Executing JUnit Test Cases as Load Test

English Edition
# Table of Contents

1. Introduction .................................................................................................................................................. 3
2. Preparing a Session ....................................................................................................................................... 3
3. Adding the JUnit Packages and the Test Package to the Session ................................................................. 6
   3.1 Adding a System Properties File ............................................................................................................. 8
4. Generating a JUnit Load Test Plug-In ........................................................................................................ 9
5. Adding a JUnit Plug-In to the Session ........................................................................................................ 14
6. Generating the Load Test Program and Executing the Load Test ............................................................... 19
7. Hints and Recommendations ..................................................................................................................... 27
   7.1 Configure Timeouts for all JUnit Test Methods ..................................................................................... 27
   7.2 Passing User-Specific Variables to JUnit Tests ...................................................................................... 28
   7.3 Modifying a JUnit Plug-In ....................................................................................................................... 30
   7.4 Integration with Apica Inside Agents ...................................................................................................... 33

Manufacturer, Sales and Support .................................................................................................................. 35
1  Introduction
Starting from ProxySniffer version 5.1 you can execute JUnit test cases as load tests by using the ProxySniffer infrastructure.

The integration of JUnit tests cases with ProxySniffer is supported by a wizard that automatically generate load test JUnit plug-ins which do not require manual programming.

The benefit of such integration is that you can run your JUnit load tests on any load generator and that also clusters of load generators are supported. This means that many thousands of JUnit test cases can run in parallel at the same time, and that all of the results are combined to one test report.

2  Preparing a Session
First you need an almost empty session that contains only one "inactive" URL call that never will be executed. If you don’t have created any session you can use the PageScanner tool to get one. Or you can load any existing session. After that remove all URL calls except the first one from that session and set the User’s Think Time to a zero value.
Next configure the URL call as "non executable"
Then return to the main menu and click on refresh. The URL call should now be shown with a yellow exclamation mark.
3 Adding the JUnit Packages and the Test Package to the Session

Call the Declare External Resources menu and add all Java *.jar files that are needed to execute the JUnit test. This means that you have to add the jar files of JUnit itself (for example junit-4.11.jar and hamcrest-core-1.3.jar) and the jar file that contains your JUnit test (for example CollectionTest.jar):
**Declare External Resources**

Add an External File to Load Test

- **File Located On**: 
  - Local System
  - Remote System (pre-installed on Exec Agent)

- **Absolute File Path**

- **Add File to Java CLASSPATH on the Exec Agent(s)**
- **Add File to Java Xbootclasspath on the Exec Agent(s)**

Add External File

---

This menu allows you to declare additional, external resources needed when executing a load test. Typically such external resources are **Java library files (*.jar files)** which are used by self-developed plugins, or **Google Protobuf File Descriptor Sets** (`.desc files`) used to parse G-PROTUBUF messages. However, any other file types can also be declared.

Note: Any declarations made in the Var Handler menu such as **Input Files** and **Main Classes of Plugins** are **not external resources** in this context and don’t need to be declared because Proxy Sniffer knows already these declarations.

---

**List of all Declared External Resources**

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
<th>CLASSPATH</th>
<th>Xbootclasspath</th>
<th>Absolute File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>local</td>
<td>yes</td>
<td>no</td>
<td>C:\Programme\Unit-4.11\JUnit-4.11.jar</td>
</tr>
<tr>
<td>✗</td>
<td>local</td>
<td>yes</td>
<td>no</td>
<td>C:\Programme\Unit-4.11\hamcrest-core-1.3.jar</td>
</tr>
<tr>
<td>✗</td>
<td>local</td>
<td>yes</td>
<td>no</td>
<td>C:\Programme\Unit-4.11\CollectionTest.jar</td>
</tr>
</tbody>
</table>
### 3.1 Adding a System Properties File

As an option you can also declare a System Properties File as external resource. Such a file will be parsed on the load generators just before the JUnit tests are executed. The key and values of such a file are stored in the Java virtual machine as system properties.

The lines of the file can contain the following commands:

1. **key = value**
2. `.includeProps = "<include-file-name>"`
3. `.includeProps = ["<include-file-name-1>", "<include-file-name-2>", ..., "<include-file-name-N>"]`

If a value ends with a backslash (\) the next line is appended to the value. Lines that are staring with a hash sign (#) are ignored.

Example:

```java
# General settings
loggingBufferSize = 1000
memoryStatsEnabled = true

# JSON data
jsonRequest = {\n    "defaultID": 1, \n    "keys": [\n        {"id":1, "type":"A25", "value":"57FCCA5C9F39BC2E73B5"}, \n    ],\n}
```

List of all Declared External Resources

<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
<th>CLASSPATH</th>
<th>Xbootclasspath:</th>
<th>Absolute File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>file</td>
<td>local</td>
<td>yes</td>
<td>C:\ProgrammeJUnit-4.11\Unit-4.11.jar</td>
</tr>
<tr>
<td>✗</td>
<td>file</td>
<td>local</td>
<td>yes</td>
<td>C:\ProgrammeJUnit-4.11\hamcrest-core-1.3.jar</td>
</tr>
<tr>
<td>✗</td>
<td>file</td>
<td>local</td>
<td>yes</td>
<td>C:\ProgrammeJUnit-4.11\JCollectionTest.jar</td>
</tr>
<tr>
<td>✗</td>
<td>file</td>
<td>local</td>
<td>no</td>
<td>C:\ProgrammeJUnit-4.11\SystemProperties.txt</td>
</tr>
</tbody>
</table>

Hint: To debug the parsing of such a file you can enable the option "debug loops" when starting a load test. The debug output is written to the *.out file of the load test job.
4 Generating a JUnit Load Test Plug-In

Once you have loaded all External Resources navigate in the Project Navigator to the Plugins directory and call the JUnit wizard.

In the wizard enter the full class name of the JUnit test:
After that you can select which JUnit test methods should be executed during the load test:
Further Input Fields:

- **Methods Execution Order**
  You can select "Java/Reflection", "Alphabetical", "Randomly per Loop" or "In Parallel Threads".
  Note: When selecting "In Parallel Threads" this means always "per simulated user". So, if you start a load test with 1,000 users containing a JUnit class with 3 test methods then 3,000 JUnit threads will be run in parallel at the same time.

- **System Properties File**
  If you have declared a System Properties File as External Resource you should select it here.

- **Combine Error Types**
  Similar measured errors that occur during the load test execution are grouped by ProxySniffer to "Error Types" in order to get a better overview. Here you can choose how the grouping should be made.
  You can select "Class + Method Name", "JUnit Failure Text", "Exception Text" or "Stack Trace".
  For example if "JUnit Failure Text" is selected then all error that contain exactly the same "JUnit Failure Text" will be in the same group.

- **Plug-in Class Name**
  The Java class name of the (new) generated JUnit plug-in.

- **Plug-in GUI Label**
  The label (text) of the plug-in that is displayed in the GUI.

- **Plug-in Input Parameter**
  Do not add or remove plug-in input parameter. Leave them "as is".

![Plug-in Input Parameter](image)
Proxy Sniffer V 5.1  

Running JUnit Test Cases as Load Test

Project Navigator - Generate JUnit Plug-In Code

JUnit Plugin Template: /Users/dfdizer/Documenten und Einstellungen/monty/ProxySniffer/MyTests/Plugins/JUnit_CollectionTest_PluginTemplate.xml

JUnit Plug-in Class Name: JUnit_CollectionTest

JUnit Plug-in Templates

JUnit CollectionTest PluginTemplate.xml

Compiled JUnit Plugins

// Most or all parts of this code have been automatically generated - copyright for generic JUnit plug-in project
// Copyright for manual written code belongs to <your name>, <your company>, <your country>
import dfischer.utils.GenericPluginInterface;
import dfischer.utils.LoadTestPluginInterface;
import dfischer.utils.RunJUnitMethodInterface;
import dfischer.utils.RunJUnitMethodThread;
import dfischer.utils.LoadTestPluginFixedSizeInputField;
import dfischer.utils.log.Vector;
import dfischer.utils.LoadTestPluginText;
import dfischer.utils.HttpLoadTest;
import dfischer.utils.PerformanceData;
import dfischer.utils.HttpTestURL;
import dfischer.utils.SystemPropertiesFile;
import dfischer.utils.bb;
import dfischer.utils.mf;
import dfischer.utils.SortStringArray;

Save JUnit Plug-In Code
Proxy Sniffer V 5.1
Running JUnit Test Cases as Load Test

© Ingenieurbüro David Fischer AG, Switzerland  All Rights Reserved  November 13, 2013  Page 13 / 35

<table>
<thead>
<tr>
<th>Project Navigator</th>
<th>Mozilla Firefox</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image_url" alt="Image of Project Navigator window" /></td>
<td></td>
</tr>
</tbody>
</table>

C:\Dokumente und Einstellungen\mutong\ProxySniffer\MyTests\Plugins

- **JUnit_CollectionTest.java**
  - Size: 13279

- **JUnit_CollectionTest_PluginTemplate.xml**
  - Size: 1908
  - Modified: 13 Nov 2013 13:23:05

- **ProxySnifferPluginDeveloperHandbookEN.pdf**
  - Size: 1072973

- **CheckWithValueClass**
  - Size: 2763

...
5 Adding a JUnit Plug-In to the Session

Once the Plug-In was generated navigate in the Project Navigator to that directory in which the load test program should be generated.

Then return to the Main Menu and add the JUnit Plug-In to the URL Call.
At this step it's also recommended to save the session.
6 Generating the Load Test Program and Executing the Load Test

Generating the Load Test Program and Executing the Load Test can be done as usual:

![Generating the Load Test Program and Executing the Load Test](image)
Proxy Sniffer V 5.1  Running JUnit Test Cases as Load Test

© Ingenieurbüro David Fischer AG, Switzerland  All Rights Reserved  November 13, 2013  Page 20 / 35
This load test program must first be zipped with other files before it can be executed.

Click on the "ZIP and execute..." button below to combine the compiled load test program together with the Plugin(s) and External Local Resources and the declarations for External Resources to a ZIP archive.

<table>
<thead>
<tr>
<th>New Zip Archive</th>
<th>TEST_01.zip</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load Test Program</td>
<td>TEST_01 class</td>
</tr>
<tr>
<td>Plugins</td>
<td>JUnit.CollectionTest.class</td>
</tr>
<tr>
<td>External Local Resources</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C:\Programme\JUnit-4.11\junit-4.11.jar</td>
</tr>
<tr>
<td></td>
<td>C:\Programme\JUnit-4.11\junit-platform-1.1.1-jar \ C:\Programme\JUnit-4.11\junit-platform-1.1.1-jar \ C:\Programme\JUnit-4.11\junit-platform-1.1.1-jar \ C:\Programme\JUnit-4.11\junit-platform-1.1.1-jar</td>
</tr>
<tr>
<td>XML File for External Resources</td>
<td>TEST_01.ExternalResources.xml</td>
</tr>
</tbody>
</table>

>> ZIP and execute...
All JUnit plug-ins support to configure a Pacing Time (0 = no Pacing Time).

The Pacing Time defines how much minimum elapsed-time a simulated user should spent in the plug-in.

**Example 1:** If you configure a pacing time of 10 seconds and a simulated user executes the plug-in in 3 seconds the user will be delayed for 7 seconds in that executed loop (3 + 7 seconds = 10 seconds).

**Example 2:** If you configure a pacing time of 10 seconds and a simulated user executes the plug-in in 2 seconds the user will be delayed for 8 seconds in that executed loop (2 + 8 seconds = 10 seconds).

**Example 3:** If you configure a pacing time of 10 seconds and a simulated user executes the plug-in in 13 seconds the user will not be delayed in that loop (13 > 10 seconds)
During the execution of the load test you can examine the measured errors at real time:
### Error Overview (Real-Time) - Local Exec Agent - Job 12

#### Latest Error Snapshots

<table>
<thead>
<tr>
<th>No.</th>
<th>Latest Error Time</th>
<th>Error Type</th>
<th>Page</th>
<th>URL Index</th>
<th>Err No.</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56989</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-1</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56988</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-2</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56990</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56991</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-4</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56992</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-5</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56993</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-6</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56994</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-7</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56995</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-8</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56996</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-9</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56997</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-10</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56998</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-11</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>56999</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-12</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>57000</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-13</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>57001</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-14</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>57002</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-15</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>57003</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-16</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>57004</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
<tr>
<td>-17</td>
<td>- 0 sec</td>
<td>secondCheck(CollectionTest) expected «&lt;jsp&gt;» but was «&lt;jsp&gt;»</td>
<td>Page #1: Apica ProxySniffer - [Profile]...</td>
<td>57005</td>
<td>GET <a href="http://92.16.4.88/">http://92.16.4.88/</a></td>
<td></td>
</tr>
</tbody>
</table>

Current Time: 13 Nov 2013 14:15:07, Elapsed Time: 1.25 min
After the load test has been completed, and you have acquired the test result, you can display the **execution time** for each JUnit method:
7 Hints and Recommendations

7.1 Configure Timeouts for all JUnit Test Methods

In order to prevent a blocking/freezing of a JUnit load test it's strongly recommended that you configure in your JUnit test cases a timeout for each test method. You can use large values for such timeouts like 15,000 milliseconds.

Example:

```java
@Test(timeout=15000)
public void sortedSet() {
...
```
7.2 Passing User-Specific Variables to JUnit Tests

JUnit by itself does not support to pass any (local) per test-run specific variables to a test method. But, for example when testing a Web service, it's often required that each simulated user, or each JUnit test run, use a different username and password for authentication.

The easiest way is to use a Text-File containing the values of such per test-run specific variables. Example:

Miller; abcd
Meier; topsecret
Sutter; password

To support that you should modify the program code of the JUnit test case as follows:

```java
import dfischer.utils.StaticInputFile; // defined in prxsniff.jar – see also ProxySniffer Java API Doc
import dfischer.utils.StaticInputFileLineContent;
import dfischer.utils.AbstractInputFileReader;
...

@Test(timeout=15000)
public void sortedSet() {
    String username = "Miller"; // default value, used for non load test execution
    String password = "abcd"; // default value, used for non load test execution

    if ((System.getProperty("PrxJUnitLoadTest") != null) &&
        System.getProperty("PrxJUnitLoadTest").equalsIgnoreCase("true")) {
        int fileId = StaticInputFile.concurrentOpen("Accounts.txt", ",", ";", true, AbstractInputFileReader.EOF_REOPEN_FILE);
        StaticInputFileLineContent line = StaticInputFile getNextLine(fileId);
        username = line.getValue(0);
        password = line.getValue(1);
    }
    ...
}
```

Note that the System property "PrxJUnitLoadTest" = "true" is automatically set by ProxySniffer when executing a JUnit load test. The file containing the user accounts must also be declared as an external resource:

```
<table>
<thead>
<tr>
<th>Type</th>
<th>Location</th>
<th>CLASSPATH</th>
<th>Xbootclasspath</th>
<th>Absolute File Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>local</td>
<td>yes</td>
<td>no</td>
<td>C:\Programs\JUnit-4.11\unit-4.11.jar</td>
</tr>
<tr>
<td>file</td>
<td>local</td>
<td>yes</td>
<td>no</td>
<td>C:\Programs\JUnit-4.11\hamcrest-core-1.3.jar</td>
</tr>
<tr>
<td>file</td>
<td>local</td>
<td>yes</td>
<td>no</td>
<td>C:\Programs\JUnit-4.11\CollectionTest.jar</td>
</tr>
<tr>
<td>file</td>
<td>local</td>
<td>no</td>
<td>no</td>
<td>C:\Programs\JUnit-4.11\System\resources.tld</td>
</tr>
<tr>
<td>file</td>
<td>local</td>
<td>no</td>
<td>no</td>
<td>C:\Programs\JUnit-4.11\accounts.tld</td>
</tr>
</tbody>
</table>
```
When using a cluster of load generators you can also split the file across the cluster members:
7.3 Modifying a JUnit Plug-In

To modify an already generated JUnit Plug-In navigate to the Plugins directory and click on the corresponding Template File:

After you have modified the plug-in generate and compile the plug-in once again. Please note that you have to confirm in Project Navigator that the old plug-in code should be overwritten and compiled (in two steps).
After that navigate to the directory where the load test program is located and update the plug-in in this directory:

Then the load test can be directly started. There is no need to generate and compile the load test program once again in such a case. This is only needed if you declare additional external resources in the load test program.
7.4 Integration with Apica Inside Agents

The measured data of JUnit load tests can also be combined with measured Operating System performance data of any machine at which an "Apica Inside Agent" is installed, such as, for example, on Application Servers and on Database Servers.

As usual you have to select the corresponding "Monitoring Controller Template" when starting the JUnit load test:
The captured system performance data will also be added to the "External Measured Data".
Manufacturer, Sales and Support

Ingenieurbüro David Fischer AG, Switzerland | A company of the Apica Group

Manufacturer's Web Site: www.proxy-sniffer.com
Support: support@apicasystem.com
Sales: sales@apicasystem.com

Note: All menus provide context specific help text, available using the Help icon: