Jmeter Performance Testing

Winter 2016

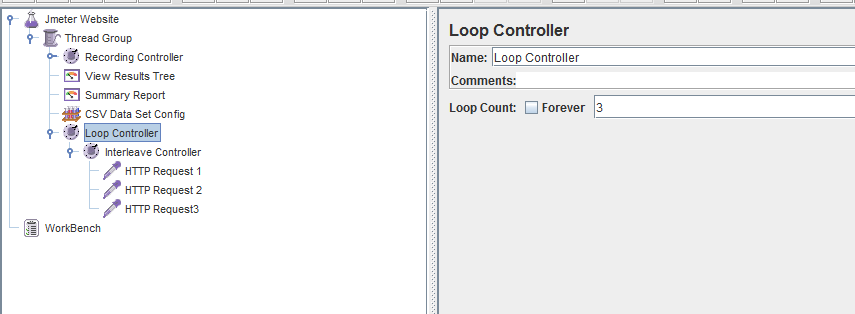
Contents

[1. Week 5: Jmeter Components 4](#_Toc467782914)

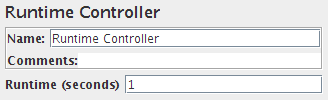
[2. Contact Us 9](#_Toc467782915)

# Week 5: Jmeter Components

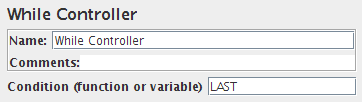
* **Logic Controller**
  + **Once Only Controller**: The Once Only Logic Controller tells JMeter to process the controller(s) inside it only once per Thread. This means if this controller is under the Loop Controller to loop 5 times, it still process only 1 times any sampler or controller inside.
  + **Interleave Controller**: if more than 1 sampler or controller under this controller, Jmeter will run each of sampler/controller one time for each loop iteration.



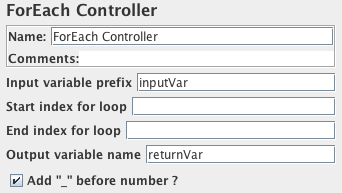
* + **Runtime Controller**: The Runtime Controller controls how long its children are allowed to run.



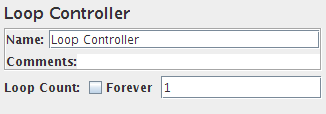
* + **While Controller**: the While Controller runs its children until the condition is "false".
    - blank - exit loop when last sample in loop fails
    - LAST - exit loop when last sample in loop fails. If the last sample just before the loop failed, don't enter loop.
    - Otherwise - exit (or don't enter) the loop when the condition is equal to the string "false



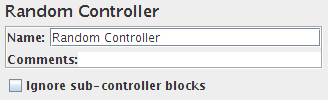
* + **Switch Controller**: the Switch Controller acts like the Interleave Controller in that it runs one of the subordinate elements on each iteration, but rather than run them in sequence, the controller runs the element defined by the switch value. The sequence of sub element starts from 0.
  + **ForEach Controller**: A ForEach controller loops through the values of a set of related variables. When you add samplers (or controllers) to a ForEach controller, every sample (or controller) is executed one or more times, where during every loop the variable has a new value. The input should consist of several variables, each extended with an underscore and a number. Each such variable must have a value. So for example when the input variable has the name inputVar, the following variables should have been defined:
    - inputVar\_1 = wendy
    - inputVar\_2 = charles
    - inputVar\_3 = peter
    - inputVar\_4 = john



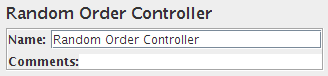
* + **Loop Controller:** loop through all the sub-elements a certain number of times



* + **Random Controller:** this controller will randomly run its sub-element.



* + **Random Order Controller:** this controller is much like a Simple Controller in that it will execute each child element at most once, but the order of execution of the nodes will be random.

 **c**

* **Config Element**: some useful and popular config element:
  + **CSV Data Set Confi**g: is used to read lines from a file, and split them into variables. It is easier to use than the \_\_CSVRead() and \_StringFromFile() functions. It is well suited to handling large numbers of variables, and is also useful for testing with "random" and unique values. Generating unique random values at run-time is expensive in terms of CPU and memory, so just create the data in advance of the test. If necessary, the "random" data from the file can be used in conjunction with a run-time parameter to create different sets of values from each run - e.g. using concatenation - which is much cheaper than generating everything at run-time.
  + **HTTP Cookie Manager:** this element stores and sends cookies just like a web browser. If you have an HTTP Request and the response contains a cookie, the Cookie Manager automatically stores that cookie and will use it for all future requests to that particular web site.
  + **HTTP Header Manager:** The Header Manager lets you add or override HTTP request headers. JMeter now supports multiple Header Managers. The header entries are merged to form the list for the sampler. If an entry to be merged matches an existing header name, it replaces the previous entry, unless the entry value is empty, in which case any existing entry is removed. This allows one to set up a default set of headers, and apply adjustments to particular samplers.
  + **User Defined Variables:** The User Defined Variables element lets you define an initial set of variables, just as in the Test Plan.
  + **Random Variable:** The Random Variable Config Element is used to generate random numeric strings and store them in variable for use later. It's simpler than using User Defined Variables together with the \_\_Random() function.

The output variable is constructed by using the random number generator, and then the resulting number is formatted using the format string. The number is calculated using the formula minimum+Random.nextInt(maximum-minimum+1). Random.nextInt() requires a positive integer. This means that maximum-minimum - i.e. the range - must be less than 2147483647, however the minimum and maximum values can be any long values so long as the range is OK.

* + **Counter:** Allows the user to create a counter that can be referenced anywhere in the Thread Group. The counter config lets the user configure a starting point, a maximum, and the increment. The counter will loop from the start to the max, and then start over with the start, continuing on like that until the test is ended
* **Timer**
  + **Constant Timer:** If you want to have each thread pause for the same amount of time between requests, use this timer.
  + **Gaussian Random Timer:** This timer pauses each thread request for a random amount of time, with most of the time intervals occurring near a particular value.
  + **Uniform Random Timer:** This timer pauses each thread request for a random amount of time, with each time interval having the same probability of occurring. The total delay is the sum of the random value and the offset value.
  + **Constant Throughput Timer:** This timer introduces variable pauses, calculated to keep the total throughput (in terms of samples per minute) as close as possible to a give figure. Of course the throughput will be lower if the server is not capable of handling it, or if other timers or time-consuming test elements prevent it. N.B. although the Timer is called the Constant Throughput timer, the throughput value does not need to be constant. It can be defined in terms of a variable or function call, and the value can be changed during a test.
  + **Synchronizing Timer:** The purpose of the SyncTimer is to block threads until X number of threads have been blocked, and then they are all released at once. A SyncTimer can thus create large instant loads at various points of the test plan.
  + **BeanShell Timer:** The BeanShell Timer can be used to generate a delay.
  + **BSF Timer** (DEPRECATED): The BSF Timer can be used to generate a delay using a BSF scripting language.
  + **Poisson Random Timer:** This timer pauses each thread request for a random amount of time, with most of the time intervals occurring near a particular value. The total delay is the sum of the Poisson distributed value, and the offset val
* **Pre Processors:**
  + **User Parameters:** Allows the user to specify values for User Variables specific to individual threads. User Variables can also be specified in the Test Plan but not specific to individual threads. This panel allows you to specify a series of values for any User Variable. For each thread, the variable will be assigned one of the values from the series in sequence. If there are more threads than values, the values get re-used. For example, this can be used to assign a distinct user id to be used by each thread. User variables can be referenced in any field of any jMeter Component.
  + **RegEx User Parameters:** Allows to specify dynamic values for HTTP parameters extracted from another HTTP Request using regular expressions. RegEx User Parameters are specific to individual threads.This component allows you to specify reference name of a regular expression that extracts names and values of HTTP request parameters. Regular expression group numbers must be specified for parameter's name and also for parameter's value. Replacement will only occur for parameters in the Sampler that uses this RegEx User Parameters which name matches
  + **JDBC PreProcessor:** to run some SQL statement just before a sample runs. This can be useful if your JDBC Sample requires some data to be in DataBase and you cannot compute this in a setup Thread group.
* **Sampler:**
  + **HTTP Request:** This sampler lets you send an HTTP/HTTPS request to a web server. It also lets you control whether or not JMeter parses HTML files for images and other embedded resources and sends HTTP requests to retrieve them. The following types of embedded resource are retrieved:
    - images
    - applets
    - stylesheets
    - external scripts
    - frames, iframes
    - background images (body, table, TD, TR)
    - background sound
  + **JDBC Request**: This sampler lets you send a JDBC Request (an SQL query) to a database. Before using this you need to set up a JDBC Connection Configuration Configuration element.
  + **Dummy Sampler:** Dummy Sampler is the most obedient of the JMeter samplers: it can generate fake samplers with defined values. This sampler makes it simpler to test when you have complex scripts or when running a test takes too much time. All you need is to just copy and paste “Response data” into “Dummy Sampler”. When you run the test, the “Dummy Sampler” generates a sampler with pasted response data.
  + **SOAP/XML-RPC Request:** This sampler lets you send a SOAP request to a webservice.
  + **Rest API Request:** When Web services use REST architecture, they are called RESTful APIs (Application Programming Interfaces) or REST API. To send a Rest API request, we use HTTP Request sampler.
  + **Debug Sampler:** will give all the element data that will help you figure out the actual problem.“Debug Sampler” has three options: “JMeter properties”, “JMeter variables”, and “System properties”. Enabling or disabling these options will print out or ignore properties.
* **Post Processors:** 
  + **BSF PostProcessor:** The BSF PostProcessor allows BSF script code to be applied after taking a sample.
  + **JDBC PostProcessor:** The JDBC PostProcessor enables you to run some SQL statement just after a sample has run. This can be useful if your JDBC Sample changes some data and you want to reset state to what it was before the JDBC sample run.
  + **Regular Expression Extractor:** Allows the user to extract values from a server response using a Perl-type regular expression. As a post-processor, this element will execute after each Sample request in its scope, applying the regular expression, extracting the requested values, generate the template string, and store the result into the given variable name.
  + **JSON Path PostProcessor:** The JSON PostProcessor enables you extract data from JSON responses using JSON-PATH syntax. This post processor is very similar to Regular expression extractor. It must be placed as a child of HTTP Sampler or any other sampler that has responses. It will alow you to extract in a very easy way text content, see JSON Path syntax.