Edit this Page

EXECUTION

Specify Directory

By convention, specifications are stored in the the `./specs/` sub-directory in the project root.Gauge scans the directory(ies) passed and picks up valid specification files.

Multiple arguments can be passed to `run` subcommand by separating them with a space. The values can be either

- path to directories that contain specifications
- path to specification files
- path to scenarios (see scenario)
- mix of above.

To run all the specifications in a given folder `specs`:

```
> gauge run specs
```

To specify a single directory in which specifications are stored:

For example,

```
> gauge run specs/
```

To specify multiple directories in which specifications are stored.

For example,

```
> gauge run specs-dir1/ specs-dir2/ specs-dir3/
```

Note

The path of specifications can also be specified through an environment variable <path>. This changes the default specification directory from `specs` to the value defined in the environment variable.

Note

Gauge specifications can also be run from within the IDE (Visual Studio Code , IntelliJ IDEA , Visual Studio)

Specify files

Gauge executes only the specification file(s) provided.

Run one (or a list of) specifications.

```
> gauge run [flags] <path-to-specs>
```

For example, execute a single specification file:

```
> gauge run specs/spec1.spec
```

Or, execute multiple specification files:

```
> gauge run specs/spec1.spec specs/spec2.spec specs/spec3.spec
```

Specify Scenarios

A single scenario of a specification can be executed by specifying the line number in the span of that scenario in the spec. The argument can be a specific scenario or a list of scenarios to execute.

To execute a `Admin Login` scenario in the following spec use `gauge run specs/login_test.spec:4` command.

```
# Configuration

## Admin Login
* User must login as "admin"
* Navigate to the configuration page
```

This executes only the scenario present at line number `3` i.e `Admin Login` in `login_test.spec`. In the above spec, specifying line numbers 3-5 will execute the same scenario because of the span.

Multiple scenarios can be executed selectively as follows :

```
> gauge run specs/helloworld.spec:3 specs/anotherhelloworld.spec:5
```

These scenarios can also belong to different specifications.

To execute scenarios, `gauge` takes path to a specification file, followed by a colon and the line number of the scenario. Any line number which the scenario spans across can be used.

For example, in the above spec file, both the below commands will run the same scenario.

```
> gauge run specs/helloworld.spec:3 # Runs scenario 'Admin Login'
> gauge run specs/helloworld.spec:5 # Runs scenario 'Admin Login'
```

Consider a specification file, `spec1.spec`:

```
# Configuration

## Admin Login
```

```
* User must login as "admin"
* Navigate to the configuration page

## User Login
* User must login as "user1"
* Navigation to configuration page is restricted.
```

To execute the second scenario of a specification file named `spec1.spec`:

```
> gauge run specs/spec1.spec:7
```

To specify multiple scenarios, add multiple such arguments. For example, to execute the first and second scenarios of a specification file named `spec1.spec`:

```
> gauge run specs/spec1.spec:3 specs/spec1.spec:7
```

Verbose reporting

By default, `gauge` reports at the specification level when executing tests. Enable verbose step-level reporting by using the `--verbose` flag. For example,

```
> gauge run --verbose specs/
```

Data driven execution

- A *data table* is defined in markdown table format in the beginning of the spec before any steps.
- The data table should have a header row and one or more data rows
- The header names from the table can be used in the steps within angular brackets `< >` to refer a particular column from the data table as a parameter.
- On execution each scenario will be executed for every data row from the table.
- Table can be easily created in IDE using template `table:<no of columns>`, and hit `Tab`.
- Table parameters are written in Multi-markdown table formats.

For example,

```
# Table driven execution

     |id| name     |
     |--|---------|
     |1 |vishnu   |
     |2 |prateek  |
     |3 |navaneeth|
```

```
## Scenario
* Say "hello" to <name>

## Second Scenario
* Say "namaste" to <name>
```

In the above example the step uses the `name` column from the data table as a dynamic parameter.

Both `Scenario` and `Second Scenario` are executed first for the first row values `1, vishnu` and then consecutively for the second and third row values from the table.

External CSV for data table

Data Tables for a specification can also be passed from an external CSV file. The parameter contains a prefix table and the path to the csv file.

Prefix : The prefix is table

Value : The value is the path to the csv file. This can be absolute file path or relative to project.

For example,

```
# Table driven execution

table: /system/users.csv

## Scenario
* Say "hello" to <name>

## Second Scenario
* Say "namaste" to <name>
```

In the above example the step uses the `name` column from the csv file.

Execute selected data table rows

By default, scenarios in a spec are run against all the data table rows. It can be run against selected data table rows with flag `--table-rows` and specifying the row numbers against which the scenarios should be executed. If there are multiple row numbers, they should be separated by commas.

For example,

```
> gauge run --table-rows "1" specs/hello.spec
> gauge run --table-rows "1,4,7" specs/hello.spec
```

Range of table rows can also be specified, against which the scenarios are run.

For example,

```
> gauge run --table-rows "1-3" specs/hello.spec
```

This executes the scenarios against table rows 1, 2, 3.

> **Note**
>
> This flag does not work well for multiple specifications, since there is no way to choose different table rows for different specifications.

Specify Tags

Tags allow filtering the specs and scenarios to be executed. The following command executes all the specs and scenarios which are labelled with certain tags:

```
> gauge run --tags tag1,tag2 specs
```

or,

```
> gauge run --tags "tag1, tag2" specs
```

This executes only the scenarios and specifications which are tagged with `tag1` and `tag2`.

Example:

```
# Search Specification

The admin user must be able to search for available products on the search page.

Tags: search,  admin

* User must be logged in as "admin"
* Open the product search page

## Successful search

Tags: successful

For an existing product name, the search result will contain the product name.

* Search for product "Die Hard"
* "Die Hard" should show up in the search results

## Unsuccessful search

On an unknown product name search, the search results will be empty

* Search for product "unknown"
* The search results will be empty
```

In the above spec, if all the scenarios tagged with “search” and “successful” should be executed, then use the following command:

```
> gauge run --tags "search & successful" SPEC_FILE_NAME # Runs scenario 'Successful
```

Execution hooks can also be filtered based on tags. See filtering hooks with tags for more information.

Tag expressions

Tags can be selected using expressions. Examples:

Tags	Selects specs/scenarios that
`!TagA`	do not have `TagA`
`TagA & TagB`	have both `TagA` and `TagB`.
`TagA & !TagB`	have `TagA` and not `TagB`.
`TagA | TagB`	have either `TagA` or `TagB`.
`(TagA & TagB) | TagC`	have either `TagC` or both `TagA` and `TagB`
`!(TagA & TagB) | TagC`	have either `TagC` or do not have both TagA and TagB
`(TagA | TagB) & TagC`	have either [`TagA` and `TagC`] or [`TagB` and `TagC`]

Note

In the command line tagged execution, the not symbol(!) has to be escaped.

Parallel Execution

Specs can be executed in parallel to run the tests faster and distribute the load.

This can be done by the command:

```
> gauge run --parallel specs
```

or,

```
> gauge run -p specs
```

This creates a number of execution streams depending on the number of cores of the machine and distribute the load among workers.

The number of parallel execution streams can be specified by `-n` flag.

Example:

```
> gauge run --parallel -n=4 specs
```

This creates four parallel execution streams.

Note

The number of streams should be specified depending on number of CPU cores available on the machine, beyond which it could lead to undesirable results. For optimizations, try parallel execution using threads .

Parallel Execution using threads

In parallel execution, every stream starts a new worker process. This can be optimized by using multithreading instead of processes. This uses only one worker process and starts multiple threads for parallel execution.

To use this, Set *enable_multithreading* env var to true. This property can also be added to the default/custom env.

```
enable_multithreading = true
```

Requirements:

- Thread safe test code.
- Language runner should support multithreading.

Note

Currently, this feature is only supported by Java language runner/plugin.

Executing a group of specification

Specifications can be distributed into groups and `--group` | `-g` flag provides the ability to execute a specific group.

This can be done by the command:

```
> gauge run -n=4 -g=2 specs
```

This creates 4 groups (provided by -n flag) of specification and selects the 2nd group (provided by -g flag) for execution.

Specifications are sorted by alphabetical order and then distributed into groups, which guarantees that every group will have the same set of specifications, no matter how many times it is being executed.

Example:

```
> gauge run -n=4 -g=2 specs
```

```
> gauge run -n=4 -g=2 specs
```

The above two commands will execute the same group of specifications.

Rerun one execution stream

Executing specs with `-n` and *-g* ` flags guarantee the same execution.

Example, execute the below command twice:

```
> gauge run -n=4 -g=2 specs
```

On both occassions, gauge will execute the same group of specifications, in the same order.

Run your test suite with lazy assignment of tests

This features dynamically allocates specs to streams during execution instead of at the start of execution.

This allows Gauge to optimise the resources on your agent/execution environment. This is useful because some specs may take much longer than other, either because of the number of scenarios in them or the nature of the feature under test

The following command will assign tests lazily across the specified number of streams:

```
> gauge run -n=4 --strategy="lazy" specs
```

or,

```
> gauge run -n=4 specs
```

As an example, if there are 100 tests, which have to be run across 4 streams/cores; lazy assignment will dynamically assign the next spec in line to the stream that has completed it's previous execution and is waiting for more work.

Lazy assignment of tests is the default behaviour.

Another strategy called `eager` can also be useful depending on need. In this case, the 100 tests are distributed before execution, thus making them an equal number based distribution.

```
> gauge run -n=4 --strategy="eager" specs
```

> **Note**
>
> The 'lazy' assignment strategy only works when you do NOT use the -g flag. This is because grouping is dependent on allocation of tests before the start of execution. Using this in conjunction with a lazy strategy will have no impact on your test suite execution.

Re-run failed tests

Gauge provides the ability to re-run only the scenarios which failed in previous execution. Failed scenarios can be run using the `--failed` flag of Gauge.

As an example if 3 scenarios failed during `gauge run specs` , the failed scenarios can be re-run instead of executing all scenarios by following command.

```
> gauge run --failed
```

This command will even set the flags which you had provided in your previous run. For example, if previous command was

```
> gauge run --env="chrome" --verbose specs
```

and 3 scenarios failed in this run, the `gauge run --failed` command sets the `--env` and `--verbose` flags to corresponding values and executes only the 3 failed scenarios. In this case `gauge run --failed` is equivalent to command

```
> gauge run --env="chrome" --verbose specs <path_to_failed_scenarios>
```

Errors during execution

Parse errors

This occurs if the spec or concept file doesn't follow the expected specifications or concepts syntax.

Example:

```
[ParseError] hello_world.spec : line no: 25, Dynamic parameter <product> could not
```

List of various Parse errors:

Parse Error	Gauge Execution Behaviour
Step is not defined inside a concept heading	Stops
Circular reference found in concept	Stops
Concept heading can only have dynamic parameters	Stops
Concept should have at least one step	Stops
Duplicate concept definition found	Stops
Scenario heading is not allowed in concept file	Stops
Table doesn't belong to any step	Ignores table,Continue
Table header cannot have repeated column values	Marks that spec as failed,Continues for others
Teardown should have at least three underscore characters	Marks that spec as failed,Continues for other
Scenario heading should have at least one character	Marks that spec as failed,Continues for other
Table header should be not blank	Marks that spec as failed,Continues for other

Multiple spec headings found in the same file	Marks that spec as failed,Continues for other
Scenario should be defined after the spec heading	Marks that spec as failed,Continues for other
Could not resolve table from file	Marks that spec as failed,Continues for other
Spec does not have any element	Marks that spec as failed,Continues for other
Spec heading not found	Marks that spec as failed,Continues for other
Spec heading should have at least one character	Marks that spec as failed,Continues for other
Dynamic param could not be resolved	Marks that spec as failed,Continues for other
Step should not be blank	Marks that spec as failed,Continues for other
Duplicate scenario definition found in the same specification	Marks that spec as failed,Continues for other

Validation Errors

These are errors for which *Gauge* skips executing the spec where the error occurs.

There are two types of validation error which can occurs

1. **Step implementation not found**
   If the spec file has a step that does not have an implementation in the projects programming language.

2. **Duplicate step implementation**
   If the spec file has a step that is implemented multiple times in the projects.

Example

```
[ValidationError] login.spec:33: Step implementation not found. login with "user" a
```

```
[ValidationError] foo.spec:11 Duplicate step implementation => 'Vowels in English l
```

TROUBLESHOOTING

Ensure that the latest version of gauge and gauge plugins .

Run `gauge update -c` to check if there are updates available for gauge and the plugins.

Validation Errors

```
[WARN] Validation failed. The following steps have errors
...
```

These generally occur if step implementation is not found for a particular step.

- Ensure the step implementation for the step has been added.
- The step template marking the step in code is case sensitive and should match the step usage in the spec file.

Compatibility errors

```
Failed to start a runner. Compatible runner version to 0.0.7 not found
```

- The language plugin installed is not compatible with the gauge version installed.
- Run `gauge install language_NAME` to install the latest compatible version. See plugin installation for more details

Execution Errors

```
Error: too many open files
```

- This error occurs when the upper limit to open the number of files is too low. To fix the error, increase the upper limit by adding the command `ulimit -S -n 2048` to your `~/.profile` file and relogin.

Troubleshooting

Product

Get started

Features

Plugins

Documentation

Blog

Press Kit

Explore

Table of Contents

Roadmap

Man page

Insights

Engage

Gitter

Google Group

Stack Overflow

Contribute

Follow us