About

- Overview
- Changes
- Known Bugs
- License
- Contributors

Download

- Download Releases
- Developer (Nightly) Builds

Documentation

- User Manual
- Javadocs
- Localisation (Translator's Guide)
- Building JMeter and Add-Ons
- JMeter Wiki
- FAQ (Wiki)

Tutorials (PDF format)

- Distributed Testing
- Recording Tests
- JUnit Sampler
- Access Log Sampler
- Extending JMeter

Community

- Get Involved
- Mailing Lists
- SVN Repositories

Index Next Prev

19. Functions and Variables

JMeter functions are special values that can populate fields of any Sampler or other element in a test tree. A function call looks like this:

```
${__functionName(var1,var2,var3)}
```

Where "__functionName" matches the name of a function.
Parentheses surround the parameters sent to the function, for example ${__time(YMD)}
The actual parameters vary from function to function. Functions that require no parameters can leave off the parentheses, for example ${__threadNum}.

If a function parameter contains a comma, then be sure to escape this with "\", otherwise JMeter will treat it as a parameter delimiter. For example:

```
${__time(EEE\, d MMM yyyy)}
```

Variables are referenced as follows:

```
${VARIABLE}
```

If an undefined function or variable is referenced, JMeter does not report/log an error - the reference is returned unchanged. For example if UNDEF is not defined as a variable, then the value of ${UNDEF} is ${UNDEF}. Variables, functions (and properties) are all case-sensitive. **Versions of JMeter after 2.3.1 trim spaces from variable names before use, so for example ${__Random(1,63, LOTTERY)} will use the variable 'LOTTERY' rather than ' LOTTERY '.**

Properties are not the same as variables. Variables are local to a thread; properties are common to all threads, and need to be referenced using the __P or __property function.

List of functions, loosely grouped into types.

Type of function	Name	Comment
Information	threadNum	get thread number
Information	machineName	get the local machine name
Information	time	return current time in various formats
Information	log	log (or display) a message (and return the value)
Information	logn	log (or display) a message (empty return value)
Input	StringFromFile	read a line from a file
Input	CSVRead	read from CSV delimited file
Input	XPath	Use an XPath expression to read from a file
Calculation	counter	generate an incrementing number
Calculation	intSum	add int numbers
Calculation	longSum	add long numbers
Calculation	Random	generate a random number
Calculation	regexFunction	parse previous response using a regular expression
Scripting	BeanShell	run a BeanShell script
Scripting	javaScript	process JavaScript (Mozilla Rhino)
Scripting	jexl	evaluate a Commons Jexl expression
Properties	property	read a property
Properties	P	read a property (shorthand method)
Properties	setProperty	set a JMeter property
Variables	split	Split a string into variables
Variables	V	evaluate a variable name
Variables	eval	evaluate a variable expression
Variables	evalVar	evaluate an expression stored in a variable

19.1 What can functions do

There are two kinds of functions: user-defined static values (or variables), and built-in functions.
User-defined static values allow the user to define variables to be replaced with their static value when a test tree is compiled and submitted to be run. This replacement happens once at the beginning of the test run. This could be used to replace the DOMAIN field of all HTTP requests, for example - making it a simple matter to change a test to target a different server with the same test.

Note that variables cannot currently be nested; i.e ${Var${N}} does not work. The __V (variable) function (versions after 2.2) can be used to do this: ${__V(Var${N})}. In earlier JMeter versions one can use ${__BeanShell(vars.get("Var${N}")}.

This type of replacement is possible without functions, but was less convenient and less intuitive. It required users to create default config elements that would fill in blank values of Samplers. Variables allow one to replace only part of any given value, not just filling in blank values.

With built-in functions users can compute new values at run-time based on previous response data, which thread the function is in, the time, and many other sources. These values are generated fresh for every request throughout the course of the test.

Functions are shared between threads. Each occurrence of a function call in a test plan is handled by a separate function instance.

19.2 Where can functions and variables be used?

Functions and variables can be written into any field of any test component (apart from the TestPlan - see below). Some fields do not allow random strings because they are expecting numbers, and thus will not accept a function. However, most fields will allow functions.

Functions which are used on the Test Plan have some restrictions. JMeter thread variables will have not been fully set up when the functions are processed, so variable names passed as parameters will not be set up, and variable references will not work, so split() and regex() and the variable evaluation functions won't work. The threadNum() function won't work (and does not make sense at test plan level). The following functions should work OK on the test plan:

- intSum
- longSum
- machineName
- BeanShell
- javaScript
- jexl
- random
- time
- property functions
- log functions

Configuration elements are processed by a separate thread. Therefore functions such as __threadNum do not work properly in elements such as User Defined Variables. Also note that variables defined in a UDV element are not available until the element has been processed.

When using variable/function references in SQL code (etc), remember to include any necessary quotes for text strings, i.e. use SELECT item from table where name='${VAR}' **not** SELECT item from table where name=${VAR} (unless VAR itself contains the quotes)

19.3 How to reference variables and functions

Referencing a variable in a test element is done by bracketing the variable name with '${' and '}'.

Functions are referenced in the same manner, but by convention, the names of functions begin with "__" to avoid conflict with user value names *. Some functions take arguments to configure them, and these go in parentheses, comma-delimited. If the function takes no arguments, the parentheses can be omitted.

Argument values that themselves contain commas should be escaped as necessary. If you need to include a comma in your parameter value, escape it like so: '\,'. This applies for example to the scripting functions - Javascript, Beanshell, Jexl - where it is necessary to escape any commas that may be needed in script method calls - e.g.

```
${__BeanShell(vars.put("name"\,"value"))}
```

Functions can reference variables and other functions, for example `${__XPath(${__P(xpath.file),${XPATH})}` will use the property "xpath.file" as the file name and the contents of the variable XPATH as the expression to search for.

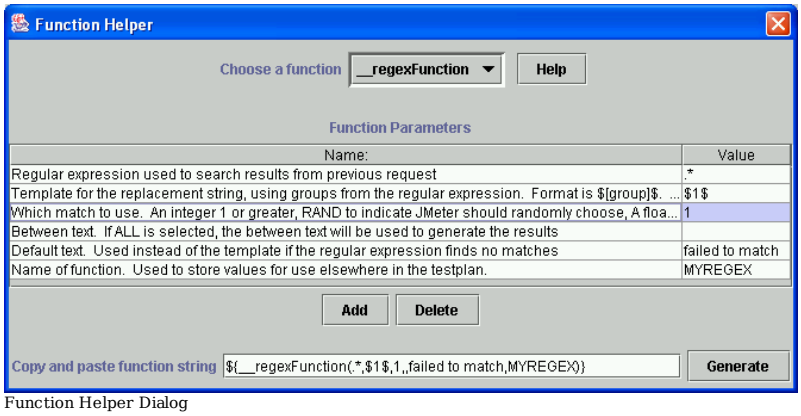
JMeter provides a tool to help you construct function calls for various built-in functions, which you can then copy-paste. It will not automatically escape values for you, since functions can be parameters to other functions, and you should only escape values you intend as literal.

The value of a variable or function can be reported using the __logn() function. The __logn() function reference can be used anywhere in the test plan after the variable has been defined. Alternatively, the Java Request sampler can be used to create a sample containing variable references; the output will be shown in the appropriate Listener. For versions of JMeter later than 2.3, there is a Debug Sampler that can be used to display the values of variables etc in the Tree View Listener.

* If you define a user-defined static variable with the same name as a built-in function, your static variable will override the built-in function.

19.4 The Function Helper Dialog

The Function Helper dialog is available from JMeter's Tools menu.



Function Helper Dialog

Using the Function Helper, you can select a function from the pull down, and assign values for its arguments. The left column in the table provides a brief description of the argument, and the right column is where you write in the value for that argument. Different functions take different arguments.

Once you have done this, click the "generate" button, and the appropriate string is generated for you to copy-paste into your test plan wherever you like.

19.5 Functions

19.5.1 __regexFunction

The Regex Function is used to parse the previous response (or the value of a variable) using any regular expression (provided by user). The function returns the template string with variable values filled in.

The __regexFunction can also store values for future use. In the sixth parameter, you can specify a reference name. After this function executes, the same values can be retrieved at later times using the syntax for user-defined values. For instance, if you enter "refName" as the sixth parameter you will be able to use:

- ${refName} to refer to the computed result of the second parameter ("Template for the replacement string") parsed by this function
- ${refName_g0} to refer to the entire match parsed by this function.
- ${refName_g1} to refer to the first group parsed by this function.
- ${refName_g#} to refer to the n th group parsed by this function.
- ${refName_matchNr} to refer to the number of groups found by this function.

Parameters

Attribute	Description	Required
First argument	The first argument is the regular expression to be applied to the response data. It will grab all matches. Any parts of this expression that you wish to use in your template string, be sure to surround in parentheses. Example: <a href="(.*)">. This will grab the value of the link and store it as the first group (there is only 1 group). Another example: <input type="hidden" name="(.*)" value="(.*)">. This will grab the name as the first group, and the value as the second group. These values can be used in your template string	Yes
Second argument	This is the template string that will replace the function at run-time. To refer to a group captured in the regular expression, use the syntax: $[group_number]$. Ie: 1, or 2. Your template can be any string.	Yes
Third argument	The third argument tells JMeter which match to use. Your regular expression might find numerous matches. You have four choices: • An integer - Tells JMeter to use that match. '1' for the first found match, '2' for the second, and so on • RAND - Tells JMeter to choose a match at random. • ALL - Tells JMeter to use all matches, and create a template string for each one and then append them all together. This option is little used. • A float number between 0 and 1 - tells JMeter to find the Xth match using the formula: (number_of_matches_found * float_number) rounded to nearest integer.	No, default=1
Fourth argument	If 'ALL' was selected for the above argument value, then this argument will be inserted between each appended copy of the template value.	No
Fifth argument	Default value returned if no match is found	No
Sixth argument	A reference name for reusing the values parsed by this function. Stored values are ${refName} (the replacement template string) and ${refName_g#} where "#" is the group number from the regular expression ("0" can be used to refer to the entire match).	No

Seventh argument	Input variable name. If specified, then the value of the variable is used as the input instead of using the previous sample result.	No

---

19.5.2 __counter

The counter generates a new number each time it is called, starting with 1 and incrementing by +1 each time. The counter can be configured to keep each simulated user's values separate, or to use the same counter for all users. If each user's values is incremented separately, that is like counting the number of iterations through the test plan. A global counter is like counting how many times that request was run.

The counter uses an integer variable to hold the count, which therefore has a maximum of 2,147,483,647.

The counter function instances are now completely independent. [JMeter 2.1.1 and earlier used a fixed thread variable to keep track of the per-user count, so multiple counter functions operated on the same value.] The global counter - "FALSE" - is separately maintained by each counter instance.

Parameters

Attribute	Description	Required
First argument	TRUE if you wish each simulated user's counter to be kept independent and separate from the other users. FALSE for a global counter.	Yes
Second argument	A reference name for reusing the value created by this function. Stored values are of the form ${refName}. This allows you to keep one counter and refer to its value in multiple places. [For JMeter 2.1.1 and earlier this parameter was required.]	No

---

19.5.3 __threadNum

The thread number function simply returns the number of the thread currently being executed. These numbers are independent of ThreadGroup, meaning thread #1 in one threadgroup is indistinguishable from thread #1 in another threadgroup, from the point of view of this function.

There are no arguments for this function.

This function does not work in any Configuration elements (e.g. User Defined Variables) as these are run from a separate thread. Nor does it make sense to use it on the Test Plan.

19.5.4a __intSum

The intSum function can be used to compute the sum of two or more integer values.

JMeter Versions 2.3.1 and earlier required the reference name to be present. The reference name is now optional, but it must not be a valid integer.

Parameters

Attribute	Description	Required
First argument	The first int value.	Yes
Second argument	The second int value.	Yes
nth argument	The nth int value.	No
last argument	A reference name for reusing the value computed by this function. If specified, the reference name must contain at least one non-numeric character otherwise it will be treated as another int value to be added.	No

---

19.5.4b __longSum

The longSum function can be used to compute the sum of two or more long values.

Parameters

Attribute	Description	Required
First argument	The first long value.	Yes
Second argument	The second long value.	Yes
nth argument	The nth long value.	No
last argument	A reference name for reusing the value computed by this function. If specified, the reference name must contain at least one non-numeric character otherwise it will be treated as another long value to be added.	No

---

19.5.5 __StringFromFile

The StringFromFile function can be used to read strings from a text file. This is useful for running tests that require lots of variable data. For example when testing a banking application, 100s or 1000s of different account numbers might be required.

See also the CSV Data Set Config test element which may be easier to use. However, that does not currently support multiple input files.

Each time it is called it reads the next line from the file. When the end of the file is reached, it will start reading again from the beginning, unless the maximum loop count has been reached. If there are multiple references to the function in a test script, each will open the file independently, even if the file names are the same. [If the value is to be used again elsewhere, use different variable names for each function call.]

If an error occurs opening or reading the file, then the function returns the string "**ERR**"

Parameters

Attribute	Description	Required
File Name	Path to the file name. (The path can be relative to the JMeter launch directory) If using optional sequence numbers, the path name should be suitable for passing to DecimalFormat. See below for examples.	Yes
Variable Name	A reference name - refName - for reusing the value created by this function. Stored values are of the form ${refName}. Defaults to "StringFromFile_".	No

Start sequence number	Initial Sequence number (if omitted, the End sequence number is treated as a loop count)	No
End sequence number	Final sequence number (if omitted, seqence numbers can increase without limit)	No

The file name parameter is resolved when the file is opened or re-opened.

The reference name parameter (if supplied) is resolved every time the function is executed.

Using sequence numbers:

When using the optional sequence numbers, the path name is used as the format string for java.text.DecimalFormat. The current sequence number is passed in as the only parameter. If the optional start number is not specified, the path name is used as is. Useful formatting sequences are:

```
# - insert the number, with no leading zeros or spaces
000 - insert the number packed out to 3 digits with leading zeros if necessary
```

Examples:
pin#'.'dat -> pin1.dat, ... pin9.dat, pin10.dat, ... pin9999.dat
pin000'.'dat -> pin001.dat ... pin099.dat ... pin999.dat ... pin9999.dat
pin'.'dat# -> pin.dat1, ... pin.dat9 ... pin.dat999

If more digits are required than there are formatting characters, the number will be expanded as necessary.
To prevent a formatting character from being interpreted, enclose it in single quotes. Note that "." is a formatting character, and must be enclosed in single quotes (though #. and 000. work as expected in locales where the decimal point is also ".")
In other locales (e.g. fr), the decimal point is "," - which means that "#." becomes "nnn,".
See the documentation for DecimalFormat for full details.
If the path name does not contain any special formatting characters, the current sequence number will be appended to the name, otherwise the number will be inserted aaccording to the fomatting instructions.
If the start sequence number is omitted, and the end sequence number is specified, the sequence number is interpreted as a loop count, and the file will be used at most "end" times. In this case the filename is not formatted.

```
${_StringFromFile(PIN#'.'DAT,,1,2)} - reads PIN1.DAT, PIN2.DAT
${_StringFromFile(PIN.DAT,,,2)} - reads PIN.DAT twice
```

Note that the "." in PIN.DAT above should not be quoted. In this case the start number is omitted, so the file name is used exactly as is.

19.5.6 __machineName

The machineName function returns the local host name

Parameters

Attribute	Description	Required
Variable Name	A reference name for reusing the value computed by this function.	No

---

19.5.7 __javaScript

The javaScript function executes a piece of JavaScript (not Java!) code and returns its value

The JMeter Javascript function calls a standalone JavaScript interpreter. Javascript is used as a scripting language, so you can do calculations etc.

For details of the language, please see Mozilla Rhino Overview

The following variables are made available to the script:

- ctx - JMeterContext object
- vars - JMeterVariables object
- threadName - String
- sampler - current Sampler object (if any)
- sampleResult - previous SampleResult object (if any)
- props - JMeter Properties object

JMeter is not a browser, and does not interpret the JavaScript in downloaded pages.

Parameters

Attribute	Description	Required
Expression	The JavaScript expression to be executed. For example: • new Date() - return the current date and time • Math.floor(Math.random()*(${maxRandom}+1)) - a random number between 0 and the variable maxRandom • ${minRandom}+Math.floor(Math.random()*(${maxRandom}-${minRandom}+1)) - a random number between the variables minRandom and maxRandom • "${VAR}"=="abcd"	Yes
Variable Name	A reference name for reusing the value computed by this function.	No

Remember to include any necessary quotes for text strings and JMeter variables. Also, if the expression has commas, please make sure to escape them. For example in: ${__javaScript('${sp}'.slice(7\,99999))} the comma after 7 is escaped.

19.5.8 __Random

The random function returns a random number that lies between the given min and max values.

Parameters

Attribute	Description	Required
Minimum value	A number	Yes
Maximum value	A bigger number	Yes
Variable Name	A reference name for reusing the value computed by this function.	No

---

19.5.8 __CSVRead

The CSVFile function returns a string from a CSV file (c.f. StringFromFile)

NOTE: versions up to 1.9.1 only supported a single file. JMeter versions since 1.9.1 support multiple file names.

In most cases, the newer CSV Data Set Config element is easier to use.

When a filename is first encountered, the file is opened and read into an internal array. If a blank line is detected, this is treated as end of file - this allows trailing comments to be used (N.B. this feature was introduced in

versions after 1.9.1)

All subsequent references to the same file name use the same internal array. N.B. the filename case is significant to the function, even if the OS doesn't care, so CSVRead(abc.txt,0) and CSVRead(aBc.txt,0) would refer to different internal arrays.

The *ALIAS feature allows the same file to be opened more than once, and also allows for shorter file names.

Each thread has its own internal pointer to its current row in the file array. When a thread first refers to the file it will be allocated the next free row in the array, so each thread will access a different row from all other threads. [Unless there are more threads than there are rows in the array.]

Note: the function splits the line at every comma by default. If you want to enter columns containing commas, then you will need to change the delimiter to a character that does not appear in any column data, by setting the property: csvread.delimiter

Parameters

Attribute	Description	Required
File Name	The file (or *ALIAS) to read from	Yes
Column number	The column number in the file. 0 = first column, 1 = second etc. "next" - go to next line of file. *ALIAS - open a file and assign it to the alias	Yes

For example, you could set up some variables as follows:

- COL1a ${__CSVRead(random.txt,0)}
- COL2a ${__CSVRead(random.txt,1)}${__CSVRead(random.txt,next)}
- COL1b ${__CSVRead(random.txt,0)}
- COL2b ${__CSVRead(random.txt,1)}${__CSVRead(random.txt,next)}

This would read two columns from one line, and two columns from the next available line. If all the variables are defined on the same User Parameters Pre-Processor, then the lines will be consecutive. Otherwise, a different thread may grab the next line.

The function is not suitable for use with large files, as the entire file is stored in memory. For larger files, use CSV Data Set Config element or StringFromFile .

19.5.9 __property

The property function returns the value of a JMeter property. If the property value cannot be found, and no default has been supplied, it returns the property name. When supplying a default value, there is no need to provide a function name - the parameter can be set to null, and it will be ignored.

For example:

- ${__property(user.dir)} - return value of user.dir
- ${__property(user.dir,UDIR)} - return value of user.dir and save in UDIR
- ${__property(abcd,ABCD,atod)} - return value of property abcd (or "atod" if not defined) and save in ABCD
- ${__property(abcd,,atod)} - return value of property abcd (or "atod" if not defined) but don't save it

Parameters

Attribute	Description	Required
Property Name	The property name to be retrieved.	Yes
Variable Name	A reference name for reusing the value computed by this function.	No

Default Value	The default value for the property.	No

19.5.10 __P

This is a simplified property function which is intended for use with properties defined on the command line. Unlike the __property function, there is no option to save the value in a variable, and if no default value is supplied, it is assumed to be 1. The value of 1 was chosen because it is valid for common test variables such as loops, thread count, ramp up etc.

For example:

```
Define the property value:
jmeter -Jgroup1.threads=7 -Jhostname1=www.realhost.edu
Fetch the values:
${__P(group1.threads)} - return the value of group1.threads
${__P(group1.loops)} - return the value of group1.loops
${__P(hostname,www.dummy.org)} - return value of property hostname or www.dummy.org if not defined
```

In the examples above, the first function call would return 7, the second would return 1 and the last would return www.dummy.org (unless those properties were defined elsewhere!)

Parameters

Attribute	Description	Required
Property Name	The property name to be retrieved.	Yes
Default Value	The default value for the property. If omitted, the default is set to "1".	No

---

19.5.11 __log

The log function logs a message, and returns its input string

Parameters

Attribute	Description	Required
String to be logged	A string	Yes
Log Level	OUT, ERR, DEBUG, INFO (default), WARN or ERROR	No
Throwable text	If non-empty, creates a Throwable to pass to the logger	No
Comment	If present, it is displayed in the string. Useful for identifying what is being logged.	No

The OUT and ERR log level names are used to direct the output to System.out and System.err respectively. In this case, the output is always printed - it does not depend on the current log setting.

```
For example:
     ${__log(Message)} - written to the log file as   "...thread Name : Message"
     ${__log(Message,OUT)} - written to console window
     ${__log(${VAR},,,VAR=)} - written to log file as "...thread Name VAR=value"
```

19.5.12 __logn

The logn function logs a message, and returns the empty string

Parameters

Attribute	Description	Required
String to be logged	A string	Yes
Log Level	OUT, ERR, DEBUG, INFO (default), WARN or ERROR	No
Throwable text	If non-empty, creates a Throwable to pass to the logger	No

The OUT and ERR log level names are used to direct the output to System.out and System.err respectively. In this case, the output is always printed - it does not depend on the current log setting.

```
For example:
     ${__logn(VAR1=${VAR1},OUT)} - write the value of the variable to the console window
```

19.5.13 __BeanShell

The BeanShell function evaluates the script passed to it, and returns the result.

Please note that the BeanShell jar file is not included with JMeter; it needs to be separately downloaded.
For full details on using BeanShell, please see the BeanShell web-site at http://www.beanshell.org/

Note that a different Interpreter is used for each independent occurence of the function in a test script, but the same Interpreter is used for subsequent invocations. This means that variables persist across calls to the function.

A single instance of a function may be called from multiple threads. However the function execute() method is synchronised.

If the property "beanshell.function.init" is defined, it is passed to the Interpreter as the name of a sourced file. This can be used to define common methods and variables. There is a sample init file in the bin directory: BeanShellFunction.bshrc.

The following variables are set before the script is executed:

- log - the logger for the BeanShell function (*)
- ctx - the current JMeter context variable
- vars - the current JMeter variables
- props - JMeter Properties object
- threadName - the threadName
- Sampler the current Sampler, if any
- SampleResult - the current SampleResult, if any

(*) means that this is set before the init file, if any, is processed. Other variables vary from invocation to invocation.

Parameters

Attribute	Description	Required
BeanShell script	A beanshell script (not a file name)	Yes
Name of variable	A reference name for reusing the value computed by this function.	No

Example:

```
${__BeanShell(123*456)} - returns 56088
${__BeanShell(source("function.bsh"))} - processes the script in function.bsh
```

Remember to include any necessary quotes for text strings and JMeter

variables

19.5.14 __split

The split function splits the string passed to it according to the delimiter, and returns the original string. If any delimiters are adjacent, "?" is returned as the value. The split strings are returned in the variables ${VAR_1}, ${VAR_2} etc. The count of variables is returned in ${VAR_n}. From JMeter 2.1.2 onwards, a trailing delimiter is treated as a missing variable, and "?" is returned. Also, to allow it to work better with the ForEach controller, __split now deletes the first unused variable in case it was set by a previous split.

Example:

```
Define VAR="a||c|" in the test plan.
${__split(${VAR},VAR),|}
This will return the contents of VAR, i.e. "a||c|" and set the following variables:
VAR_n=4 (3 in JMeter 2.1.1 and earlier)
VAR_1=a
VAR_2=?
VAR_3=c
VAR_4=? (null in JMeter 2.1.1 and earlier)
VAR_5=null (in JMeter 2.1.2 and later)
```

Parameters

Attribute	Description	Required
String to split	A delimited string, e.g. "a|b|c"	Yes
Name of variable	A reference name for reusing the value computed by this function.	Yes
Delimiter	The delimiter character, e.g. **|** . If omitted, **,** is used. Note that **,** would need to be specified as **\,** .	No

---

19.5.15 __XPath

The XPath function reads an XML file and matches the XPath. Each time the function is called, the next match will be returned. At end of file, it will wrap around to the start.

Note that the entire file is held in memory, so this function should not be used for very large files.

Example:

```
${__XPath(/path/to/build.xml, //target/@name)}
This will match all targets in build.xml and return the contents of the next name attribute
```

Parameters

Attribute	Description	Required
XML file to parse	a XML file to parse	Yes
XPath	a XPath expression to match nodes in the XML file	Yes

19.5.16 __setProperty

The setProperty function sets the value of a JMeter property. The default return value from the function is the empty string, so the function call can be used anywhere functions are valid.

The original value can be returned by setting the optional 3rd parameter to

"true".

Properties are global to JMeter, so can be used to communicate between threads and thread groups

Parameters

Attribute	Description	Required
Property Name	The property name to be set.	Yes
Property Value	The value for the property.	Yes
True/False	Should the original value be returned?	No

19.5.17 __time

The time function returns the current time in various formats.

Parameters

Attribute	Description	Required
Format	The format to be passed to SimpleDateFormat. The function supports various shorthand aliases, see below.	No
Name of variable	The name of the variable to set.	No

If the format string is omitted, then the function returns the current time in milliseconds. Otherwise, the current time is passed to SimpleDateFormat. The following shorthand aliases are provided:

- YMD = yyyyMMdd
- HMS = HHmmss
- YMDHMS = yyyyMMdd-HHmmss
- USER1 = whatever is in the Jmeter property time.USER1
- USER2 = whatever is in the Jmeter property time.USER2

The defaults can be changed by setting the appropriate JMeter property, e.g. time.YMD=yyMMdd

---

19.5.18 __jexl

The jexl function returns the result of evaluating a Commons JEXL expression . See links below for more information on JEXL expressions.

- JEXL syntax description
- JEXL examples

Parameters

Attribute	Description	Required
Expression	The expression to be evaluated. For example, 6*(5+2)	Yes
Name of variable	The name of the variable to set.	No

The following variables are made available to the script:

- ctx - JMeterContext object
- vars - JMeterVariables object
- props - JMeter Properties object
- threadName - String
- sampler - current Sampler object (if any)
- sampleResult - previous SampleResult object (if any)

19.5.19 __V

The V (variable) function returns the result of evaluating a variable name expression. This can be used to evaluate nested variable references (which are not currently supported).

For example, if one has variables A1,A2 and N=1:

- ${A1} - works OK
- ${A${N}} - does not work (nested variable reference)
- ${__V(A${N})} - works OK. A${N} becomes A1, and the __V function returns the value of A1

Parameters

Attribute	Description	Required
Variable name	The variable to be evaluated.	Yes

19.5.20 __evalVar

The eval function returns the result of evaluating an expression stored in a variable.

This allows one to read a string from a file, and process any variable references in it. For example, if the variable "query" contains "select ${column} from ${table}" and "column" and "table" contain "name" and "customers", then ${__evalVar(query)} will evaluate as "select name from customers".

Parameters

Attribute	Description	Required
Variable name	The variable to be evaluated.	Yes

19.5.21 __eval

The eval function returns the result of evaluating a string expression.

This allows one to interpolate variable and function references in a string which is stored in a variable. For example, given the following variables:

- name=Smith
- column=age
- table=birthdays
- SQL=select ${column} from ${table} where name='${name}'

then ${__eval(${SQL})} will evaluate as "select age from birthdays where name='Smith'".

This can be used in conjunction with CSV Dataset, for example where the both SQL statements and the values are defined in the data file.

Parameters

Attribute	Description	Required
Variable name	The variable to be evaluated.	Yes

19.6 Pre-defined Variables

Most variables are set by calling functions or by test elements such as User Defined Variables; in which case the user has full control over the variable name that is used. However some variables are defined internally by JMeter. These are

listed below.

- cookiename - contains the cookie value
- JMeterThread.last_sample_ok - whether or not the last sample was OK - true/false
- START variables (see next section)

19.6 Pre-defined Properties

The set of JMeter properties is initialised from the system properties defined when JMeter starts; additional JMeter properties are defined in jmeter.properties, user.properties or on the command line.

Some built-in properties are defined by JMeter. These are listed below. For convenience, the START properties are also copied to variables with the same names.

- START.MS - JMeter start time in milliseconds
- START.YMD - JMeter start time as yyyyMMdd
- START.HMS - JMeter start time as HHmmss
- TESTSTART.MS - test start time in milliseconds

Please note that the START variables / properties represent JMeter startup time, not the test start time. They are mainly intended for use in file names etc.

Index Next Prev