

User Guide

Thank you for 3

Local machine setup: 3

Remote machine setup 3

Properties files 3

Running BOLT Test via command line 4

Running BOLT Test via Intellij 4

Defining elements 4

Utilizing JMeter 5

Specification “Spec” files 5

Concept files 5

Adding custom steps 5

Adding custom actions 5

Using Tags 5

Using Tables for data driven testing 5

Using csv files for data driven testing 5

Using a database for data driven testing 6

Setup steps 6

Teardown steps 6

Parameters 6

Variables 6

Spec comments 6

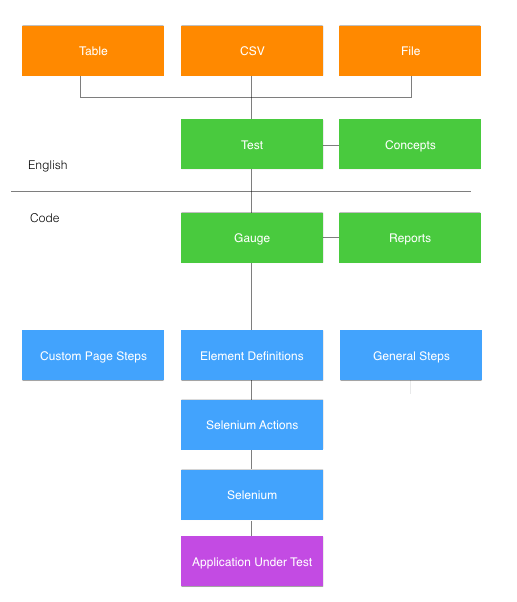
Thank you for Using BOLT!

This user guide will go through the steps to set up and run BOLT.



# BOLT Test

# Framework Design





# Local machine setup:

1. Install Java 8 <https://www3.ntu.edu.sg/home/ehchua/programming/howto/JDK_Howto.html>
2. Download and install Intellij (community edition): <https://www.jetbrains.com/idea/download/>
3. Install Gauge: <https://gauge.org/get-started.html>
4. Add the Gauge plugin to Intellij: <https://docs.gauge.org/using.html#intellij-idea>
5. Clone the BOLT Test sample project: {future link}
6. Open Intellij and create a new project from an existing source.
7. Select the BOLT Test sample project folder as the source.
8. Select “Import project from external model” and select “Maven”, then click “Next”.
9. Continue clicking “Next” until prompted to select the project SDK. At this point, if “1.8” is not already displayed, click the “+” in the top left corner, navigate to where java was installed, and select the folder for java 8 (1.8). Once that is complete, click “Next” and then “Finish”.
10. At this point, it will take a few minutes for Intellij to download all of the maven dependencies.
11. Once that is completed, go to the pom.xml file, find the com.swatsolutions.bolt dependency, and add your username and password. It is recommended to use a hashed version of the password for security reasons. <https://maven.apache.org/guides/mini/guide-encryption.html>

Once this has been setup, it is recommended to create a repository to store the automation code. This will aid in maintenance and source control. It also puts the code in the cloud where it can then be used in the CI/CD pipeline.

# 

# Remote machine setup

Contact Swat Solutions for an expert to setup BOLT Build or to help integrate BOLT Test with an existing CI/CD pipeline.

# Recommendations:

* As steps are not held to a strict form, it is highly recommended to setup standards as to how new steps should be built.
* When building new steps/actions, it is recommended to let the step class call a different method to conduct interactions with the webpage.

# Quick definitions:

* Specification (spec): Business layer test cases written in the business language. Typically one spec per page or feature. A spec must also contain at least one scenario.
* Step: A single action that is performed for testing. A step always begins with a ‘\*’.
* Scenario: A single flow of steps to complete a test case.
* Concept: A way to combine multiple small steps into one step.
* Parameter: A value that is passed into a step so steps can be re-used with different values.
* Tag: A way to associate labels with specs or scenarios. This allows tests with only a specific tag to be run.
* Tear Down Steps: Steps defined at the end of a spec that are run after every scenario in a spec.
* Comments/notes: A comment or note can be added to a spec by simply writing the comment or note on its own line in a spec.

# Properties files

Properties files are used to easily change environmental properties. When running via command line, a user can simply specify what environment to run on and the properties from that environment will be stored. Each environment that can be specified relates to a folder inside the “env” folder. Each folder can contain multiple .proeprties files and when specifying that environment, properties from all files will be used. The default folder is used to store default properties. These properties will always be used as well, but if an environment is specified, the properties defined in that environment will only add properties that were not defined in that environment.

|  |  |
| --- | --- |
| Property | Description |
| REMOTE\_SYSTEM | Defines which remote system to be using, saucelabs, browserstack, or openshift. |
| BROWSER | Defines the browser to test within. |
| BROWSER\_VERSION | Available to specify a specific browser version. Selects the latest available version if empty. |
| REMOTE | True to run on the defined remote system |
| REMOTEURL | Default remote system url to use if remote\_system is empty |
| BROWSERSTACK\_URL | URL to connect to for a browserstack test run |
| SAUCELABS\_URL | URL to connect to for a saucelabs test run |
| OPENSHIFT\_URL | URL to connect to for an openshift test run |
| ELEMENT\_DEFINITIONS | File location of the element definitions csv file |
| ELEMENT\_WAIT\_TIME | Maximum time to wait for an element |
| IP | IP address of the AUT (used for JMeter testing) |
| PORT | Port of the AUT (used for JMeter testing) |
| AUT\_URL | URL of the AUT (Application Under Test) |
| BUTTON\_TYPE | WebElement type of buttons in the AUT |
| LABEL\_TYPE | WebElement type of labels in the AUT |
| SPECIAL\_TEXT\_ATTRIBUTE | WebElement attribute for element text if text is stored in an attribute instead of in a standard manor. |
| PAGE\_HEADING\_LEVEL | Heading level of the page titles (h1,h2,h3, etc) |
| TEXT\_FIELD\_TYPE | WebElement type of text fields |
| TEXT\_AREA\_TYPE | WebElement type of text areas |
| CHECKBOX\_TYPE | WebElement type of checkboxes |
| FIELD\_DEFAULT\_FIELD\_ATTRIBUTE | WebElement attribute of text field default values |
| TEXTAREA\_DEFAULT\_FIELD\_ ATTRIBUTE | WebElement attribute of text area default values |
| INITIAL\_AUT\_LOAD\_TIME\_MS | Optional wait for initial loading of the AUT |
| FLASH\_SERVER\_PORT | Port for the flash report to persist upon |
| REMOTE\_HOST\_# | Location of remote nodes to use with jmeter testing. |
|  |  |
| DB.PROPERTIES | Properties for pulling data from a database to use for specs. |
| DB\_USERNAME | Database username |
| DB\_PASSWORD | Database password |
| DB\_URL | Database URL |
| DB\_TYPE | Type of database (SQL) |
| QUERY\_# | The query to make (‘#’ is the index starting at 1) |
| FILENAME\_# | The filename to store the query results |
| URL\_# | Override of the primary database URL if different for a query |
| DB\_USERNAME\_# | Override of the primary database username if different |
| DB\_PASSWORD\_# | Override of the primary database password if different |
| DB\_TYPE\_# | Override of the primary database type if different for a query |

# Running BOLT Test via command line

Running BOLT Test via command line is best for when running regression, api, or performance tests. This option allows the environment to be specified, parallel execution, running tests with given tag(s), and more. The following chart displays the options that are available.

|  |  |
| --- | --- |
| Command/configuration | Description |
| mvn gauge:execute | Primary Command |
| -DspecsDir= | A specific spec or folder of specs can be specified to run |
| -DinParallel= | “true or false”, specifies to run in parallel or not. |
| -Dnodes= | The maximum number of nodes to use. Uses all nodes if not included. |
| -Dtags= | “tag1 & tag2” Run tests only with the given tag(s). |
| -Denv= | How to specify what environment to run the tests against. |
| -Ddir= | Working directory for gauge. Default is project.base.dir |
| -Dflags=”” | Add additional flags to the execution |

Example: mvn gauge:execute -DspecsDir=specs

# Running BOLT Test via Intellij

Running a test or spec in Intellij is very simple. Simply open the spec file that is to run or the spec file with the test that is to be run. Once the spec is open, on the left side next to the line numbers will be a green play button. The one at the top next to the spec name is used to run the whole spec, while the ones next to each test are used to run that given test. Once clicking on the play button, the option is given to run in debugger mode if needed.

# Defining elements

Most of the time defining elements will not be needed due to BOLT Tests’ smart element technology that can find elements without any identifiers. In the time that an element needs to be defined, there is a file named “element\_definition.csv” that is used to store all of the elements and how to find those elements. The csv can be edited directly, but for easier viewing, it can be opened by your favorite spreadsheet editor. Just be sure to save it as a csv again.

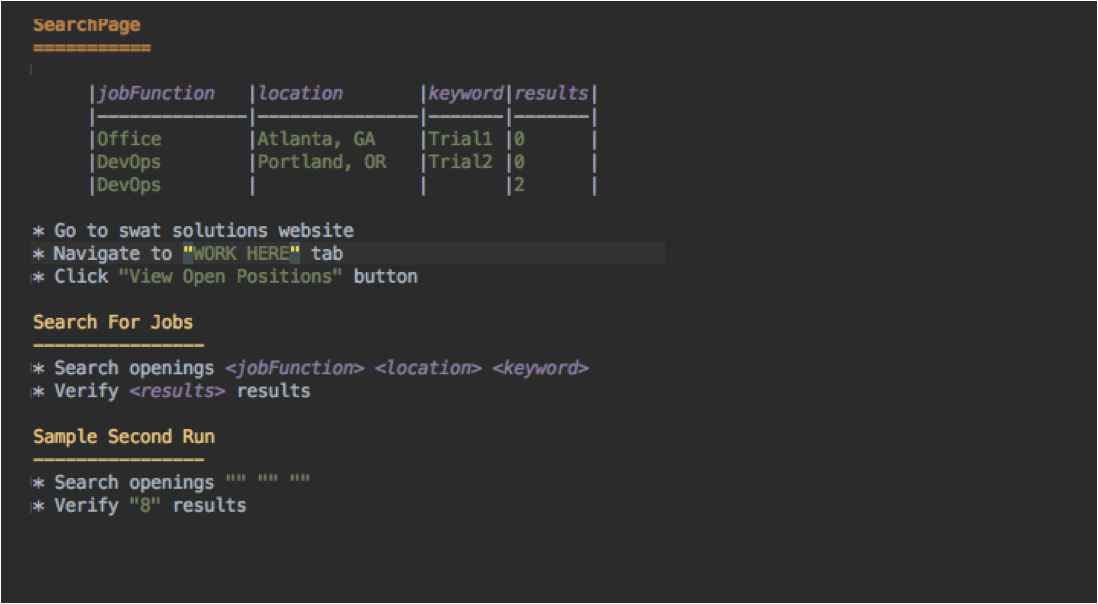
There are four columns, page, object, type, and value. Unless the element is a symbol, the page needs to relate to the name of the spec that will be using it. Object is used to define what the element is, a button, field, etc. For symbols, the object will need to match how it is referenced in tests. Type specifies what is being used to define the element (id, class, name, xpath, etc). Value is the value of the specified type for the element.

# Utilizing JMeter

BOLT Test is able to manage running jmeter tests for both api and performance testing. As each JMeter test is custom for each application, all jmeter steps will need to be custom built. When building the JMeter test, each variable can be either hard-coded in the test, or it can be left as a variable and passed in from BOLT Test. It is highly recommended to have the thread count, ramp up period, and loop count marked as variables with a default value set for them in the JMeter test. Within the JMeter test, variables from BOLT Test are passed in as “bolt.variableNameInStep”.

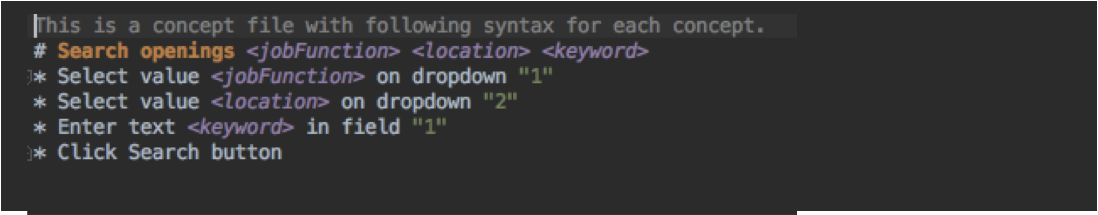
# Specification “Spec” files

Business layer test cases written in the business language. Typically one spec per page or feature. A spec must also contain at least one scenario.



# Concept files

A way to combine multiple steps into one step. This is very helpful when there are a series of steps that are used frequently.



# Adding custom steps

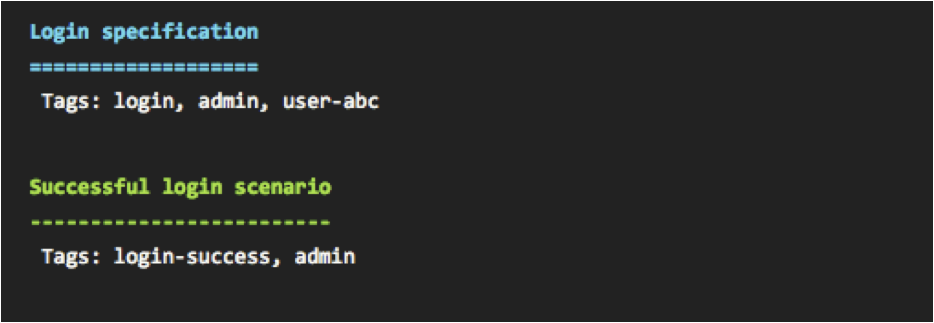
There are many steps already built into BOLT Test, but when another step is needed, there is an easy way to create it. To build a new step, there are a few examples in “CustomClientSteps.java” that can be followed. The text for a step is listed using “@Step” and the method directly below that is what will be run for the given step. Steps should not directly interact with the website or webdriver. Instead, an action should be used or created.

# Adding custom actions

As with steps, there are many prebuilt actions in BOLT Test, but additional actions can be easily created in “CustomActions.java” file.

# Using Tags

Tags are a very powerful tool that can be used with BOLT Test. Tags can tie tests or specs to a specific feature, type of test, etc. When running tests via command line, one or more tags can be selected to run and any test that has one of the listed tags will be run.



# Using Tables for data driven testing

Tables can be added to the top of a spec file and this allows a test to be run with each set of variables.

# Using csv files for data driven testing

In many cases, a table is not ideal to store different sets of data to run tests with. Instead of a table, a csv file can be created and listed at the top of a given test. This will then pull the data from the csv and run as if it was just a table.

# Using a database for data driven testing

Data for tests can easily be pulled in from a database for testing. In the spec file, simply create a file path/name for a csv that will be created/populated with the data from the database. After completing that step, look at the “db.properties” file in “env/default”. If there are no existing queries of this type, simply fill out the “DB\_USERNAME”, “DB\_PASSWORD”, “DB\_URL”, and “DB\_TYPE”. Then, fill out “QUERY\_1” with the query to get the needed information and ensure the query is on one line. Also fill out “FILENAME\_1” with the same file path/name added to the test.

To add additional queries, simply add “QUERY\_2” with the second query and add “FILENAME\_2” with the file path/name for the results to go to. The additional options (URL\_1, DB\_USERNAME\_1, DB\_PASSWORD\_1, DB\_TYPE\_1) can be used to use a different database for a query if needed.

# Setup steps

Setup steps are steps that are run before every test in a spec file. These steps are listed after a table or csv definition and before the first test.

# Teardown steps

Just as there are setup steps, there are teardown steps that are run after each test. These steps exist at the end of a spec file and are listed after a line with at least three dashes “---“.

# Parameters

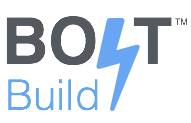
Steps can take in parameters. Parameters are surrounded by quotations and are passed into the step when run.

# Variables

Variables are similar to parameters, but instead of text being entered between quotes, a reference is used between “<>”. The variable gets data from a table or csv file in the spec.

# Spec comments

Adding a comment to a spec file is very simple. Instead of starting a line with an asterisk as with steps, simply just type the comment on its own line.



# Full Stack

