PerfCake 5.x

User Guide

Pavel Macík
Martin Večeřa

PerfCake 5.x: User Guide

by Pavel Macík and Martin Večeřa

Table of Contents

List of Figures

List of Tables

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

Some components also work only with an active Internet connection. For example chart output is based on Google Charts and they do not work offline. We are considering to provide a different solution and make PerfCake work completely offline in the future.

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).

In case of building from 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 18:35:51,709 INFO  {org.perfcake.scenario.ScenarioBuilder}
 Scenario configuration: file:~/perfcake-5.0/resources/scenarios/
http.xml
2015-09-22 18:35:52,174 INFO  {org.perfcake.ScenarioExecution} ===
 Welcome to PerfCake 5.0 ===
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.
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 2.2.1, “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).

```xml
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:5.0">
  <run type="${perfcake.run.type:time}"
       value="${perfcake.run.duration:30000}"/>
  <generator class="DefaultMessageGenerator"
       threads="${perfcake.thread.count:100}"/>
  <sender class="HttpSender">
     <target>http://${server.host}/post</target>
     <property name="method" value="POST"/>
  </sender>
  <reporting>
     <reporter class="IterationsPerSecondReporter">
        <destination class="ConsoleDestination">
           <period type="time" value="1000" />
        </destination>
     </reporter>
  </reporting>
  <messages>
     <message uri="plain.txt"/>
  </messages>
</scenario>
```

As you can see, the simplest scenario runs for 30000ms = 3 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-5.0-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
 usage: ScenarioExecution -s <SCENARIO> [-sd <SCENARIOS_DIR>] [-md
 <MESSAGES_DIR>] [-D<property=value>]*
 -D <property=value> system properties
 -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
 -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 community [2] to welcome you in our commmunity.

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:`

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

`http-echo.xml:`

```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:5.0">
   <run type="time" value="10000"/>
```

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

```
  <generator class="DefaultMessageGenerator"
             threads="${thread.count}"/>
  <sender class="HttpSender">
     <target>http://your-app.com/cool-app</target>
  </sender>
</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:`

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

`http-reporting.xml:`

```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:5.0">
   <run type="time" value="10000"/>
   <generator class="DefaultMessageGenerator"
              threads="${thread.count:10}"/>
   <sender class="HttpSender">
      <target>http://your-app.com/cool-app</target>
      <property name="method" value="GET"/>
   </sender>
   <reporting>
      <reporter class="ResponseTimeStatsReporter">
         <destination class="ChartDestination">
            <period type="time" value="1000"/>
            <property name="name" value="Response Time"/>
            <property name="group" value="rt"/>
            <property name="yAxis" value="Response Time [ms]"/>
            <property name="attributes" value="Result,Average"/>
         </destination>
         <destination class="ConsoleDestination">
            <period type="time" value="1000"/>
         </destination>
      </reporter>
   </reporting>
</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:

```
2015-09-23 13:30:11,310 INFO {org.perfcake.ScenarioExecution} ===
 Welcome to PerfCake 5.0 ===
2015-09-23 13:30:11,311 INFO {org.perfcake.util.TimerBenchmark}
 Benchmarking system timer resolution...
2015-09-23 13:30:11,312 INFO {org.perfcake.util.TimerBenchmark} This
 system is able to differentiate up to 605ns.
A single thread is now able to measure maximum of 1652892 iterations/
second.
2015-09-23 13:30:11,331 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Starting to
 generate...
2015-09-23 13:30:11,390 INFO {httl} Using slf4j logger for httl.,
 httl: 1.0.11, jvm: 1.8.0_31, os: Linux 3.14.27-100.fc19.x86_64 amd64
[0:00:02][1823 iterations][29%] [3.152025 ms] [Threads => 10] [Average
 => 6.367008149088894 ms] [warmUp => false]
[0:00:03][5060 iterations][39%] [2.698051 ms] [Threads => 10] [Average
 => 4.245464461477511 ms] [warmUp => false]
[0:00:04][9423 iterations][49%] [1.940839 ms] [Threads => 10] [Average
 => 3.323919806458463 ms] [warmUp => false]
[0:00:05][14684 iterations][59%] [1.555495 ms] [Threads => 10]
 [Average => 2.805643889532502 ms] [warmUp => false]
[0:00:07][23586 iterations][74%] [1.401873 ms] [Threads => 10]
 [Average => 2.373560429316712 ms] [warmUp => false]
[0:00:08][32674 iterations][89%] [3.777534 ms] [Threads => 10]
 [Average => 2.1659369298922284 ms] [warmUp => false]
[0:00:09][38948 iterations][99%] [1.776092 ms] [Threads => 10]
 [Average => 2.0701102116441534 ms] [warmUp => false]
2015-09-23 13:30:21,332 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Reached test
 end.
[0:00:10][39293 iterations][100%] [1.888512 ms] [Threads => 10]
 [Average => 2.067882293274619 ms] [warmUp => false]
```

```
2015-09-23 13:30:21,332 INFO {org.perfcake.reporting.ReportManager}
 Reporting final results:
2015-09-23 13:30:21,677 INFO
 {org.perfcake.message.generator.DefaultMessageGenerator} Shutting
 down execution...
2015-09-23 13:30:21,678 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.

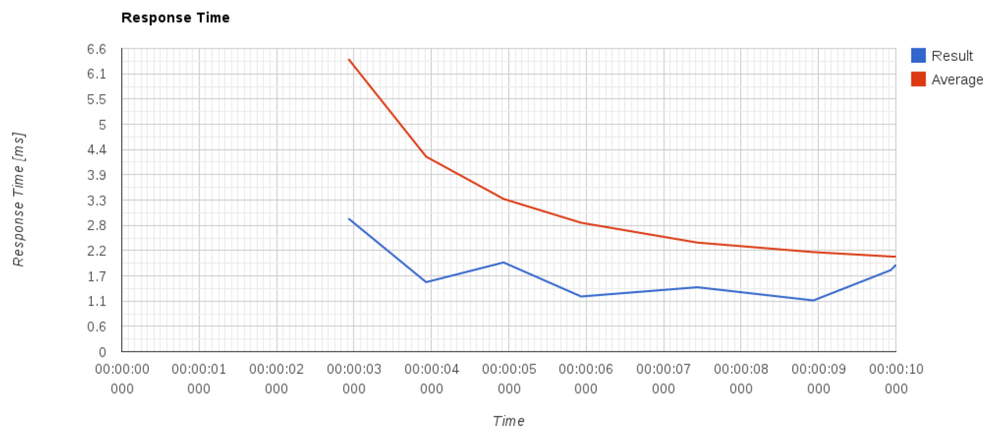

Figure 1.1. RampUpDownGenerator time chart

Chapter 2. PerfCake Features

2.1. 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.

2.1.1. XML scenario

As you can see from the following listing the XML scenario is defined by `urn:perfcake:scenario:5.0` namespace [1] . The scenario is divided into several sections: Properties, Run, Generator, Sender, Reporting, Messages and Validation.

Scenario structure

```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:5.0">
   <!-- Scenario properties (optional) -->
   <properties>
      <property name="..." value="..."/>
      ...
   </properties>

   <!-- Run section (required) -->
   <run ... >
      ...
   </run>

   <!-- Generator section (required) -->
   <generator ... >
      ...
   </generator>

   <!-- Sender section (required) -->
   <sender ... >
      ...
   </sender>

   <!-- Reporting section (optional) -->
   <reporting>
      ...
   </reporting>

   <!-- Messages section (optional) -->
   <messages>
      ...
```

[1] Schema can be found at http://schema.perfcake.org/perfcake-scenario-5.0.xsd

```
   </messages>

   <!-- Validation section (optional) -->
   <validation>
      ...
   </validation>
</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 2.1.4, “Filtering properties” for more details.).

Run

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

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

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” .

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 4.4, “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 4.3, “What - Messages” describes the messages in more details.

Validation

Validation module allows to validate the response messages.

The validation capabilities are described in Section 4.5, “Validation” .

2.1.2. 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.

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

2.1.3. 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.

2.1.4. Filtering properties

It is possible to use property placeholders in scenarios (and in messages too, 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.

The properties are loaded from system properties and environment properties (in this order). To limit just to one of these sources, one can specify a namespace prefix *env.* for environment properties and *props.* for system properties.

A placeholder has the following format:

```
'${' + <property name> + (':' + <default value>) + '}'
```

An example of a scenario with a property placeholder without the default value

```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:
${perfcake.scenario.version}">
   ...
</scenario>
```

An example of a scenario with a property placeholder with the default value of "5.0"

```
<?xml version="1.0" encoding="utf-8"?>
<scenario xmlns="urn:perfcake:scenario:${perfcake.version:5.0}">
   ...
</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 simmple string replacement for scenario files.

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.

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

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

Another way is to configure an environment property in the operating system. In case the same property exists both as a system property and as an environment property, the system property takes precedence.

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
perfcake.messages.dir	Messages directory
perfcake.plugins.dir	Plugins directory
perfcake.properties.file	Custom properties file
perfcake.run.timestamp	A timestamp of a moment when the scenario execution started
perfcake.scenario	A name of the scenario
perfcake.scenarios.dir	Scenarios directory

Table 2.1. Available PerfCake internal properties

2.2. Configuration

2.2.1. Command line parameters

PerfCake is a command line tool so it provides a CLI for executing a PerfCake scenario (see Section 2.1, “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
usage: ScenarioExecution -s <SCENARIO> [-sd <SCENARIOS_DIR>]
                        [-md <MESSAGES_DIR>] [-D<property=value>]*
 -D <property=value>                          system properties
 -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
 -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 4.3, “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 2.1.4, “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	Default PerfCake logging level.	No	info
-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
-D<property>=<value>	Sets a single system property with a given name and value. Multiple property arguments are allowed.	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

Table 2.2. PerfCake CLI arguments

Some interesting 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.	-

Property name	Description	Default value
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, PerfCake can also run in JRE (the templating is the only part of PerfCake depending on JDK).	false

Table 2.3. PerfCake recognized properties

2.2.2. 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 2.2.1, “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. Handling issues during excution

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 isused 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 Exit codes.

2.4. 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]

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

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.
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.

Table 2.4. PerfCake Exit Codes

2.5. Migrating scenarios to latest version

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

2.5.1. 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`

2.5.2. 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 2.5. 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

```
<reporter class="AverageThroughputReporter">
  ...
  (destinations)
  ...
</reporter>
```

is replaced by

```
<reporter class="ThroughputStatsReporter">
  <property name="minimumEnabled" value="false"/>
  <property name="maximumEnabled" value="false"/>
  ...
  (destinations)
  ...
</reporter>
```

ResponseTimeReporter -> ResponseTimeStatsReporter

```
<reporter class="ResponseTimeReporter">
  ...
  (destinations)
  ...
</reporter>
```

is replaced by

```
<reporter class="ResponseTimeStatsReporter">
  <property name="minimumEnabled" value="false"/>
  <property name="maximumEnabled" value="false"/>
  ...
  (destinations)
  ...
</reporter>
```

WindowResponseTimeReporter -> ResponseTimeStatsReporter

```
<reporter class="WindowResponseTimeReporter">
  <property name="windowSize" value="10"/>
  ...
  (destinations)
  ...
</reporter>
```

is replaced by

```
<reporter class="ResponseTimeStatsReporter">
  <property name="windowSize" value="10"/>
  <property name="minimumEnabled" value="false"/>
  <property name="maximumEnabled" value="false"/>
  ...
  (destinations)
  ...
</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

```
<sender class="RequestResponseJMSSender">
  ...
  <property name="connectionFactory"
            value="${jms.connection.factory}"/>

  <property name="username" value="${username}"/>
  <property name="password" value="${password}"/>
  <!-- JNDI -->
  <property name="jndiContextFactory"
            value="${jndi.ctx.factory}"/>

  <property name="jndiUrl" value="${jndi.url}"/>

  <property name="jndiSecurityPrincipal"
            value="${jndi.username}"/>

  <property name="jndiSecurityCredentials"
            value="${jndi.password}"/>
  ...
</sender>
```

is replaced by

```
<sender class="RequestResponseJmsSender">
  ...
  <property name="connectionFactory"
            value="${jms.connection.factory}"/>

  <property name="username" value="${username}"/>
  <property name="password" value="${password}"/>
  <property name="responseUsername" value="${username}"/>
  <property name="responsePassword" value="${password}"/>
  <!-- JNDI -->
  <property name="jndiContextFactory"
            value="${jndi.ctx.factory}"/>

  <property name="jndiUrl" value="${jndi.url}"/>

  <property name="jndiSecurityPrincipal"
            value="${jndi.username}"/>

  <property name="jndiSecurityCredentials"
            value="${jndi.password}"/>

  <property name="responseJndiSecurityPrincipal"
            value="${jndi.username}"/>

  <property name="responseJndiSecurityCredentials"
            value="${jndi.password}"/>
  ...
</sender>
```

Scenario conversion using XSLT

After understanding the 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`.

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

2.5.3. 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.

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.

2.5.4. 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.

```
<generator class="...">
  <run type="..." value="..."/>
  ...
  (properties)
  ...
</reporter>
```

is replaced by

```
<run type="..." value="..."/>
<generator class="...">
  ...
  (properties)
  ...
```

```
</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.

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

is replaced by

```
<sender class="...">
  <target>...</target>
  ...
  (properties)
  ...
</sender>
```

Chapter 3. General Usage

3.1. Determining application sweet spot

(TODO: fill in)

3.2. Sending messages

(TODO: fill in) multiple messages, http headers, messages in files or in scenario

3.3. Application warm-up

(TODO: fill in) what happens if the application cannot get warmed

3.4. Monitoring memory

(TODO: fill in) heap dump, gc before reporting, memory leak detection, agent protocol

3.5. Charts output

(TODO: fill in)

3.6. Response validation

(TODO: fill in)

3.7. Message templates and sequences

(TODO: fill in)

3.8. Basics of measurement

(TODO: fill in) response time vs throughput - what are the differences in numbers and the role of threads

3.9. Reporting results

(TODO: fill in) csv, log4j

3.10. Result aggregation

(TODO: fill in) accumulators, time based windows

3.11. Slowly adding clients/threads

(TODO: fill in) rampup/down generator

3.12. Providing JDBC or JMS adapter

(TODO: fill in)

3.13. Using PerfCake API

(TODO: fill in)

3.14. Adding custom plugin - Selenium

(TODO: fill in)

3.15. Universal and simple Groovy sender

(TODO: fill in)

3.16. Importing CSV data to LibreOffice or Microsoft Office

(TODO: fill in)

3.17. Properties to enable/disable destinations

(TODO: fill in) how commandline properties can disable destinations and some other elements in xml

3.18. Keep open connection

(TODO: fill in)

3.19. Using SSL with HTTP

(TODO: fill in)

3.20. Testing with DummySender

(TODO: fill in)

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 4.3, “What - Messages”) and after that it returns the sender back to the sender pool. The reporting module (See Section 4.4, “Reporting”) treats this as a single iteration.

An example of the `run` configuration in a scenario:

```
<run type="time" value="10000"/>
<generator class="..." threads="...">
   ...
   (properties)
   ...
</generator>
```

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

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

4.1.2. 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.

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

An example of DefaultMessageGenerator configuration

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

4.1.3. ConstantSpeedMessageGenerator

This generator is based on `DefaultMessageGenerator` (SeeSection 4.1.2, “DefaultMessageGenerator”). 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.	No	5000

Table 4.3. ConstantSpeedMessageGenerator properties

An example of ConstantSpeedMessageGenerator configuration

```
<run type="time" value="60000"/>
<generator class="ConstantSpeedMessageGenerator" threads="100">
   <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.

4.1.4. RampUpDownGenerator

The generator is based on the `DefaultMessageGenerator` (SeeSection 4.1.2, “DefaultMessageGenerator”) but 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” .

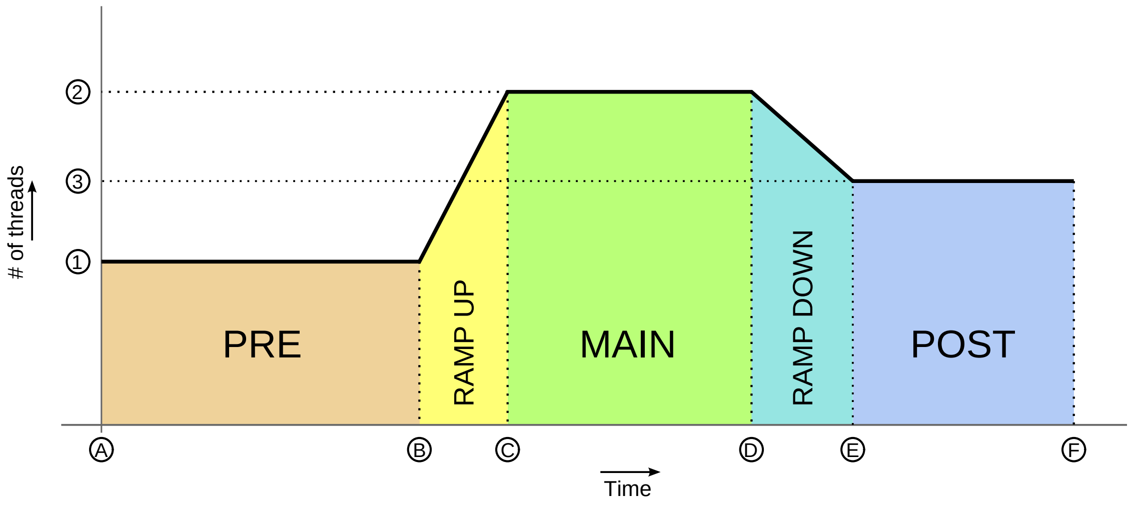

Figure 4.1. RampUpDownGenerator time chart

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

Property name	Description	Required	Default value
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 4.4. RampUpDownGenerator properties

An example of RampUpDownGenerator configuration

```
<run type="time" value="60000"/>
<generator class="RampUpDownGenerator" threads="20">
  <property name="preThreadCount" value="10"/>
  <property name="preDuration" value="10000"/>
  <property name="rampUpStep" value="2"/>
  <property name="rampUpStepPeriod" value="1000"/>
  <property name="mainThreadCount" value="20"/>
  <property name="mainDuration" value="20000"/>
  <property name="rampDownStep" value="1"/>
  <property name="rampDownStepPeriod" value="1000"/>
  <property name="postThreadCount" value="15"/>
</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.

4.2.1. An example of a Sender configuration

```
<sender class="...">
  <target>...</target>
  ...
  sender properties
  ...
</sender>
```

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

4.2.2. 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. The supported values are `stdin` or `arguments`	No	"stdin"

Table 4.5. CommandSender properties

An example of CommandSender configuration

```
<sender class="CommandSender">
  <target>/tmp/script.sh</target>
</sender>
...
<messages>
  <message>
     <property name="GREETINGS" value="Be well!"/>
  </message>
</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!`".

4.2.3. 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 4.6. DummySender properties

An example of DummySender configuration

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

4.2.4. 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

Property name	Description	Required	Default value
classpath	Classpath for groovy executable - specifies where to find the class or groovy files.	No	-

Table 4.7. GroovySender properties

An example of GroovySender configuration

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

4.2.5. 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 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"

[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 4.8. HttpSender properties

An example of HttpSender configuration

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

4.2.6. HttpsSender

HttpsSender is similar to the HttpSender (see Section 4.2.5, “HttpSender”), thus it shares 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 4.9. HttpsSender properties

An example of HttpsSender configuration

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

4.2.7. ChannelDatagramSender

Sends messages through NIO [1] `DatagramChannel` [2].

The sender's target is an URL of a datagram 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"
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 4.10. ChannelDatagramSender properties

4.2.8. ChannelFileSender

Sends messages through NIO [1] `FileChannel` [3].

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

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 4.11. ChannelFileSender properties

4.2.9. ChannelSocketSender

Sends messages through NIO [1]> `SocketChannel` [4] .

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 4.12. ChannelSocketSender properties

4.2.10. 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 4.13. JdbcSender properties

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

An example of JdbcSender configuration

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

4.2.11. JmsSender

The JmsSender 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.

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	-
jndiSecurityPrincipal	A JNDI username	Yes	
jndiSecurityCredentials	A JNDI password	Yes	
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 trans-	No	""

Property name	Description	Required	Default value
	port will be performed unsecured.		

Table 4.14. JmsSender properties

An example of JmsSender configuration

```xml
<sender class="JmsSender">
  <target>jms/queue/YourQueue</target>
  <property name="receivingTimeout" value="30000"/>
  <property name="receiveAttempts" value="1"/>
  <property name="connectionFactory" value="jms/ConnFactory"/>
  <property name="username" value="KingRoland"/>
  <property name="password" value="12345"/>
  <!-- JNDI properties-->
  <property name="jndiContextFactory" value="pkg.InitCtxFactory"/
>
  <property name="jndiUrl" value="remote://${server.host}:4447"/>
  <property name="jndiSecurityPrincipal" value="KingRoland"/>
  <property name="jndiSecurityCredentials" value="12345"/>
</sender>
```

4.2.12. LdapSender

The sender is able to query an LDAP server [5] .

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 4.15. HttpSender properties

An example of LdapSender configuration

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

[5] http://en.wikipedia.org/wiki/Lightweight_Directory_Access_Protocol

```
</sender>
```

4.2.13. MqttSender

The sender is capable to send MQTT [6] 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 message delivery from the sender to the remote broker. Supported values are `EXACTLY_ONCE`, `AT_LEAST_ONCE` and `AT_MOST_ONCE`.	No	`EXACTLY_ONCE`
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.

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

Table 4.16. MqttSender properties

An example of MqttSender configuration

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

[6] http://mqtt.org/

4.2.14. 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>"

An example of PlainSocketSender configuration

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

4.2.15. RequestResponseJmsSender

The RequestResponseJmsSender 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 JmsSender (see Section 4.2.11, “JmsSender”) so it inherits its properties. The following table contains additional properties of the `RequestResponseJmsSender`:

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	-
receivingTimeout	A time duration in ms of specifying how long the sender will wait to receive the response message.	No	"1000"

Property name	Description	Required	Default value
receiveAttempts	A Number of attempts to receive the message.	No	"5"

Table 4.17. RequestResponseJmsSender properties

An example of RequestResponseJmsSender configuration

```
<sender class="RequestResponseJmsSender">
  <target>jms/queue/RequestQueue</target>
  <property name="responseTarget" value="jms/queue/ResponseQueue"/>
  <property name="receivingTimeout" value="30000"/>
  <property name="receiveAttempts" value="1"/>
  <property name="connectionFactory" value="jms/ConnFactory"/>
  <property name="username" value="KingRoland"/>
  <property name="password" value="12345"/>
  <property name="responseUsername" value="KingRoland"/>
  <property name="responsePassword" value="12345"/>
  <!-- JNDI properties-->
  <property name="jndiContextFactory" value="pkg.InitCtxFactory"/>
  <property name="jndiUrl" value="remote://${server.host}:4447"/>
  <property name="jndiSecurityPrincipal" value="KingRoland"/>
  <property name="jndiSecurityCredentials" value="12345"/>
  <property name="responseJndiSecurityPrincipal" value="KingRoland"/>
  <property name="responseJndiSecurityCredentials" value="12345"/>
</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.

4.2.16. SslSocketSender

SslSocketSender is similar to the PlainSocketSender (see Section 4.2.14, “PlainSocketSender”) and so it shares the same properties. The difference is that the SslSocketSender uses a SSL socket.

An example of SslSocketSender configuration

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

4.2.17. 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 5.x.

Table 4.18. WebSocketSender properties

An example of WebSocketSender configuration

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

4.3. 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.

4.3.1. An example of a simple message configuration:

```
<messages>
  ...
  <message content="Greetings stranger!"/>
  ...
</messages>
```

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

4.3.2. An example of multiple messages configuration:

```xml
<messages>
  <message uri="message1.txt">
     <header name="header.name" value="header.value"/>
     <header name="header2" value="you-know"/>
     <property name="Empire.State.Building" value="A lot of $"/>
  </message>
  <message uri="message2.xml" multiplicity="2"/>
</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.

4.3.3. 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 and uses HTTL Framework. [7] More complex replacements are possible including math operators, string concatenation etc. 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 through the seq. namespace prefix.

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}`

4.4. 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

[7] http://httl.github.io/en/

- 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.

An example reporting configuration:

```
<reporting>
  <reporter class="ThroughputStatsReporter">
     <property name="minimumEnabled" value="false"/>
     <property name="maximumEnabled" value="false"/>
     <destination class="ConsoleDestination">
        <period type="time" value="1000"/>
     </destination>
     <destination class="CsvDestination">
        <period type="time" value="2000"/>
        <property name="path" value="test-average-throughput.csv"/>
     </destination>
  </reporter>

  <reporter class="MemoryUsageReporter">
     <property name="agentHostname" value="localhost"/>
     <property name="agentPort" value="8850"/>
     <destination class="CsvDestination">
        <period type="time" value="2000"/>
        <property name="path" value="test-memory-usage.csv"/>
     </destination>
  </reporter>
</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.

An example of a disabled reporter:

```
<reporting>
  <reporter class="ThroughputStatsReporter">
     ...
  </reporter>

  <reporter class="MemoryUsageReporter" enabled="false">
     ...
  </reporter>
</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.

4.4.2. Reporters

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 4.19. IterationsPerSecondReporter result names

An example of IterationsPerSecondReporter configuration

```
<reporter class="IterationsPerSecondReporter">
  ...
  (destinations)
  ...
</reporter>
```

In the example above there is a `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"
port	A port number where PerfCake agent is listening.	No	"8850"

Table 4.20. 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.

JVM argument to attach PerfCake agent to the tested JVM

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

PerfCake JVM argument example

```
JAVA_OPTS="... -javaagent:$PERFCAKE_HOME/lib/perfcake-
agent-5.0.jar=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.	No	"false"

Property name	Description	Required	Default value
	The MemoryUsageReporter will send a command to PerfCake agent that will create a heap dump.		
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 4.21. 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.

Result name	Description
MemoryLeak	A boolean value indicating whether a possible memory leak has been detected yet.

Table 4.22. MemoryUsageReporter result names

An example of MemoryUsageReporter configuration

```
<reporter class="MemoryUsageReporter">
  <property name="agentHostname" value="localhost"/>
  <property name="agentPort" value="8850"/>
  ...
  (destinations)
  ...
</reporter>
```

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
windowSize	A window where the data for the statistics are taken from.	No	Integer.MAX_VALUE

Table 4.23. 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 time window).
Maximum	The minimal response time in ms measured so far (in a given time window).
Average	The average response time in ms measured so far (in a given time window).

Table 4.24. ResponseTimeStatsReporter result names

An example of ResponseTimeStatsReporter configuration

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

```
<property name="minimumEnabled" value="false"/>
<property name="maximumEnabled" value="false"/>
  ...
  (destinations)
  ...
</reporter>
```

ThroughputStatsReporter

The reporter is able to report the statistics - current, minimal, maximal and average value of a 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 throughput.

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
windowSize	A window where the data for the statistics are taken from.	No	Integer.MAX_VALUE

Table 4.25. 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).

Table 4.26. ThroughputStatsReporter result names

An example of ThroughputStatsReporter configuration

```
<reporter class="ThroughputStatsReporter">
<property name="minimumEnabled" value="false"/>
<property name="maximumEnabled" value="false"/>
  ...
  (destinations)
  ...
</reporter>
```

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

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.

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 iteration 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"

Table 4.27. WarmUpReporter properties

An example of WarmUpReporter configuration

```xml
<reporter class="WarmUpReporter" enabled="true">
  <property name="minimalWarmUpCount" value="1000"/>
  <property name="minimalWarmUpDuration" value="10000"/>
  <property name="relativeThreshold" value="0.005"/> <!-- 0.5% -->
  <property name="absoluteThreshold" value="0.5"/>
</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.

4.4.3. 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 4.28. Destination `period` options

An example of the `period` configuration in a destination:

```
<destination class="...">
  <period type="time" value="1000"/>
  ...
  (properties)
  ...
</destination>
```

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

ChartDestination

Creates nice charts from the results. The charts are generated under directory specified by the `outputDir` property. The charts in the same `group` can be later combined together based on the names of the columns (reporter's result names). All previously generated charts in the `outputDir` directory are placed in the final report.

Use with caution as the big number of results can take too long to load in the browser. Each chart has a quick view file where the results can be seen while the test is still running.

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.	Yes	-
name	Name of the chart for this measurement.	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	No	"default"

Property name	Description	Required	Default value
	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.		
xAxis	X axis legend.	No	"Time"
yAxis	Y axis legend.	No	"Iterations"
outputDir	A name of the directory where the charts are stored.	No	"perfcake-chart"

Table 4.29. ChartDestination properties

An example of ChartDestination configuration

```
<reporter class="MemoryUsageReporter">
  ...
  <destination class="ChartDestination">
     <period type="time" value="5000"/>
     <property name="name" value="Memory Usage"/>
     <property name="group" value="${perfcake.scenario}_memory"/>
     <property name="yAxis" value="Memory Usage [MiB]"/>
     <property name="outputDir" value="${perfcake.scenario}-charts"/>
     <property name="attributes" value="Used,Total"/>
  </destination>
<reporter>
```

In the example above there is a `MemoryUsageReporter` configured to publish memory usage report into a chart. The memory usage data is gathered each 5 seconds 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” .

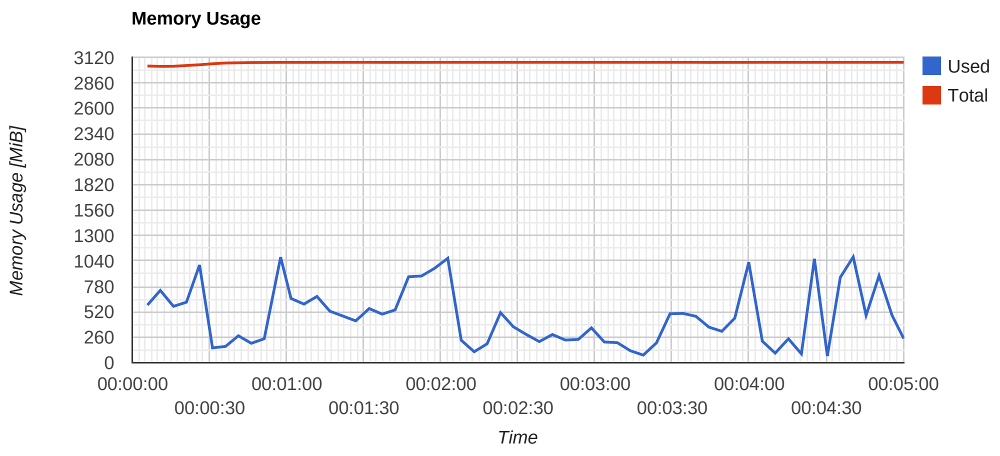

Figure 4.2. ChartDestination example chart

ConsoleDestination

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

The destination has no properties to be configured.

An example of ConsoleDestination configuration

```xml
<destination class="ConsoleDestination">
  <period type="time" value="1000"/>
</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.

The following table describes the CsvDestination's properties

Property name	Description	Required	Default value
path	A path to the output CSV file.	No	perfcake-results-${perfcake.run.timestamp}[a].csv
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 `forceAppend` is for appending new results to the original file.	No	"rename"

[a] See Table 2.1, “Available PerfCake internal properties”

Table 4.30. CsvDestination properties

An example of CsvDestination configuration

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

Log4jDestination

The destination appends the measurements to Log4j to category `prg.perfcake.reporting.destinations.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 4.31. Log4jDestination properties

An example of Log4jDestination configuration

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

4.5. 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 `en-abled` 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` .

An example of validation configuration:

```
<messages>
  ...
  <message uri="...">
     <validatorRef id="validator1"/>
     ...
  </message>
  <message uri="...">
     <validatorRef id="validator1"/>
     <validatorRef id="validator2"/>
     ...
  </message>
  ...
```

```
</messages>
    ...
<validation fastForward="false" enabled="true">
  <validator id="validator1" class="...">
    ...
  </validator>
  <validator id="validator2" class="...">
    ...
  </validator>
</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

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

4.5.2. Validators

DictionaryValidator (TODO: describe)

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 to 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 he 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.

An example of DictionaryValidator configuration

```
<validator id="rimmerValidator" class="DictionaryValidator">
  <property name="pattern">
     <text><![CDATA[.*"I'm a [Ff]ish"!\.*]]></text>
  </property>
</validator>
```

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.

An example of RegExpValidator configuration

```
<validator id="rimmerValidator" class="RegExpValidator">
```

```
  <property name="pattern">
     <text><![CDATA[.*"I'm a [Ff]ish"!\.*]]></text>
  </property>
</validator>
```

RulesValidator

This validator uses Drools [8] engine to assert the validator rules. The rules for this validator have their own DSL [9] in a form of human readable sentences. The rules are listed in a following listing:

RulesValidator rules

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

An example of RulesValidator configuration

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

ScriptValidator (TODO: describe)

...

An example of ScriptValidator configuration

```
<validator id="rimmerValidator" class="ScriptValidator">
  <property name="pattern">
     <text><![CDATA[.*"I'm a [Ff]ish"!\.*]]></text>
  </property>
</validator>
```

[8] http://www.jboss.org/drools/
[9] Domain Specific Language

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] .

If you want to add a new component to be used by PerfCake, all you need to do is to take the jar file, where the component is packaged (along with the jar files on which the component depends on), and place it all under 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.

Chapter 8. Changelog

8.1. Release 5.0

8.1.1. Features

- https://github.com/PerfCake/PerfCake/issues/23 - Add check for code-style, code coverage and license
- https://github.com/PerfCake/PerfCake/issues/53 - Implement missing senders
- https://github.com/PerfCake/PerfCake/issues/73 - Implement Eclipse plugin to operate PerfCake
- https://github.com/PerfCake/PerfCake/issues/80 - Implement IntelliJ IDEA plugin to operate PerfCake
- https://github.com/PerfCake/PerfCake/issues/138 - Add support for classpath definition to the Groovy sender
- https://github.com/PerfCake/PerfCake/issues/149 - Review logging
- https://github.com/PerfCake/PerfCake/issues/151 - Review exception handling
- https://github.com/PerfCake/PerfCake/issues/182 - Create a script to automatically incerase PerfCake version
- https://github.com/PerfCake/PerfCake/issues/190 - Implement SoapSender
- https://github.com/PerfCake/PerfCake/issues/194 - Remove generated classes from JaCoCo.
- https://github.com/PerfCake/PerfCake/issues/195 - Develop CommandSender test
- https://github.com/PerfCake/PerfCake/issues/197 - Add support for sequences
- https://github.com/PerfCake/PerfCake/issues/205 - Reorganize pom files
- https://github.com/PerfCake/PerfCake/issues/206 - Strip debug info from binary releases
- https://github.com/PerfCake/PerfCake/issues/207 - Generate aggregated javadoc
- https://github.com/PerfCake/PerfCake/issues/209 - Implement constant speed message generator
- https://github.com/PerfCake/PerfCake/issues/210 - Implement MQTT sender
- https://github.com/PerfCake/PerfCake/issues/214 - Review XML scenario format
- https://github.com/PerfCake/PerfCake/issues/215 - Migrate testing features out of DummySender
- https://github.com/PerfCake/PerfCake/issues/216 - Implement Iteration per seconds reporter
- https://github.com/PerfCake/PerfCake/issues/218 - Http Sender must be able to switch Http methods
- https://github.com/PerfCake/PerfCake/issues/219 - Create thread ID sequencer and make it a default one
- https://github.com/PerfCake/PerfCake/issues/220 - Create current timestamp sequencer and make it a default one
- https://github.com/PerfCake/PerfCake/issues/221 - Verify Java version in startup scripts

- https://github.com/PerfCake/PerfCake/issues/222 - Find out whether JRE is sufficient for running PerfCake
- https://github.com/PerfCake/PerfCake/issues/223 - Make sure sequences can be used in message headers
- https://github.com/PerfCake/PerfCake/issues/226 - Switch httl to Javassist
- https://github.com/PerfCake/PerfCake/issues/231 - Mark components' mandatory properties as such.
- https://github.com/PerfCake/PerfCake/issues/237 - Include number of threads from run info into reporting
- https://github.com/PerfCake/PerfCake/issues/242 - Create XSL transformation for scenarios from version 4.0 to 5.0

8.1.2. Bug fixes

- https://github.com/PerfCake/PerfCake/issues/204 - Fix path handling in binary distribution
- https://github.com/PerfCake/PerfCake/issues/212 - `messageNumberingEnabled` property of `DefaultMessageGenerator` is missing from User Guide
- https://github.com/PerfCake/PerfCake/issues/213 - javadoc.perfcake.org is empty
- https://github.com/PerfCake/PerfCake/issues/227 - Sequence manager raise condition
- https://github.com/PerfCake/PerfCake/issues/228 - Timestamp configuration skipped when using API to run scenario
- https://github.com/PerfCake/PerfCake/issues/229 - RegExp validator does not consider empty payload
- https://github.com/PerfCake/PerfCake/issues/233 - README.md and CHANGES.md files are missing from the root of the repository
- https://github.com/PerfCake/PerfCake/issues/236 - Scenario loading fails if there is a directory with the same name as scenario present in PerfCake home
- https://github.com/PerfCake/PerfCake/issues/243 - Broken build - com.sun.xml.bind

8.2. Release 4.1

8.2.1. Features

- https://github.com/PerfCake/PerfCake/issues/200 - Rename agent's artifact ID to "perfcake-agent" to increase lucidity
- https://github.com/PerfCake/PerfCake/issues/187 - Make sure PerfCakeAgent can be attached to various JDK versions.
- https://github.com/PerfCake/PerfCake/issues/186 - Introduce Generator interface

8.2.2. Bug fixes

- https://github.com/PerfCake/PerfCake/issues/202 - Fix timezone handling in chart destination

- https://github.com/PerfCake/PerfCake/issues/199 - Useless test dependencies are present in PerfCake distro
- https://github.com/PerfCake/PerfCake/issues/193 - Sources JAR found in distro can cause main script to fail
- https://github.com/PerfCake/PerfCake/issues/192 - Source jars are part of binary distribution

8.3. Release 4.0

8.3.1. Features

- Scenario schema namespace changed to `"urn:perfcake:scenario:4.0"`.
- https://github.com/PerfCake/PerfCake/issues/68 - Allow to set Logger threshold by parameter or similar way
- https://github.com/PerfCake/PerfCake/issues/116 - Add Google Charts templates
- https://github.com/PerfCake/PerfCake/issues/136 - Implement expression language
- https://github.com/PerfCake/PerfCake/issues/140 - Develop Validators integration test
- https://github.com/PerfCake/PerfCake/issues/141 - Refactor Reporter Contract test
- https://github.com/PerfCake/PerfCake/issues/142 - Review members visibility and usage of final
- https://github.com/PerfCake/PerfCake/issues/143 - Update maven-javadoc-plugin to the latest version
- https://github.com/PerfCake/PerfCake/issues/145 - Archive signatures using website SSL key
- https://github.com/PerfCake/PerfCake/issues/148 - Review Javadoc Maven plugin configuration
- https://github.com/PerfCake/PerfCake/issues/150 - Review JavaDoc enhancement
- https://github.com/PerfCake/PerfCake/issues/152 - Optimize test groups and run time
- https://github.com/PerfCake/PerfCake/issues/153 - Investigate possibility to initiate a heapdump once possible memory leak is detected by `MemoryUsageReporter`.
- https://github.com/PerfCake/PerfCake/issues/158 - Add possibility to show validation statistics when validation is enabled.
- https://github.com/PerfCake/PerfCake/issues/159 - Improve integration with Sonatype enhancement
- https://github.com/PerfCake/PerfCake/issues/160 - Enable possibility to perform a garbage collection for `MemoryUsageReporter`
- https://github.com/PerfCake/PerfCake/issues/163 - Extend tests to cover message templating
- https://github.com/PerfCake/PerfCake/issues/167 - Extend `RegExpValidator` to be able to configure `Pattern.compile` [1] flags
- https://github.com/PerfCake/PerfCake/issues/168 - Publish documentation in HTML format documentation

[1] http://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html#compile-java.lang.String-int-

- https://github.com/PerfCake/PerfCake/issues/172 - Try using Timer from Faban as a precise source of time
- https://github.com/PerfCake/PerfCake/issues/173 - Investigate possibilities of using Goldman Sachs collections
- https://github.com/PerfCake/PerfCake/issues/174 - Extend accumulators tests
- https://github.com/PerfCake/PerfCake/issues/175 - Update embedded AS Arquillian tests to run with Java 8
- https://github.com/PerfCake/PerfCake/issues/176 - Convert `CSVDestination` to use `FileChannel` [2]
- https://github.com/PerfCake/PerfCake/issues/177 - Document all result names for each reporter
- https://github.com/PerfCake/PerfCake/issues/178 - Log validators separately
- https://github.com/PerfCake/PerfCake/issues/180 - Allow only alphanumeric digits and an underscore in group name for `ChartDestination`

8.3.2. Bug fixes

- https://github.com/PerfCake/PerfCake/issues/156 - XSLT transformation adds an empty property to sender
- https://github.com/PerfCake/PerfCake/issues/157 - `MemoryUsageReporter` does not reset correctly
- https://github.com/PerfCake/PerfCake/issues/161 - Issue with space in `JAVA_HOME`
- https://github.com/PerfCake/PerfCake/issues/164 - `RampUpDownGenerator` produces NPE
- https://github.com/PerfCake/PerfCake/issues/165 - `RampUpDownGenerator` does not have implemented `mainThreadCount` getters/setters
- https://github.com/PerfCake/PerfCake/issues/166 - `RampUpDownGenerator` logs current phase too often
- https://github.com/PerfCake/PerfCake/issues/179 - Log4j config is missing in binary distro

8.4. Release 3.3

8.4.1. Features

- https://github.com/PerfCake/PerfCake/issues/94 - First Technical Preview of DSL scenario specification support

8.4.2. Bug fixes

- https://github.com/PerfCake/PerfCake/issues/146 - The XSLT for scenario conversion from v2 to v3 does not cover RequestResponseJmsSender properly.

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

8.5. Release 3.2

8.5.1. Bug fixes

- Workaround for https://jira.codehaus.org/browse/MJAVADOC-408 .
- Scenario conversion XSLT is missing from binary distribution.

8.6. Release 3.1

8.6.1. Bug fixes

- JBoss Maven repository is missing - needed for JMS based tests.

8.7. Release 3.0

8.7.1. Features

- Scenario schema namespace changed to`"urn:perfcake:scenario:3.0"`.
- Added `content` attribute to `message` element in scenario XML for specifying message content directly from scenario definition.
- `AverageThrouhgputRepoter`, `ResponseTimeReporter` and `WindowResponseTimeReporter` removed and replaced by more sophisticated `ThroughputStatsReporter` and`ResponseTimeStatsReporter`.
- Extended `CommandSender` to set message headers and properties as environmental variables for the executing script.
- Extended `CSVDestination` features - ability to specify line prefix, suffix, end-of-line and to skip the file header.
- Added a PIT (http://pitest.org/) plugin for mutation testing.
- Improved performance.
- Increased test coverage.
- Improved validators - renamed some classes, simplified Validator interface, updated RulesValidator (KIE 6.1.0.Beta3), created ScriptValidator (JSR-223), both original and response messages are passed to the validator, validation can run fast from the very beginning (fastForward in scenario), validation can be disabled in scenario.
- Compatibility with JDK 8.
- Properties in Scenario can have arbitrary XML elements in it, components can declare setter that accepts org.w3c.dom.Element
- Implemented `LDAPSender` .
- `RequestResponseJMSSender` has the ability to set different connection for request and response queues.

- Created XSLT for scenario XML conversion from v 2.0 to 3.0.

8.7.2. Bug fixes

- https://github.com/PerfCake/PerfCake/issues/60 - Refactor validators
- https://github.com/PerfCake/PerfCake/issues/104 - Fix CSVDestination synchronization problem.
- https://github.com/PerfCake/PerfCake/issues/111 - PerfCakeAgent hangs when configured to monitor local JVM process.
- https://github.com/PerfCake/PerfCake/issues/123 - Review and update tests to become independent to each other.
- https://github.com/PerfCake/PerfCake/issues/127 - CommandSender throws `NullPointerException` when message is not used.
- https://github.com/PerfCake/PerfCake/issues/128 - Fix ext.dirs path to also include JRE libraries.
- https://github.com/PerfCake/PerfCake/issues/130 - Early log messages are not shown.
- https://github.com/PerfCake/PerfCake/issues/132 - Rename those classes that do not follow the naming conventions so they do.

8.8. Release 2.1

8.8.1. Features

- Added site.xml making use of Maven Fluido Skin.

8.8.2. Bug fixes

- https://github.com/PerfCake/PerfCake/issues/119 - Message properties and headers are ignored.

8.9. Release 2.0

8.9.1. Features

- Scenario schema namespace changed to `"urn:perfcake:scenario:2.0"`.
- Added `-pf` parameter to CLI for specifying a property file.
- Added a possibility to add extra jars to the classpath (lib/ext directory).
- Added a mechanism for extending PerfCake via plugins (lib/plugins directory).
- Added `appendStrategy` property to `CSVDestination` .
- Added a new generator - `RampUpDownGenerator` .
- Added a new sender - `WebSocketSender` .
- Implemented a memory leak detection feature into `MemoryUsageReporter` .
- Improved sender pool implementation.

- Improved performance.
- Increased test coverage.

8.9.2. Bug fixes

- https://github.com/PerfCake/PerfCake/issues/77 - Exceptions thrown by senders are not processed by the logging subsystem.
- https://github.com/PerfCake/PerfCake/issues/84 - CSVDestination appends new records into the output file if the one exists and losts the warmUp column.
- https://github.com/PerfCake/PerfCake/issues/86 - MemoryUsageReporter collects data for memory leak detection analysis incorrectly.
- https://github.com/PerfCake/PerfCake/issues/87 - Bad maxIterations handling after warm up.
- https://github.com/PerfCake/PerfCake/issues/97 - Accumulating the results leads to NPE because of concurrency.

8.10. Release 1.0.1

8.10.1. Bug fixes

- https://github.com/PerfCake/PerfCake/issues/82 - GroovySender resolves default groovy path wrong.
- https://github.com/PerfCake/PerfCake/issues/83 - Default scenarios and messages directories are resolved wrong in binary distribution.

8.11. Release 1.0

8.11.1. Features

- Basic performance testing functionality including Generators, Senders, Reporters, Destinations and Validators.

8.11.2. Bug fixes

- Many bugs fixed

8.12. Release 0.3

8.12.1. Features

- First released version - everything has changed.

8.13. Changes in scenario format

All the changes in scenario format are described in details and can be found in Section 2.5, “Migrating scenarios to latest version” .