender is intended to work as a dummy sender and to be used for testing scenarios and developing purposes. It does not actually send any message. It can simulate a synchronous waiting for a reply by setting the `delay` property.

While the target of the sender is mandatory for all senders, it is irrelevant for the `DummySender` so it can be anything.

Property name	Description	Required	Default value
delay	Time duration in milliseconds that the sender will simulate waiting for a response. If set to 0 (default) it will not wait at all.	No	0

Table 4.8. DummySender properties

Example 4.11. An example of DummySender configuration

```
1   <sender class="DummySender">
2      <target>out there!</target>
3      <property name="delay" value="500"/>
4   </sender>
```

4.2.5. GroovySender

Groovy sender can be used to implement the message sending in a Groovy script. By default the sender uses the `groovy` installed on the system, where the PerfCake is executed. PerfCake would try to find it at the following path: `$GROOVY_HOME/bin/groovy` on UNIX systems or `%GROOVY_HOME%\bin\groovy` in case of Windows, where `GROOVY_HOME` is an environment variable.

The Groovy binary doesn't have to be the one, that is installed on the system. There is a possibility to use `groovyExecutable` property to specify, what Groovy binaries would be used to execute the target script.

The message configured in the scenario is passed to the Groovy script through standard input, so it can be accessed via `System.in` . The response is supposed to be passed back through a standard output via `System.out` .

The target of the sender is a path or URL to the Groovy script that is to be executed.

The following table describes all the GroovySender's properties.

Property name	Description	Required	Default value
groovyExecutable	Path to the Groovy binary, that would be used to execute the script.	No	$GROOVY_HOME/bin/groovy
classpath	Classpath for groovy executable - specifies where to find the class or groovy files.	No	-

Table 4.9. GroovySender properties

Example 4.12. An example of GroovySender configuration

```
1     <sender class="GroovySender">
2        <target>/tmp/script.groovy"</target>
3        <property name="classpath" value="~/common-classes"/>
4     </sender>
```

4.2.6. HttpSender

The `HttpSender` can be used to send messages via HTTP protocol using POST method as default to the URL specified by the 'target' property. The HTTP method can be changed using the `method` property.

Various scenarios can lead to different HTTP response codes that are expected. The scenario can be set to expect one or more response codes. It can be set via the `expectedResponseCodes` property. Default expected response code is 200.

To set headers of the HTTP request use the `header` element of the particular `message` element in the `messages` section of the scenario definition.

The `Content-Type` header is set automatically by the sender for all messages to the value of `text/plain; charset=utf-8` by default. Also `Content-Length` header is generated and set automatically to the value of message payload size if the payload is not empty.

The target of the sender is an URL, where the message is send.

Following table shows the properties of the HttpSender:

Property name	Description	Required	Default value
expectedResponseCodes	A comma separated list of HTTP response code(s) that is expected to be returned.	No	200
method	An HTTP method to be used. [a]	No	POST

Property name	Description	Required	Default value
storeCookies	Should the individual threads store cookies between requests? When true, the cookies are stored.	No	false

[a] See http://docs.oracle.com/javase/7/docs/api/java/net/HttpURLConnection.html#setRequestMethod(java.lang.String) for a complete list of available methods.

Table 4.10. HttpSender properties

Example 4.13. An example of HttpSender configuration

```
1    <sender class="HttpSender">
2       <target>http://domain.com/cool-url</target>
3       <property name="method" value="GET"/>
4       <property name="expectedResponseCodes" value="200,202"/>
5    </sender>
```

4.2.7. HttpsSender

HttpsSender is similar to the HttpSender (see Section 4.2.6, “HttpSender”), thus it inherits the same properties. The difference is that the HttpsSender uses HTTPS instead of plain HTTP protocol to send messages.

The HttpsSender's properties are described in the following table:

Property name	Description	Required	Default value
keyStore	Path to the key store	No	-
keyStorePassword	Key store password	No	-
trustStore	Path to the trust store	No	-
trustStorePassword	Trust store password	No	-

Table 4.11. HttpsSender additional properties

Example 4.14. An example of HttpsSender configuration

```
1    <sender class="HttpsSender">
2       <target>https://domain.com/secured_url</target>
3       <property name="method" value="POST"/>
4       <property name="trustStore" value="${my.keystores}/
cacert.jks"/>
5       <property name="trustStorePassword" value="ts_passsword"/>
6    </sender>
```

4.2.8. ChannelDatagramSender

Sends messages through NIO [3] `DatagramChannel` [4] .

The sender's target is an URL of a datagram socket in a form of `<host>:<port>`.

[3] http://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html#package.description
[4] http://docs.oracle.com/javase/8/docs/api/java/nio/channels/DatagramChannel.html

Property name	Description	Required	Default value
awaitResponse	Determines whether we should wait for the response from the channel.	No	`false`
maxResponseSize	Expected maximum response size. Defaults to -1 which means to instantiate the buffer of the same size as the request messages.	No	-1

Table 4.12. ChannelDatagramSender properties

4.2.9. ChannelFileSender

Sends messages through NIO [3] `FileChannel` [5] .

The sender's target is a path to the file to which message is sent (written) or received (read).

Property name	Description	Required	Default value
awaitResponse	Determines whether we should wait for the response from the channel.	No	`false`
maxResponseSize	Expected maximum response size. Defaults to -1 which means to instantiate the buffer of the same size as the request messages.	No	-1

Table 4.13. ChannelFileSender properties

4.2.10. ChannelSocketSender

Sends messages through NIO [3]> `SocketChannel` [6] .

The sender's target is an URL of a socket in a form of`<host>:<port>`.

Property name	Description	Required	Default value
awaitResponse	Determines whether we should wait for the response from the channel.	No	`false`
maxResponseSize	Expected maximum response size. Defaults to -1 which means to instantiate the buffer of the same size as the request messages.	No	-1

Table 4.14. ChannelSocketSender properties

[5] http://docs.oracle.com/javase/8/docs/api/java/nio/channels/FileChannel.html
[6] http://docs.oracle.com/javase/8/docs/api/java/nio/channels/SocketChannel.html

4.2.11. JdbcSender

As the name of the sender clues JdbcSender is meant to be used to send JDBC queries. It can handle any query the JDBC is capable of.

The target of the sender is a JDBC URL of the target, where the query is send.

Property name	Description	Required	Default value
driverClass	A fully qualified JDBC Driver class.	Yes	-
username	A database user	no	""
password	A database password	no	""

Table 4.15. JdbcSender properties

Example 4.15. An example of JdbcSender configuration

```
1     <sender class="JdbcSender">
2        <target>jdbc:postgresql://localhost:5432/db</target>
3        <property name="username" value="me-the-first"/>
4        <property name="password" value="guess_me"/>
5        <property name="driverClass" value="org.postgresql.Driver"/>
6     </sender>
```

4.2.12. Jms[11]Sender

There are two versions of JMS (Java Message Service) sender. An older one supporting JMS API 1.1 - `Jms11Sender` (introduced in PerfCake 7.0, previously known as `JmsSender`). And a new one supporting JMS API 2.0 - `JmsSender` (new in PerfCake 7.0). Except for the API version they use, there is no difference in their functionality and properties. They should also have the same performance, depending on the client driver used.

The Jms[11]Sender can be used to send a single JMS message.

The target of the sender is a JNDI name of the JMS destination where the JMS message is send. Both JNDI and messaging can be secured independently.

For the sender to work properly, you need to have a JMS client adaptor on the class path. In PerfCake, this means placing the messaging client driver to the `lib/ext` directory.

The following table describes the JmsSender's properties:

Property name	Description	Required	Default value
connectionFactory	A name of a JMS Connection factory.	Yes	-
jndiContextFactory	A fully qualified name of the JNDI ContextFactory class.	Yes	-
jndiUrl	A JNDI location URL.	Yes	-

Property name	Description	Required	Default value
jndiSecurityPrincipal	A JNDI username	No	
jndiSecurityCredentials	A JNDI password	No	
autoAck	Indicates whether the received message will be auto-acknowledged.	No	true
replyTo	The destination where the reply is supposed to be sent by server. JMS 'replyTo' header.	No	""
persistent	Indicate whether the message is to be persisted by JMS provider.	No	true
transacted	Indicate whether the message transport is to be transacted.	No	false
messageType	Indicate the type of the message. The supported values are `object` , `string` and `bytearray` .	No	string
username	A JMS security username. If not provided - JMS transport will be performed unsecured.	No	""
password	A JMS security password. If not provided - JMS transport will be performed unsecured.	No	""
safePropertyNames	Use safe message property names because some messaging implementations (e.g. Apache Artemis) do not allow anything but valid Java identifiers. When set to true (which is the default), anything else than letters, numbers and underscores is changed to underscores.	No	true

Table 4.16. JmsSender properties

Example 4.16. An example of JmsSender configuration

```
1    <sender class="JmsSender">
2       <target>jms/queue/YourQueue</target>
3       <property name="connectionFactory" value="jms/ConnFactory"/>
4       <property name="username" value="KingRoland"/>
5       <property name="password" value="12345"/>
6       <!-- JNDI properties-->
```

```
 7
  <property name="jndiContextFactory" value="pkg.InitCtxFactory"/>
 8        <property name="jndiUrl" value="remote://
${server.host}:4447"/>
 9        <property name="jndiSecurityPrincipal" value="KingRoland"/>
10        <property name="jndiSecurityCredentials" value="12345"/>
11     </sender>
```

4.2.13. LdapSender

The sender is able to query an LDAP server [7] .

The target of the sender is an URL of the LDAP server in the form of `ldap://<server-host>:<server-port>`.

Following table shows the properties of the HttpSender:

Property name	Description	Required	Default value
searchBase	DN (distinguished name) to use as the search base	Yes	-
filter	The search filter	Yes	-
ldapUsername	LDAP server user	No	-
ldapPassword	LDAP server password	No	-

Table 4.17. HttpSender properties

Example 4.17. An example of LdapSender configuration

```
1     <sender class="LdapSender">
2        <target>ldap://localhost:389</target>
3        <property name="searchBase" value="dc=example,dc=org"/>
4        <property name="filter" value="(objectclass=*)"/>
5     </sender>
```

4.2.14. MqttSender

The sender is capable to send MQTT [8] messages to a remote broker and receiving responses if needed. The response is expected only if `responseTarget` property is set.

The target of the sender is an URL of the remote broker with the topic name in a form of `<protocol>://<host>:<port>/<topic name>` , where the message is send.

Following table shows the properties of the MqttSender:

Property name	Description	Required	Default value
qos	Quality of Service level [a] - guarantee for the	No	`EXACTLY_ONCE`

[7] http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol
[8] http://mqtt.org/

Property name	Description	Required	Default value
	message delivery from the sender to the remote broker. Supported values are EXACTLY_ONCE, AT_LEAST_ONCE and AT_MOST_ONCE.		
userName	MQTT broker user name	No	-
password	MQTT broker password	No	-
responseTarget	An URL of the remote broker and a topic name, from which the response is received. If ordinary string is presented (not in a form of an URL), the string as a whole is considered to be a response topic name in the same broker where the original message was sent.	No	-
responseQos	Quality of service level for receiving a response. (see qos property description for more details).	No	The value of qos property.
responseUserName	MQTT broker user name for response	No	-
responsePassword	MQTT broker password for response	No	-

[a] http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718099

Table 4.18. MqttSender properties

Example 4.18. An example of MqttSender configuration

```
1    <sender class="MqttSender">
2       <target>tcp://localhost:1883/request.topic</target>
3       <property name="responseTarget" value="response.topic"/>
4    </sender>
```

I the example above there is a MqttSender configured to send messages to topic with the name of request.topic in a broker listening on tcp://localhost:1883. The response is expected and received from the topic with the name of response.topic of the same broker. The quality of service level for both sending and receiving is default EXACTLY_ONCE.

4.2.15. OauthHttpSender

OauthHttpSender is similar to the HttpSender (see Section 4.2.6, “HttpSender”), thus it inherits the same properties. The difference is that the OauthHttpSender allows token authentication to an HTTP based service using OAuth2[9] protocol.

[9] See https://oauth.net/2/ for the protocol details.

The OauthHttpSender's properties are described in the following table, where the defaults works with Keycloak:[10]:

Property name	Description	Required	Default value
tokenServerUrl	URL of the server granting us a token. Default value works with Keycloak, supposing your realm name is `demo`.	No	http://127.0.0.1:8180/auth/realms/demo/protocol/openid-connect/token
tokenServerData	Data to send to the server to request a token. Default value works with Keycloak.	No	grant_type=password&client_id=jboss-javaee-webapp&username=marvec&password=abc123
responseParser	RegEx to create a capture group around the token value in the server's response.	No	.*"access_token":"([^"]*)".*
oauthHeader	Header where the token is passed to the target service. The default value works with Keycloak.	No	Authorization
oauthHeaderFormat	String formatting the value of authorization header. The token is placed instead of `%s`. The default value works with Keycloak.	No	Bearer %s
tokenTimeout	How long in milliseconds is a token valid before a new one is needed. Defaults to 60s.	No	60000L

Table 4.19. HttpsSender additional properties

Example 4.19. An example of OauthHttpSender configuration

```
1     <sender class="OauthHttpSender">
2        <target>http://domain.com/secured_url</target>
3        <property name="method" value="POST"/>
4
  <property name="tokenServerUrl" value="http://127.0.0.1:8180/auth/
realms/demo/protocol/openid-connect/token"/>
5
  <property name="tokenServerData" value="grant_type=password&amp;client_id=jboss-
javaee-webapp&amp;username=marvec&amp;password=abc123"/>
6
  <property name="responseParser" value=".*&quot;access_token&quot;:&quot;
([^&quot;]*)&quot;.*"/>
7        <property name="oauthHeader" value="Authorization"/>
8        <property name="oauthHeaderFormat" value="Bearer %s"/>
9        <property name="tokenTimeout" value="60000"/>
10    </sender>
```

[10] See http://www.keycloak.org for more information about the Keycloak server.

4.2.16. PlainSocketSender

The sender uses a socket to send the message to and receive a response from.

The target of the sender is a socket in a form of "<host>:<port>"

Example 4.20. An example of PlainSocketSender configuration

```
1    <sender class="PlainSocketSender">
2       <target>127.0.0.1:12345</target>
3    </sender>
```

4.2.17. RequestResponseJms[11]Sender

Similarily to JmsSender, there are two variants of this sender for both JMS API 1.1 (`RequestResponseJms11Sender`, new in PerfCake 7.0, formerly known as `RequestResponseJmsSender`) and 2.0 (`RequestResponseJmsSender`, new in PerfCake 7.0).

The `RequestResponse[11]JmsSender` is supposed to be used in request-response scenarios. First it sends a JMS message to the target destination (specified by the `target` property) and then waits to receive the response message from the response destination (specified by the `responseTarget` property).

It is based on `Jms[11]Sender` (see Section 4.2.12, “Jms[11]Sender”) and inherits all its configuration properties. Some properties, when not specified, use the same value as those present in `Jms[11]Sender`. However, security credentials always need to be set separately. The following table contains additional properties of the `RequestResponseJms[11]Sender`:

Property name	Description	Required	Default value
responseTarget	A JMS destination where the sender will wait and receive a response message from.	Yes	-
responseConnectionFactory	A name of a JMS Connection factory for response.	No	Value of `connectionFactory` of `JmsSender`
responseJndiContextFactory	A fully qualified name of the JNDI ContextFactory class for response.	No	Value of `jndiContextFactory` of `JmsSender`
responseJndiUrl	A JNDI location URL for response.	No	Value of `jndiUrl` of `JmsSender`
responseJndiSecurityPrincipal	A JNDI username for response	No	-
responseJndiSecurityCredentials	A JNDI password for response	No	-
responseUsername	A JMS security username for response. If not provided - JMS transport will be performed unsecured.	No	-
responsePassword	A JMS security password for response. If not provided - JMS transport will be performed unsecured.	No	-

Property name	Description	Required	Default value
receivingTimeout	A time duration in ms of specifying how long the sender will wait to receive the response message.	No	1000
receiveAttempts	A Number of attempts to receive the message.	No	5

Table 4.20. RequestResponseJms[11]Sender properties

Example 4.21. An example of RequestResponseJmsSender configuration

```
 1     <sender class="RequestResponseJmsSender">
 2        <target>jms/queue/RequestQueue</target>
 3        <property name="responseTarget" value="jms/queue/
ResponseQueue"/>
 4        <property name="receivingTimeout" value="30000"/>
 5        <property name="receiveAttempts" value="1"/>
 6        <property name="connectionFactory" value="jms/ConnFactory"/>
 7        <property name="username" value="KingRoland"/>
 8        <property name="password" value="12345"/>
 9        <property name="responseUsername" value="KingRoland"/>
10        <property name="responsePassword" value="12345"/>
11        <!-- JNDI properties-->
12
  <property name="jndiContextFactory" value="pkg.InitCtxFactory"/>
13        <property name="jndiUrl" value="remote://
${server.host}:4447"/>
14        <property name="jndiSecurityPrincipal" value="KingRoland"/>
15        <property name="jndiSecurityCredentials" value="12345"/>
16
  <property name="responseJndiSecurityPrincipal" value="KingRoland"/>
17
  <property name="responseJndiSecurityCredentials" value="12345"/>
18     </sender>
```

In the example above there is a `RequestResponseJmsSender` configured to send messages to the queue `jms/queue/RequestQueue` using a connection factory found at `jms/ConnFactory` in JNDI. After sending a JMS message the sender waits for the response from the response queue `jms/queue/ResponseQueue` for at most 30s. It uses the same JMS provider for both request and response, so the request and response credentials (both JMS and JNDI) are the same. Connection factory, JNDI context factory and JNDI URL for the response are inherited from the respective request properties.

4.2.18. ScriptSender

The sender can be used to send a single message using a JSR 223 [11] compliant script.

The target of the sender is a path to the file where the script is implemented. To specify the scripting language engine there is a `engine` property. The message and it's headers are passed to the script via script binding and should be available as variables or objects instances, according to the scripting language.

[11] See https://www.jcp.org/en/jsr/detail?id=223 for more details on JSR 223 specification.

The following table shows the senders properties.

Property name	Description	Required	Default value
engine	A scripting language engine name. `groovy` is available out-of-the-box.	No	-

Table 4.21. ScriptSender properties

Example 4.22. An example of ScriptSender configuration

Scenario:

```
1    <sender class="ScriptSender">
2       <target>/tmp/script.groovy</target>
3       <property name="engine" value="groovy"/>
4    </sender>
5    ...
6    <messages>
7       <message content="Soylent green">
8          <header name="what" value="people"/>
9       </message>
10   </messages>
```

script.groovy file:

```
1 println message.payload + " is made out of " +
message.headers['what'];
```

4.2.19. SslSocketSender

SslSocketSender is similar to the PlainSocketSender (see Section 4.2.16, “PlainSocketSender”) and so it shares the same properties. The difference is that the SslSocketSender uses a SSL socket.

Property name	Description	Required	Default value
keyStore	Path to the key store.	No	-
keyStorePassword	Key store password.	No	-
trustStore	Path to the trust store.	No	-
trustStorePassword	Trust store password.	No	-

Table 4.22. SslSocketSender additional properties

If neither `keyStore` or `trustStore` are set, the default Java `SSLSocketFactory` is used. Its configuration depends on system properties.

Example 4.23. An example of SslSocketSender configuration

```
1    <sender class="SslSocketSender">
2       <target>127.0.0.1:12345</target>
3    </sender>
```

4.2.20. WebSocketSender

The WebSocketSender can be used to send a simple messages via websocket protocol to a remote websocket server endpoint. The websocket technology creates a full-duplex connection over TCP and is suitable for applications requiring fast responses. It was standardized as RFC 6455 and it is a part of Java EE 7 (JSR-356). Connections are initiated by sending an HTTP upgrade header. It is event driven on both client and server sides. Load of requests is generated after the connection session is successfully established.

There are two forms of remote endpoints, which can be set - basic and async. To set content of message payload, set the `message` element in the `messages` section of the scenario definition. Payload can be set text, binary and ping. In PerfCake 5.x only text messages are supported.

The target of the sender is an URL of the remote endpoint, where the message is send.

The following table shows the properties of the WebSocketSender:

Property name	Description	Required	Default value
remoteEndpointType	One of `basic` or `async` .	No	basic
payloadType	One of `text` , `binary` or `ping` . [a]	No	text

[a] Only text messages are supported in PerfCake 7.x.

Table 4.23. WebSocketSender properties

Example 4.24. An example of WebSocketSender configuration

```
1     <sender class="WebSocketSender">
2        <target>ws://domain.com/ws-ctx/ws-ep</target>
3        <property name="remoteEndpointType" value="basic"/>
4        <property name="payloadType" value="text"/>
5     </sender>
```

4.3. Receiving messages

A `Receiver` is a component used to receive responses from a separate message channel. It is though possible to send requests to some protocol and use a completely different protocol to receive responses.

The `Receiver` always needs to know how many threads it should use to receive responses and from where it should read them. So there is a mandatory attribute `threads`, and two mandatory tags `source` and `correlator`. For *Correlators* see Section 4.4, “Correlating messages”.

Some *Senders* allow generating requests without specifying a message in the scenario. However, this does not work with *Receivers* and *Correlators*. There always needs to be a message specified (be it a message with an empty body) for the *Correlator* to be able to compute (and possibly store) the correlation ID. PerfCake insists on user specifying the message for them to realize the configuration is different (in other words, PerfCake does not perform any magic hidden operations on your configuration).

Example 4.25. An example of a Receiver configuration

```
1     <receiver class="..." threads="...">
2        <correlator class="..." />
```

```
3        <source>...</source>
4        ...
5        receiver properties
6        ...
7     </receiver>
```

When specifying the receiver class, unless you enter a fully classified class name, the default package `org.perfcake.message.receiver` is assumed.

In the following sections you can find a complete description of all *Receivers* that can be used by PerfCake including all of their properties.

4.3.1. HttpReceiver

The HttpReceiver can be used to receive responses from an HTTP channel. It opens an HTTP port and listens for responses. After receiving a single response it immediatelly cresponds with the given response, status code and status message.

Following table shows the properties of the HttpReceiver:

Property name	Description	Required	Default value
source	<address>[:<port>] where to bind the HTTP server and listen for the messages. The default port, if not specified, is 8088. Please note that for lower port numbers you might need to run PerfCake as a superuser (root, administrator).	Yes	-
httpStatusCode	The HTTP status code to return to the client sending the message.	No	200
httpStatusMessage	A status message to return to the client sending the message. By default, no status message is set.	No	-
httpResponse	A response to return to the client sending the message. By default, no response is sent.	No	-

Table 4.24. HttpReceiver properties

Example 4.26. An example of HttpReceiver configuration

```
1     <receiver class="HttpReceiver" threads="100">
2        <correlator class="GenerateHeaderCorrelator" />
3        <source>localhost:3000</source>
4        <property name="httpResponse" value="Hello this is
PerfCake!"/>
```

```
5     </receiver>
```

4.4. Correlating messages

All messages/responses received via a `Receiver` are passed to a `Correlator` which notifies the correct `SenderTask`. The `Correlator` component can extract a *correlation ID* from both the requesnt and the response. It can also have an active role and actually create a *correlation ID* and store it in the request message.

`Correlator` is always bound to a `Receiver` and does not make any sense without it.

Example 4.27. An example of a correlator configuration

```
1     <receiver class="..." threads="...">
2        <correlator class="...">
3           ...
4           correlator properties
5           ...
6        </correlator>
7        <source>...</source>
8        ...
9        receiver properties
10       ...
11    </receiver>
```

When specifying the correlator class, unless you enter a fully classified class name, the default package `org.perfcake.message.correlator` is assumed.

In the following sections you can find a complete description of all correlators that can be used by PerfCake including all of their properties.

4.4.1. GenerateHeaderCorrelator

`GenerateHeaderCorrelator` creates a new UUID style *correlation ID* and stores it in a request message header and in message attributes. It then searches for the same value in the response message headers. The header name used for storing the UUID is `perfcake.correlation.id`.

`GenerateHeaderCorrelator` does not have any configurable properties.

Example 4.28. An example of GenerateHeaderCorrelator usage

```
1     <receiver class="HttpReceiver" threads="10">
2        <correlator class="GenerateHeaderCorrelator" />
3        <source>localhost:8080</source>
4     </receiver>
```

4.4.2. PrefixCorrelator

`PrefixCorrelator` uses a message prefix - a string from the beginning of the message to a prefix boundary (`:` by default) - as a correlation ID. The prefix boundary can be configured via `prefixBoundary` property of the correlator.

Following table shows the properties of the PrefixCorrelator:

Property name	Description	Required	Default value
prefixBoundary	A character or a string specifying the prefix boundary.	No	:

Table 4.25. PrefixCorrelator properties

Example 4.29. An example of PrefixCorrelator usage

```
1    <receiver class="HttpReceiver" threads="10">
2       <correlator class="PrefixCorrelator">
3          <property name="prefixBoundary" value=":"/>
4       </correlator>
5       <source>localhost:8080</source>
6    </receiver>
```

In the example, when a message `prefix-01:What is the meaning of life?` is sent through a sender, the correlator extract the prefix `prefix-01` according to the prefix boundary `:` and uses it as a correlation ID. The receiver then listens on HTTP address `http://localhost:8080` and waits for a response message with the same prefix (e.g. `prefix-01:I don't know, the computers are down.`).

4.5. What - Messages

A message is an actual payload, that is sent by a sender. To specify what will be sent, you can use the `uri` attribute of the particular message element in the scenario configuration. The `uri` can be an absolute file path in a form of `file://...` , URL or just a file name, in which case PerfCake will look for the file in the messages directory (See https://www.perfcake.org/quickstart/).

The simple payload of the message can be specified directly in the scenario by using `content` attribute instead of the`uri`. The value of the `content` attribute will be the actual message payload.

The scenario can be configured to send more that one message or to send a message more than once in a single iteration. To specify multiple different messages you just need to add more `message` elements in the `messages` configuration. To send a particular message more than once use the `multiplicity` attribute of the respective `message` element.

Example 4.30. An example of a simple message configuration:

```
1    <messages>
2       ...
3       <message content="Greetings stranger!"/>
4       ...
5    </messages>
```

In the example above there is a single message defined. The payload of the message is the value of the `content` attribute .

Example 4.31. An example of multiple messages configuration:

```
1    <messages>
2       <message uri="message1.txt">
3          <header name="header.name" value="header.value"/>
4          <header name="header2" value="you-know"/>
```

```
 5            <property name="Empire.State.Building" value="A lot of
$"/>
 6         </message>
 7         <message uri="message2.xml" multiplicity="2"/>
 8     </messages>
```

In the example above there are 2 messages defined. In the case of the first message the payload is taken from the file `message1.txt` and the message has two headers and one property specified. The second message is taken from the file `message2.xml` and will be sent two times in each iteration.

4.5.1. Filtering and templates

Messages stored in separate files can take an advantage of more complex filtering than those with body specified directly in the scenario file. This is optimized for performance. However, take into account that this might have a negative impact on the maximal message generation speed. It is recommended to create a comparative test without any templates to be sure there is no significant drop in troughput.

All system properties are must be prefixed with the *props.* namespace declaration. All environment properties must be prefixed with the *env.* namespace declaration. Internal PerfCake properties (i.e. message number) do not need any prefix. Sequences are accessible without any prefix, unless you defined with one.

While loading the message body from a file, all placeholders with the dollar sign are replaced first (for example`${env.JAVA_HOME}`). This is done only once and no later changes to the property values take any effect. There is an option to be able to replace a placeholder with a fresh property value each tima a new message is created/being sent. Such a property placeholder uses the *at* sign (for example`@{props.counter}`).

To specify a default value for a property, the double pipe character is used. For instance: `${property_name||default_value}`

Any occurrence of the placeholder can be escaped using the backslash sign. This means that the following placeholders will never get replaced: `\${non-replaceable} \@{ignored-placeholder}`

For more details see Section 2.2.5, “Filtering properties”.

4.6. Sequences

Sequences are used to create a sequence of values that change for each sent message set (a set of messages are all the messages specified in a single scenario). Whenever a message set is to be sent, a current snapshot of all sequences is created. Then the values are replaced in the message template and are also passed to `MessageSender.preSend()` method in `messageAttributes`. The property names/keys correspond to the configured sequence `id`.

We suggest to configure your sequence ids with a unique prefix (like *seq.*) to avoid possible conflicts with another properties.

Possible usages of sequences are:

- in the message templates to make each message unique or to make it route through a different path,
- to influence the behaviour of a message sender which can decide based on the sequence's current value (values can for example cycle over a few different values),
- to send the messages to different targets, supposing the message sender does not cache connections and uses `AbstractSender.getSafeTarget()` when openning a new connection.

Example 4.32. An example of a Sequence configuration

```
1    <sequences>
2       <sequence id="..." class="...">
3          ...
4          sequence properties
5          ...
6       </sequence>
7    </sequences>
```

When specifying the sequence class, unless you enter a fully classified class name, the default package `org.perfcake.message.sequence` is assumed.

In the following sections you can find a complete description of all sequences, that can be used by PerfCake including all of their properties.

4.6.1. PrimitiveNumberSequence

Just an ever increasing number sequence starting at 0. There are no properties available but provides high performance.

Example 4.33. An example of PrimitiveNumberSequence configuration

```
1    <sequence id="mySequence" class="PrimitiveNumberSequence" />
```

4.6.2. NumberSequence

Long number sequence with the ability to specify boundaries, direction (increasing or decreasing) of the counter and the step size.

Following table shows the properties of `NumberSequence`:

Property name	Description	Required	Default value
start	Beginning of the sequence counter.	No	0
end	End of the sequence counter (when reached, the counter can start from `start` depending on the `cycle` property). With positive (negative) step, `Long.MIN_VALUE` (`Long.MAX_VALUE`) disables checking for reaching the end value.	No	`Long.MIN_VALUE`
step	The step of the counter, can be either positive or negative.	No	1
cycle	The counter starts from beginning when reaching end again. Otherwise the counter	No	`true`

Property name	Description	Required	Default value
	stays at the end value for the rest of the test.		

Table 4.26. NumberSequence properties

Example 4.34. An example of NumberSequence configuration

```
1    <sequence id="mySequence" class="NumberSequence">
2       <property name="start" value="10" />
3       <property name="end" value="0" />
4       <property name="step" value="-2" />
5    </sequence>
```

4.6.3. RandomSequence

This sequence generates random numbers in the given range `<min, max)`. i.e. including the `min` value but not the `max` value.

Following table shows the properties of `RandomSequence`:

Property name	Description	Required	Default value
min	Lower boundary of the interval of random number (inclusive).	No	0
max	Upper boundary of the interval of random numbers (exclusive).	No	100

Table 4.27. RandomSequence properties

Example 4.35. An example of RandomSequence configuration

```
1    <sequence id="mySequence" class="RandomSequence">
2       <property name="min" value="10" />
3       <property name="max" value="256" />
4    </sequence>
```

4.6.4. RandomUuidSequence

This sequence generates random UUID identifiers. It has no configuration parameters. All UUIDs have the same format. Sample UUID is `259f1e8a-816e-4fa0-a5eb-15e2aefe57a6`.

Example 4.36. An example of RandomUuidSequence configuration

```
1    <sequence id="mySequence" class="RandomUuidSequence"/>
```

4.6.5. ThreadIdSequence

Returns the value of `Thread.currentThread().getId()`. No configuration properties are available.

Example 4.37. An example of ThreadIdSequence configuration

```
1   <sequence id="mySequence" class="ThreadIdSequence" />
```

4.6.6. TimeStampSequence

Returns the value of System.currentTimeMillis(). No configuration properties are available.

Example 4.38. An example of TimeStampSequence configuration

```
1   <sequence id="mySequence" class="TimeStampSequence" />
```

4.6.7. FileLinesSequence

Every single line in a given input file specifies a value of this sequence. Once the end of the file is hit, the sequence starts from beginning. The whole file is read in the memory in advance. Please make sure the file is of a reasonable size given your expectations and memory limits.

Following table shows the properties of `FileLinesSequence`:

Property name	Description	Required	Default value
fileUrl	Location of the file with sequence values.	Yes	-

Table 4.28. FileLinesSequence properties

Example 4.39. An example of FileLinesSequence configuration

```
1   <sequence id="mySequence" class="FileLinesSequence">
2      <property name="fileUrl" value="sequence-lines.txt" />
3   </sequence>
```

4.6.8. FilesContentSequence

Every single line in a given input index file specifies another file with sequence value. Once the end of the index file is hit, the sequence starts from beginning. The content of individual files is cached by default. Please make sure the files are of a reasonable size given your expectations and memory limits.

Following table shows the properties of `FilesContentSequence`:

Property name	Description	Required	Default value
fileUrl	Location of the index file with references to files with sequence values.	Yes	-
cacheContent	When true (the default value) the content of individual files is cached in memory.	No	true

Table 4.29. FilesContentSequence properties

Example 4.40. An example of FilesContentSequence configuration

```
1   <sequence id="mySequence" class="FilesContentSequence">
```

```
2        <property name="fileUrl" value="sequence-index.txt" />
3     </sequence>
```

4.7. Reporting

This chapter is about PerfCake's reporting abilities. It is configured using <reporting> element in the scenario definition.

The configuration consists of the following steps:

- Configure a reporter
- Configure the reporter's destinations

A reporter represents a different type of the reports such as average throughput or memory usage. By configuring the destinations you tell where output should be directed by the reporter (e.g. console, CSV file, etc.).

A reporter can publish multiple results (e.g. current value, average value, etc.) each with a particular name. The actual names of the results are described with the particular reporter.

When specifying the reporter class, unless you enter a fully classified class name, the default package `org.perfcake.reporting.reporter` is assumed. For the destination class, the default package is `org.perfcake.reporting.destination`.

Example 4.41. An example reporting configuration:

```
1     <reporting>
2        <reporter class="ThroughputStatsReporter">
3           <property name="minimumEnabled" value="false"/>
4           <property name="maximumEnabled" value="false"/>
5           <destination class="ConsoleDestination">
6              <period type="time" value="1000"/>
7           </destination>
8           <destination class="CsvDestination">
9              <period type="time" value="2000"/>
10             <property name="path" value="test-average-
throughput.csv"/>
11          </destination>
12       </reporter>
13
14       <reporter class="MemoryUsageReporter">
15          <property name="agentHostname" value="localhost"/>
16          <property name="agentPort" value="8850"/>
17          <destination class="CsvDestination">
18             <period type="time" value="2000"/>
19             <property name="path" value="test-memory-usage.csv"/>
20          </destination>
21       </reporter>
22    </reporting>
```

With this configuration the 2 reporters are specified - `ThroughputStatsReporter` and `MemoryUsageReporter` . First one will report to console each 1 second and to CSV file each 2 seconds, while the second one will report memory usage of the tested system into CSV file each 2 seconds.

Each reporter can be enabled/disabled by the optional boolean attribute called `enabled` . The disabled reporter is just ignored by PerfCake just like it wouldn't be there at all. If not specified the reporter is enabled by default.

Example 4.42. An example of a disabled reporter:

```
1   <reporting>
2      <reporter class="ThroughputStatsReporter">
3         ...
4      </reporter>
5
6      <reporter class="MemoryUsageReporter" enabled="false">
7         ...
8      </reporter>
9   </reporting>
```

In the example above there are two reporters configured, `ThroughputStatsReporter` which is enabled and `MemoryUsageReporter` which is disabled.

The following sections contain a description of reporters and destinations that can be used in PerfCake.

4.7.1. Reporters

ClassifyingReporter

The reporter can monitor a selected message attribute (i.e. a sequence), classify and count the number of appearance of individual values. The values are then reported by the class name with an optional prefix.

Property name	Description	Required	Default value
attribute	The name of the message attribute to classify.	Yes	
prefix	A prefix used in the result map for individual class names.	No	class_

Table 4.30. ClassifyingReporter properties

Example 4.43. An example of ClassifyingReporter configuration

```
1   <sequences>
2      <sequence class="ThreadIdSequence" id="threadId" />
3   </sequences>
4   ...
5   <reporting>
6      <reporter class="ClassifyingReporter">
7         <property name="attribute" value="threadId" />
8         <property name="prefix" value="thread_" />
9         <destination class="ConsoleDestination">
10           <period type="iteration" value="1000"/>
11        </destination>
12     </reporter>
```

```
13    </reporting>
14    ...
```

In the example above there is a `ClassifyingReporter` configured to report the utilization of individual threads used to to send messages. The number of classes is equal to the number of threads configured. The sum of the values is the number of iterations passed in total.

An example output for the above example can be as follows.

Example 4.44. An example of the output when ClassifyingReporter is used

```
[0:00:00][60000 iterations][60%] [warmUp => false] [Threads => 10]
 [failures => 0] [thread_22 => 4381]↵
[thread_23 => 3734] [thread_24 => 7607] [thread_14 => 8016] [thread_15
 => 3338] [thread_16 => 3204]↵
[thread_17 => 7790] [thread_18 => 7215] [thread_19 => 8493] [thread_21
 => 6222]
```

GeolocationReporter

The reporter figures out geo-location information from a 3rd party service (http://ipinfo.io) and stores the returned values in the results. The values are obtained just once for the whole test execution so it does not make much sense to make this reporter report more than once. This can be achieved by setting the reporting period to a very large number - the first iteration is always reported and no others will be.

The reporter has just a single configuration property. It allows you to switch to a different geo-location service provider. However its configuration is questionable because it expects very specific output from the 3rd service. The best possibility if to provide your own service with the exact same JSON result format.

PerfCake needs internet access for this reporter to work. As a bonus, the reporter count average iterations per second in the same way as `IterationsPerSecondReporter` does.

Property name	Description	Required	Default value
serviceUrl	The location of the 3rd party geo-location service.	No	http://ipinfo.io/json

Table 4.31. GeolocationReporter properties

The following table describes result names of `GeolocationReporter`:

Result name	Description
Result	The current throughput in iterations/s
ip	Public address of the IP address where PerfCake runs (or your provider's IP address)
hostname	Your or your provider's hostname.
city	Estimated city where PerfCake runs.
region	Estimated region where PerfCake runs.
country	Estimated country where PerfCake runs.
lat	Estimated latitude (+ means north, - means south).

Result name	Description
lon	Estimated longitude (+ means east, - means west).

Table 4.32. GeolocationReporter result names

Example 4.45. An example of GeolocationReporter configuration

```
1    <reporter class="GeolocationReporter">
2       ...
3       (destinations)
4       ...
5    </reporter>
```

IterationsPerSecondReporter

The reporter reports a plain high-level throughput (in the means of the number of iterations per second) from the beginning of the measuring to the moment when the results are published. The result is computed from the current number of processed iterations at the moment of publishing result and the time duration from the beginning (or warmup).

The reporter does not have any specific properties.

The following table describes result names of `IterationsPerSecondReporter`:

Result name	Description
Result	The current throughput in iterations/s

Table 4.33. IterationsPerSecondReporter result names

Example 4.46. An example of IterationsPerSecondReporter configuration

```
1    <reporter class="IterationsPerSecondReporter">
2       ...
3       (destinations)
4       ...
5    </reporter>
```

In the example above there is an `IterationsPerSecondReporter` configured to report the current and the average value of the throughput.

MemoryUsageReporter

The reporter is able to report the current memory usage of the tested system at the moment when the results are published. It requires PerfCake agent to be installed in the tested system's JVM.

To be able to use MemoryUsageReporter you need to attach PerfCake agent to the tested system's JVM. The PerfCake agent is a part of binary distribution (PerfCake Agent's JAR archive). The agent is configurable by following properties:

Property name	Description	Required	Default value
hostname	IP address of hostname where PerfCake agent is listening.	No	localhost

Property name	Description	Required	Default value
port	A port number where Perf-Cake agent is listening.	No	8850

Table 4.34. PerfCake agent properties

To attach the agent to the tested system's JVM, append the following JVM argument to the executing java command or use JAVA_OPTS environment variable. It is also possible to attach the agent to an already running JVM since Java 7 (supposing you have `tools.jar` on the classpath).

Example 4.47. JVM argument to attach PerfCake agent to the tested JVM

```
"... -
javaagent:<perfcake_agent_jar_path>=hostname=<hostname>,port=<port>"
```

Example 4.48. PerfCake JVM argument example

```
JAVA_OPTS="... -javaagent:$PERFCAKE_HOME/lib/perfcake-
agent-7.x.jar=port=8850"
```

Example 4.49. Attaching to running JVM

```
java -cp $JAVA_HOME/lib/tools.jar:perfcake-agent-7.x.jar \
  org.perfcake.agent.PerfCakeAgent <PID>
 hostname=<hostname>,port=<8850>
```

Once you have started the tested system up, you should see the following line in the system's console output:

```
...
PerfCakeAgent > Listening at localhost on port 8850
...
```

Once you have the PerfCake agent attached and the tested system is up and running you can use the MemoryUsageReporter to measure the memory usage of the tested system.

MemoryUsageReporter is capable of possible memory leak detection. It is disabled by default. Once enabled, the reporter periodically gathers memory usage from the tested system using via an ordinary way (using the PerfCakeAgent) and remembers a window of N last measured values. Once the window is filled, the reporter uses a linear regression analysis over the data from the time window to compute an used memory trend. The possible memory leak is considered detected when the slope of the memory trend exceeds the specified slope threshold. All the period, the window size and the slope threshold are configurable via particular reporter's properties.

The reporter is also able to dump memory when a possible memory leak is detected. The feature can be enabled by the `memoryDumpOnLeak` property and the memory dump is then saved in a file which can be specified by the `memoryDumpFile` property. If not specified the dump name will be generated as `"dump-" + System.currentTimeMillis() + ".bin"` in case of the Java agent that is part of PerfCake.

The reporter can ask the agent to perform a garbage collection each time the memory usage of the tested system is measured and published. Since the garbage collection is CPU intensive operation be careful to enable it and to how often the memory usage is measured because it will have a significant impact on the measured system and naturally the measured results too.

The reporter has the following properties:

Property name	Description	Required	Default value
agentHostname	An IP address of hostname where the PerfCake agent is listening.	No	localhost
agentPort	A port number where the PerfCake agent is listening.	No	8850
memoryDumpOnLeak	The property to make a memory dump, when possible memory leak is detected. The MemoryUsageReporter will send a command to PerfCake agent that will create a heap dump.	No	false
memoryDumpFile	The property specifying the name of the memory dump file. The full "file:" URI is supported.	No	-
memoryLeakDetectionEnabled	Enables or disables the memory leak detection.	No	false
memoryLeakDetectionMonitoringPeriod	A time period in ms in which the memory leak detection mechanism gathers memory usage data.	No	500
memoryLeakSlopeThreshold	The used memory trend slope threshold. The slope's unit is a byte per second.	No	1024
performGcOnMemoryUsage	The property is used to enable/disable performing garbage collection each time the memory usage of the tested system is measured and published. Since the garbage collection is CPU intensive operation be careful to enable it and to how often the memory usage is measured because it will have a significant impact on the measured system and naturally the measured results too.	No	false
usedMemoryTimeWindowSize	The used memory time window size. (Number of records in the memory data set used for the statistical analysis).	No	100

Table 4.35. MemoryUsageReporter properties

The following table describes result names of MemoryUsageRepoter:

Result name	Description
Used	The amount of currently used memory in the Java Virtual Machine.
Total	The total amount of memory in the Java virtual machine in MiB.
Max	The maximum amount of memory that the Java virtual machine will attempt to use in MiB
UsedTrend	The memory usage regression line slope in B/s.
MemoryLeak	A boolean value indicating whether a possible memory leak has been detected yet.

Table 4.36. MemoryUsageReporter result names

Example 4.50. An example of MemoryUsageReporter configuration

```
1    <reporter class="MemoryUsageReporter">
2       <property name="agentHostname" value="localhost"/>
3       <property name="agentPort" value="8850"/>
4       ...
5       (destinations)
6       ...
7    </reporter>
```

ResponseTimeHistogramReporter

Reports response time in milliseconds using HDR Histogram [https://github.com/HdrHistogram/HdrHistogram] that can computationally correct the Coordinated omission problem.

The following paragraphs are based on the HDR Histogram documentation [https://github.com/HdrHistogram/HdrHistogram/blob/master/README.md].

This reporter depends on the features introduced by HDR Histogram to correct the coordinated omission. This problem occurs when all sending threads are blocked waiting for a response from a system under test that suddenly stopped responding for a relatively long time (longer than it did in the past). Under these conditions, no additional bad results with high response time are recorded while the system is still blocked. To have the balanced result, we should have approximately the same number of measurements for each time interval during the test execution.

To compensate for the loss of sampled values when a recorded value is larger than the expected, interval between value samples, HDR Histogram will auto-generate an additional series of decreasingly-smaller value records. The values go down to the `expectedValue` in case of the `user` correction mode, or down to the average response time in case of the `auto` correction mode.

The reporter tries to divide the time range between shortest and longest response time into intervals of similar length and calculate the percentiles for the intervals.

For example, the reporter can be configured to track the counts of observed response times in milliseconds between 0 and 3,600,000 (`maxExpectedValue`) while maintaining a value precision of 3 (`precision`) significant digits across that range. Value quantization within the range will thus be no larger than 1/1,000th (or 0.1%) of any value. This example reporter could be used to track and analyze the counts of observed response times ranging between 1 millisecond and 1 hour in magnitude, while maintaining a value resolution of 1 millisecond (or better) up to one second, and a resolution of 1 second (or better)

up to 1,000 seconds. At its maximum tracked value (1 hour), it would still maintain a resolution of 3.6 seconds (or better).

Property name	Description	Required	Default value
precision	Precision of the resulting histogram (number of significant digits) in range 0 - 5. This determines the memory used by the reporter. Also, for low precision, numbers are recorded in less precise ranges.	No	2
detail	Detail level of the result (the number of iteration steps per half-distance to 100%). Must be greater than 0.	No	2
maxExpectedValue	The maximum expected value to better organize the data in the histogram. The response time reported must never exceed this value, otherwise the result will be skipped, an error reported and the output will be invalid. -1 turns the optimization off. It is valuable to set some reasonable number like 3,600,000 which equals to the resolution from 1 millisecond to 1 hour.	No	-1 (unspecified)
correctionMode	The correction of coordinated omission in the resulting histogram. `auto` is the default value and this means that the histogram is corrected according to the average measured value. In the `user` correction mode, the values are corrected according to the expectedValue specified by user. This is useful when you know thhe expected response time in advance. The correction for coordinated omission is turned off by	No	auto

Property name	Description	Required	Default value
	setting `none` to this property.		
expectedValue	The value of normal/typical/expected response time in ms to correct the histogram while the `user` correction mode is turned on.	Only when `correctionMode` is set to `user`, however the default value can still be used.	1
prefix	String prefix used in the result map for histogram entries. This prefix is followed by the percentile for the corresponding range cumulatively. E.g. `perc0.98834000=14` means that in 98.834% of measurements, the response time was 14 or better.	No	perc
filter	When true, it tries to minimize the number of reported values while keeping the same level of information. For instance, instead of reporting `perc0.0=2`, `perc0.5=2`, `perc0.75=2`, `perc0.882=3`, just the values `perc0.75=2`, `perc0.882=2` are reported. It is then obvious that all percentiles under 0.75 are equal to 2.	No	false

Table 4.37. ResponseTimeHistogramReporter properties

ResponseTimeHistogramReporter can be best used with *CsvDestination* and *ChartDestination* as you can see in the following example.

Example 4.51. An example of ResponseTimeHistogramReporter configuration

```
1    <reporter class="ResponseTimeHistogramReporter">
2       <property name="detail" value="1" />
3       <property name="precision" value="1" />
4       <property name="maxExpectedValue" value="100" />
5       <property name="correctionMode" value="user" />
6       <property name="expectedValue" value="2" />
7       ...
8       <destination class="ChartDestination">
9          <period type="time" value="500"/>
```

```
10            <property name="yAxis" value="HDR Response time [ms]"/>
11
 <property name="group" value="${perfcake.scenario}_hdr_resp"/>
12            <property name="name" value="HDR Response Time
(${threads:25} threads)"/>
13            <property name="attributes" value="*, warmUp"/>
14            <property name="autoCombine" value="false" />
15            <property name="chartHeight" value="1000" />
16            <property name="outputDir" value="target/
${perfcake.scenario}-charts"/>
17         </destination>
18         <destination class="CsvDestination">
19            <period type="time" value="500"/>
20            <property name="expectedAttributes" value="*" />
21            ...
22         </destination>
23         ...
24      </reporter>
```

In the example above there is an instance of `ResponseTimeHistogramReporter` configured to report at slightly lower precision and detail level than default with correction for coordinated omission expecting the system under test to response within 2ms and no response time larger than 100ms is expected. The results will be writen as a chart and into a CSV file.

ResponseTimeStatsReporter

The reporter is able to report the statistics - current, minimal, maximal and average value of a response time (in miliseconds) from the beginning of the measuring to the moment when the results are published (default) or in a specified window. The default result of this reporter is the current response time.

Property name	Description	Required	Default value
minimumEnabled	Enables minimal value measuring.	No	true
maximumEnabled	Enables maximal value measuring.	No	true
averageEnabled	Enables average value measuring.	No	true
requestSizeEnabled	Enables measuring of total size of requests sent.	No	true
responseSizeEnabled	Enables measuring of total size of responses received.	No	true
windowSize	A window where the data for the statistics are taken from. The value unit depends on the window type specified by the `windowType` property.	No	Integer.MAX_VALUE
windowType	A type of the window. It is either number of last iterations of an amount of time in milliseconds. The values	No	iteration

Property name	Description	Required	Default value
	of `iteration` or `time` is supported.		
histogram	A comma separated list of values where the histogram is split to individual ranges.	No	
histogramPrefix	String prefix used in the result map for histogram entries. This prefix is followed by the mathematical representation of the particular range.	No	in

Table 4.38. ResponseTimeStatsReporter properties

The following table describes result names of ResponseTimeStatsReporter:

Result name	Description
Result	The current response time in ms - of the latest iteration.
Minimum	The minimal response time in ms measured so far (in a given sliding window).
Maximum	The minimal response time in ms measured so far (in a given sliding window).
Average	The average response time in ms measured so far (in a given sliding window).
RequestSize	The size of all requests sent so far (in a given sliding window).
ResponseSize	The size of all responses received so far (in a given sliding window).
`${histogramPrefix}<from:to)`	If `histogram` is used, there is a result with the value of histogram for each range. Example: `in<100.0:200.0)` for a value range between 100.0 and 200.0 and the `histogramPrefix` set to "`in`".

Table 4.39. ResponseTimeStatsReporter result names

Example 4.52. An example of ResponseTimeStatsReporter configuration

```
1 <reporter class="ResponseTimeStatsReporter">
2   <property name="minimumEnabled" value="false"/>
3   <property name="maximumEnabled" value="false"/>
4   ...
5   (destinations)
6   ...
7 </reporter>
```

Example 4.53. An example of ResponseTimeStatsReporter configuration with histogram

```
1    <reporter class="ResponseTimeStatsReporter">
```

```
2       <property name="histogram" value="100,200"/>
3       <property name="histogramPrefix" value="in"/>
4
5       <destination class="ConsoleDestination">
6          <period type="time" value="5000" />
7       </destination>
8       ...
9       (destinations)
10      ...
11   </reporter>
```

In the example above a `ResponseTimeStatsReporter` is configured to report all statistics with the following output:

```
2016-06-13 23:20:22,158 INFO  {org.perfcake.ScenarioExecution} ===
 Welcome to PerfCake 7.5 ===
2016-06-13 23:20:22,159 INFO  {org.perfcake.util.TimerBenchmark}
 Benchmarking system timer resolution...
2016-06-13 23:20:22,160 INFO  {org.perfcake.util.TimerBenchmark} This
 system is able to differentiate up to 356ns. A single thread is now
 able to measure maximum of 2808988 iterations/second.
2016-06-13 23:20:22,177 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Starting to
 generate...
[0:00:00][1 iterations][0%] [1.084454 ms] [warmUp => false] [Threads
 => 10] [ResponseSize => 256.00 B] [Minimum => 1.084454 ms] [Maximum
 => 1.084454 ms] [failures => 0] [RequestSize => 256.00 B] [Average =>
 1.084454 ms]
2016-06-13 23:20:22,201 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Reached test
 end. All messages were prepared to be sent.
2016-06-13 23:20:22,201 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Waiting for
 all messages to be sent...
[0:00:00][100 iterations][10%] [1.059404 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 25.00 KiB] [Minimum => 1.009627 ms]
 [Maximum => 2.501367 ms] [failures => 0] [RequestSize => 25.00 KiB]
 [Average => 1.11484413 ms]
[0:00:00][200 iterations][20%] [1.057651 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 50.00 KiB] [Minimum => 1.008045 ms]
 [Maximum => 3.143239 ms] [failures => 0] [RequestSize => 50.00 KiB]
 [Average => 1.1479294649999998 ms]
[0:00:00][300 iterations][30%] [1.06057 ms] [warmUp => false] [Threads
 => 10] [ResponseSize => 75.00 KiB] [Minimum => 1.008045 ms] [Maximum
 => 3.143239 ms] [failures => 0] [RequestSize => 75.00 KiB] [Average
 => 1.12146921 ms]
[0:00:00][400 iterations][40%] [1.063183 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 100.00 KiB] [Minimum => 1.008045 ms]
 [Maximum => 3.143239 ms] [failures => 0] [RequestSize => 100.00 KiB]
 [Average => 1.1061472125 ms]
[0:00:00][500 iterations][50%] [1.114687 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 125.00 KiB] [Minimum => 1.008045 ms]
 [Maximum => 3.143239 ms] [failures => 0] [RequestSize => 125.00 KiB]
 [Average => 1.1018137200000009 ms]
```

```
[0:00:00][600 iterations][60%] [1.031199 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 150.00 KiB] [Minimum => 1.008045 ms]
 [Maximum => 3.988934 ms] [failures => 0] [RequestSize => 150.00 KiB]
 [Average => 1.1051423050000004 ms]
[0:00:00][700 iterations][70%] [1.272101 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 175.00 KiB] [Minimum => 1.008045 ms]
 [Maximum => 3.988934 ms] [failures => 0] [RequestSize => 175.00 KiB]
 [Average => 1.1001457528571437 ms]
[0:00:00][800 iterations][80%] [1.059498 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 200.00 KiB] [Minimum => 1.007744 ms]
 [Maximum => 3.988934 ms] [failures => 0] [RequestSize => 200.00 KiB]
 [Average => 1.0959077025000006 ms]
[0:00:00][900 iterations][90%] [1.079325 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 225.00 KiB] [Minimum => 1.007744 ms]
 [Maximum => 6.986077 ms] [failures => 0] [RequestSize => 225.00 KiB]
 [Average => 1.1047625722222227 ms]
[0:00:00][1000 iterations][100%] [1.05882 ms] [warmUp => false]
 [Threads => 10] [ResponseSize => 250.00 KiB] [Minimum => 1.007744 ms]
 [Maximum => 6.986077 ms] [failures => 0] [RequestSize => 250.00 KiB]
 [Average => 1.104236193000001 ms]
2016-06-13 23:20:23,310 INFO  {org.perfcake.reporting.ReportManager}
 Checking whether there are more results to be reported...
2016-06-13 23:20:23,313 INFO  {org.perfcake.ScenarioExecution} ===
 Goodbye! ===
```

ThroughputStatsReporter

The reporter is able to report the statistics - current, minimal, maximal and average value of a pure throughput (in the means of the number of iterations per second) from the beginning of the measuring to the moment when the results are published (default) or in a specified window. The default result of this reporter is the current pure throughput.

The pure throughput is how much iterations per second would the tested system be able to process under the current load if the overhead was zero. It is computed from a response time simply by inverting the value of the response time and multiplying by the number of threads.

Property name	Description	Required	Default value
minimumEnabled	Enables minimal value measuring.	No	true
maximumEnabled	Enables maximal value measuring.	No	true
averageEnabled	Enables average value measuring.	No	true
requestSizeEnabled	Enables measuring of total size of requests sent.	No	true
responseSizeEnabled	Enables measuring of total size of responses received.	No	true
windowSize	A window where the data for the statistics are taken from. The value unit depends on the window type	No	Integer.MAX_VALUE

Property name	Description	Required	Default value
	specified by the `windowType` property.		
windowType	A type of the window. It is either number of last iterations of an amount of time in milliseconds. The values of `iteration` or `time` is supported.	No	iteration
histogram	A comma separated list of values where the histogram is split to individual ranges.	No	
histogramPrefix	String prefix used in the result map for histogram entries. This prefix is followed by the mathematical representation of the particular range.	No	in

Table 4.40. ThroughputStatsReporter properties

The following table describes result names of ThroughputStatsReporter:

Result name	Description
Result	The current throughput in iterations/s - of the latest iteration.
Minimum	The minimal throughput in iterations/s measured so far (in a given time window).
Maximum	The minimal throughput in iterations/s measured so far (in a given time window).
Average	The average throughput in iterations/s measured so far (in a given time window).
RequestSize	The size of all requests sent so far (in a given sliding window).
ResponseSize	The size of all responses received so far (in a given sliding window).
`${histogramPrefix}<from:to)`	If `histogram` is used, there is a result with the value of histogram for each range. Example: `in<100.0:200.0)` for a value range between 100.0 and 200.0 and the `histogramPrefix` set to "`in`".

Table 4.41. ThroughputStatsReporter result names

Example 4.54. An example of ThroughputStatsReporter configuration with a sliding window over last 30 iterations

```
1     <reporter class="ThroughputStatsReporter">
2        <property name="minimumEnabled" value="false"/>
3        <property name="maximumEnabled" value="false"/>
4        <property name="windowSize" value="30"/>
5        ...
6        (destinations)
7        ...
```

```
8    </reporter>
```

In the example above there is a `ThroughputStatsReporter` configured to report the current and the average value of the throughput in a sliding window of 30 iterations.

Example 4.55. An example of output with the above configuration

```
[0:00:01][50 iterations][10%] [68.56845152517322 iterations/s] [warmUp
 => false] [Threads => 10] [ResponseSize => 19.24 KiB] [failures => 0]
 [RequestSize => 650.00 B] [Average => 56.58369666441687 iterations/s]
[0:00:02][125 iterations][20%] [72.44633269853925 iterations/
s] [warmUp => false] [Threads => 10] [ResponseSize => 19.24
 KiB] [failures => 0] [RequestSize => 650.00 B] [Average =>
 71.35868254120271 iterations/s]
[0:00:03][188 iterations][30%] [73.01081433991678 iterations/
s] [warmUp => false] [Threads => 10] [ResponseSize => 19.24
 KiB] [failures => 0] [RequestSize => 650.00 B] [Average =>
 68.11154264877203 iterations/s]
[0:00:04][253 iterations][40%] [72.21612698050471 iterations/
s] [warmUp => false] [Threads => 10] [ResponseSize => 19.24
 KiB] [failures => 0] [RequestSize => 650.00 B] [Average =>
 71.15376567483213 iterations/s]
[0:00:05][309 iterations][50%] [72.30052923842801 iterations/s]
 [warmUp => false] [Threads => 10] [ResponseSize => 19.24 KiB]
 [failures => 0] [RequestSize => 650.00 B] [Average => 72.50585514834
 iterations/s]
[0:00:06][351 iterations][60%] [74.03760369891869 iterations/
s] [warmUp => false] [Threads => 10] [ResponseSize => 19.24
 KiB] [failures => 0] [RequestSize => 650.00 B] [Average =>
 73.84493230828234 iterations/s]
[0:00:07][390 iterations][70%] [73.47836429196032 iterations/
s] [warmUp => false] [Threads => 10] [ResponseSize => 19.24
 KiB] [failures => 0] [RequestSize => 650.00 B] [Average =>
 73.86340270431613 iterations/s]
[0:00:08][426 iterations][80%] [73.44844565461368 iterations/
s] [warmUp => false] [Threads => 10] [ResponseSize => 19.24
 KiB] [failures => 0] [RequestSize => 650.00 B] [Average =>
 74.35713799467462 iterations/s]
[0:00:09][464 iterations][90%] [75.16546154258003 iterations/
s] [warmUp => false] [Threads => 10] [ResponseSize => 19.24
 KiB] [failures => 0] [RequestSize => 650.00 B] [Average =>
 73.90578951833827 iterations/s]
[0:00:10][499 iterations][100%] [75.45269086855991 iterations/
s] [warmUp => false] [Threads => 10] [ResponseSize => 19.24
 KiB] [failures => 0] [RequestSize => 650.00 B] [Average =>
 73.98895045521625 iterations/s]
```

WarmUpReporter

The reporter is able to determine when the tested system is warmed up. The warming is enabled/disabled by the presence of enabled WarmUpReporter in the scenario. It does not publish any results to destinations. The minimal iterations count and the warm-up period duration can be tweaked by the respective properties `minimalWarmUpCount` with the default value of 10,000 and `minimalWarmUpDuration` with the default value of 15,000 ms).

The reporter internally keeps track of current throughput - each second checks the number of processed iterations and computes the current throughput as the difference in number of iterations per checking period (second). It also remembers the current throughput from the previous checking period to calculate a difference in throughput. The throughput is considered NOT changing in time "much" when the relative difference in current throughput between the current checking period and the previous one is less than `relativeThreshold` value or the absolute difference in current throughput between the current checking period and the previous one is less than `absoluteThreshold` value.

Normally, the maximal length of the warm-up period is determined by the length of the performance test itself. This can be further limited (by using `maximalWarmUpDuration` and `maximalWarmUpType` properties) for the test not to waste time when the system under test cannot get warmed-up within a reasonable time frame.

The system is considered warmed up when all of the following conditions are satisfied: The current throughput is not changing much over the time, the minimal iterations count has been executed and the minimal duration from the very start has exceeded.

Property name	Description	Required	Default value
minimalWarmUpDuration	A minimal amount of time (in milliseconds) of the warm-up period.	No	15000
minimalWarmUpCount	A minimal number of iterations in the warm-up period.	No	10000
relativeThreshold	A relative difference threshold to determine whether the throughput is not changing much.	No	0.002
absoluteThreshold	An absolute difference threshold to determine whether the throughput is not changing much.	No	0.2
maximalWarmUpDuration	Maximal tolerance of waiting for the end of the warm-up period. If we run out of this time/percentage/iteration count (determined by maximalWarmUpType), we simply break the test and do not waste any more time. `-1` means that the check is disabled.	No	-1
maximalWarmUpType	The unit in which we measure the maximal warm-up count. Can be iteration, time, or percentage.	No	`iteration`

Table 4.42. WarmUpReporter properties

Example 4.56. An example of WarmUpReporter configuration

```
1    <reporter class="WarmUpReporter" enabled="true">
```

```
2        <property name="minimalWarmUpCount" value="1000"/>
3        <property name="minimalWarmUpDuration" value="10000"/>
4        <property name="relativeThreshold" value="0.005"/> <!-- 0.5%
-->
5        <property name="absoluteThreshold" value="0.5"/>
6     </reporter>
```

In the example above the system would be considered warmed up when at least 1000 iterations is processed AND the scenario is executed for at least 10 seconds AND, the relative change in the throughput is less then 0.5% or the absolute change in throughput is less than 0.5 iterations per second.

RawReporter

This is a very specific reporter for advanced users. `RawReporter` simply stores the complete test execution reporting data in a given file. The file can be later replayed with any provided scenario. The replay just reuses the reporting configuration in a scenario and emulates the test execution with the data recorded previously. `RawReporter` does not support any destinations to be added to it.

The results file is a compressed (gzip) serialization of `RunInfo` as a heaader and all `MesurementUnits` received by the reporter.

The following table describes configuration parameters of `RawReporter`:

Property name	Description	Required	Default value
outputFile	The file where the results will be recorded.	No	perfcake-measurement-${timestamp}.raw

Table 4.43. RawReporter properties

Example 4.57. An example of RawReporter configuration

```
1     <reporter class="RawReporter">
2        <property name="outputFile" value="results.raw" />
3     </reporter>
```

In the example above there is a `RawReporter` configured to report the results to results.raw file. There are no other configuration possibilities and no destinations can be specified.

4.7.2. Destinations

A destination is a representation of places where the measurements from the reporters are published. Each destination is configured to publish the results of reporter's measurements during the scenario execution periodically with a period specified by the `period` element in the scenario definition. Destination can have multiple periods but each destination has to have at least one period configured.

The following table shows the destination `period` options:

Destination `period` type	Value description
time	Time period in milliseconds
iteration	Number of iterations
percentage	The relative percentage of the scenario run

Table 4.44. Destination `period` options

Example 4.58. An example of the `period` configuration in a destination:

```
1    <destination class="...">
2       <period type="time" value="1000"/>
3       ...
4       (properties)
5       ...
6    </destination>
```

The following sections describe the destinations that can be used by reporters.

ChartDestination

Creates nice charts from the results using the C3.js [http://c3js.org/] library. The charts are quite powerful and we recommend reading this section thoroughly to fully discover their capabilities.

A user typically needs to specify the location where the chart(s) will be stored. This is set by the `outputDir` property. Next, a list of attributes that should be put in the chart is specified in the `attributes` property. This is a comma separated list of attributes that contains numbers. The charts cannot work with enumerations and text attributes. The names of the attributes can be seen in the console using *ConsoleDestination*. For example, in the listing below you can see the following attributes listed (not all of them need to be used in *ChartDestination*, you can select any subset suitable): *Threads*, *Minimum*, *Maximum*, *failures*, *Average*.

```
[0:00:08][97112 iterations][97%] [2.068165 ms] [warmUp => false]
 [Threads => 25] [Minimum => 2.004637 ms] [Maximum => 13.024963 ms]
 [failures => 0] [Average => 2.1204630348576172 ms]
```

Each chart also has some basic configuration properties specifying its name, the description of X and Y axeses, and the height of the chart image in pixels. The output is written as HTML files. The data files created are:

- `${outputDir}/data/${name}-${timeStamp}.*` - containing chart meta data in JSON format, chart data in a JavaScript file and a preview HTML file for each chart,
- `${outputDir}/index.html` - the final report generated at the end of a performance test,
- `${outputDir}/src/*` - HTML resources needed to render the charts even in the offline mode.

The destination can work in two modes. First is that all data are immediately written to the file system and you can find a preview file (see the list above) of the current state during the performance testing. Once you open the preview HTML file in the browser, it is automatically reloaded every 5 seconds. This mode can be only used when we specify precise names of the attributes to be recorded.

There is also an option to use prefix wildcard in the list of attributes. This is very useful when used with *ResponseTimeHistogramReporter* which reports attributes in the format `<some prefix><percentile value>` (e.g. `perc0.0100000`, `perc0.2500000`...). For this particular use case, we need to specify the attribute as `perc*`. We can also use just `*` to record all the available attributes. However, because we do not know all the attributes in advance, the preview is not available and no data are written to the file system until the performance test is completed.

One attribute cannot be replaced with a wildcard and this is `warmUp`. *ChartDestination* can be configured to completely ignore the warm-up period of the test when the `warmUp` attribute is not specified in the list of attributes. Or it records all attributes specified in the list during both warm-up and normal test phase. During the warm-up phase, the values are recorded into separate data series with the `_warmUp` suffix.

For example, for the attribute list `Result, Average, warmUp`, the following data series will be created (supposing the test has *WarmUpReporter* configured): *Result_warmUp*, *Average_warmUp*, *Result*, *Average*. If we set the attribute list to `*`, the following data series would be created: *Result*, *Average*, *Threads* (supposing there are no other attributes reported). If we wanted to create data series even for the warm-up phase, we would need to specify the attributes list as `*, warmUp`.

It is possible to keep recording new charts to the same directory location as was used for previous performance test runs. In this case, ChartDestination can automatically create combined charts comparing the same data series (according to their name) from all the charts recorded so far and having the same `group` property value. This behavior can be switched off by setting the `autoCombine` property to `false`.

Please use the charts with caution as the big number of results or charts recorded in the same report can take too long to load in the browser.

The following table describes the ChartDestination properties:

Property name	Description	Required	Default value
attributes	Attributes that will be stored in the chart. Each attribute is a result name of the reporter from which the results are published. Prefix wildcards (e.g. `perc*`) can be used. Using wildcards turns off chart previews. To record during the warm-up phase, the `warmUp` attribute needs to be specified explicitely.	Yes	-
name	Name of the chart for this measurement. There must not be two charts with the same name.	No	PerfCake Results
group	Group of this chart. Charts in the same group can be later matched for the column names. The group name can contain only alphanumeric characters and underscores and it is not allowed to begin with a number digit. If the group name does not follow the naming conventions, it would be converted to do so.	No	default
xAxis	X axis legend.	No	Time
yAxis	Y axis legend.	No	Iterations
type	The chart can have two visual types - a line chart or a bar chart. The bar chart is not recommended for reporting many values. It is more suitable for a few (or even a single) records (e.g. HDR his-	No	line

Property name	Description	Required	Default value
	togram). Possible values are `line` and `bar`.		
outputDir	A name of the directory where the charts are stored.	No	perfcake-chart
chartHeight	Height in pixels of each individual chart graphics in the HTML report. This is useful when the legend is too long.	No	400
autoCombine	Specifies whether the newly created chart should be automatically combined with the previously recorded data.	No	`true`

Table 4.45. ChartDestination properties

Example 4.59. An example of ChartDestination configuration

```
 1     <reporter class="MemoryUsageReporter">
 2        ...
 3        <destination class="ChartDestination">
 4           <period type="time" value="1000"/>
 5           <property name="name" value="Memory Usage"/>
 6
 <property name="group" value="${perfcake.scenario}_memory"/>
 7           <property name="yAxis" value="Memory Usage [MiB]"/>
 8           <property name="outputDir" value="${perfcake.scenario}-
charts"/>
 9           <property name="attributes" value="Used,Total"/>
10        </destination>
11     <reporter>
```

In the example above there is a `MemoryUsageReporter` configured to publish memory usage report into a chart. The memory usage data is gathered every single second and the chart is supposed to be showing used and total memory (results taken from `Used` and `Total` attributes of the`MemoryUsageReporter`. The resulting chart is shown in Figure 4.2, “ChartDestination example chart” .

Memory Usage (created: 3.5.2016 21:52:43)

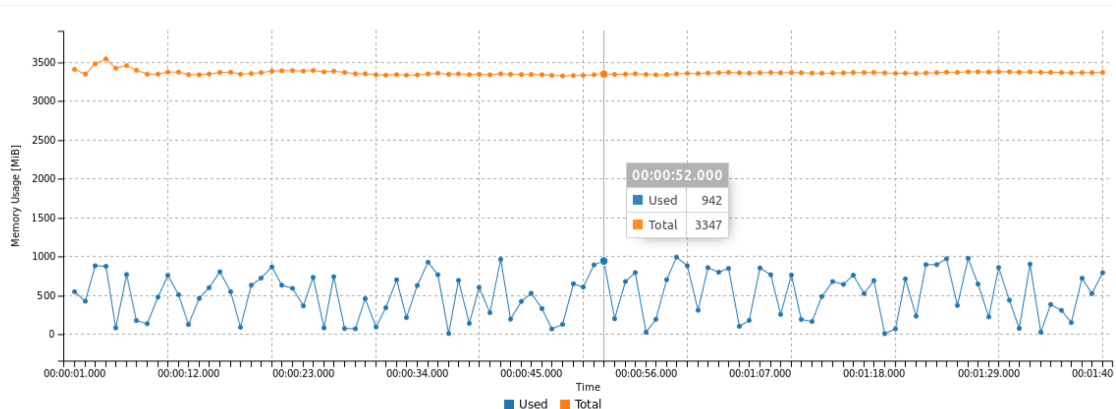

Figure 4.2. ChartDestination example chart

ConsoleDestination

A simple destination that appends the measurements to the PerfCake's console output.

Warning

Console output is not written into PerfCake log file and is lost once you close your terminal. If you want to keep the output, use the section called “Log4jDestination” or redirect PerfCake output to a file.

It is possible to setup a prefix to each line on the output to differentiate between several instances of this destination in a single performance test scenario. The `ConsoleDestination` can also send ANSI codes to change output color, however, this works only on certain operating systems and terminals. For example Microsoft Windows is known to support this feature since version 10.

The color codes are determined by the terminal configuration and can vary on different platforms.

The following table lists the typical colors, two values (normal and bright) each.

Code	0 and 8	1 and 9	2 and 10	3 and 11	4 and 12	5 and 13	6 and 14	7 and 15
Color	Black	Red	Green	Yellow	Blue	Magenta	Cyan	White

The following table lists the available properties of `ConsoleDestination`.

Property name	Description	Required	Default value
prefix	A prefix string that is written to each output line.	No	-
foreground	Output foreground color as a number in range 0 - 15. The real color depends on the terminal configuration. The range 8 - 15 means bold or bright version of the same colors as 0 - 7.	No	-
background	Output background color as a number in range 0 - 7. The real color depends on the terminal configuration. Background does not support bold/bright colors.	No	-

Table 4.46. ConsoleDestination properties

Example 4.60. An example of ConsoleDestination configuration

```
1    <destination class="ConsoleDestination">
2       <period type="time" value="1000"/>
3       <property name="prefix" value="===[Throughput]===>"/>
4       <property name="foreground" value="11"/>
5    </destination>
```

CsvDestination

This destination can be used to publish the measurements into a CSV file. Each result in the measurement is treated as a column in the file and the name of the result is used to name the column.

CsvDestination in its minimal configuration simply streams out all the measured data during the test execution and the CSV result file is immediately available. It always writes out *Time*, *Iteration* and *Result* attributes. These attributes cannot be requested in the scenario configuration and cannot be removed.

To get better idea on what attributes can be requested in your scenario configuration, use *ConsoleDestination* which always outputs all of them. You can then pick those that suite your needs.

By default, *CsvDestination* writes out all the attributes observed in the first measurement it receives. Changing the CSV result file to add a data column while the performance test is in progress would cause too much overhead. The file header once written remains unchanged. Attributes added later to the measurement are ignored. This is especially the case when it takes very long for the first result to arrive and the *CsvDestination* already reports to the CSV file. This would practically block any results from being reported.

To handle the situation, it is possible to specify wildcards in the form of <prefix>*. This is very useful when used with histogram reporters and we do not know the names of the attributes in advance. It is also possible to use just * as a wildcard. It is not possible to use the asterisk wildcard in the middle of an attribute name. The wildcard does not replace the warmUp attribute. However, such a configuration leads to storing all the results in the memory and writing the final CSV result file after the test is successfully finished. In case of a failure, no results are written.

If there are any attributes missing from the records, *CsvDestination* can either skip such a record, or it can fill in the missing values with null.How to handle the

The following table describes the configuration properties of CsvDestination.

Property name	**Description**	**Required**	**Default value**
path	A path to the output CSV file.	No	perfcake-results-${perfcake.run.timestamp}.csv [a]
delimiter	A CSV record delimiter.	No	;
appendStrategy	A strategy that is used in case, that the output file exists. overwrite means that the file is overwritten, rename means that the current output file is renamed by adding a number-based suffix and append is for appending new results to the original file.	No	rename
expectedAtrributes	A comma separated list of attributes to be recorded in the CSV result file. This is useful when the attributes are not present in every measurement or it takes too long for the first measurement to arrive.	No	Empty string. The attributes reported by default are *Time*, *Iteration*, *Result* and all others observed in the first measurement (supposing it arrives sooner than the *CsvDestination* reports for the first time).

Property name	Description	Required	Default value
missingStrategy	Specifies what to do in case of a missing attribute value. It can either replace it with `null` when the strategy is set to `null`, or it can completely skip such a record when set to `skip`.	No	null
linePrefix	The prefix prepended to all lines in the CSV result file. This can facilitate creating JSON like records for example.	No	(empty string)
lineSuffix	The suffix appended to all lines in the CSV result file. This can facilitate creating JSON like records for example.	No	(empty string)
lineBreak	Line separator used to add new entry to the CSV result file.	No	\n (new line character)
skipHeader	Skips writing the header in the CSV result file if set to `true`.	No	`false`

[a] See Table 2.2, “Available PerfCake internal properties”

Table 4.47. CsvDestination properties

Example 4.61. An example of CsvDestination configuration

```
1    <destination class="CsvDestination">
2       <period type="time" value="1000"/>
3       <property name="path" value="${perfcake.scenario}-
output.csv"/>
4       <property name="appendStrategy" value="overwrite"/>
5    </destination>
```

Example 4.62. Sample output of CsvDestination

```
Time;Iterations;Result;warmUp;Threads;ResponseSize;Minimum;Maximum;failures;Reques
0:00:00;1;1.306239;false;10;0;1.306239;1.306239;0;0;1.306239
0:00:00;1000;1.069117;false;10;0;1.014015;10.468919;0;0;1.1494209689999997
0:00:00;2000;1.059257;false;10;0;1.012308;10.468919;0;0;1.1295661935000005
0:00:00;3000;1.069961;false;10;0;1.012308;10.468919;0;0;1.1466806060000005
```

ElasticsearchDestination

This destination stores results in the Elasticsearch database. The reported data have information about the test progress (time in milliseconds since start, percentage and iteration), real time of each result, and the complete results map. Quantities are stored without their unit for them to be parseable as numbers.

To properly search through the data, we need to set the mapping (to be able to interpret time as time). However, this needs to be done just once for each index and type. Another attempts to set the mapping lead to an error from the server (this does not break the test execution) because Elasticsearch cannot change existing mapping.

Property name	Description	Required	Default value
serverUrl	Comma separated list of Elasticsearch servers including protocol and port number. Port is typically 9292.	Yes	
index	Elasticsearch index name.	No	perfcake
type	Elasticsearch type name.	No	results
tags	Comma separated list of tags to be added to results. This is useful to differentiate results from multiple test runs for example.	No	
userName	Elasticsearch user name. Authentication is only used when the `userName` property is specified.	No	
password	Elasticsearch password.	No	
timeout	Elasticsearch client timeout in milliseconds. When getting too many missed records in the log, try increasing this value.	No	3000
configureMapping	True when the mapping should be configured prior to writing any data. This needs to be done only once for each index and type.	No	`true`
keyStore	Enables a SSL connection to the server. Sets the location of the key store created with Java `keytool`. The default loaction is specified by the system property `perfcake.keystores.dir` which defaults to `resources/keystores`.	No	
keyStorePassword	Password to the key store.	No	
trustStore	See keyStore property. The only difference is that this is for the trust store.	No	
trustStorePassword	Password to the trust store.	No	

Table 4.48. ElasticsearchDestination properties

Example 4.63. An example of a ElasticsearchDestination configuration

```
1    <destination class="ElasticsearchDestination">
2       <period type="iteration" value="500"/>
3       <property name="serverUri" value="http://localhost:9292" />
4       <property name="index" value="perfcake" />
5       <property name="tags" value="tag1,tag2" />
6       <property name="timeout" value="5000" />
7    </destination>
```

InfluxDbDestination

This destination stores results in the InfluxDb database. The reported data have information about the test progress (time in milliseconds since start, percentage and iteration), real time of each result, and the complete results map. Quantities are stored without their unit for them to be parseable as numbers. It supports SSL connection and the database is by default created on connection.

Property name	Description	Required	Default value
serverUri	InfluxDb server including protocol and port number. Supports SSL. Port is typically `8086`.	Yes	
database	InfluxDb database.	No	perfcake
measurement	InfluxDb measurement (serves as a database table).	No	results
tags	Comma separated list of tags to be added to results. This is useful to differentiate results from multiple test runs for example.	No	
userName	InfluxDb user name. There always need to be some user. InfluxDb does not support empty field, however, you can configure your server to let anybody in.	Yes	admin
password	InfluxDb password. There always need to be some password. InfluxDb does not support empty field, however, you can configure your server to let anybody in.	Yes	admin
createDatabase	True when the database should be created on connection. If the database already exists, nothing happens (all data and tables remain there).	No	`true`
keyStore	Enables a SSL connection to the server. Sets the lo-	No	

Property name	Description	Required	Default value
	cation of the key store created with Java `keytool`. The default loaction is specified by the system property `perfcake.keystores.dir` which defaults to `resources/keystores`.		
keyStorePassword	Password to the key store.	No	
trustStore	See keyStore property. The only difference is that this is for the trust store.	No	
trustStorePassword	Password to the trust store.	No	

Table 4.49. InfluxDbDestination properties

Example 4.64. An example of a InfluxDbDestination configuration

```
1    <destination class="InfluxDbDestination">
2       <period type="iteration" value="500"/>
3       <property name="serverUri" value="http://localhost:8086" />
4       <property name="database" value="perfcake" />
5       <property name="tags" value="tag1,tag2" />
6       <property name="userName" value="admin" />
7       <property name="password" value="abc123" />
8    </destination>
```

Log4jDestination

The destination appends the measurements to Log4j to category `prg.perfcake.reporting.destination.Log4jDestination`. The appropriate configurations to customize its output should be done. You can configure a separate appender only for this category for instance. Logging level can be set through the `level` property.

The following table describes the Log4jDestination's properties

Property name	Description	Required	Default value
level	The logging level for the destination.	No	INFO

Table 4.50. Log4jDestination properties

Example 4.65. An example of Log4jDestination configuration

```
1    <destination class="Log4jDestination">
2       <period type="time" value="1000"/>
3       <property name="level" value="INFO"/>
4    </destination>
```

4.8. Validation

Validation can be used to check if the response received by a sender is valid. It is optional and if it is used, it is performed in a gentle way (only once in a 0.5 sec) not to add any significant overhead and not to affect the measuring.

The validation is configured in a scenario in two places. First there is a kind of validator pool where all validators available in the scenario are placed. The validator pool is configured in the `validation` section of the scenario. Each validator has a unique ID by which it can be referenced. Once at least one of the validators is configured, it can be referenced by a `message` 's sub-element `validatorRef`, which is the second place the validation is configured. By adding a validator refecence to the messages we tell PerfCake that the response to that particular messages should be validated by that particular validator. Any message can have multiple validator references and each validator must be passed in order to set the response valid. Any validator can be referenced from more than one message.

The validation as a whole can be enabled or disabled by setting the optional boolean attribute called `enabled` on the `validation` element to the value of `true`. By default the validation is disabled.

The validation thread has some sleep for to wait between validations not to influence the measured results. At the end, when there is nothing else to do, it goes through the remaining responses faster. This behavior can be controlled via the optional `fastForward` attribute on the `validation` element. If the value is `true` the sleep periods are ignored and the validation goes as fast as possible. However, that influences the measured results. For that reason the default value is `false`.

Example 4.66. An example of validation configuration:

```
1     <messages>
2        ...
3        <message uri="...">
4           <validatorRef id="validator1"/>
5           ...
6        </message>
7        <message uri="...">
8           <validatorRef id="validator1"/>
9           <validatorRef id="validator2"/>
10          ...
11       </message>
12       ...
13    </messages>
14    ...
15    <validation fastForward="false" enabled="true">
16       <validator id="validator1" class="...">
17          ...
18       </validator>
19       <validator id="validator2" class="...">
20          ...
21       </validator>
22    </validation>
```

In the example above there are two validators (`validator1` and `validator2`) and two messages configured. The first message has just the `validator1` attached so the response message received to the first message will be validated just by the `validator1`.

On the other hand there is a second message that has both validators attached. So the response to the second message will be validated by both of the validators.

The validation is enabled and the fast validation mode will be activated once the performance test finishes.

When specifying the validator class, unless you enter a fully classified class name, the default package `org.perfcake.validation` is assumed.

The rest of the chapter will present all the available validators that can be used in PerfCake.

4.8.1. Validators

DictionaryValidator

Dictionary validator can create a dictionary of valid responses and use this to validate them in another run. It is also possible to create the dictionary manually, however, this is too complicated task and we always recommend running the validation in record mode first. Any manual changes can be done later.

The validator creates an index file and a separate file for each response. A writable directory must be specified using the `dictionaryDirectory` property. The default index file name can be redefined. The response file names are based on hash codes of the original messages. Empty, null or equal messages will overwrite the file but this is not the intended use of this validator. Index file is never overwritten, if you really insist on recreating it, please rename or delete the file manually (this is for safety reasons). It is not sufficient to store just the index as it is likely that the correct messages will be manually modified after they are recorded.

The idea of usage is to first activate the record mode, run the whole test, switch the record mode off and then reuse the created `dictionaryDirectory` with future test runs.

Following table shows the properties of the DictionaryValidator:

Property name	Description	Required	Default value
dictionaryDirectory	The directory where the dictionary is/will be store.	Yes	-
record	Activates the record mode.	No	`false`

Table 4.51. DictionaryValidator properties

Example 4.67. An example of DictionaryValidator configuration

```
1     <validator id="rimmerValidator" class="DictionaryValidator">
2        <property name="dictionaryDirectory" value="/home/rimmer/
work/tests" />
3        <property name="record" value="true" />
4     </validator>
```

PrintingValidator

Just prints out the original message and its response for the validation by human eyes. Please note that the validation output is normally stored in a separate log file and not printed to all other logging output. All messages controlled by this validator are stated valid.

There are no configuration properties available to this validator.

Example 4.68. An example of ScriptValidator configuration

```
1     <validator id="print" class="PrintingValidator" />
```

RegExpValidator

The text validator treats each rule line as a regular expression and checks if the response message matches that regular expression. The response message must match all lines in order to be successfully validated.

Note

Please note that the value of pattern is passed through a template engine. The benefit of this is the ability to use standard properties in the pattern like `${prop1}` and `@{prop2}`. The disadvantage is that each backslah character used to escape an RegExp character needs to be escaped as well. In reality this means that instead of writing `\.` to escape the dot character, one need to enter `\\.`.

Example 4.69. An example of RegExpValidator configuration

```
1     <validator id="rimmerValidator" class="RegExpValidator">
2        <property name="pattern">
3           <text><![CDATA[.*"I'm a [Ff]ish"!\\.*]]></text>
4        </property>
5     </validator>
```

RulesValidator

This validator uses Drools [12] engine to assert the validator rules. The rules for this validator have their own DSL [13] in a form of human readable sentences. The rules are listed in a following listing:

Example 4.70. RulesValidator rules

```
1 Message body contains "{string}".
2 Message body equals "{string}".
```

Example 4.71. An example of RulesValidator configuration

```
1     <validator id="fareWellValidator" class="RulesValidator">
2        <property name="pattern">
3           <text><![CDATA[Message body contains "Farewell
World!".]]></text>
4        </property>
5     </validator>
```

ScriptValidator

Validates messages using Java Script Engine and the provided script. The script engine must be installed in the extensions directory. The original message is passed to the script in the `originalMessage` property, the response is inserted as `message`, the attributes used for the particular message instance are passed as `attributes`, and logger is passed as `log`, all using script bindings. The script return value is evaluated for validation success (`true` for passed, `false` for failed).

Following table shows the properties of the ScriptValidator:

[12] http://www.jboss.org/drools/
[13] Domain Specific Language

Property name	Description	Required	Default value
engine	The name of the Java Scripting Engine (please do not confuse with JavaScript).	Yes	-
script	The code of the validation script. Takes priority over `scriptFile`.	Either `script` or `scriptFile` must be present	-
scriptFile	The location of the validation script file.	Either `script` or `scriptFile` must be present	-

Table 4.52. ScriptValidator properties

Example 4.72. An example of ScriptValidator configuration

```
1    <validator id="groovyValidator" class="ScriptValidator">
2       <property name="engine" value="groovy" />
3       <property name="script">
4          <![CDATA[
5             log.info("Be groovy")
6             return message.payload.toString().contains('Pepa')
7          ]]>
8       </property>
9    </validator>
```

Chapter 5. Result repository

Our community developed a special PerfRepo application that serves as a repository for storing performance test results. We will be integrating PerfRepo into our organization at GitHub and providing instructions on configuring ans using it.

Chapter 6. Extending PerfCake

PerfCake has the extension mechanism, that allows user to add specific client libraries or a new functionality. The following sections describes those possibilities.

6.1. Client libraries

Some of the PerfCake's components need additional stuff to work properly. For example:

- JMS based senders need JMS provider's specific client jar files,
- JDBC based sender needs database specific JDBC driver.

To add the necessary client libraries and the dependencies, place it all under the `$PERFCAKE_HOME/lib/ext` directory.

6.2. Custom components - Plugins

PerfCake has the ability to extend its functionality by adding plugins - new custom components such as senders, generators, reporters, destinations or validators. [1]

To add a new component to PerfCake, all you need to do is to take the jar file with the packaged component (along with all its dependencies) and place them in the `$PERFCAKE_HOME/lib/plugins` directory.

[1] To get the details about creating custom components (plugins), see the PerfCake Developers' Guide.

Chapter 7. Troubleshooting PerfCake

In this chapter we will try to address common pitfalls users can hit with PerfCake.

7.1. Running in Virtual Environment

In a virtual environment, the host might have some system level tweaks to optimize its performance. Among others, some of the operating system real timer ticks or PIT/HPET timers might be skipped. This can lead in a poor PerfCake time measurement resolution. Pay close attention to the time benchmark executed during PerfCake startup. This reveals the maximal performance PerfCake is able to measure. All faster systems will look like having an infinite performance to PerfCake. PerfCake will also start displaying warning when such a speed is achieved and the test results will be flawed.

7.2. Too many open HTTP connections

When using *HttpSender*, there are plenty of short-lived connections. When your test uses a lot of threads, you migth quickly run into the limits of the underlying system. You can either increase the number of connections, or on Linux system, you can make sure that the connections are reused like follows:

```
$ echo "1" >/proc/sys/net/ipv4/tcp_tw_reuse
$ echo "1" >/proc/sys/net/ipv4/tcp_tw_recycle
```

7.3. "java.net.BindException: Address already in use: connect” issue on Windows

Windows will only make outbound TCP/IP connections using ports 1024-5000 by default, and takes up to 4 minutes to recycle them. So in the case of runnung a load type performance tests the port range is filled pretty fast.

To allow higher values for ports on Windows add this parameter to the system registers:

1. Start Registry Editor
2. Find the `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters` subkey in the registry.
3. Right-clicking on the *Parameters* create a new *DWORD* value with the name of `MaxUserPort`.
4. Right-clicking on the *MaxUserPort* edit the value to `65534` as *decimal*.
5. Reboot the Windows.

Chapter 8. Changelog

All the changes for the releases are tracked an recorded at https://github.com/PerfCake/PerfCake/blob/devel/CHANGES.md.