Package ‘cucumber’

June 29, 2024

Type Package
Title Behavior-Driven Development for R
Version 1.0.4
Description Write executable specifications in a natural language that describes how your code should behave.
Write specifications in feature files using 'Gherkin' language and execute them using functions implemented in R.
Use them as an extension to your 'testthat' tests to provide a high level description of how your code works.
License MIT + file LICENSE
URL https://github.com/jakubsob/cucumber,
https://jakubsob.github.io/cucumber/
BugReports https://github.com/jakubsob/cucumber
Encoding UTF-8
Suggests mockery, box, shinytest2, chromote, covr, knitr, rmarkdown
Config/testthat/edition 3
Imports checkmate, dplyr, fs, glue, purrr, rlang, stringr, testthat
(>= 3.0.0), tibble, withr
RoxygenNote 7.3.2
VignetteBuilder knitr
NeedsCompilation no
Author Jakub Sobolewski [aut, cre]
Maintainer Jakub Sobolewski <jakupsob@gmail.com>
Repository CRAN
Date/Publication 2024-06-29 19:30:02 UTC

Contents

  define_parameter_type ........................................... 2
  step ........................................................................ 3
  test ...................................................................... 4
The following parameter types are available by default:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>Matches integers, for example 71 or -19. Converts value with as.integer.</td>
</tr>
<tr>
<td>float</td>
<td>Matches floats, for example 3.6, .8 or -9.2. Converts value with as.double.</td>
</tr>
<tr>
<td>word</td>
<td>Matches words without whitespace, for example banana (but not banana split).</td>
</tr>
<tr>
<td>string</td>
<td>Matches single-quoted or double-quoted strings, for example &quot;banana split&quot; or 'banana split' (but not banana split).</td>
</tr>
</tbody>
</table>

To use custom parameter types, call `define_parameter_type` before `cucumber::test` is called.

**Usage**

```r
define_parameter_type(name, regexp, transformer)
```

**Arguments**

- **name**
  - The name of the parameter.

- **regexp**
  - A regular expression that the parameter will match on. Note that if you want to escape a special character, you need to use four backslashes.

- **transformer**
  - A function that will transform the parameter from a string to the desired type. Must be a function that requires only a single argument.

**Value**

An object of class `parameter`, invisibly. Function should be called for side effects.

**Examples**

```r
define_parameter_type("color", "red|blue|green", as.character)
define_parameter_type(
  name = "sci_number",
  regexp = "\[[+-]?\d*\.?\d+?e[+-]?\d+\]",
  transform = as.double
)

## Not run:
# tests/testthat/test-cucumber.R
cucumber::define_parameter_type("color", "red|blue|green", as.character)
cucumber::test(".", "/steps")
## End(Not run)
```
Define a step

Description

Provide a description that matches steps in feature files and the implementation function that will be run.

Usage

given(description, implementation)
when(description, implementation)
then(description, implementation)

Arguments

description  A description of the step. A simple version of a Cucumber expression. The description is used by the cucumber::test function to find an implementation of a step from a feature file. The description can contain placeholders in curly braces, e.g. "I have {int} cucumbers in my basket". If no step definition is found an error will be thrown. If multiple steps definitions for a single step are found an error will be thrown. Make sure the description is unique for each step.

implementation  A function that will be run when the step is executed. The implementation function should always have the last parameter named context. It holds the environment where state should be stored to be passed to the next step.

If a step has a description "I have {int} cucumbers in my basket" then the implementation function should be a function(n_cucumbers, context). The {int} value will be passed to n_cucumbers, this parameter can have any name. If a table or a docstring is defined for a step, it will be passed as an argument after placeholder parameters and before context. The function should be a function(n_cucumbers, table, context). See an example on how to write implementation that uses tables or docstrings.

Details

Placeholders in expressions are replaced with regular expressions that match values in the feature file. The regular expressions are generated during runtime based on defined parameter types. The expression "I have {int} cucumbers in my basket" will be converted to "I have [+-]?(?<![.]):digit:+(?![.]) cucumbers in my basket". The extracted value of {int} will be passed to the implementation function after being transformed with as.integer.

To define your own parameter types use define_parameter_type.
Value

A function of class step, invisibly. Function should be called for side effects.

See Also

define_parameter_type()

Examples

given("I have {int} cucumbers in my basket", function(n_cucumbers, context) {
  context$n_cucumbers <- n_cucumbers
})

given("I have {int} cucumbers in my basket and a table", function(n_cucumbers, table, context) {
  context$n_cucumbers <- n_cucumbers
  context$table <- table
})

when("I eat {int} cucumbers", function(n_cucumbers, context) {
  context$n_cucumbers <- context$n_cucumbers - n_cucumbers
})

then("I should have {int} cucumbers in my basket", function(n_cucumbers, context) {
  expect_equal(context$n_cucumbers, n_cucumbers)
})

---

test Run all Cucumber tests

Description

This command runs all Cucumber tests. It takes all .feature files from the features_dir and runs them using the steps from the steps_dir.

Usage

test(
  features_dir,
  steps_dir,
  steps_loader = .default_steps_loader,
  test_interactive = getOption("cucumber.interactive", default = FALSE)
)
Arguments

features_dir  A character string of the directory containing the feature files.
steps_dir    A character string of the directory containing the step files.
steps_loader A function that loads the steps implementations. By default it sources all files
              from the steps_dir using the built-in mechanism. You can provide your own
              function to load the steps. The function should take one argument, which will
              be the steps_dir and return a list of steps.

test_interactive
              A logical value indicating whether to ask which feature files to run.

Value

None, function called for side effects.
Index

define_parameter_type, 2, 3
define_parameter_type(), 4
given (step), 3
step, 3
test, 4
then (step), 3
when (step), 3