The PowerTools engine built-in instructions

Introduction

This document describes the instructions that are built into the PowerTools engine and are available to all tests. This introduction explains about the different types of instructions. The instructions reference table then lists the available instructions for each type. The body of the document describes each of the instructions in detail and provides examples of their usage.

If any of the contents of this document is incorrect, is missing or could otherwise be improved, please get in touch.

Two forms

The exact form of an instruction depends on the source that it is read from. Currently supported are:

- Regular instructions (with mixed in arguments, used with FitNesse), and
- Keywords (followed by arguments, used with MS Excel).

FitNesse is a Wiki, where instructions are read from HTML tables. It is quite suitable for writing instructions in (almost) natural language, with arguments mixed in with the instruction name. An example:

set	URL	to	http://www.myWebSite.com

The instruction name here is 'set … to …', and the variable name ('URL') and its new value are arguments that will be used to execute the instruction.

In a spreadsheet, which is not as flexible as an HTML table in terms of layout, the above line looks somewhat different, with the instruction in the form of a keyword in the first column:

	name	value
set	URL	http://www.myWebSite.com

The first line names the arguments. Since it has an empty first column and therefore no keyword, it is treated as a comment and ignored. Comment lines are optional, so you can choose not to add such a comment line above an instruction. While this makes the test span fewer lines and easier to oversee, it can reduce readability of individual lines. The second line contains the instruction name 'set' in the first column and only arguments in the other columns.

Reference table of instructions

Instruction	Keyword
Instruction sets	
use instruction set <class name>	
use instruction set <class name> as <instruction set name>	use instruction set <class name> <instruction set name>
Constants and variables	
define constant <name> as <value>	define constant <name> <value>
define variable <name>	
define variable <name> as <value>	define variable <name> <value>
set <name> to <value>	set <name> <value>
Structures	
define global structure <name>	define global structure <name>
define structure <name>	define structure <name>
set <name> to <value>	set <name> <value>
copy structure <source> to <target>	copy structure <source> <target>
clear structure <name>	clear structure <name>
Sequences	
define number sequence <name>	
define number sequence <name> from <value>	define number sequence <name> <value>
define string sequence <name>	define string sequence <name>
add <string> to sequence <name>	add sequence string <name> <string>
Value checks	
check that <boolean value >	check that <boolean value>
check that <text> contains <text>	check that string contains <text> <text>
define variable <name> as <value>	define variable <name> <value>
set <name> to <value>	set <name> <value>
Roles	
Role <role> User name <user name> Password <password>	
System <system> Role <role> Domain <domain> User name <user name> Password <password>	declare role <system> <role> <domain> <user name> <password>
Miscellaneous	
wait <number> milliseconds	wait milliseconds <number>
wait <number> seconds	wait seconds <number>
wait <number> minutes	wait minutes <number>
	run sheet <sheet name>

Instruction details

Instruction sets

<table>
<tr><td>Name</td><td>Instruction: use instruction set <class name>
use instruction set <class name> as <instruction set name>

Keyword: use instruction set <class name> <instruction set name></td></tr>
<tr><td>Description</td><td>Registers a Java class as an instruction set – a class that implements instructions. Creates an instance of the class and registers it with the engine, so that it will be used to look for the method that implements an instruction. The specified class must be on the class path. If its constructor has a single parameter of type RunTime, it will receive the runtime object of the engine; otherwise the constructor with no parameters will be used. The class name and instruction set name must be unique.

An instruction set name can be provided using this instruction, or it can be set by the code of the instruction set itself. Either way, the instruction set name can normally be ignored as it is only needed to distinguish between identical instructions in more than one instruction set. So as long as this duplication can be avoided, there is no need to specify or refer to instruction set names. If duplicate instruction names do occur, the intended one can be specified using <instruction set name>.<instruction name>. (Notice the period in between.)</td></tr>
<tr><td>Example</td><td>Instruction:
<table>
<tr><td>use instruction set</td><td>com.company.product.X</td><td></td><td></td></tr>
<tr><td>use instruction set</td><td>com.company.product.Y</td><td>as</td><td>z</td></tr>
</table>
Keyword:
<table>
<tr><td></td><td>Class name</td><td>Name</td></tr>
<tr><td>use instruction set</td><td>org.powerTools.web.WebDriverLibrary</td><td>web</td></tr>
<tr><td>use instruction set</td><td>MyInstructions</td><td></td></tr>
</table></td></tr>
</table>

Symbol instructions

A symbol is a named data item like a constant or variable. It is used to keep information for reference later in the test. The instructions described below define or act upon a symbol.

The name of a symbol must always consist of a letter, followed by any number of letters and digits (for instance: city, clientName and addressLine2). Other characters, like spaces and underscores, are not allowed in symbol names. References to a symbol must use the exact same spelling of its name as when it was defined, with the same capitalization. While names consisting of only capital letters are allowed, their use is discouraged because they are less readable.

How long a symbol exists after being defined depends on where it is defined:

- Any symbol that is defined within a scripted instruction only exists in that scripted instruction, so it can only be used there. This includes parameters, constants, variables and structures.
- Any symbol that is defined outside a scripted instruction exists from the point in time where it is defined until the end of the test. This includes constants, variables and structures, but not parameters as these only exist inside scripted instructions.

Symbols are usually referenced in an expression that starts with a question mark. Expressions can be simple references to symbols (like '?orderNumber') but can also use operators (like '?(nrOfLeftShoes + nrOfRightShoes) / 2'). Expressions are evaluated before the instruction is invoked, so the instruction receives the result of the evaluations for its parameters and is not aware that a value was provided using an expression.

Constants and variables

<table>
<tr><td>Name</td><td>Instruction: define constant <name> as <value>

Keyword: define constant <name> <value></td></tr>
<tr><td>Description</td><td>Defines a constant – a symbol that can't be given a new value after it is defined. Using a constant instead of a variable prevents unintended overwriting of a value that should remain fixed during the whole test.</td></tr>
<tr><td>Example</td><td>Instruction:
<table><tr><td>define constant</td><td>url</td><td>as</td><td>http://www.mycompany.com</td></tr></table>
Keyword:
<table><tr><td></td><td>name</td><td>value</td></tr><tr><td>define constant</td><td>url</td><td>http://www.mycompany.com</td></tr></table></td></tr>
</table>

<table>
<tr><td>Name</td><td>Instruction: define variable <name>
define variable <name> as <value>

Keyword: define variable <name> <value></td></tr>
<tr><td>Description</td><td>Defines a variable – a symbol that can be given a new value at any time. If no value is specified, the variable is initially empty.</td></tr>
<tr><td>Example</td><td>Instruction:
<table><tr><td>define variable</td><td>emptyVariable</td></tr></table></td></tr>
</table>

	<table><tr><td>define variable</td><td>city</td><td>as</td><td>Amsterdam</td></tr></table> Keyword: <table><tr><td></td><td>Name</td><td>Value</td></tr><tr><td>define variable</td><td>emptyVariable</td><td></td></tr><tr><td>define variable</td><td>city</td><td>Amsterdam</td></tr></table>

Name	Instruction: **set** <name> **to** <value> Keyword: **set** <name> <value>
Description	Assigns a new value to a variable. Also used for structures (see below).
Example	Instruction: <table><tr><td>define variable</td><td>city</td><td>as</td><td>Amsterdam</td></tr><tr><td>set</td><td>city</td><td>to</td><td>Rotterdam</td></tr></table> Keyword: <table><tr><td></td><td>Name</td><td>Value</td></tr><tr><td>define variable</td><td>city</td><td>Amsterdam</td></tr><tr><td>set</td><td>city</td><td>Rotterdam</td></tr></table>

Structures

<table>
<tr><td>Name</td><td>define global structure <name>
define structure <name></td></tr>
<tr><td>Description</td><td>Defines a structure – data storage that can hold more than one item of data. A field in a structure can be created or assigned a new value using 'set <name> to <value>' (see below). A global structure is visible anywhere in the test once it has been defined, even if this instruction was executed from a scripted procedure. A local structure is only visible in the scripted instruction where it is defined (or anywhere, if 'define structure <name>' is used outside a scripted procedure).</td></tr>
<tr><td>Example</td><td>Instruction:
<table>
<tr><td>define global structure</td><td>officeAddress</td></tr>
<tr><td>define structure</td><td>clientAddress</td></tr>
</table>
Keyword:
<table>
<tr><td></td><td>Name</td></tr>
<tr><td>define global structure</td><td>officeAddress</td></tr>
<tr><td>define structure</td><td>clientAddress</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>Name</td><td>Instruction: set <name> to <value>

Keyword: set <name> <value></td></tr>
<tr><td>Description</td><td>Creates a field in a structure or sets an existing field to a new value. The name of the structure and the name of the field are separated by a dot. Also used for regular variables (see above).</td></tr>
<tr><td>Example</td><td>Instruction:
<table>
<tr><td>define structure</td><td colspan="3">address</td></tr>
<tr><td>set</td><td>address.street</td><td>to</td><td>My street</td></tr>
<tr><td>set</td><td>address.number</td><td>to</td><td>123</td></tr>
<tr><td>set</td><td>address.zipcode</td><td>to</td><td>1234AB</td></tr>
<tr><td>set</td><td>address.city</td><td>to</td><td>Amsterdam</td></tr>
</table>
Keyword:
<table>
<tr><td></td><td>Name</td><td>Value</td></tr>
<tr><td>define structure</td><td>address</td><td></td></tr>
<tr><td>set</td><td>address.street</td><td>My street</td></tr>
<tr><td>set</td><td>address.number</td><td>123</td></tr>
<tr><td>set</td><td>address.zipcode</td><td>1234AB</td></tr>
<tr><td>set</td><td>address.city</td><td>Amsterdam</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>Name</td><td>Instruction: copy structure <source> to <target>

Keyword: copy structure <source> <target></td></tr>
</table>

<table>
<tr><td>Description</td><td>Copies all fields in a structure to another structure. The source can be a whole structure or part of a structure (a field that contains other fields). The destination can also be a structure or a field. Existing fields in the destination structure are not removed but may be overwritten.</td></tr>
<tr><td>Example</td><td>Instruction:
<table>
<tr><td>copy structure</td><td>anAddress</td><td>to</td><td>newAddress</td></tr>
<tr><td>copy structure</td><td>client.address</td><td>to</td><td>oldAddress</td></tr>
<tr><td>copy structure</td><td>newAddress</td><td>to</td><td>client.address</td></tr>
<tr><td>copy structure</td><td>client.address</td><td>to</td><td>recipient.address</td></tr>
</table>
Keyword:
<table>
<tr><td></td><td>Source</td><td>Target</td></tr>
<tr><td>copy structure</td><td>anAddress</td><td>newAddress</td></tr>
<tr><td>copy structure</td><td>client.address</td><td>oldAddress</td></tr>
<tr><td>copy structure</td><td>newAddress</td><td>client.address</td></tr>
<tr><td>copy structure</td><td>client.address</td><td>recipient.address</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>Name</td><td>clear structure <name></td></tr>
<tr><td>Description</td><td>Clears (part of) a structure by removing the fields in there.</td></tr>
<tr><td>Example</td><td>Instruction:
<table>
<tr><td>clear structure</td><td>address</td></tr>
</table>
Keyword:
<table>
<tr><td></td><td>Name</td></tr>
<tr><td>clear structure</td><td>address</td></tr>
</table></td></tr>
</table>

Sequences

A sequence is a symbol that cannot be set but returns a different value each time it is referenced. There are two types of them: number sequences and string sequences. A number sequence returns the next number each time it is referenced. A string sequence must first be filled with strings that can be used for tests and then returns these in the order in which they were added.

<table>
<tr><td>Name</td><td>Instruction: define number sequence <name>
define number sequence <name> from <number>

Keyword: define number sequence <name> <number></td></tr>
<tr><td>Description</td><td>Defines a number sequence symbol that will yield a new value every time it is evaluated. If no initial value is specified, the first value will be 1. A number sequence if often used to create unique values in order to prevent issues with duplicate keys.</td></tr>
<tr><td>Example</td><td>Instruction:
<table>
<tr><td>define number sequence</td><td colspan="3">counter1</td></tr>
<tr><td>define number sequence</td><td>counter2</td><td>from</td><td>1000</td></tr>
</table>
Keyword:
<table>
<tr><td></td><td>Name</td><td>Value</td></tr>
<tr><td>define number sequence</td><td>counter1</td><td></td></tr>
<tr><td>define number sequence</td><td>counter2</td><td>1000</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>Name</td><td>define string sequence <name></td></tr>
<tr><td>Description</td><td>Defines a string sequence symbol that will contain a new value every time it is evaluated. The sequence is initially empty and must be filled with strings using 'add <string> to sequence <name>' (see below) before it is evaluated. After each of the added strings has been used, evaluation of the sequence will fail, so it should contain enough strings for the whole test.</td></tr>
<tr><td>Example</td><td>Instruction:
<table>
<tr><td>define string sequence</td><td>licenseNumber</td></tr>
</table>
Keyword:
<table>
<tr><td></td><td>Name</td></tr>
<tr><td>define string sequence</td><td>licenseNumber</td></tr>
</table></td></tr>
</table>

<table>
<tr><td>Name</td><td>Instruction: add <string> to sequence <name>

Keyword: add sequence string <name> <string></td></tr>
<tr><td>Description</td><td>Adds a string to a string sequence symbol, that will a new value every time it is evaluated. All values must be added to the string sequence before it is</td></tr>
</table>

<table>
<tr><td></td><td>referenced. A string sequence is referenced in the same way as variables. It if often used to create unique values in order to prevent issues with duplicate keys.</td></tr>
<tr><td>Example</td><td>Instruction:
<table>
<tr><td>define string sequence</td><td colspan="3">licenseNumber</td></tr>
<tr><td>add</td><td>AB-12-CD</td><td>to</td><td>licenseNumber</td></tr>
<tr><td>add</td><td>EF-34-GH</td><td>to</td><td>licenseNumber</td></tr>
</table>
Keyword:
<table>
<tr><td></td><td>Name</td><td>Value</td></tr>
<tr><td>define string sequence</td><td>licenseNumber</td><td></td></tr>
<tr><td>add sequence string</td><td>licenseNumber</td><td>AB-12-CD</td></tr>
<tr><td>add sequence string</td><td>licenseNumber</td><td>EF-34-GH</td></tr>
</table></td></tr>
</table>

Value checks

<table>
<tr><td>Name</td><td>check that <boolean value></td></tr>
<tr><td>Description</td><td>Checks that a boolean expression evaluates to true. The boolean operators can be used to perform many kinds of checks. The operators that return a boolean value are: ‘=’, ‘<>’, ‘<’, ‘<=’, ‘>’, ‘>=’, ‘and’, ‘or’ and ‘not’.</td></tr>
<tr><td>Example</td><td>Instruction:
<table><tr><td>check that</td><td>?answer = 42</td></tr></table>
Keyword:
<table><tr><td></td><td>Expression</td></tr><tr><td>check that</td><td>?answer = 42</td></tr></table></td></tr>
</table>

<table>
<tr><td>Name</td><td>Instruction: check that <text> contains <text>

Keyword: check that text contains <text> <text></td></tr>
<tr><td>Description</td><td>Checks that one string contains another. This is the case if the second string is either a substring of or identical to the first string.</td></tr>
<tr><td>Example</td><td>Instruction:
<table><tr><td>check that</td><td>?phoneNumber</td><td>contains</td><td>020-</td></tr></table>
Keyword:
<table><tr><td></td><td>Text</td><td>Substring</td></tr><tr><td>check that text contains</td><td>?phoneNumber</td><td>020-</td></tr></table></td></tr>
</table>

<table>
<tr><td>Name</td><td>Instruction: check that <value1> does not contain <value2>

Keyword: check that text does not contain <value1> <value2></td></tr>
<tr><td>Description</td><td>Checks that one string does not contain another. This is the case if the second string is neither a substring of nor identical to the first string.</td></tr>
<tr><td>Example</td><td>Instruction:
<table><tr><td>check</td><td>?phoneNumber</td><td>does not contain</td><td>020-</td></tr></table>
Keyword:
<table><tr><td></td><td>Text</td><td>Substring</td></tr><tr><td>check that text does not contain</td><td>?phoneNumber</td><td>020-</td></tr></table></td></tr>
</table>

<table>
<tr><td>Name</td><td>Instruction: check that <value1> is within <margin> of <value2>

Keyword: check value is within margin <value1> <margin> <value2></td></tr>
<tr><td>Description</td><td>Checks that one numeric value is within a certain margin of another value. This is used most to check floating point numbers, as checking these for an exact value often fails.</td></tr>
</table>

Example	Instruction: check | ?result | is within | 0,001 | of | ?expectedResult Keyword: | Value1 | Margin | Value2 check value is within margin | ?result | 0,001 | ?expectedResult

Name	Instruction: **check that** <value1> **is not within** <margin> **of** <value2> Keyword: **check value is not within margin** <value1> <margin> <value2>
Description	Checks that one numeric value is not within a certain margin of another value. This is used most to check floating point numbers, as checking these for an exact number often fails.
Example	Instruction: check | ?result | is not within | 0,001 | of | ?expectedResult Keyword: | Value1 | Margin | Value2 check value is not within margin | ?result | 0,001 | ?expectedResult

Name	Instruction: **check that** <value> **is between** <min> **and** <max> Keyword: **check that value is in range** <value> <min> <max>
Description	Checks that a numeric value is between a lower and an upper bound value (inclusive).
Example	Instruction: check | ?itemNr | is between | 1 | and | ?nrOfItems Keyword: | Value | Min | Max check that value is in range | ?itemNr | 1 | ?nrOfItems

Name	Instruction: **check that** <value> **is not between** <min> **and** <max> Keyword: **check value is not in range** <value> <min> <max>
Description	Checks that a numeric value is not between a lower and an upper bound value (inclusive).
Example	Instruction: check | ?itemNr | is not between | 1 | and | ?nrOfItems Keyword: | Value | Min | Max

check that value is not in range	?itemNr	1	?nrOfItems

Roles

Role instructions make user accounts available in a test while hiding their details from the test cases. They are called roles rather than accounts because, in a test, accounts are usually meant to be used for a specific purpose or role (that may not be apparent from the account name).

Name	Instruction: **role** <role> **user name** <user name> **password** <password>
Description	Defines a role by associating a user name and password with a role name. The user name and password are made available as symbols: • roles.<role name>.username and • roles.<role name>.password.
Example	Instruction (data-driven): Role | User name | Password operator | Alice | secret manager | Bob | important

Name	Instruction: **system** <system name> **role** <role name> **domain** <domain> **user name** <user name> **password** <password> Keyword: **define role** <system> <role> <domain> <user name> <password>
Description	Defines a role by associating a user name and password with a role name. The user name and password are made available as symbols: • roles.<system name>.<role name>.domain, • roles.<system name>.<role name>.username, and • roles.<system name>.<role name>.password.
Example	Instruction (data-driven): System | Role | Domain | User name | Password frontend | operator | xyz | Alice | secret frontend | manager | xyz | Bob | important backend | operator | | Alice | forgotten backend | reviewer | | Carol | whatever Keyword: | System | Role | Domain | User name | Password define role | frontend | operator | xyz | Alice | secret define role | frontend | manager | xyz | Bob | important define role | backend | operator | | Alice | forgotten define role | backend | reviewer | | Carol | whatever

Miscellaneous

<table>
<tr><td>Name</td><td>Instruction: wait <number> milliseconds
wait <number> seconds
wait <number> minutes

Keyword: wait milliseconds <number>
wait seconds <number>
wait minutes <number></td></tr>
<tr><td>Description</td><td>Pauses test execution for the specified duration.

Note that waiting for a specific event is usually a more reliable way to synchronize with an application than these simple waits. One example is explicitly waiting for a web page element to become visible or enabled. This not only avoids the risk of waiting too short (or too long, wasting time) but also makes the intent clearer.</td></tr>
<tr><td>Example</td><td>Instruction:<table><tr><td>wait</td><td>200</td><td>milliseconds</td></tr><tr><td>wait</td><td>5</td><td>seconds</td></tr><tr><td>wait</td><td>2</td><td>minutes</td></tr></table>Keyword:<table><tr><td></td><td>Duration</td></tr><tr><td>wait milliseconds</td><td>200</td></tr><tr><td>wait seconds</td><td>5</td></tr><tr><td>wait minutes</td><td>2</td></tr></table></td></tr>
</table>

<table>
<tr><td>Name</td><td>Keyword: run sheet <sheet name></td></tr>
<tr><td>Description</td><td>Selects the specified sheet to retrieve the next instruction. When all instructions from the new sheet have been executed, control will return to the original sheet. A sheet name can include the file name, in which case the file name and sheet name are separated by an ‘@’ character. Otherwise, the sheet will be opened in the same file.</td></tr>
<tr><td>Example</td><td>Keyword:<table><tr><td></td><td>Name</td></tr><tr><td>run sheet</td><td>common.xls@environment</td></tr><tr><td>run sheet</td><td>transfers</td></tr></table></td></tr>
</table>