



Fresher Academy



BeanShell Programming in Jmeter

Agenda

- What is BeanShell?
- Why do we need BeanShell in Jmeter.
- JMeter Beanshell Components
- BeanShell : Create and use variables
- Print data on console & Comments
- BeanShell PreProcessor
- BeanShell Sampler
- BeanShell PostProcessor
- BeanShell Assertion
- Dashboard Generator

What is BeanShell

BeanShell is a small, free, embeddable Java source interpreter with object scripting language features, written in Java.

BeanShell runs in the Java Runtime Environment and makes use of a variation of the Java syntax.

BeanShell has been used in many applications like Apache Jmeter, WebLogic Server and Apache OpenOffice.

Why do we need BeanShell in Jmeter

BeanShell is one of the most advanced JMeter built-in components. It supports Java syntax and extends it with scripting features like loose types, commands, and method closures. If your test case is uncommon and implementation via embedded JMeter components becomes tricky or even impossible, BeanShell can be an excellent option to achieve your objectives.

BeanShell entities in JMeter have access to both internal JMeter APIs and any external classes that are loaded into the JMeter classpath (be sure to drop necessary jars into **/lib/ext** folder of your JMeter installation and place all necessary “**import**” statements at the top of your BeanShell scripts).

JMeter Beanshell Components

JMeter provides the following components that can be used to write BeanShell scripts

BeanShell PreProcessor

BeanShell Sampler

BeanShell PostProcessor

BeanShell Assertion

JMeter Beanshell Components

The following table shows some of the common variables used by the BeanShell components:

Variable name	Description
ctx	It holds context information about the current thread that includes sampler and its results.
vars	This is a thread local set of variables stored in a map used by BeanShell components in the same thread.
props	These are variables loaded as properties from an external file (<i>jmeter.properties</i>) stored in the classpath.

JMeter Beanshell Components

ctx

ctx is the most powerful variable exposed to BeanShell. It represents the **org.apache.jmeter.threads.JMeterContext** class, which is virtually JMeter itself. It provides read/write access to the underlying JMeter engine, samplers, and their results as well as variables/properties.

Modifier and Type	Method and Description
void	cleanAfterSample() Clean cached data after sample Internally
void	clear() Internally called by JMeter, never call it c
Sampler	getCurrentSampler()
StandardJMeterEngine	getEngine()
SampleResult	getPreviousResult()
Sampler	getPreviousSampler() Returns the previousSampler.
java.util.Properties	getProperties()
java.util.Map<java.lang.String,java.lang.Object>	getSamplerContext() Sampler context is cleaned up as soon as
JMeterContext.TestLogicalAction	getTestLogicalAction()
JMeterThread	getThread()
AbstractThreadGroup	getThreadGroup()
int	getThreadNum()
JMeterVariables	getVariables() Gives access to the JMeter variables for t

JMeter Beanshell Components

vars

vars (JMeter variables) is the most frequently used component. It's an instance of the `org.apache.jmeter.threads.JMeterVariables` class and provides read/write access to current variables, is capable of enumerating/changing existing variables, creating new ones, and obtaining nested properties. All JMeter variables are Java strings. If you need to put something else to a JMeter variable, you'll need to cast it to string first. The following code snippet demonstrates how to save previous sampler response data into a JMeter variable.

JMeter Beanshell Components

vars

Modifier and Type	Method and Description
<code>java.util.Set<java.util.Map.Entry<java.lang.String,java.lang.Object>></code>	<code>entrySet()</code>
<code>java.lang.String</code>	<code>get(java.lang.String key)</code> Gets the value of a variable, converted to a S
<code>int</code>	<code>getIteration()</code>
<code>java.util.Iterator<java.util.Map.Entry<java.lang.String,java.lang.Object>></code>	<code>getIterator()</code> Gets a read-only Iterator over the variables.
<code>java.lang.Object</code>	<code>getObject(java.lang.String key)</code> Gets the value of a variable (not converted to
<code>java.lang.String</code>	<code>getThreadName()</code>
<code>void</code>	<code>incIteration()</code> Increase the current number of iterations
<code>void</code>	<code>put(java.lang.String key, java.lang</code> Creates or updates a variable with a String v
<code>void</code>	<code>putAll(JMeterVariables vars)</code> Updates the variables with all entries found
<code>void</code>	<code>putAll(java.util.Map<java.lang.Stri</code> Updates the variables with all entries found
<code>void</code>	<code>putObject(java.lang.String key, java</code> Creates or updates a variable with a value th
<code>java.lang.Object</code>	<code>remove(java.lang.String key)</code>

JMeter Beanshell Components

props

Basically, this is the same as “vars,” but it exposes JMeter properties instead. See JavaDoc on `java.util.Properties` and JMeter documentation on JMeter properties for more information. The primary distinction between props and vars is that props have a “global” scope, whereas the scope of “vars” is limited to the current thread group.

JMeter Beanshell Components

log

log represents the org.apache.log.Logger class and can be used to append a message into jmeter.log file. See JavaDoc on logger for more details. The following is the sample use case:

```
log.info("This is a message with INFO level");  
log.error("This is a message with ERROR level");
```

BeanShell : Create and use variables

Variable type	The amount of memory in bytes	Valid range of values
int	4	from -2147483648 to 2147483647
short	2	from -32768 to 32767
long	8	from -9223372036854775808 to 9223372036854775807
byte	1	from -128 to 127

BeanShell : Create and use variables

Script (see below for variables that are defined)

```
1 int a = 2147483647;
2 short b = 32767;
3 long c = 9223372036854775807L;
4 byte d = 127;
5
6 float e = 3.4028234f;
7 float f = 3.4028234F;
8 double g = 1.7976931348623157;
9
10 char h = 'N';
11 char i = 78;
12
13 boolean k = true;
14 boolean l = false;
15
16 String m = "It is string";
17 String n = new String("It is string");
```

Print data on console & Comments

```
18
19 log.info(a + " - It is int");
20 log.info(b + " - It is short");
21 log.info(c + " - It is long");
22 log.info(d + " - It is byte");
23 /*
24 log.info(e + " - It is float");
25 log.info(f + " - It is float");
26 log.info(g + " - It is double");
27 */
28 log.info(h + " - It is char");
29 log.info(i + " - It is not char");
```

Print data on console & Comments

```
18
19 log.info(a + " - It is int");
20 log.info(b + " - It is short");
21 log.info(c + " - It is long");
22 log.info(d + " - It is byte");
23 /*
24 log.info(e + " - It is float");
25 log.info(f + " - It is float");
26 log.info(g + " - It is double");
27 */
28 log.info(h + " - It is char");
29 log.info(i + " - It is not char");
```

Beanshell Preprocessor

PreProcessors are Jmeter elements that are used to execute actions before the sampler requests are executed in the test scenario. **PreProcessors** can be used for different performance testing needs, like fetching data from a database, setting a timeout between sampler execution or before test data generation.

To add the *BeanShell* PreProcessor to the Sampler request:

Right Click on the Sampler -> Add -> Pre Processors -> BeanShell PreProcessor

Beanshell Preprocessor

BeanShell PreProcessor

Name: BeanShell PreProcessor

Comments:

Reset bsh.Interpreter before each call

Reset Interpreter: ☐ False

Parameters to be passed to BeanShell (=> String Parameters and String []bsh.args)

Parameters:

Script file (overrides script)

File Name:

Script (variables: ctx vars props prev sampler log)

Script:

1

Beanshell Preprocessor

BeanShell PreProcessor configs:

Name - the name of the sampler shown in the Thread Group tree

Reset bsh.Interpreter before each call - Resets the interpreter before each call and cleans occupied memory. Setting this option as “True” might be useful for long running scripts, since repeated invocations might consume a lot of memory

Parameters - JMeter parameters that will be passed to the BeanShell script. You need to keep in mind that you cannot use JMeter variables in the BeanShell preprocessor if they are not specified in this configuration field. If a parameter is specified, you can use it in the preprocessor like this:

```
String BS_Variable_Name = vars.get("JMeterVariable");
```

File Name - the path to an external BeanShell script that needs to be run

Beanshell Preprocessor

Script:

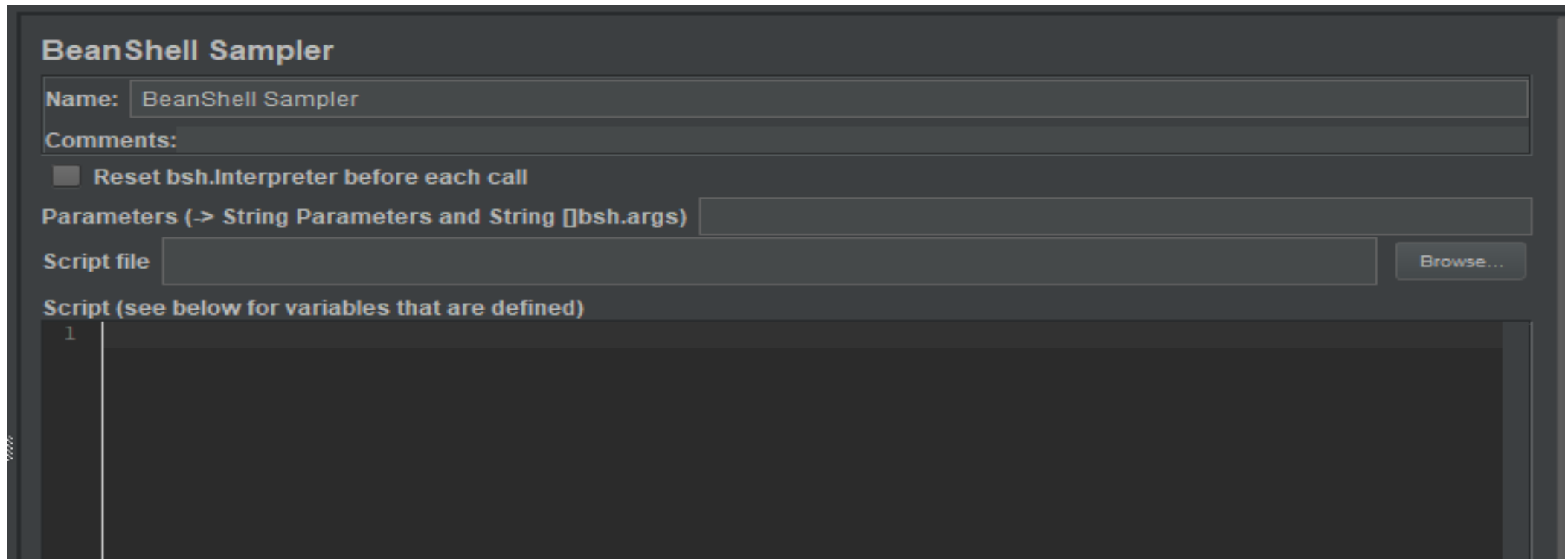
```
1 import java.util.Random;
2
3 chars = "1234567890abcdefghijklmnopqrstuvwxyz-";
4 int string_length = 36;
5 randomstring = "";
6
7 for (int i=0; i < string_length; i++) {
8     Random randomGenerator = new Random();
9     int randomInt = randomGenerator.nextInt(chars.length());
10    randomstring += chars.substring(randomInt,randomInt+1);
11 }
12 log.info("-----BeanShell PreProcessor-----");
13 log.info(randomstring);
14 log.info("-----BeanShell PreProcessor-----");
15 vars.put("RANDOM_STRING",randomstring);
```

```
done! Engine: All thread groups have been started
Thread: Thread started: Thread Group 1-1
allTestElement: -----BeanShell PreProcessor-----
allTestElement: fib4ktapx5rs6e-75x38lwbe8wkd1685-iuk
allTestElement: -----BeanShell PreProcessor-----
allTestElement: -----
```

BeanShell Sampler

In order to manipulate and work on the response data, BeanShell sampler would be a great element.

To add BeanShell Sampler, right click, **Add-> Sampler-> BeanShell sampler**



The screenshot shows the configuration window for a BeanShell Sampler. The title bar reads "BeanShell Sampler". Inside the window, there is a "Name:" field with the text "BeanShell Sampler". Below it is a "Comments:" text area. A checkbox labeled "Reset bsh.Interpreter before each call" is currently unchecked. The "Parameters (-> String Parameters and String []bsh.args)" field is empty. The "Script file" field is also empty, with a "Browse..." button to its right. At the bottom, there is a "Script (see below for variables that are defined)" text area with a line number "1" visible on the left margin.

BeanShell Sampler

Script (see below for variables that are defined)

```
1 import java.net.HttpURLConnection;
2 import java.net.URL;
3 import java.nio.charset.StandardCharsets;
4 import org.apache.commons.io.IOUtils;
5
6 URL url = new URL("https://api.tiki.vn/personalization/v2/products?page=1&limit=10");
7 HttpURLConnection con = (HttpURLConnection) url.openConnection();
8 con.setRequestMethod("GET");
9 String response = IOUtils.toString(con.getInputStream(), StandardCharsets.UTF_8);
10 return response;
```

The screenshot shows the JMeter BeanShell Sampler interface. On the left, a list of samplers is shown, all named "BeanShell Call tiki.vn". The right pane displays the "Response data" for the selected sampler, showing a JSON response body. The response body contains an array of product data, including details like default_spid, discount_rate, id, name, price, and thumbnail_url.

Text

BeanShell Call tiki.vn

BeanShell Call tiki.vn

BeanShell Call tiki.vn

Sampler result Request Response data

Response Body Response headers

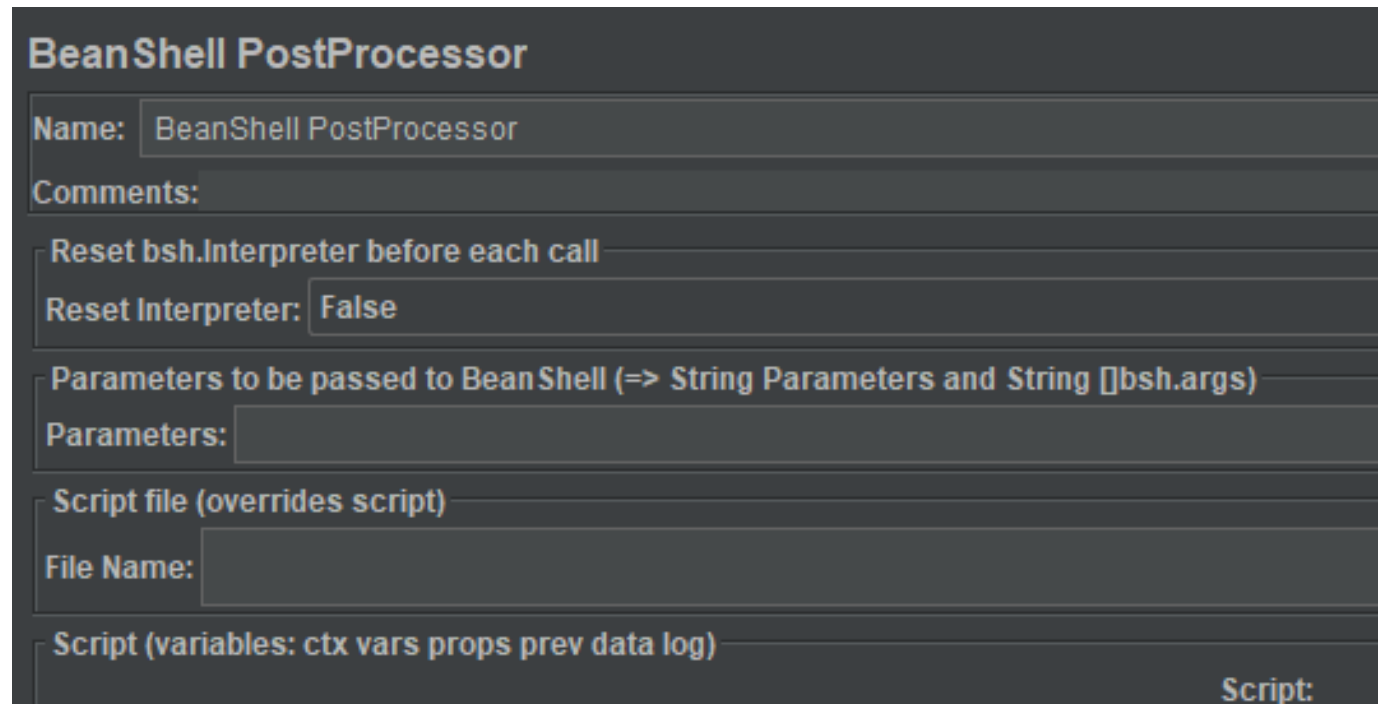
Response data

```
{
  "data": [
    {
      "default_spid": "146473",
      "discount_rate": 33.0,
      "id": 440702,
      "name": "Canon 750D + Lens 18-55 IS STM (L\u00e0m s\u00e1n h\u00e0ng)",
      "price": 20490000,
      "thumbnail_url": "https://salt.tikicdn.com/cache/280x280/00/47/df/b02b462394bc3c59e5876ec0d9cb55-is-stm-le-bao-minh-p440702",
      "type": "configurable",
      "url_key": "dien-thoi-iphone-x-64gb-128gb",
      "url": "https://api.tiki.vn/personalization/v2/products?page=1&limit=10"
    },
    {
      "default_spid": "684982",
      "discount_rate": 49.0,
      "id": 732475,
      "name": "iPhone 7 32GB VN/A",
      "price": 29990000,
      "thumbnail_url": "https://salt.tikicdn.com/cache/280x280/b7/15/fa/892e3ng",
      "type": "configurable",
      "url_key": "dien-thoi-iphone-x-64gb-128gb",
      "url": "https://api.tiki.vn/personalization/v2/products?page=1&limit=10"
    }
  ]
}
```

BeanShell PostProcessor

Consider a simple example: Jmeter send HTTP Request to the web server under test and get the response.

To add PostProcessor, right click the BeanShell Sampler, **Add-> PostProcessor > BeanShell PostProcessor**



The screenshot shows the configuration window for a BeanShell PostProcessor. The title bar reads "BeanShell PostProcessor". The "Name" field is set to "BeanShell PostProcessor". The "Comments" field is empty. There is a checkbox labeled "Reset bsh.Interpreter before each call" which is currently unchecked. Below it, the "Reset Interpreter:" field is set to "False". There is a checkbox labeled "Parameters to be passed to BeanShell (=> String Parameters and String []bsh.args)" which is currently unchecked. Below it, the "Parameters:" field is empty. There is a checkbox labeled "Script file (overrides script)" which is currently unchecked. Below it, the "File Name:" field is empty. At the bottom, there is a checkbox labeled "Script (variables: ctx vars props prev data log)" which is currently unchecked. To the right of this checkbox is a label "Script:".

BeanShell PostProcessor

```
//access to previous result  
log.info("Previous Response Message is: " + ctx.getPreviousResult().getResponseMessage());  
log.info("Previous Response Code is: " + ctx.getPreviousResult().getResponseCode());  
log.info("Previous Response URL is: " + ctx.getPreviousResult().getURL());  
log.info("Previous Response Time is: " + ctx.getPreviousResult().getTime());
```

BeanShell PostProcessor

```
import net.minidev.json.JSONArray;
import net.minidev.json.JSONObject;
import net.minidev.json.parser.JSONParser;
import org.apache.commons.lang.StringUtils;
import org.apache.jmeter.threads.JMeterVariables;
```

```
//Get product total count
int totalStoreNumber = StringUtils.countMatches(new String(data), "name");
log.info("Total Number of product are: " + totalStoreNumber);

if (totalStoreNumber > 0) {
    String jsonString = new String(data);
    JSONParser parser = new JSONParser(JSONParser.MODE_JSON_SIMPLE);
    JSONObject store = (JSONObject) parser.parse(data);
    JSONArray storeArray = (JSONArray) store.get("data");
    for( int i=0; i<totalStoreNumber; i++ ) {
        log.info("-----" + i);
        log.info("id: " + ((JSONObject) storeArray.get(i)).getAsString("id"));
        log.info("name: " + ((JSONObject) storeArray.get(i)).getAsString("name"));
        log.info("price: " + ((JSONObject) storeArray.get(i)).getAsString("price"));
        log.info("-----");
    }
    vars.put("totalStoreNumber", totalStoreNumber+"");
} else {
    log.info("No Product");
}
```


BeanShell Assertion

BeanShell Assertion

Name: BeanShell Assertion

Comments:

☐ **Reset bsh.interpreter before each call**

Parameters (-> String Parameters and String []bsh.args)

Script file

Script (see below for variables that are defined)

```
1 String totalStoreNumber = vars.get("totalStoreNumber");
2
3 Failure = !totalStoreNumber.equals("9");
4
5 if (Failure) {
6     FailureMessage = "Variables are not equal. Expected 9 products , actual:\'" + totalStoreNumber + "\'";
7 }
```

Name - Name of the BeanShell Assertion

Comments - you can provide any comments regarding this assertion

Reset bsh.interpreter before each call - In case this option is selected, then the interpreter will be recreated for each samples and it is useful only when your scripts runs for longer time.

Parameters - Specified parameters to pass into your Beanshell script

Script File - In case you have script file ,you can run your assertion with the file with proper script in the file and path of the script file.

Script - This area provides to option to create a script to validate the response of samples.

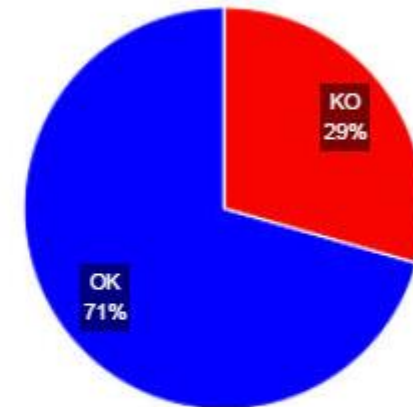
Dashboard Generator

Apache JMeter is an open source performance tool which is helpful in testing load test for WebApplications and WebServices, REST API services. In latest JMeter 3.0 version apache software included Generate Jmeter Report Dashboard using APDEX(Application Performance Index).

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.329	500 ms	1 sec 500 ms	Total
0.000	500 ms	1 sec 500 ms	HP ALM QC
0.395	500 ms	1 sec 500 ms	Appium Interview Questions
0.458	500 ms	1 sec 500 ms	Selenium Tutorials
0.465	500 ms	1 sec 500 ms	QTP Tutorials

Requests Summary



Dashboard Generator

DATETIME: \${__time(MM-dd-yyyy-HH-mm-ss,,)}
report/\${REPORT_BASE}/\${DATETIME}/Summary.csv

User Defined Variables

Name: User Defined Variables

Comments:

User Defined Variables	
Name:	Value
REPORT_BASE	FPT
DATETIME	\${__time(MM-dd-yyyy-HH-mm-ss,,)}

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: report/\${REPORT_BASE}/\${DATETIME}/Summary.csv

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	T
BeanShell Cal...	1	333	333	333	0.00	0.00%	
TOTAL	1	333	333	333	0.00	0.00%	

Jmeter Report Generation Configuration


In Apache jmeter 3.0 version it has given reportgeneration properties file ,if you open that file it will display all report generation configuration details,simply copy those **jmeter.reportgenerator** configuration in **User Properties file** as below.

reportgenerator.properties	3/10/2019 10:08 AM	PROPERTIES File	12 KB
saveservice.properties	3/10/2019 10:08 AM	PROPERTIES File	25 KB
shutdown	3/10/2019 8:36 AM	Windows Comma...	2 KB
shutdown.sh	3/10/2019 8:36 AM	SH File	2 KB
stoptest	3/10/2019 8:36 AM	Windows Comma...	2 KB
stoptest.sh	3/10/2019 8:36 AM	SH File	2 KB
system.properties	3/10/2019 10:08 AM	PROPERTIES File	5 KB
Test Plan	4/14/2019 8:28 PM	JMX File	8 KB
threaddump.sh	3/10/2019 8:36 AM	SH File	2 KB
upgrade.properties	3/10/2019 10:08 AM	PROPERTIES File	8 KB
user.properties	3/10/2019 10:08 AM	PROPERTIES File	7 KB

Dashboard Generator

Test Results - CSV file:

is PC > Documents > Automation Tester > Automation Tester > apache-jmeter-5.1.1 > apache-jmeter-5.1.1 > bin > report > FPT > 04-19-2019-00-03-42

Name	Date modified	Type	Size
 Summary	4/19/2019 12:03 AM	Microsoft Excel C...	1 KB

```
Summary - Notepad
File Edit Format View Help
timestamp,elapsed,label,responseCode,responseMessage,threadName,dataType,success,failureMes
1555607022569,164,BeanShell Call tiki.vn,200,OK,Thread Group 1-1,text,true,,3744,0,1,1,null
```

Dashboard Generator

Now open Command Prompt.

Go to ApacheJmeter3.0/bin path as below

Enter below command to generate the reports as per test results csv file.

jmeter -g E:\JMETER_Tutorials\Summary.csv -o E:\JMETER_Tutorials\HTMLReports

```
C:\Users\ADMIN\Documents\Automation Tester\Automation Tester\apache-jmeter-5.1.1\apache-jmeter-5.1.1\bin>jmeter -g C:\Users\ADMIN\Documents\report\FPT\04-19-2019-00-03-42\Summary.csv -o C:\Users\ADMIN\Documents\report\FPT\04-19-2019-00-03-42\HTMLReports
C:\Users\ADMIN\Documents\Automation Tester\Automation Tester\apache-jmeter-5.1.1\apache-jmeter-5.1.1\bin>
```

Dashboard Generator

This PC > Documents > report > FPT > 04-19-2019-00-03-42 > HTMLReports				
	Name	Date modified	Type	Size
★	content	4/19/2019 12:09 AM	File folder	
★	sbadmin2-1.0.7	4/19/2019 12:09 AM	File folder	
★	index	4/19/2019 12:09 AM	Chrome HTML Do...	10 KB
★	statistics.json	4/19/2019 12:09 AM	JSON File	1 KB

Dashboard Generator

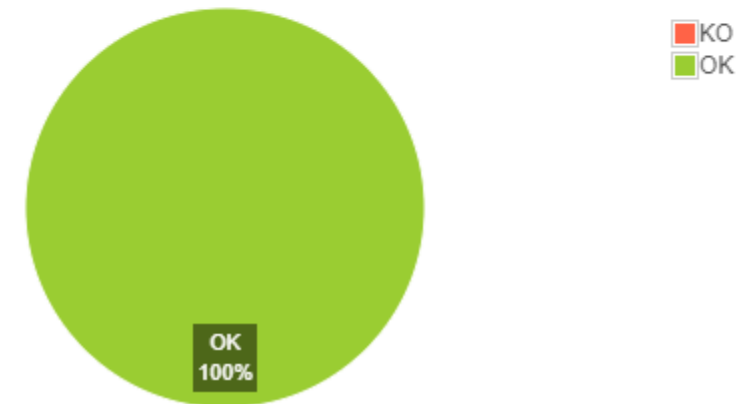
Test and Report informations

Source file	"Summary.csv"
Start Time	"4/19/19 12:03 AM"
End Time	"4/19/19 12:03 AM"
Filter for display	""

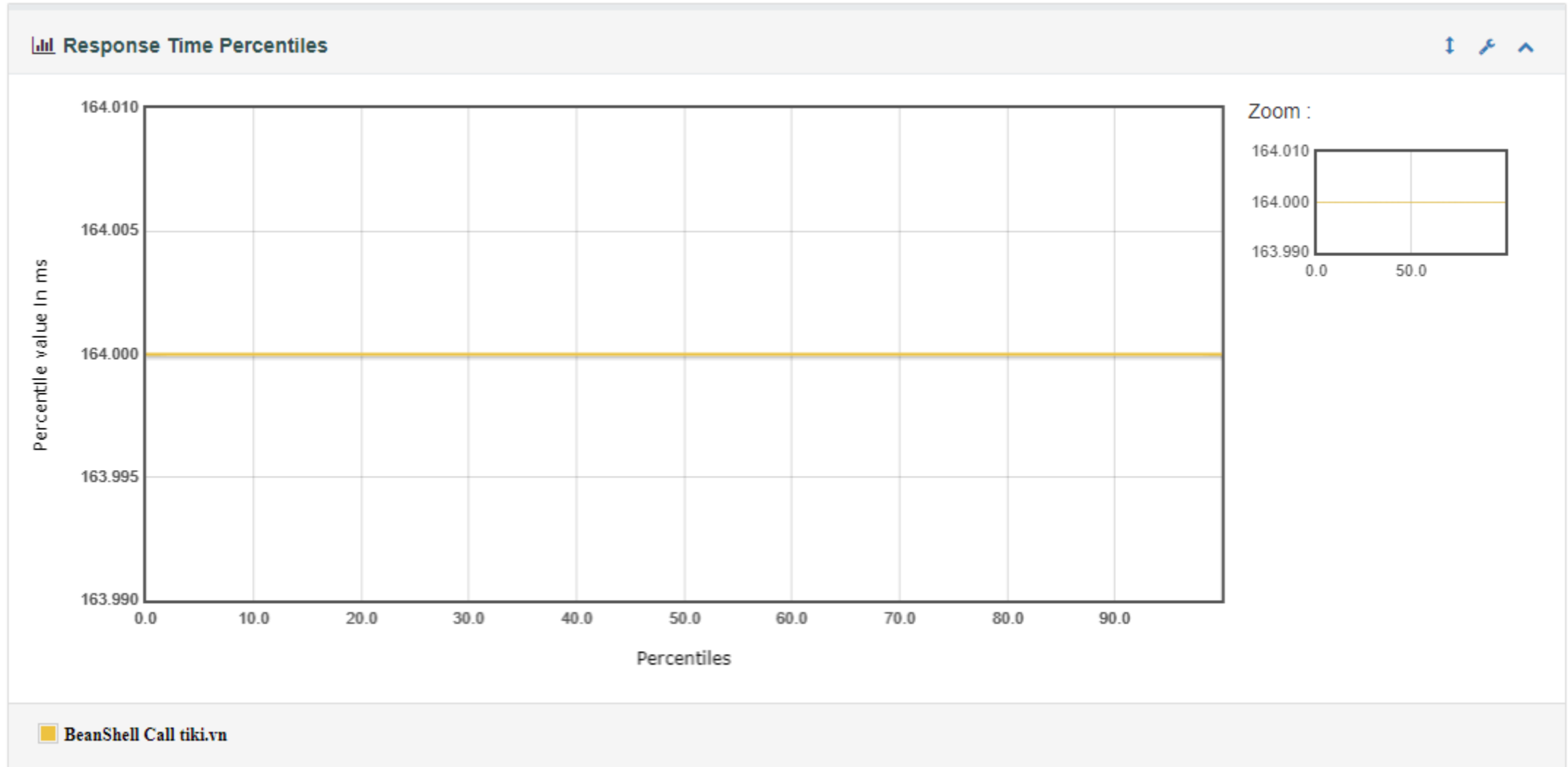
APDEX (Application Performance Index)

Apdex ▲	T (Toleration threshold) ◆	F (Frustration threshold) ◆	Label ◆
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	BeanShell Call tiki.vn







Requests Summary



Dashboard Generator



Dashboard Generator

Apache JMeter Dashboard	
 Dashboard	
 Charts 	
Over Time	
Throughput	
Response Times 	
Response Time Percentiles	
Response Time Overview	
Time Vs Threads	
Response Time Distribution	
 Customs Graphs 	



Fresher Academy



Happy Coding!