Getting Started guide

Introduction to testRigor

testRigor is an AI Agent that allows anyone to create end-to-end tests from an end user's perspective using plain English, therefore eliminating excessive test maintenance related to locator changes. testRigor supports testing on the following platforms:

- Web testing (Windows, MacOS, Ubuntu) and Mobile Web testing on iOS and Android
- Native and Hybrid Mobile App testing for iOS and Android
- Native Desktop applications testing

With testRigor, you can perform various types of testing, including:

- Acceptance testing
- Smoke testing
- Regression testing
- System (end-to-end) testing
- API testing
- Visual testing
- SMS and phone call testing
- 2FA and Captcha testing

To create your end-to-end tests, you have several options:

- Leverage testRigor's Generative AI to create tests based on descriptions
- Write tests from scratch using plain English commands (See this documentation for help)
- Use testRigor's record-and-playback tool

Setting Up an Account

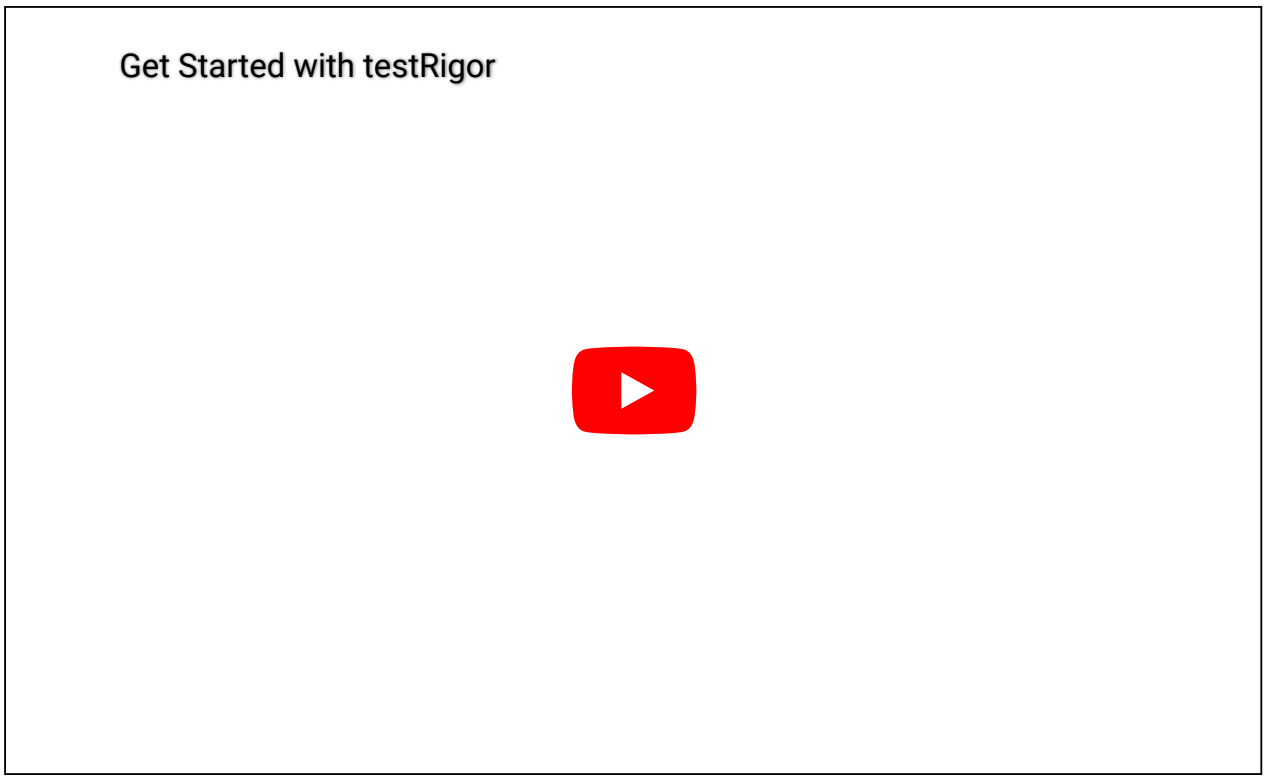
testRigor can be highly customized according to your needs. For the most recent pricing information, please contact our friendly sales team. Here are some options to consider:

- Cloud (default) or on-premise
- Windows, MacOS, Ubuntu, iOS, Android, the number of devices, etc.
- Number of parallelizations for faster test execution speed

To set up an account, visit https://testrigor.com/sign-up/. Select a free public plan or a free trial depending on your preference.

Creating Your First Codeless Test Cases

Tutorial for creating your first test case:



All video tutorials: https://testrigor.com/tutorials/

All recent public test cases https://app.testrigor.com/public-tests.

See more details on step-by-step test creation here.

testRigor English-language support documentation

The goal of testRigor is to allow you to write your tests in your way of saying it in plain English. It is extendable by allowing you to support your own phrases. And you might want to get familiar with test automation best practices and, more importantly testRigor Best Practices.

You can click on elements with `click "Submit"` , enter data with `enter "Peter" into "First Name"` and validate with `check that page contains "Welcome, Peter!"` .

Table of Basic Commands

We support a vast variety of commands. The list below contains many examples of how to express them.

Action	Options	Example

click	double, right, middle, long, times, in a context of, using the mouse, using javascript, without scrolling, using OCR, using OCR only	`click "cart"`
		example
		or:
		`click on the 3rd "hello" 5 times`
		example
		or:
		`click "chrome" in the context of "Pixel 3 XL"`
		example
		or:
		`click in the middle of the screen`
		example
		or:
		`double click on the 3rd "hello"`
		example
		or:
		`right click on the 3rd "hello"`
		example
		or:
		`middle click on the 3rd "hello"`
		Note: You can use middle click to open links in a new tab
		example
		or:
		`click "Best value plan" using OCR`
		example
		or:
		`long click on the 3rd "hello"`
		For web testing you can specify the way we click on an ele as follows:
		`click on "hello" using javascript`
		example
		or:
		`click on "hello" using the mouse`
		example
		Note: In headless mode this option will be ignored as the only met available now is javascript
		or:
		`click on "hello" without scrolling`
		Note: You can use "without scrolling" to prevent testRigor's defaul behavior (scroll to the element before the click).
		or:

Action	Options	Example

		`click on image from stored value "Logo"`
		`click on image from stored value "Logo" with less th`
		`"10" % discrepancy`
		`click on the 6th element by image from stored value`
		`"logo" with less than "10" % discrepancy`
		
		Note: This sample uses an image stored with the name "Logo" in ' Data". The image is used to identify the location of the click. For e this type of click is useful when you need to click on an image map default discrepancy is 20%.
		We also support mobile-specific commands to go to the ho screen or review recently used apps.
		To click at Home Button is available to mobile Android and using the commands:
		
		`press home`
		`click home`
		`press home button`
		`click home button`
		
		To click the Recent Button is available on mobile Android u the following commands (Android only as iOS doesn't have button):
		
		`press recent`
		`click recent`
		`press recent button`
		`click recent button`

Action	Options	Example

generate	Simple Template, unique email, unique name, RegEx, Google authenticator code	Generate a unique email in testrigor-mail.com domain:
		`generate unique email, then enter into "Email" and save as "newEmail"`
		example
		Generate a unique name:
		`generate unique name, then enter into "Name" and sav as "generatedName"`
		example
		Generate a unique phone number:
		`generate from template "###-###-####", then enter in "Phone" and save as "generatedPhone"`
		example
		Generate a unique string of letters/numbers:
		`generate from template "%**********", then enter int "Description" and save as "generatedDescription"`
		example
		Generate unique the date and time with unique parameters
		`generate from template by string with escaped parameters "${nowDateTimeIso}-**********", then ente into "Data" and save as "generatedData"`
		example
		Generate an email address in a custom domain:
		`generate from template "$**********@testrigor-mail.com" then enter into "Email" and save as "newEmail"`
		example
		Generate multi-line:
		`generate from regex text starting from next line and ending with [END]`
		`[a-z0-9]{18}`
		`[END], then enter into "JumboInput" and save as "JumboInput"`
		example
		Generate and type:
		`generate from template "%$$$$$$$$$$", then enter and save as "generatedName"`
		example
		Generate and save only:
		`generate from regex "[A-Z][a-z]{30}", and save as "generatedName"`
		example
		Generate Google authenticator time-based one-time passw (TOTP) code for 2-step or multi-factor authentication. It nee user QRCode saved in a stored value or the user text secre generate the code:

Action	Options	Example

		`generate google code from stored value "qr-code-imag`
		`and save it as "code"`
		`generate totp code using "mysecret" and save it as "code"`
		`generate pin with totp from stored value "myStoredSecret" and save it as "code"`
		
		setup and example

enter	enter, tab, escape, delete, backspace, ctrl+a, arrow right, arrow left, arrow up, arrow down, go, search, send, next, previous, done	`enter stored value "actionNotes" into 3rd "Notes"`
		
		example
		Note: Enter supports selects/dropdowns/checkboxes/etc.
		For checkboxes, you can do the following:
		
		`enter "1" into "my_checkbox"`
		
		example
		For enter, tab, escape, delete, backspace, ctrl+a, arrows or Android-specific go, search, send, next, previous, done:
		
		`enter enter into "Notes"`
		
		example
		For multi-line inputs:
		
		`enter text starting from next line and ending with [END]`
		`line1`
		`line2[END] into "Notes"`
		
		example
		For selects or dropdowns:
		
		`select "code or value" from "MySelect"`
		
		example
		***Note:** Avoid using the "enter into" command when the curs already focused on the desired field. For these cases, use "type" command instead.
		It is also possible to select an option by position in selects dropdowns:
		
		`select 1st option from "MySelect"`
		`select second option from "MySelect"`
		`select option 10 from "MySelect"`
		
		***Note:** Due to the differences required for testRigor to inter with desktop applications (i.e., browsers) versus the web application UI, the "select" commands were created in orde interact with native browser dropdowns/selects that are tag <select> in the dom. If your dropdown is a custom dropdow preferable to interact with it as a manual user would (i.e., v clicks, scrolls, etc).

Action	Options	Example

type	`enter, tab, escape, delete, backspace, ctrl+a, arrow right, arrow left, arrow up, arrow down, go, search, send, next, previous, done`	It should be mainly used for typing free text into input fields may type text without referencing an input field if the previc step has already placed the cursor within the input field you to use:
		`click on "Notes"`
		`type "This is a test."`
		example
		For multi-line inputs:
		`click on "Notes"`
		`type text starting from next line and ending with [E`
		`line1`
		`line2`
		`line3[END]`
		example
		Press keyboard keys:
		`type enter`
		example
		Or you can press arrow keys like this:
		`type arrow right`

copy and paste	`text`	Copy selected text (highlighted by either double clicking or dragging the mouse) to the clipboard:
		`copy selection to clipboard`
		`copy to clipboard`
		`copy selection`
		`copy`
		Copy text from string (non-headless only):
		`copy to clipboard value "text_to_copy"`
		`copy value "text_to_copy"`
		example
		Copy text from saved value (non-headless only):
		`copy to clipboard from "variableName"`
		`copy from "variableName"`
		example
		In order to paste copied text click on the input field and use 2 commands:
		`paste from clipboard`
		or simply
		`paste`

Action	Options	Example

check	`page`	Check page content by string:
		`check that page contains stored value from "actionNotes"`
		example
		or
		`check that page contains "Hello"`
		example
		Check page content (including the OCR recognized texts) b string:
		`check that page contains "Best value plan" using OCR`
		example
		Check page content using OCR recognized texts only:
		`check that page contains "Best value plan" using OCR only`
		example
		Check that page did not change after some action:
		`check that page didn't change compared to the previo step`
		example
		`check that page didn't change`
		example
		Note: This action compares page images pixel to pixel
		Check that page doesn't have a fourth button:
		`check that page doesn't contain 4th button`
		Validate page with Vision AI:
		`check that page "contains a positive message" using`
		Look for UI/UX issues using Vision AI:
		`check page for UI errors`
		`check page for UI errors reporting major errors or higher`
		`check page for UI errors treating errors as major or lower`
		`check page for UI errors reporting major errors or higher treating errors as major or lower`
		`check page for UI errors treating errors as major or lower reporting major errors or higher`
		Priority options are: minor, major, critical, blocker

Action	Options	Example

check	`element`	`check that "element" contains stored value from "actionNotes"`
		example
		and
		`check that "payout" contains "1000.00"`
		example
		You can also validate whether an input/checkbox/button/etc disabled, clickable, or enabled
		`check that button "Add to Cart" is disabled`
		example
		Check stored value itself:
		`check that stored value "createdName" itself contain "James"`
		example
		Check that an element looks the same as it did on the sam during last successful run:
		`compare image of "my_div" to previous version with allowance of "5%"`
		Check that the entire screen looks the same as during last successful run:
		`compare screen to previous version`
		`compare screen to previous version with allowance of "5%" treating error as "minor"`
		`compare screen to stored value "Saved Screenshot" treating error as "minor"`
		example
		Check that SMS is received:
		`check that sms to "+12345678902" matches regex "Code\:\d\d\d\d" and save it as "sms"`
		Check that file is downloaded and check its content:
		`check that file "instruction.pdf" was downloaded`
		example
		`check that downloaded file contains "app"`
		example
		`check that downloaded file "agreement.pdf" contains "agreement"`
		example
		`check that downloaded file "agreement.pdf" does not contain "liability"`
		example
		`check that file contains saved value "variableName"`
		example
		You can also reuse the downloaded file or multiple files to u them later in the test. For this you need the following actior

Action	Options	Example

		`check that file was downloaded and save it as "DownloadedFiles"`
		
		The variable "DownloadedFiles" will contain all files downlc after the last check for downloaded files. Then to upload th on some other page, just use the following action
		
		`enter stored value "DownloadedFiles" into "upload-image"`
		
		example
		You can also extract text from a downloaded file and save i variable.
		
		`check that file was downloaded and save text as "file_text_1"`
		
		Check that a container contains a nth element:
		
		`check that "container" contains 2nd "element"`
		
		Check that an element is changing. This is useful to check some element that contains animation or video is updating (desktop web browser testing and Android only):
		
		`check that "sparkles" is changing`
		`check that video is playing`
		`check that video "movie" is playing`
		
		You can also use a negative check:
		
		`check that "sparkles" is not changing`
		`check that video is not playing`
		`check that video "movie" is not playing`
		
		Validate element with Vision AI:
		
		`check that "element" "contains a positive message" using ai`

check	`statement is true/following statement is correct`	Utilize Vision AI tovalidate statement about the page `check that statement is true "page contains TestRigo logo"`
long press		`long press on the 3rd "element"`
hover		`hover over 3rd "element"` example
open new tab		`open new tab` example

Action	Options	Example

switch tab		`switch to tab 3`
		example
		or
		`switch to tab "popup"`
		Note: In order to easily keep track of tab numbers, testRigor's defa setting is to begin each test case in a new browser window. Theref 1 is always the tab the test case starts in, and tab 2, 3 and so on a new tabs in the order that they are opened throughout the test cas Note that quotation marks are not needed for tab numbers. Quotat marks are needed for tab names only. For pop-ups, instead of a ta number, you will need to use the popup window name or title.

close tab		`close tab` example
go back		`go back` example
go forward		`go forward` example
reload		`reload` example
reset to home		`reset to home` example
restart app		Restarts application without clearing data. `restart app` **Note:** This action is for mobile devices only.

Press **Ctrl+F** to search

LANGUAGE DOCUMENTATION

- Getting Started guide
- Basic Commands
- Reusable Rules (Subroutines)
- Referencing locations
- Using variables
- Loops
- Validations
- Visual Testing
- Working with Tables
- Conditional execution
- API Testing
- Uploading files
- Phone calls
- SMS messages (Phone Text)
- Login support
- Email testing
- Browser cookies, localStorage, sessionStorage, userAgent
- Comments
- Audio Testing
- Database Query
- Chrome Extension Testing
- Captcha resolution
- Scan QR Code

ADVANCED

AUTOMATE TESTING TABLES

JSONPATH OVERVIEW

REGEX SUPPORT

SIMPLE TEMPLATES

WHAT IS BOOLEAN LOGIC

Action	Options	Example

drag	element, file	`drag "element1" to "element2"`
		example
		Or file:
		`drag file "<URL>" onto "element"`
		example
		`drag file from saved value "File to upload" onto "element"`
		example
		Or draw on a canvas:
		`drag "canvas1" with offset "0,0" to "canvas1" with offset "50,50"`
		Drag mouse to multiple points without releasing the click:
		`drag "canvas1" with offset "0,0" to "canvas1" with offset "50,0" via "canvas1" with offset "0,50" throu "canvas1" with offset "50,50"`
		`drag "canvas1" with offset "0,0" via "canvas1" with offset "0,50" through "canvas1" with offset "50,50" "canvas1" with offset "50,0"`
		If you need to drag a folder with files, you need to zip it anc upload in Test Data section and use it like so:
		`drag folder from saved value "Zipped Folder" onto "element"`
		example
		Note: This action does NOT work in headless mode. It also does N work with Internet Explorer

Action	Options	Example

scroll	down, up, left, right	`scroll down`
		example
		Or:
		`scroll down on "right_panel"`
		example
		Partial scroll:
		`scroll down by 1/2 of the screen`
		example
		You can also scroll directly to a specific part of the page:
		`scroll down until page contains "Submit"`
		example
		Or:
		`scroll down until page contains link exactly "Contac Us"`
		example
		Using the mouse:
		For pages with several scroll areas, you can also scroll up down on a specific portion of text using the mouse wheel:
		`scroll down on "text" until page contains "Sign here using the mouse`
		`scroll down on "text" using the mouse until page contains "Sign here!"`
		If the target text is very far from the starting point of the scr can focus scrolls on the **area** of the text instead of the text by using mouse wheel action in a loop:
		`scroll down on "text" using the mouse up to 15 times until page contains "Sign here!"`
		Important Note: The `scroll until page contains` method w best in `Visible first` mode. `Visible first` prioritizes only visible in the viewport/screenshot. `Batched mode` opens testRig visibility to anything that is loaded on the page whether it is visible end user or not. (To toggle this setting, you can find the dropdown Settings->Speed optimizations->Performance->Getting visibility of elements approach)

swipe	down, up, left, right	`swipe right`
		example
		Or:
		`swipe down on "right_panel"`

wait	time	`wait 3 sec` example **Note:** 2 min max

Press **Ctrl+F** to search

LANGUAGE DOCUMENTATION

- Getting Started guide
- Basic Commands
- Reusable Rules (Subroutines)
- Referencing locations
- Using variables
- Loops
- Validations
- Visual Testing
- Working with Tables
- Conditional execution
- API Testing
- Uploading files
- Phone calls
- SMS messages (Phone Text)
- Login support
- Email testing
- Browser cookies, localStorage, sessionStorage, userAgent
- Comments
- Audio Testing
- Database Query
- Chrome Extension Testing
- Captcha resolution
- Scan QR Code

ADVANCED

AUTOMATE TESTING TABLES

JSONPATH OVERVIEW

REGEX SUPPORT

SIMPLE TEMPLATES

WHAT IS BOOLEAN LOGIC

Action	Options	Example

call api	URL	`call api "https://testrigor.ai/api" and save it as "result"`
		example
		Or with more parameters:
		`call api post "http://dummy.restapiexample.com/api/v1/create" with headers "Content-Type:application/json" and "Accept:application/json" and body "{\"name\":\"James\",\"salary\":\"123\",\"age\":\"32\" and get "$.data.name" and save it as "createdName" check that stored value "createdName" itself contain "James"`
		example

mock api call	URL, returning (headers, body or http status code)	`mock api call "https://jsonplaceholder.typicode.com/todos/1" returning body "{\"mock\": \"mocked response\"}"`
		example
		Or with more parameters:
		`mock api call POST "https://jsonplaceholder.typicode.com/todos" with headers "Content-Type:application/json" and "Accept:application/json" and body "{\"desc\":\"New Todo\"}" returning payload "Todo created" and http code 200`
		example

grab value	Simple Template, RegEx, element	`grab value from "element" and save it as "variableName"`
		example
		or, to grab a value from `UserName:53vhhsw1fi@testrigor-mail.com`
		`grab value of "(?<=UserName\:)[^ ]+" and save it as "generatedUsername"`
		or:
		`grab value of "(?<=UserName\:)[^ ]+" from "generated_section" and save it as "generatedUsernam`
		example
		or, to grab a value using simple template:
		`grab value by template "(###) ###-####" and save it "phoneNumber"`
		or:
		`grab value by template "(###) ###-####" from element below "Phone" and save it as "phoneNumber"`
		You can also grab multiple values from a table row or colun value will be stored as a JSON Array)
		`grab values from "my-table" at first column and save it as "first-column-values"`
		`grab values from "my-table" at first row and save it as "first-row-values"`

Action	Options	Example

extract value by regex or template	element	`extract value by regex "regex" from saved value "variableFrom" and save it as "variableTo"`
		For example, to extract a value of username from variable containing `UserName:53vhhsw1fi@testrigor-mail.com` you use:
		`extract value by regex "(?<=UserName\:)[^ ]+" from "var1" and save it as "generatedUsername"`
		example
		`extract value by template "######" from stored value "variableFrom" and save it as "variableTo"`
		For example, to extract the ID number from variable `var1` containing `userID:123456` , you could use:
		`extract value by template "######" from stored value "var1" and save it as "idNumber"`

save value		`save value "Peter" as "name"` example
store clipboard		Clipboard current value can be saved into a variable for lat usage: `store clipboard value as "variable"` example
open url	URL	`open url "https://testrigor.ai?d=\"` example **Note:** The domain must be whitelisted!
grab url		You can grab the browser's current URL and save it into a variable: `grab url and save it as "variableName"` example

Press **Ctrl+F** to search

LANGUAGE DOCUMENTATION

Getting Started guide

Basic Commands

Reusable Rules (Subroutines)

Referencing locations

Using variables

Loops

Validations

Visual Testing

Working with Tables

Conditional execution

API Testing

Uploading files

Phone calls

SMS messages (Phone Text)

Login support

Email testing

Browser cookies, localStorage, sessionStorage, userAgent

Comments

Audio Testing

Database Query

Chrome Extension Testing

Captcha resolution

Scan QR Code

ADVANCED

AUTOMATE TESTING TABLES

JSONPATH OVERVIEW

REGEX SUPPORT

SIMPLE TEMPLATES

WHAT IS BOOLEAN LOGIC

Action	Options	Example

`send email`		Here is how to send a simple email:
		`send email to "user@customer.com" with subject "Tes message" and body "Hi, this is a test, this is just test message."`
		example
		To send an email with an attachment, you can upload the fi testRigor store in "Test Data" section, then use it by name a stored value. Alternatively you can use your own URL. If yo choose to attach a file from your own URL, the link should l downloable.
		`send email from "sender@customer.com" to "recipient@customer.com" with subject "Test message and body "Hi, this is a test, this is just a test message.", and attachment from saved value "Sample File"`
		example
		`send email from "sender@customer.com" to "recipient@customer.com" with subject "Test message and body "Hi, this is a test, this is just a test message.", and attachment "http://online.com/file/name.pdf"`
		example

Action	Options	Example

`check email`		`check that email from "user@customer.com" is deliver`
		
		Or
		
		`check that email to saved value "newEmail" was recei`
		
		example
		By default testRigor renders emails in a desktop browser. I want to render it in the mobile device (available for Android iOS), you need to add the following suffix: `and show in mo`
		E.g.:
		
		`check that email to "user@customer.com" and "Confirm in subject was received and show in mobile`
		
		example
		You can use both - sender and recipient
		
		`check that email from "user@customer.com" to saved value "newEmail" was received`
		
		example
		By default we assume that the user expects one email mes Multiple emails will trigger an error message. It is possible customize this behavior by specifying exactly how many en are expected or just say "one or more".
		
		`check that email to "user@customer.com" and "Confirm in subject was received`
		
		example
		
		`check that two emails to "user@customer.com" and "Confirm" in subject were delivered`
		
		example
		
		`check that one or more emails to "user@customer.com" and "Confirm" in subject were delivered`
		
		example
		Also it is possible to filter messages by subject
		
		`check that email to "user@customer.com" and "Confirm in subject was received`
		
		example
		Notes:
		1. This user action will render the email as if it is a web page. T address here is not the same as "To" address in the email. `that email to` will validate that email is delivered to one c email recipients (To, Cc, Bcc). If there are more than one rec several actions may check for every recipient's address.
		2. When you specify a subject, it is treated as a regular expres match a possible email subject. E.g. "Welcome \w+" will mat subject starting with "Welcome " and following some word, lik "Welcome John", "Welcome Henry", etc.

Action	Options	Example

reply to email		`reply to email from "user@customer.com" with "confirmed"`
		Or
		`reply to email to saved value "newEmail" with "confirmed"`
		example
		`reply to email to saved value "newEmail" from "user@customer.com" and subject "Confirm" with "confirmed"`
		`reply to email to saved value "newEmail" and subject <regular expression>" with "confirmed"`
		Note: this command will render the received email (not the respon it is a web page

call	phone number	`call "+15344297154" and check it was picked up`
		example
		Or, simply:
		`call "+15344297154"`

sms	phone number	`sms to +15344297154 with body "hello"`
		Or
		`message +15344297154 with body "hi from testRigor" a verify it was delivered`

set geo location (GPS Coordinates)		`set geo location "40.7128,74.0060"`

start browser / start device	name	Starts a new browser with its own new session.
		`start browser "User 2"`
		example
		Start and switch:
		`start browser "User 2" and switch`
		example
		For device:
		`start device "User 2"`
		example

Press **Ctrl+F** to search

LANGUAGE DOCUMENTATION

- Getting Started guide
- Basic Commands
- Reusable Rules (Subroutines)
- Referencing locations
- Using variables
- Loops
- Validations
- Visual Testing
- Working with Tables
- Conditional execution
- API Testing
- Uploading files
- Phone calls
- SMS messages (Phone Text)
- Login support
- Email testing
- Browser cookies, localStorage, sessionStorage, userAgent
- Comments
- Audio Testing
- Database Query
- Chrome Extension Testing
- Captcha resolution
- Scan QR Code

ADVANCED

AUTOMATE TESTING TABLES

JSONPATH OVERVIEW

REGEX SUPPORT

SIMPLE TEMPLATES

WHAT IS BOOLEAN LOGIC

Action	Options	Example

switch to browser / switch to device / switch to remote desktop	name	`switch to browser "User 2"`
		example
		For device:
		`switch to device "User 2"`
		example
		Get back to the first browser:
		`switch to browser "default"`
		example
		For remote desktop:
		`switch to remote desktop "default"`
		Note: This command is only available for Windows Remote Deskto suites. It will return to the first remote desktop if the user has switc browser at some point during the test case execution.

switch context to native / switch context to browser		Switch the context of the following actions to the entire mob device (as for application testing) or start the mobile brows use only its content (as for web testing).

Action	Options	Example

compare	type of element	Compares current elements on the current screen with the appearing the previous time the screen was shown.
		`compare inputs`
		`compare texts`
		`compare buttons`
		example
		The following sentence compares element on the current te screen with the one appearing on the reference screen or s as a file pixelwise. If the element appears to be different, b baseline and current (failed) run element images will be sa the artifacts for manual investigation.
		`compare image of "my_div" to previous version with allowance of "5%"`
		example
		The sentence must contain an image definition, such as `ir of text "pressed"` , `image of 3rd "peter"` , etc. Optiona may contain an allowance value in percent. If missing, then assumed to be zero. If element size changed, centered intersection of the element is compared. Pixels are compar using mean squared error.
		Silimarly you can make a screenshot of an image, save it, a then use as a reference like so:
		`compare image of "logo" to file "https://testrigor.com/assets/images/logo.png" with allowance of "5%"`
		example
		`compare image of "logo" to stored value "logoFile" with allowance of "5%"`
		example

login		`login` example Complex action. Identifies and performs the necessary step login.

crawl sitemap / go through sitemap		testRigor supports XML and Text sitemap formats.
		`crawl sitemap "https://app.testrigor.com/sitemap.xml`
		example
		`crawl sitemap "http://online.com/sitemap.txt"`

execute JavaScript in the browser		testRigor supports execution of vanilla Javascript in the brc
		`execute JavaScript in the browser text starting from next line and ending with [END]`
		`document.querySelector("#input1").value = "John Doe"`
		`[END]`
		example

Action	Options	Example
accept alert/prompt	`prompt value`	Browser alert/prompt can be accepted with a value: `accept prompt with value "John Doe"` example

change device orientation	`portrait, landscape`	Device orientation can be changed to portrait/landscape on android/iOS device:
		`change device orientation to landscape`
		`change device orientation to portrait`

zoom	`in, out, open, close`	In native mobile testing (android/iOS), zoom can be applied the pinch gesture:
		`zoom in`
		`zoom out`
		The zoom value is set to 50% by default, and it is applied fr center of the screen. It is possible to specify the screen or element from which the motion will depart, as well as an offs
		`zoom in "20" % from the middle of the screen`
		`zoom out "10" % from "element" with offset "10,10"`
		Another syntax option is to use pinch open/close instead of in/out; both methods function the same.
		`pinch open "20" % from the middle of the screen`
		`pinch close "10" % from "element" with offset "10,10`

open deeplink	`URL, package or bundle`	In mobile testing (web and native), deeplinks can be opened passing the URL:
		`open deeplink "URL"`
		For package/bundle information, you can pass that explicitly command.
		Example: opening a deeplink to a specific place using the " app on Android:
		`open deeplink "geo:0,0?q=eiffel+tower+paris" with package "com.google.android.apps.maps"`
		For Android, if no package information is provided, then the default application package will be used.
		Example: opening up accessibility options inside the setting on iOS 17:
		`open deeplink "app-prefs:ACCESSIBILITY"`
		For iOS, if no bundle information is provided, then the defa application for the given url scheme is going to be used.

install application	`alias of mobile application`	Extra mobile application can be uploaded under Settings->Multiple applications and installed on testRigor android/iO device. `install application "whatsapp"`

Action	Options	Example

hold key, release key	Control, Shift, Alt, Command, F1, F2,...F12, Enter, Space, Delete, Backspace	Hold key could be use in combination with a click.
		`hold key command and click "button"`
		`hold key ctrl and long click "item 2"`
		`hold key command and click exactly "Item 9"`
		Example of use of Hold and Release
		`hold key command`
		`click "item 1"`
		`check that page contains "item 5"`
		`click "item 5"`
		`click "item 9"`
		`release key command`

Reusable Rules (Subroutines)

If you have a sequence that will be used often, you can save them as a Reusable Rule and refer to it with the name of your choosing.

Example: The first 6 steps of the example below will take you to your checkout page. If you frequently perform this process, create one rule to simulate all of those steps.

```
login
click "Men Clothing"
scroll down
click "Men's cargo shorts"
click "brown"
click "Size 34"
click "Add to cart"
check that page contains "Your order is nearly complete!"
```

example

Then, go to the Reusable Rules section in testRigor, assign the name "go to checkout page" to your rule, and add the first 6 steps above. From then on, you can simply use the syntax below to trigger all 6 steps:

```
go to checkout page
check page contains "Add to cart"
```

You can also create rules with dynamic parameters.

Rule name:

```
search "product" click on the link and then press "button"
```

Steps: (We're going to create variables with the same name you define between quotes.)

```
enter stored value "product" into "search"
click link stored value "product"
click stored value "button"
```

example

Then you can call the rule like:

```
search "Computer" click on the link and then press "Add to Cart"
```

The variables defined are scoped and can only be accessed inside the rule.

Referencing Locations and Elements

All references must be in double quotes, which can be escaped by backslash. There are several ways to refer to elements (for checks, clicks, entering data, etc.).:

Attributes

The following list contains the supported **attributes** that testRigor can interact with. **Attributes are searched for without the need to refer to them in the script.** The command `check that page contains "peter"` will search through elements with the following attributes:

Attributes for Desktop web browser and mobile web browser testing

Desktop web browser and mobile web browser testing support the following attributes:

1. Nested text - example
2. Placeholder - example
3. Value - example
4. data-tid/data-testid/data-test-id/Id/resource-id/data-id - example
5. Name - example
6. aria-label - example
7. CSS Class - example
8. Label from ML classification - example
9. Hint/Title/Tooltip - example
10. Alt/Src - example
11. For inputs/edits/dropdowns/selects/etc. will also search corresponding label - example

*__Note:__ In addition to the above mentioned attributes, custom attributes can be added for desktop web browser testing in ***Settings*** => ***Advanced*** => ***Custom attributes to consider for finding elements new line separated***.

Attributes for native mobile application testing

Native mobile applications support the following attributes:

Android

1. content-desc
2. class (usually named something like `android.widget.TextView`)
3. resource-id
4. text/label

iOS

1. accessibility-id
2. XCUIElementType
3. name
4. text/label

*__Note:__ Hybrid applications tested on testRigor infrastructure use the attributes both from web browser applications and native mobile applications.

Attributes for remote desktop application testing (only available if Windows Application Driver is properly set up)

1. AutomationId
2. Name

Generic Indexes

Generic indexes are allowed for when multiple instances of the same element are on the page. For example:

```
check that second "peter" color is "ffaabb"
check that 2nd "peter" color is "ffaabb"
```

example

Types

Types allow us to differentiate elements that have the same name on the screen. For example, if there is both a button and an input placeholder named "Search" on the page and you want to click the button, the way to specify it would be `click button "Search"` . Supported types are:

1. text - example
2. label - example
3. button - example
4. link - example
5. input (or "edit" or "field") - example
6. dropdown (or "select") - example
7. checkbox (or "switch") - example
8. radiobutton - example
9. file input (or "input file" or "edit file" or "input type file"): specifically an input of type file - example

Typed indexes

Typed indexes are a combination of general indexes and types. For example, `check that second input "peter" color is "ffaabb"` will find the second input/edit named "peter".

example

Stored value for controls

Stored value for controls is the use of a variable to find and element or control. For example, `check that stored value "peter" color is "ffaabb"` will resolve the stored value for "peter" and use that resolved value to find the control.

example

Image class

Image class uses image recognition technology to identify common icons. For example, `click on "cart"` . testRigor classifies images on the screen and if there is a button which looks like a shopping cart it will click on it. Complete list of all button image categories is here.

example

Stored value for data

For example, `check that "peter" contains stored value "actionNote"` will find the second input/edit related to "peter".

example

All the ways to refer an element described above apply to all types of actions like check, enter, click, etc. The options above can be combined together.

Multiple references support

Multiple references allows us to use `or` in instances where the targeted element has two possible names. In certain cases, certain elements might be expected to have different names but mean the same thing. For example:

```
click "checkout" or "submit"
```

example

It can also be combined with "if exists" to avoid failing if the element doesn't exist:

```
click "checkout" or "submit" if exists
```

example

Selecting elements in reference to other elements

You can select elements based on it's position in relation to some other element (otherwise known as the "anchor"). For example, you can refer to elements in certain sections like:

```
click on button "Delete" below "Actions"
```

Supported relative locations are:

1. `to the left of` - example
2. `to the right of` - example
3. `above` - example
4. `below` - example
5. `on the right top of` - example
6. `on the left top of` - example
7. `on the right bottom of` - example
8. `on the left bottom of` - example
9. `near` - example

You can also refer to the elements by 2 references:

```
click on button "Delete" below "Section Name" to the right of "label"
```

example

By default, testRigor will consider elements that are located at least 30% between the the target area (the yellow lines that extend from the anchor on screenshots). When the element is located less than 30% between the target area, certain keywords can be used to specify how the relative position should be used:

1. `roughly` - used to search for an element anywhere on the screen in the direction of the relative location specified starting from the anchor (e.g., anywhere below the anchor for "below", anywhere to the right of the anchor for "to the right of") - example
2. `with at least "10" percent overlap` - used to specify when less than 30% of the element falls between the target area - example
3. `completely` - As the first relative location in a command defaults to "roughly" when there multiple anchors, this keyword allows the first relative location to retain it's default meaning (i.e., it is only necessary when more than one anchor is specified and you need to keep the default meaning of the relative location) - example

For example:

```
enter "Peter" into roughly below "Section"
enter "Peter" into element with at least "1" percent overlap on the right of
"Description"
```

However, if there are 2 anchors specified, the first anchor defaults to "roughly" and the second one uses the default behavior. For example,

```
enter "Peter" into "Section" below "Type" and on the right of "Description"
```

is equivalent to:

```
enter "Peter" into "Section" roughly below "Type" and on the right of "Description"
```

however, if you want "below" to keep its default meaning, you would need to say:

```
enter "Peter" into "Section" completely below "Type" and on the right of
"Description"
```

Selecting elements in the context of other elements

You can select elements within the context of other elements. For example, you can pinpoint row at the table and ask to click a button in the context of that row:

```
click on "Delete" within the context of table "actions" at row containing "id1" and column "Actions"
```

example

You can use references to other elements (previous section) in the context itself

```
click on "Delete" within the context of "sectionTwo" below "Actions" and to the right of "rowName"
```

Selecting elements using reference to other elements and in the context of other elements

You can narrow where to look for an element using a combination of references to and in the context of other elements.

1. Context is processed first, narrowing what to consider to what is inside that element
2. References to other elements is processed next, only considering what was delimited by the context

```
click on "Button" below "Title" in the context of "sectionOne"
```

example

```
click on "Button" below "Title" and to the right of "leftHeader" in the context of "sectionTwo"
```

example

You can even specify up to 4 references to other elements using this combination

```
click on "Button" below "Subtitle" and to the right of "leftHeader" in the context of "sectionOne" below "Title" and above "Second Title"
```

example

Specifying the type of an element

You can force the system to only deal with a certain type of elements. The following command will not click on any text that is not a button or link:

```
click on strictly button "Delete"
```

example

Case sensitivity and exact matches

You can choose to deal with only "exactly" matches (i.e., case sensitive full strings match, or just say "case sensitive" to make matching case sensitive):

```
click on exactly "Delete"
```

example

It can be combined with the strict type selection like so:

```
click on strictly button exactly "Delete"
```

example

All supported modifiers to a string being searched:

1. `case sensitive`
2. `exactly` (case sensitive full string must be exactly the same)
3. `full string` (not necessarily case sensitive)
4. `contains template` contains substring which matches a simple template

5. `contains regex` contains substring which matches a regular expression
6. `matches template` the full string must match a simple template
7. `matches regex` the full string must match a regular expression
8. `strictly less` lexicographically strictly earlier (empty strings are always not less)
9. `less or equal` lexicographically earlier or equal (empty strings are always not less or equal)
10. `strictly more` lexicographically strictly later (empty strings are always not more)
11. `more or equal` lexicographically later or equal (empty strings are always not more or equal)

Fine-tuning access

In some cases, the target might be too wide, in this case you can force it to go deeper:

```
click deepest element "Delete"
```

example
It will issue click on the deepest element in the element tree that has the text.
You can also to force it go shallow elements. It is useful when the "Prioritize the deepest element" setting is enabled and we don't want to click on the deepest element:

```
click enclosing element "Delete"
```

example
It will issue click on the enclosing element in the element tree that has the text.

Specifying position on screen

It is possible but highly discouraged (for stability reasons) to use offsets to click on specific part of an element:

```
click on "Delete" with offset "20,10"
```

offset is calculated from top left corner of the element, horizontal coordinate first. It is also possible to specify some generic positions like:

```
click in the middle of the screen
```

The supported positions are:

1. `in the middle of the screen` - example
2. `in the top quarter of the screen` - example
3. `in the second top quarter of the screen` - example
4. `in the bottom quarter of the screen` - example
5. `in the second bottom quarter of the screen` - example
6. `in the top third of the screen` - example
7. `in the bottom third of the screen` - example
8. `in the left quarter of the screen` - example
9. `in the second left quarter of the screen` - example
10. `in the right quarter of the screen` - example
11. `in the second right quarter of the screen` - example
12. `in the left third of the screen` - example
13. `in the right third of the screen` - example
14. `in the left side of the screen` - example
15. `in the right side of the screen` - example
16. `in the top of the screen` - example
17. `in the bottom of the screen` - example
18. `on the left edge of the screen` - example
19. `on the right edge of the screen` - example
20. `on the top edge of the screen` - example
21. `on the bottom edge of the screen` - example
22. `in the left top corner of the screen` - example

23. `in the right top corner of the screen` - example
24. `in the left bottom corner of the screen` - example
25. `in the right bottom corner of the screen` - example

Saved values (variables) support

To use a variable instead of an explicit string, add "stored value" or "saved value" before the variable:

```
validate that stored value "actionNotes" color is "ffaabb"
```

example

```
check that "peter" contains stored value "actionNote"
```

example

```
click on stored value "actionNotes"
```

example

```
check that page contains stored value from "actionNotes"
```

example

testRigor supports generating random values based on RegEx , saving them and using them later in the test. For example:

```
generate from regex "[a-z]{10,18}", then enter into "Notes" and save as
"actionNotes"
```

example

There are 2 special stored values: `"username"` and `"password"` which come from Application-specific credentials settings for your test suite. With it you can do:

```
enter stored value "username" into "username_field"
```

example

```
enter stored value "password" into "password_field"
```

example

These stored credentials are also used by `login` command. Also, you can use stored variables as parameters in most commands by adding a `$` and curly brackets (`${variableName}`):

```
enter from the string with parameters "${homePrefix}/my/path" into "urlPath"
```

example

*Note that using `$` and curly brackets with keywords `string with parameters` allows users to concatenate or join variables with variables or variables with static values in the portion between quotation marks.

Calculations and equations

And calculate expressions like this:

```
check that expression "${a} + ${b}" itself is equal as a number to "42"
```

example

or like this:

```
save expression "${a} + ${b}" as "answer"
```

example

testRigor supports ECMAScript 5.1 compatible expressions.

For example, you can calculate to check that a year from 30 days ago is present on the screen:

```
check that page contains expression "var aDate=new
Date();aDate.setDate(aDate.getDate()-30);''+aDate.getFullYear()"
```

Pre-defined Variables

testRigor supports the following pre-defined saved values:

1. username - `enter stored value "username" into "Email"` example
2. password - `enter stored value "password" into "Password"` example
3. homeDomain - (from URL in Test Suite Details: www.example.com) `open URL from string with parameters "https://${homeDomain}/cart/checkout/confirm"` example
4. homeFile - (from URL in Test Suite Details: /cart/checkout/confirm) `open URL from string with parameters "https://www.example.com${homeFile}"` example
5. homePrefix - (from URL in Test Suite Details: https://www.example.com) `open URL from string with parameters "${homePrefix}/cart/checkout/confirm"` example
6. todayYear - (2023) `enter stored value "todayYear" into "YYYY"` example
7. todayYearShort - (23) `enter stored value "todayYearShort" into "Year"` example
8. todayMonthNumber - (9) `enter stored value "todayMonthNumber" into "Month"` example
9. todayMonthNumberTwoDigits - (09) `enter stored value "todayMonthNumberTwoDigits" into "Month"` example
10. todayMonth - (September) `click stored value "todayMonth"` example
11. todayMonthShort - (Sep) `click stored value "todayMonthShort"` example
12. todayDayOfMonth - (2) `enter stored value "todayDayOfMonth" into "Month"` example
13. todayDayOfMonthTwoDigits - (02) `enter stored value "todayDayOfMonthTwoDigits" into "Month"` example
14. todayDayOfWeek - (Monday) `enter stored value "todayDayOfWeek" into "Day"` example
15. todayDayOfWeekShort - (Mon) `enter stored value "todayDayOfWeekShort" into "Day"` example
16. nowHour - (2 or 14) `select stored value "nowHour" from "Start time"` example
17. nowHourTwoDigits - (02 or 14) `select stored value "nowHourTwoDigits" from "Start time"` example
18. nowHourAmPm - (2) `select stored value "nowHourAmPm" from "Itinerary"` example
19. nowHourTwoDigitsAmPm - (02) `select stored value "nowHourTwoDigitsAmPm" from "Itinerary"` example
20. nowAmPm - (PM) `select stored value "nowAmPm" from "Time of day"` example
21. nowMinute - (5) `enter stored value "nowMinute" into "Start time"` example
22. nowMinuteTwoDigits - (05) `enter stored value "nowMinuteTwoDigits" into "Start time"` example
23. nowSecond - (7) `enter stored value "nowSecond" into "Sec"` example
24. nowSecondTwoDigits - (07) `enter stored value "nowSecondTwoDigits" into "Sec"` example
25. nowNanosecond - (355881000) `save string with parameters "email${nowNanosecond}@testrigor-mail.com" as "newEmail"` example
26. nowDateIso - (2023-09-02) `save stored value "nowDateIso" as "currentDate"` example
27. nowTimeIso - (02:05:07) `save stored value "nowTimeIso" into "currentTime"` example
28. nowDateTimeIso - (2023-09-02T02:05:07.165068-07:00[America/Los_Angeles]) `save stored value "nowDateTimeIso" as "currentDateTime"` example
29. nowMillisecondsFrom1970 - (1637061365335) `save stored value "nowMillisecondsFrom1970" as "timeStamp"` example
30. nowDateTimeRFC1123UTC - (Mon, 16 Sep 2023 16:32:41 GMT) `save stored value "nowDateTimeRFC1123UTC" as "currentDateTime"` example
31. nowUnixTime - (1637061404) `save stored value "nowUnixTime" as "currentUnixTime"` example
32. testSuiteParentFolder - `enter stored value "testSuiteParentFolder" into "Field Name"` example

33. testSuitePath - `enter stored value "testSuitePath" into "Field Name"` example
34. testSuiteName - `enter stored value "testSuiteName" into "Field Name"` example
35. testCaseName - `enter stored value "testCaseName" into "Field Name"` example
36. testCaseExecutionLink - `enter stored value "testCaseExecutionLink" into "Field Name"` example
37. currentUrl - `save stored value "currentUrl" as "url"` example
38. browser - `save stored value "browser" as "variable name"`
39. device - `save stored value "device" as "variable name"`
40. provider - `save stored value "provider" as "variable name"`
41. os - `save stored value "os" as "variable name"`
42. osVersion - `save stored value "osVersion" as "variable name"`
43. abi - `save stored value "abi" as "variable name"`

testRigor also supports grabbing values from elements on the screen and saving them into variables for later usage. For instance:

```
grab value from "some-element" and save it as "my-email"
grab value from input "some-edit" and save it as "my-email"
```

example

You can also grab a set of values from a table row or column (the value will be stored as a JSON Array)

```
grab values from table "some-table" at column "some-column" and save it as "some-column-values"
grab values from table "some-table" at first row and save it as "first-row-values"
```

You can set variables directly without entering it anywhere. For instance:

```
generate from regex "[a-z]{10,18}" and save as "actionNotes"
```

example

```
save value "Peter" as "name"
```

example

Executing actions in a loop

testRigor has limited support for executing commands until a certain condition is true. For example:

```
click "Next" until page contains stored value "previously generated id"
```

example

It can be used for going through pages in long lists or scrolling down until a certain text is visible or certain button is visible. By default the action will be executed up to 10 times, but you can extend maximum number of times by adding "up to":

```
click "Next" up to 12 times until page contains strictly button "Place order"
```

example

You can do the same thing with rules as well:

```
Go To The Next Page until page contains stored value "previously generated id"
Do Something up to 12 times until page contains strictly button "Place order"
```

Validations

You can make multiple types of validations as described below. Validations are by default are "soft" validations, meaning that the execution of the test case will continue even if the validation failed. You can have "hard" validations by adding `and stop test if fails` to the end of the validation. Validations are supported for multiple things like:

1. Finding something on the screen/page
2. Validating properties or content of an element/finding element with given properties and content
3. Visual validations (colors, matching images to of elements to later versions)
4. Downloaded files validations
5. Audio validations (Ubuntu only)
6. Video validations (desktop web browser testing only)
7. Email validations
8. SMS validations
9. Phone call validations
10. Mathematical validations/calculations of formulas (even for data that just looks like financial data like *-$30,000.23*)
11. Built-in API validations
12. Chrome Extensions Testing
13. AI-based validations where you can use AI to assess if certain statements are true
14. Exploratory validations where you can ask AI to look at the screen and find obvious issues
15. And many more

For **full-page validations, URL and page title validations**, we support the following positive and negative assertions:

1. contains/doesn't contain (text). For example, `check that page contains "Error"`
2. contains template/doesn't contain template (text). For example, `check that page contains template "###-###-####"`
3. contains regex/doesn't contain regex (text). For example, `check that page contains regex "(+\d )?\d{3}-\d{3}-\d{4}"`
4. return code is. For example, `check that page return code is "404"`
5. did not change compared to the previous step. For example, `check that page did not change compared to the previous step`
6. url starts with/url doesn't start with. For example, `check that url starts with "https"`
7. url contains/url doesn't contain. For example, `check that url contains "testrigor.com"`
8. url is/url is not. For example, `check that url is "https://testrigor.com/docs/language/"`
9. url ends with/url doesn't ends with. For example, `check that url is "https://testrigor.com/docs/language/"`
10. url matches regex/url doesn't match regex. For example, `check that url matches regex "https://testrigor\.com/docs/\w+/"`
11. title is. For example, `check that page title is "testRigor - Documentation"`
12. title contains. For example, `check that page title contains "testRigor"`

For **element-specific validations**, we support the following positive and negative assertions:

1. contains/doesn't contain (text) - example/example
2. is blank/is not blank - example/example
3. matches regex/doesn't match regex - example/example
4. matches simple template/doesn't match simple template - `check that "section1" matches simple template "$########"`
5. contains simple template/doesn't contain simple template - `check that "section1" contains simple template "$########"`
6. contains regex/doesn't contain regex - `check that "section1" contains regex "[A-Z][a-z]+"`
7. lexicographically before - `check that "section1" lexicographically before "zzzz"` (empty string is not lexicographically before anything)
8. lexicographically before or the same as - `check that "section1" lexicographically before or the same as "zzzz"` (empty string is not lexicographically before or the same as anything)
9. lexicographically after - `check that "section1" lexicographically after "AAAA"` (empty string is not lexicographically after anything)

10. lexicographically after or the same as - `check that "section1" lexicographically after or the same as "AAAA"` (empty string is not lexicographically after or the same as anything)
11. has value/doesn't have value (for inputs/text areas) - example/example
12. is checked/is not checked - example/example
13. is disabled/is enabled - example/example
14. is visible/is invisible - example/example
15. color is - example
16. is clickable/is not clickable - example/example
17. cursor is - example
18. has CSS class - example
19. background color is - example
20. has property - example
21. has options selected (for dropdowns when tagged as `<select>`) - example

```
check that "element" color is "ffaabb"
check that input "input" has value "value"
check that checkbox "Keep me signed in" is checked and stop test if fails
check that checkbox "Keep me signed in" is not checked
```

Validate that an input has no value:

```
check that input "input" has value ""
```

Validate that element has some property:

```
check that property "background-color" of "my-div" is equal to "black"
check that property "width" of "my-div" is equal as a number to "310px"
check that property "height" of "my-div" is greater or equal than "170"
```

For stored values, including: API return value validations, values grabbed from the screen, values extracted from text. We support (positive and negative):

1. matches regex/doesn't match regex - example/example
2. contains/doesn't contain - example/example
3. is equal to/is not equal to - example/example
4. is null/is not null - example/example
5. is blank/is not blank - example/example

You should use the word `itself` to perform validation on a stored value:

```
call api "https://testrigor.com" and save it as "variableName"
check that stored value "variableName" itself contains "James"
```

example

The key to the testing API results is to save it to stored value and then perform validation on stored value itself (with keyword `itself`) as shown above.

Both element and stored value validations support:

1. is equal as a number to/is not equal as a number to - example/example
2. is greater than - example
3. is greater or equal than - example
4. is less than - example
5. is less or equal than - example

You can also enjoy validations that apply to the whole screen/page:

1. page contains/page doesn't contain - example/example
2. page has regex/page doesn't have regex - example/example
3. page's return code is
4. title is - example
5. title contains - example
6. url is/url is not - example/example
7. url contains/url doesn't contain - example/example
8. url starts with/url doesn't start with - example/example
9. url ends with/url doesn't end with - example/example
10. url matches regex/url doesn't match regex - example/example

For example, you can combine stored values for validation:

```
check that url matches regex from the string with parameters
"${homePrefix}/product/${type}/[0-9A-Za-z\-]+"
```

AI-based testing

You can use 2 types of pure-AI-based testing. You have to be cautious since AI can have hallucinations and provide incorrect answer sometimes.

- Asking AI specific questions like `check that page "contains a positive message in the chat response" using ai` - example
- Getting UI issues from the screen `check page for UI errors` - example

Specific AI validations on the screen

You can set validations to confirm that a statement is true about the screen you are currently on like this:

```
check that page "contains a positive message in the chat response" using ai
```

See example. Or, alternatively like so:

```
check that statement is true "page contains TestRigor logo"
```

Or, you can ask AI to validate statements about specific elements on the screen like so: `check that "element" "contains a positive message" using ai` See example

AI discovers issues itself

You can also ask AI to find issues on a screen like this, you can also specify the limit for the severity for the issues:

```
check page for UI errors
check page for UI errors reporting major errors or higher
check page for UI errors treating errors as major or lower
check page for UI errors reporting major errors or higher treating errors as major
or lower
check page for UI errors treating errors as major or lower reporting major errors or
higher
```

See example

Visual Testing

You can verify visual changes on the screen using the following sample steps:

```
compare screen
compare screenshot to previous version
compare screen to previous version with allowance of "1%" treating error as "minor"
compare screen to previous version treating error as "minor"
```

*Remember that this verifies that the entire screen looks the same as it did at this step during last successful run.
It is possible to save a sample screenshot in test data and use it as the point of comparison instead of the previously saved screen image:

```
compare screen to stored value "Saved Screenshot"
```

Regarding the severity parameter, the allowed values are "minor", "major", "critical", or "blocker". The default value is "critical", which marks the test case as failed. Similarly, there is an option to use the "allowance" clause, a minimum acceptable difference in percentages. Any difference below this value will be ignored. You may need to experiment with this parameter. In any case, you can find actual discrepancy in "Extra info" for a particular step. On mobile devices we ignore the status bar on top. This is so because there is a time stamp that may be different for each screenshot.

Clicking on specific images with visual testing

testRigor also offers visual support for clicking on images by indexing for instances where images or icons do not have a simple way to be referred to. To accomplish this, crop the desired image from a screenshot or file in its full resolution and save it in the Test Data Section. To then refer to it in your case, use the syntax below:

```
click on the 6th element by image from stored value "logo" with less than "10" % discrepancy
```

If there is only one image that you want testRigor to recognize, use the following:

```
click by image from stored value "logo" with less than "10" % discrepancy
```

You may need to toggle discrepancy percentages in order to allow testRigor to recognize the image.

Working with Tables

You can refer to table cells by the intersection of the row and column by providing value of the first cell in the row and value of the header/top cell in the column. For example, command:

```
click on table "actions" at row "103" and column "Action"
```

example

For the table

#	Id	Name	Actions	Additional Data
Filter by				
101	york1	Yorktown	Arrive Cancel	Looks like a trap
102	spk2	Spock	Listen to Ignore	
103	nyo3	Nyota	Open channel Promote	

Actions

will result in a click on "Open channel".

You can also specify row by saying that row should contain a certain value. This way we will check all values of every row to find the one which matches. For example, for the same table above, and command:

```
click on table "actions" at row containing "spk2" and column "Actions"
```

example

will result in a click on the first action "Listen to". To click on second action "Ignore" you can leverage our context feature:

```
click on "Ignore" within the context of table "actions" at row containing "spk2" and column "Actions"
```

example

You can also work with multiline headers by referring to them wither "header" word:

```
enter "york1" into first table at the second header row and column "Id"
```

example

testRigor supports tables for all kinds of operations including but not limited to: validations (checks), clicks, hover, entering data (enter ... into), drag and drop, etc. Examples:

```
check that table "actions" at row "102" and column "Name" contains "Spock"
```

example

```
check that table "actions" at row "101" and column "Additional Data" has value
"Looks like a trap"
```

example

```
enter "This is a trap!" into table "actions" at row "101" and column "Additional
Data"
```

example

```
click "Open channel" within the context of second table at row "103" and column
"Action"
```

example

```
check that the second table at row containing "Nyota" and column "Action" contains
link "Open channel"
click the first button within the context of second table at row containing "Nyota"
and column "Action"
```

Row or column value aggregation

You can check certain aggregations and comparations with the data contained in an entire row or column

Order of values

To check the order of the values contained in a row or column

```
check that table "actions" at column "Name" has values sorted in ascending order
check that table "actions" at column "Name" has values sorted in descending order
```

Number of values (count)

To check the value count in a row or column

```
check that table "actions" at column "Name" the value count is greater than "2"
check that table "actions" at column "Name" the value count is less than "2"
check that table "actions" at column "Name" the value count is equals to "2"
```

Sum of values

To check the sum of values in a row or column

```
check that table "actions" at column "#" the sum of values is greater than "100"
check that table "actions" at column "#" the sum of values is less than "500"
check that table "actions" at column "#" the sum of values is equals to "306"
```

Average of values

To check the average of values in a row or column

```
check that table "actions" at column "#" the average of values is greater than "100"
check that table "actions" at column "#" the average of values is less than "300"
check that table "actions" at column "#" the average of values is equals to "155"
```

Conditional execution

testRigor supports conditional execution of commands and rules.

Inline Conditional Command

You can execute a command only if a condition is met, like this:

```
click "element" if page contains "bla"
```

example

```
enter "macbook" into "search" if "Search in" has value "bla"
```

example

In certain cases, certain elements might appear randomly on the screen. For such cases, there is an `if exists` clause. For example:

```
click "element" if exists
```

example

```
enter "bla" into "element" if exists
```

example

The above commands will not fail if element is not found and will be skipped silently.

Inline Conditional Rule

You can execute a rule only if a condition is met, like this:

```
My Rule if page contains "bla"
Purchase Product if "Search in" has value "Products"
```

Conditional blocks

In addition to inline conditionals, which provide a quick way to execute a command or a rule depending on the result of a validation, we also provide conditional blocks to allow you to test multiple conditions with multiple commands and/or rules.
You can execute multiple commands on one condition. For example:

```
if page contains "bla" then
  click "element"
  wait 10 sec
  enter "some text" into "some other element"
end
```

example

Also if you add an `else` clause you could execute other commands when the condition is not met.
For example:

```
if page contains "bla" then
  click "element"
  wait 10 sec
else
  enter "some text" into "search"
  wait 10 sec
end
```

example

Test as many conditions as you want using the `elseif` clause. For example:

```
if page contains "bla" then
  click "element"
  wait 10 sec
else if page contains "search" then
  enter "some text" into "search"
  wait 10 sec
else if page contains "foo" then
  enter "foo bar" into "element"
  wait 10 sec
  click "some button"
else
  wait 10 sec
  logout
end
```

example

In addition to commands you can execute rules inside the conditional blocks. For example:

```
if page contains "bla" then
  check for some words
  open the main menu
  clear all
end
```

Conditional blocks are very flexible cause you can mix commands and rules on every condition. For example:

```
if page contains "Welcome" then
  accept all cookies
  wait 20 sec
  open the main menu
else if page contains "Search" then
  click "Search"
  enter "Big TV's"
  purchase the first item on the list
else
  login
end
```

NOTE: Conditional blocks cannot be nested, we suggest you to use rules to nest complex conditions.

Fail the test on purpose

There is a way to fail the test at will as in the following example:

```
fail
```

example

```
fail with "error"
```

example

API Testing

testRigor supports calling to API's, getting value and saving result as a stored value:

```
call api <TYPE> "<API_URL>" with headers "a:a" and "b:b" and body "body" and get
"JsonPath" and save it as "variableName"
```

For example:

```
call api post "http://dummy.restapiexample.com/api/v1/create" with headers "Content-
Type:application/json" and "Accept:application/json" and body "
{\"name\":\"James\",\"salary\":\"123\",\"age\":\"32\"}" and get "$.data.name" and
save it as "createdName"  and then check that http code is 200
```

example

In the example above, testRigor would validate that return http code is 200. Parameters `"JsonPath"` and `"variableName"` are both optional. However, if `"JsonPath"` is present,

`"variableName"` will be mandatory. Here is one more example,

```
call api "api" and save it as "variableName"
```

JsonPath is the way to refer to parts of JSON and described with examples here. If "JsonPath" is not defined in the action, the complete result of the request will be stored in "variableName".
testRigor supports all HTTP actions for call <TYPE>:

1. get - example
2. post - example
3. put - example
4. patch - example
5. head - example
6. delete - example
7. options - example
8. trace - example

testRigor supports multiple headers separated by "and" like so: `with headers "a:a" and "b:b"`
You can pass JSON into the body of POST message, with double quotes characters escaped like so: "\"". You could use tools like this to escape it for you. For example:

```
and body "{\"name\":\"James\",\"salary\":\"123\",\"age\":\"32\"}"
```

You can use parameters and multi-line strings for constructing a call:

```
call api post from the string with parameters "${homePrefix}/api/v1/create" with
headers "Content-Type:application/json" and "Accept:application/json" and body text
starting from next line and ending with [END]
{
  "param": "value",
  "param2": "value2"
}
[END] and get "$.data.name" and save it as "createdName"
```

And you can pass parameters:

```
call api post from the string with parameters "${homePrefix}/api/v1/create" with
headers "Content-Type:application/json" and "Accept:application/json" and body from
the string with parameters text starting from next line and ending with [END]
{
  "param": "value",
  "param2": "${dynamicValue}"
}
[END] and get "$.data.name" and save it as "createdName"
```

After the API call is executed, if the result was stored into saved value it could be later tested like so:

```
check that stored value "createdName" itself contains "James"
```

Validation of API calls made by the browser (Chrome/Edge only)

Web browsers perform several operations behind the scenes in order to render a page or application, these include multiple requests to retrieve necessary files such as HTML, style sheets, scripts, etc. And also communicating with APIs (application programming interfaces) which provide data in a more dynamic way. testRigor allows you to verify the correct functioning of the mentioned request directly from the steps of the test case through a validation command.
The command accepts several parts for you to filter a particular request or to validate more:

- Request URL: We will find requests to a target URL starting with your input
- Request method: GET, POST, PUT, PATCH, DELETE, etc
- Request headers: filtering among the requests those with headers containing your input
- Request body: filtering requests sending data containing your input
- Response status code: finding a response received with that status code
- Response headers: finding a response received with headers containing the input

testRigor will look for the requests/responses that match the parameters specified on your command, using the priority determined by the following list:

- Filter by request URL matching
- Filter by request method matching
- Filter by request headers matching
- Filter by request body matching
- Filter by response status code
- Filter by response headers

Note: these filters act as a pipeline, so each one will receive the output of the one before.
A validation with all the parameters would look like:

```
check that api call was made to POST from stored value "myURL" with headers containing "some-header-value" and "\"Content-Type\":\"application/json\"" and text starting from next line and ending with [END]
"Referer":"https://testrigor.com"[END] and the request body containing "some data" and the response code was "200" and response headers containing "resp-header-value"
```

Simpler ones can be made of course:

```
check that the browser called api "https://testrigor.com" and response code was "200"
```

```
check that api call was made to url "https://testrigor.com" with headers containing from stored value "myHeader"
```

```
check that api call was made with response headers "some-value" and from stored value "myHeaderVariable"
```

Request body types to filter

testRigor allows the user to specify multiple body parameters to filter requests. Each body parameter can be a TEXT, a JSON or a JSON PATH.

Text body parameter

Text is the default body parameter type, so if the user does not specify a type, we will use the text type.

```
check that api call was made to POST "https://testrigor.com" with body containing "some_data1" and  text "some_data2"
```

JSON body parameter

To use a JSON body parameter, the user must specify the word **json** before the parameter value. The parameter value must be composed of a key and a value divided by a colon (**:**) as in the example below.

```
check that api call was made to POST "https://testrigor.com" with body containing json "key1 : value1" and json "\"key2\":\"value2\""
```

JSON PATH body parameter

To use a JSON PATH body parameter, the user must specify the word **json path** before the parameter value. The parameter value must be a valid json path.

```
check that api call was made to POST "https://testrigor.com" with body containing json path "$.some_path[?(@.value < 10)]"
```

The user can use more than one type of parameter in the same step.
example

Mocking API calls

testRigor supports mocking response data (headers, body and/or http status code) for API calls made inside your application.

```
mock api call <TYPE> "<API_URL>" returning body "<MOCK_BODY>"
```

For example:

```
mock api call get "https://dummy.restapiexample.com/api/v1/employees" with headers
"a:a" returning body "This is a mock response" with http status code 200
```

In the example above any GET calls to the endpoint "https://dummy.restapiexample.com/api/v1/employees" with the headers "a:a" will respond with the testRigor mock, with status 200.
Some useful cases are:

- You might want to use mocking APIs if you are using third-party API calls. Those calls can be charged and expensive, so you can mock the responses instead of calling the real service.
- You can test your application individually if servers are unstable or down.
- You can test specific scenarios, for example, if you want to test a scenario where the server returns error.

Uploading files

File upload is supported out of the box. Just use it like the following:

```
enter stored value "keyName" into input file "fileField"
```

example

```
enter "<FILE_URL>" into file input "fileField"
```

example

The most common way to upload files is without specifying the name of the input:

```
enter stored value "myFile" into input type file
```

You can also specify the `input type file` field by indexing or relative location:

```
enter stored value "myFile" into 3rd input type file
enter stored value "myFile" into input type file in the context of "Select file"
```

*Note: In most cases, clicking on the button that opens the file directory is not necessary to upload the file and should be omitted. If this does not work, you can try clicking on the button, entering the file, and then closing the directory with a command appending `using OCR only using the mouse`.
Where "fileField" is a file input element (i.e., `input type="file" ...`), you can upload files up to 10MB into testRigor storage in the "Test Data" section, then use it by name as a stored value. Alternatively you can use your own URL. If you choose to upload from your own URL, the link should be downloadable. Since files must always be uploaded during executions from the testRigor environment (whether stored in test data or called by URL) and cannot be directly uploaded from users' local storage, only in very rare cases is it necessary to include commands in the script to click the button that prompts the file directory or file finder.
You can upload a file into a testRigor mobile device using the following action:

```
upload file from saved value "sampleFile"
upload file "https://some-page.com/path-to-file" to mobile device
```

The file will be uploaded into `Downloads` folder on the Android device.

Multiple files

Multiple files upload can be done like so:

```
enter from the string with parameters text starting from next line and ending with
[END]
${file_var_one}
<FILE_URL>
[END] into file input "myFileInput"
```

example

Working with folders

If the input element has a `webkitdirectory` attribute, e.g. - `<input type="file" webkitdirectory ...` this element is expecting a folder. If you try uploading a file, we will put it into a temporary folder and enter that value. However, if you want to upload an actual folder, you can zip the folder and use it as an input. In such case, we will unzip the file, preserving directory structure and enter the root path. This is similar to the drag folder action.

Note: This does NOT work in headless mode. It also does NOT work with Internet Explorer

Template files

You can also specify files containing comma separated values (CSV) and parameterize those with variables (any stored value) in the form ${nowDateIso}.

The file should be in CSV or TXT format and contain the values and variables in the form:

```
some value, ${variable1}, another value
${variable2}, fixed value, ${variable3}
```

example

To allow templating you need to use the word "template" like the following

```
enter template stored value "keyName" into input file "fileField"
enter template "<FILE_URL>" into file input "fileField"
```

example

Phone calls

testRigor supports making a call through Twilio . There is a section in the application configuration under
Integrations for setting Twilio parameters; it is required for making calls.
After the integration is setup you can add custom steps like:

```
call "+15344297154" and validate it was picked up
```

example

```
make call to "+15344297154" and check it was answered
```

example

```
call to +15344297154
```

example

```
call "+15344297154" and check it was completed
```

example

```
call "+15344297154" from "+15551234567" and verify it is ringing
```

example

```
check that phone call from "+15344297154" is ringing
```

example

```
check that phone call from "+15344297154" was answered
```

example

Number to call from is optional and may need to be allocated prior to using. Numbers to call from and to can be from stored values or parameterized strings:

```
call from stored value "allocatedNumber" to stored value "answerPhoneNumber"
```

example

SMS messages (Phone Text)

You can send messages thanks to Twilio integration. There is a section in the application configuration under Integrations for setting Twilio parameters; it is required for sending messages. Then you can add custom steps like:

```
sms "+15344297154" with body "this is content" and validate it was sent
```

example

```
send sms to "+15344297154" with content "this is content"
send message to "+15344297154" with text "this is content" and check it was
delivered
```

example

send message from +15551234567 to "+15344297154" with text "this is content" and check it was failed message from +15551234567 to "+15344297154" with body "this is content" and check it was not delivered
Number from which to send message is optional and may need to be allocated prior to using.
You can also check SMS messages and validate and store their contents:

```
check that sms from "+12345678901" to "+12345678902" contains "Code" and matches
regex "Code\:\d\d\d\d" and save it as "sms"
```

You can then additionally apply `extract value` command to get just the code:

```
check that sms to "+12345678902" matches regex "Code\:\d\d\d\d" and save it as "sms"
extract value by regex "(?<=Code\:)[0-9]{4}" from "sms" and save it as
"confirmationCode"
```

Number from, number to, and body can be taken from stored values or parameterized strings. Additionally, message body may be a multiline string:

```
send sms from stored value "allocatedNumber" to stored value "answerPhoneNumber"
with content from string with parameters starting from next line and ending with
<END>
${answerCode} b
<END>
```

example

You can request a temporary phone number from Twilio . There is a section in the application configuration under
Integrations for setting Twilio parameters; it is required for making calls.
After the integration is setup you can add custom steps like:

```
allocate a temporary number and save it as "newNumber"
```

This custom step will request a new phone number (Twilio charges will apply) and save it in a variable "newNumber", which you can use to check for incoming calls or messages. When the test run finished, this number will be released automatically.
Here is an example of testing a 2FA login with SMS:

```
click "Sign in"
enter "jacob" into "Username"
enter "jacobs-secure-password" into "Password"
click "Verify me"
check that sms to "+12345678902" is delivered and matches regex "Code\:\d\d\d\d" and
save it as "sms"
extract value by regex "(?<=Code\:)[0-9]{4}" from "sms" and save it as
"confirmationCode"
enter saved value "confirmationCode" into "code"
click "Continue to Login"
check that page contains text "Welcome, Jacob!"
```

Login support

testRigor supports login with a single command like so:

```
login
```

example

This action identifies and performs the necessary steps required to login to the application automatically. After running successfully the first time, it creates a rule named Autogenerated Login for the application containing the identified steps, that you can override by creating a reusable rule with the same name.

For example, for an application with a simple email and password login, the steps executed and included in the rule would be:

```
enter stored value "username" into "email"
enter stored value "password" into "password"
click "Log In"
```

Note that login command relies on having login credentials configured for your application on testRigor.

Email testing

testRigor supports testing with both sending emails as well as receiving and validating emails:

```
send email to "user@customer.com" with subject "Test message" and  body "Hi, this is
a test, this is just a test message."
check that email from "user@customer.com" is delivered
```

example

Sending emails also supports sending attachments by referencing a URL of a file you'd like to attach:

```
send email from "sender@customer.com" to "recipient@customer.com" with  subject
"Test message", and  body "Hi, this is a test, this is just a test message.", and
attachment from saved value "Sample File"
```

example

```
send email from "sender@customer.com" to "recipient@customer.com" with  subject
"Test message", and  body "Hi, this is a test, this is just a test message.", and
attachment "http://online.com/file/name.pdf"
```

example

Notes:

1. "to" and "from" can be used at the same time. If not specified, "from" address will default to "noreply@testrigor-mail.com".
2. A stored file defined in "Test Data" can be used as an optional attachment. An attachment can also be a link to a file available online without username/password.
3. Both "from" and "to" addresses can be filled from stored values (e.g., "to saved value "newEmail" and/or "from saved value "newEmail")

Checking email will automatically open and render it as an HTML page in a desktop browser window:

```
check that email to "<random-user>@testrigor-mail.com" from "user@customer.com" is delivered
```

example

```
check that two emails to "user@customer.com" and "Confirm" in subject were delivered
```

example

```
check that one or more emails to "user@customer.com" and "Confirm" in subject were delivered
```

example

```
check that email to saved value "newEmail" was received
```

example

```
check that email to saved value "newEmail" from "admin@customer.com" and "Confirm" in subject was received
```

example

```
check that email to saved value "newEmail" and "<regular expression>" in subject was received
```

If you need to render the message in the browser of the same mobile device where you test your app or website (if tested on a mobile device), e.g. - when validating a sign up test case, or just want to see how it looks on a small screen, you can mention this in the action like so:

```
check that email to "<random-user>@testrigor-mail.com" and "Confirm" in subject was received and show in mobile
```

example

In some cases you may need to check that email was not sent, here is how to do this:

```
check that email to "<random-user>@testrigor-mail.com" from "user@customer.com" was not delivered
```

example

Notes:

1. If the email count is not specified, we assume that only one message is expected. Multiple emails are treated as an error; so in order to avoid receiving the error message, specify the number of emails that should be expected or simply inform the system that more than one can be sent ad done below:

   ```
   check that one or more emails to saved value "<random-user>@testrigor-mail.com" were received
   ```

   example
2. The user should send a message to the following address: `<random-user>@testrigor-mail.com` , where `<random-user>` cab be any valid email handle. In order to avoid a conflict it is recommended that the user handle will contain customer name and some random part, e.g. hfynerifj@testrigor-mail.com. This is important when you are running email tests in multiple browsers and/or on multiple machines at the same time. In this case it is your responsibility to ensure that you are not getting messages, meant for another test case run.
3. The user is expected to add some wait time before checking email. This check action will validate delivery, but not wait for the message to arrive. For example:

   ```
   wait 2 minutes
   ```
4. Once the action has been executed successfully, the message is deleted from the mailbox and rendered in a new tab. Multiple emails will be marked as error, but each one will be rendered in a separate tab. You can switch between tabs using `switch to tab "1"` , `switch to tab "2"` , etc. Note that old content will stay in the tab before the first rendered email message. In most cases it is tab "1", unless you opened more tabs.

5. Each email tab is rendered as an html page, so you can use any actions that you would use on an html page. There are a few useful element ids that will help getting specific parts of an email:
    - "message-froms" a list of "From" addresses (usually one address)
    - "message-subject" subject line
    - "message-date" message sent date and time in Pacific Time zone.
    - "message-tos" a list of "To" addresses
    - "message-ccs" a list of "CC" addresses
    - "message-bccs" a list of "BCC" addresses
    - "message-reply-to" "Reply To" address
    - "message-text" the message text
    - "message-attachments" a list of attachments (file names only)
6. If you want to check again, you need to re-send the mail. Your custom actions cannot contain multiple "check" actions for the same message.
7. Multiple recipients can be separated with "and":

```
send email to saved value "newEmail" and saved value "newEmail2" with subject "Test" and body "Hi"
```

example

8. When checking emails we only take into account those sent after a specific test case has started. This is to avoid a conflict when an email was left unchecked from a previous, unfinished run.

Example of a test for a sign-up flow:

```
click "Sign up"
generate unique email, then enter into "Email" and save as "generatedEmail"
generate unique name, then enter into "Name" and save as "generatedName"
enter "PasswordSuperSecure" into "Password"
click "Submit"
check that email to stored value "generatedEmail" was delivered
click "Confirm email"
check that page contains "Email was confirmed"
check that page contains expression "Hello, ${generatedName}"
```

Browser cookies, localStorage, sessionStorage, userAgent

The user can set/get/clear browser cookies like this:

```
set cookie "cookie value" as "cookie-name"
```

example

```
set cookie from saved value "variableName" as "cookie-name"
```

example

```
get cookie "cookie-name" and save it as "variableName"
```

example

```
clear cookies
```

example

set/get/clear browser localStorage and sessionStorage items like so:

```
set item "item-data" in sessionstorage as "item-name"
```

example

```
get item "item-name" from session storage and save it as "varName"
```

example

```
clear sessionstorage
```

example

```
set item "item-data" in localstorage as "item-name"
```

example

```
get item "item-name" from local storage and save it as "varName"
```

example

```
clear localstorage
```

example

and set/unset a custom userAgent value like this:

```
set user agent to "My User Agent"
```

example

```
unset user agent
```

example

Comments support

testRigor supports one-line comments separated by `//` like so:

```
click "my-cryptic-button" // actually clicks "add to cart"
```

example

Audio testing

You can test audio by recording it and then comparing it with another recording in the same test or an external file. (Audio testing is currently available for Linux/Ubuntu devices only).

Recording

You have the option to record all the audio that is being reproduced in a tab. The command below in an example:

```
record audio through 10 seconds after clicking "audio-trigger" and save as "my-recording"
```

You can also record the audio for a specific audio tag as in the example below:

```
record audio from "my-audio-tag" through 20 seconds after clicking "audio-trigger" and save as "my-specific-recording"
```

Comparing

After having a recording that is saved as a variable, you can compare it against another recording. For example:

```
check that audio from "my-recording" is "70%" similar to "my-specific-recording"
```

You can also compare it against an external file:

```
check that audio from "my-recording" is "85%" similar to "https://some-page.com/path-to-file"
```

The supported file extension for external files is **.wav**
When comparing we can test positive or negative by using **similar** or **different**, for example:

```
check that audio from "my-recording" is "99%" similar to "my-specific-recording"
```

or

```
check that audio from "my-recording" is "1%" different to "my-specific-recording"
```

Playing

If you need to reproduce audio as it was someone speaking through a microphone you could load the .wav that is going to be reproduced from a remote file.
For tests running in linux, virtual devices are used (microphone/speakers) otherwise we attach to existing
< audio > tags
You can reference a remote file to download

```
play audio "https://some-page.com/path-to-file"
```

Or a reference from test data

```
play audio "test-data-ref"
```

If more control about when to play the audio and when to stop it is needed, you can use

```
play audio "sound-to-play-ref" after clicking "start-recording" then click "stop-recording"
```

Validating

If you need to check that audio is produced (e.g., that a video is playing correctly with sound), you can use the following commands:

```
check that audio is playing
```

Negative test is also available:

```
check that audio is not playing
```

In both cases, a 10-second audio sample will be recorded, which will be available in the artifacts for download for manual validation if needed.

Database Query Support

You can connect to certain external databases through a JBDC driver and execute commands directly from your tests.

Managing Multiple Connections

It's possible to configure more than one connection. In this case, you can indicate the connection to be used by name:

```
run sql query "select top 1 UserID, LastName, FirstName from Users;" using connection "connectionName"
run no-sql query "" using connection "anotherConnectionName"
```

Note: If the connection name is not passed, the application will use the first configured.

Overriding the settings

You can update the connection settings at runtime using the following variables: "connectionName:usernameJDBC", "connectionName:passwordJDBC" and "connectionName:connStringJDBC":

```
save value "user" as "connectionName:usernameJDBC".
save value "pass12345" as "connectionName:passwordJDBC
save value "jdbc:mysql:host:port" as "connectionName:connStringJDBC"
run sql query "select top 1 UserID, LastName, FirstName from Users;" using connection "connectionName"
```

SQL Database Support

We have driver support for the most popular databases such as *MySQL*, *PostgreSQL*, *SQL Server*, *Snowflake*, and *Grid Gain*.
You can use SQL commands in the test steps to retrieve, insert, update, and delete rows.

Select

The following example will get the first row and save each pair of column-name/value into a stored value with the column name as the key:

```
run sql query "select top 1 UserID, LastName, FirstName from Users;"
```

example
Then you can use the stored values as follows:

```
enter stored value "FirstName" into "First name"
check that stored value "LastName" itself contains "Doe"
```

example

Insert

Executes the defined insert statement, for example:

```
run sql query "insert into Users(UserID, FirstName, LastName) values (3, 'Jon',
'Doe');"
```

example

No-SQL Database Support

We have driver support for *MongoDB* databases.
You can use MongoDB commands in the test steps to retrieve, insert, update, and delete documents.

Find

The following example will get the first document and save each pair of property-name/value into a stored value with the property name as the key:

```
run no-sql query "{\"find\": \"Users\", \"limit\": 1, \"projection\": {\"userId\":
1, \"firstName\": 1, \"lastName\": 1}}"
```

Then you can use the stored values as follows:

```
enter stored value "FirstName" into "First name"
check that stored value "LastName" itself contains "Doe"
```

Insert

Executes the defined insert command, for example:

```
run no-sql query "{\"collection\": \"Users\", \"documents\": [{\"userId\": 3,
\"firstName\": \"Jon\", \"lastName\": \"Doe\"}]}"
```

Chrome Extensions Testing

You can use your extension to test it or test the integration with your/other pages.
A CRX file of the extension is needed for this feature. Upload it in Settings -> Integrations -> Chrome Extensions, and assign a name to it as it will be needed for its use in the commands to open the extension.
To open the extension in a new tab, use the following command:

```
open extension "myextension"
```

To open the extension in a pop up or to simply enable the extension, you can use this command:

```
activate extension "myextension"
```

Captcha resolution (not included by default)

You can resolve image to text and Google Re-Captcha V2 and V3 directly from your test case steps. Keep in mind this is an extra feature and you will have to contact our sales team to include it in your plan.

Image to text type of captcha

This type of captcha is very common and consists mainly of an image showing a sequence of distorted letters and numbers. To solve it, just indicate in your test the type of captcha as image and point to the element containing the captcha or captcha image. The result of the operation will be saved as a variable called "captchaTest" and you can enter it on the corresponding input.

```
resolve captcha of type image from "captcha element"
```

Then, you can use the extracted text as follows:

```
enter stored value "captchaText" into "Enter captcha text here..."
click "Validate"
```

Google Re-Captcha type of captcha

Google Re-Captcha comes in many flavors and is usually in the page as a custom script that renders (or not depending on the settings) a button to verify if the user is a human. In this case you don't have to specify an element since we're going to detect the captcha on the page and solve it. If the configuration of the captcha has a callback, we will even call it for you. Also, we're going to store the resulting token into a variable called "captchaToken" so you can enter it on the corresponding input or use it inside a javascript snippet to validate it if such callback is not defined.

```
resolve captcha of type recaptcha
```

Then, if a callback was not detected into the configuration of the captcha, you can use the extracted token as follows:

```
enter stored value "captchaToken" into "g-recaptcha-response"
click "Validate"
```

Scan QR Code

testRigor supports scan QR Code, getting value and saving result as a stored value.
You can use the Scan QR Code feature to read a QR Code image saved as variable or to read a QR Code image from screen.

Scan QR Code from stored image

```
scan qr code value from stored value "saved-qr-code" and save as "code"
```

In the command above we are reading the QR Code image stored as "saved-qr-code" and saving the result in the variable "code"
example

Scan QR Code from screen

You can specify any element from screen to scan as QR Code.

```
scan qr code value from "code-container" on the right of "QR Code 2" and save as "code"
```

In the command above we are reading the QR Code image from element "code-container" that is to the right of "QR Code 2" on the screen and saving the result in the "code" variable

example

testRigor advanced topics

testRigor provides you full interface to JavaScript with ability to access the full power of testRigor via interface.

JavaScript Support

testRigor allows you to use JavaScript and refer to testRigor's commands like so:

```
store value "hello" as "var1"
execute JavaScript text starting from next line and ending with [END]
  if (testRigor.hasStoredValue("var1")) {
    testRigor.execute('enter stored value "var1" into "Message"');
  }
  testRigor.execute('click "Update"');
  testRigor.execute('check that page contains text from stored value "var1"');
[END]
```

JavaScript supported is fully ECMAScript 5.1 compatible.

testRigor interface

testRigor interface is as following:

```
interface TestRigor {
  /**
   * True if variable had been assigned a value.
   */
  boolean hasStoredValue(String storedValueName);
  /**
   * Gets the value of the variable.
   */
  String getStoredValue(String storedValueName);
  /**
   * Sets the value of variable.
   */
  String putStoredValue(String storedValueName, String value);
  /**
   * Extracts the full variable-to-value map.
   */
  Map<String, String> getReadOnlyStoredValuesMap();
  /**
   * Gets tree of elements for the current screen.
   */
  UiElement getNodeTree();
  /**
   * Execute the commands. Can be multiple commands new-line separated.
   */
  boolean execute(String commands);
}
```

Screen/Page structure

The interface UiElement is as following:

```
interface UiElement {
  String getElementName();
  String getHtmlNodeName();
  double getY();
  double getX();
  double getHeight();
  double getWidth();
  UiElement getParent();
  String getText();
  String getPath();
  String getShortPath();
  String getFullPath();
  String getName();
  String getLabel();
  String getHint();
  String getTitle();
  String getClassName();
  String getResourceId();
  String getOcrText();
  String getPlaceholder();
  String getAlternative();
  String getType();
  String getValue();
  String getAriaLabel();
  String getRole();
  boolean isVisible();
  boolean isEnabled();
  boolean isClickable();
  boolean isChecked();
  boolean hasProperty(String propertyName);
  String getProperty(String propertyName);
}
```

Above, most of the properties are directly from XML/HTML, except calculated ones:

1. x
2. y
3. height
4. width
5. visible
6. ocrText - only available if OCR is enabled
7. shortPath - short version of XPath leveraging ids where possible
8. fullPath - direct full XPath from the root
9. path - one of the above based on the settings

Note, that id is stored in resourceId attribute

Element interactions

Referencing elements is parsed using ML-based NLP, but can be thought as the following stricture in simplified Pseudo-EBNF:

```
command [index] [typeOfElement] ["containing"] ["saved value" | "parameterized string"] [elementReference] ["in the context of", elementReference]
```

Where:

```
elementReference = \"elementName\", {"or", elementReference} [insideTableReference]
```

Where:

```
insideTableReference = "at" ["row", elementReference] ["column", elementReference]
```

For example:

```
click on the first button "delete" or "remove" in the context of the first table "visible" at the row containing saved value "generatedId" and column "actions"
```

Or you can refer elements directly through:

```
click XPath "/html/body/div[3]"
```

Or:

```
click CSS Selector "a[target=_blank]"
```

Automate testing tables

Selecting a row in a table

testRigor has a very powerful way of dealing with whatever looks like a table for a user.
You can refer to elements of the table by specifying row and column. And to specify a row there is a powerful SQL-like language is available. For example, you can say things like:

```
click "Delete" inside of table at row containing "my unique id" and column "Actions"
```

Here we didn't specify the column where the "my unique id" should reside, but rather indicated that it could be in any column at that row.

You can also construct SQL-like expressions containing one or multiple columns with different validations. For example:

```
check that table with "Country" equal to "United States" and "City" equal to "Saint Petersburg" and "State" equal to "Florida" at column "Status" contains "Flourishing"
```

Here we used 3 different conditions to identify the row connected with the boolean operator AND. The following boolean operators are supported:

- AND
- OR
- NOT
- Brackets ()

and you can also use brackets. The condition calculation will follow the standard boolean logic. The condition above could also be expressed like so:

```
check that table with not("Country" not equal to "United States" or "City" not equal to "Saint Petersburg" or "State" not equal to "Florida") at column "Status" contains "Flourishing"
```

You can learn more about how boolean operators work here.

The exciting thing about testRigor is that these instructions would work regardless of how your table is rendered. It could be rendered as an HTML TABLE tag with TRs and TDs today and completely div-based tomorrow. It doesn't matter, your code will continue to function anyway.

For an example, let's consider the following table:

#	Id	Name	Ship	Status	Actions
1	york1	Yorktown	Enterprise	Alright	Arrive Cancel
2	spk2	Spock	Enterprise	Alright	Listen to Ignore
3	nyo3	Nyota	Enterprise	Alright	Open channel Promote
4	spk2	Spock	Kelvin	Alright	Listen to Ignore
5	nyo3	Nyota	Kelvin	Alright	Open channel Promote

Actions

In the example above, rows have no unique meaningful identifiers. We only have row numbers which might be generated automatically to show the index of a row. We can't rely on them consistently because they will change the next time the page refreshes. How are we going to on a button with Spock and Kelvin? Here is how:

```
click on the first button in the context of table with "Name" equal to "Spock" and
"Ship" equals to "Kelvin" at column "Actions"
```

Calculating aggregates

You can calculate and work with aggregate values on columns or specify a condition that would validate against all values in a column. Here are some examples:

Order of values

To check the order of the values contained in a row or column

```
check that table "actions" at column "Name" has values sorted in ascending order
check that table "actions" at column "Name" has values sorted in descending order
```

Number of values (count)

To check the value count in a row or column

```
check that table "actions" at column "Name" the value count is greather than "3"
check that table "actions" at third column the value count is less than "20"
check that table "actions" at column "Name" the value count is equals to "5"
check that table "actions" at third column the value count is equals to stored value
"expectedValueCount"
```

Sum of values

To check the sum of values in a row or column

```
check that table "actions" at column "#" the sum of values is greater than "10"
check that table "actions" at first column the sum of values is less than "50"
check that table "actions" at column "#" the sum of values is equals to "15"
check that table "actions" at first column the sum of values is less than stored
value "expectedValueSummatory"
```

Average of values

To check the average of values in a row or column

```
check that table "actions" at column "#" the average of values is greater than "1"
check that table "actions" at first column the average of values is less than "6"
check that table "actions" at column "#" the average of values is equals to "3"
check that table "actions" at first column the average of values is equals to stored
value "expectedValueAvg"
```

JsonPath Overview

testRigor uses JsonPath as a way to help you to refer to elements of returned JSON. JsonPath expressions always refer to a JSON structure in the same way as XPath expression are used in combination with an XML document. The "root member object" in JsonPath is always referred to as `$` regardless if it is an object or array. JsonPath expressions can use the dot-notation

```
$.store.book[0].title
```

or the bracket-notation

```
$['store']['book'][0]['title']
```

Operators

Operator	Description
`$`	The root element to query. This starts all path expressions.
`@`	The current node being processed by a filter predicate.
`*`	Wildcard. Available anywhere a name or numeric are required.
`..`	Deep scan. Available anywhere a name is required.
`.<name>`	Dot-notated child
`['<name>' (, '<name>')]`	Bracket-notated child or children
`[<number> (, <number>)]`	Array index or indexes
`[start:end]`	Array slice operator
`[?(<expression>)]`	Filter expression. Expression must evaluate to a boolean value.

Functions

Functions can be invoked at the tail end of a path - the input to a function is the output of the path expression. The function output is dictated by the function itself.

Function	Description	Output
`min()`	Provides the min value of an array of numbers	Double
`max()`	Provides the max value of an array of numbers	Double
`avg()`	Provides the average value of an array of numbers	Double
`stddev()`	Provides the standard deviation value of an array of numbers	Double
`length()`	Provides the length of an array	Integer
`sum()`	Provides the sum value of an array of numbers	Double

Filter Operators

Filters are logical expressions used to filter arrays. A typical filter would be `[?(@.age > 18)]` where `@` represents the current item being processed. More complex filters can be created with logical operators `&&` and `||`. String literals must be enclosed by single or double quotes (`[?(@.color == 'blue')]` or `[?(@.color == "blue")]`).

Operator	Description
`==`	left is equal to right (note that 1 is not equal to '1')
`!=`	left is not equal to right
`<`	left is less than right
`<=`	left is less or equal to right
`>`	left is greater than right

Operator	Description
`>=`	left is greater than or equal to right
`=~`	left matches regular expression `[?(@.name =~ /foo.*?/i)]`
`in`	left exists in right `[?(@.size in ['S', 'M'])]`
`nin`	left does not exists in right
`subsetof`	left is a subset of right `[?(@.sizes subsetof ['S', 'M', 'L'])]`
`anyof`	left has an intersection with right `[?(@.sizes anyof ['M', 'L'])]`
`noneof`	left has no intersection with right `[?(@.sizes noneof ['M', 'L'])]`
`size`	size of left (array or string) should match right
`empty`	left (array or string) should be empty

Path Examples

Given the JSON


```
{
  "store": {
      "book": [
          {
              "category": "reference",
              "author": "Nigel Rees",
              "title": "Sayings of the Century",
              "price": 8.95
          },
          {
              "category": "fiction",
              "author": "Evelyn Waugh",
              "title": "Sword of Honour",
              "price": 12.99
          },
          {
              "category": "fiction",
              "author": "Herman Melville",
              "title": "Moby Dick",
              "isbn": "0-553-21311-3",
              "price": 8.99
          },
          {
              "category": "fiction",
              "author": "J. R. R. Tolkien",
              "title": "The Lord of the Rings",
              "isbn": "0-395-19395-8",
              "price": 22.99
          }
      ],
      "bicycle": {
          "color": "red",
          "price": 19.95
      }
  },
  "expensive": 10
}
```


The following results will be returned:

JsonPath	Result
`$.store.book[*].author`	The authors of all books

JsonPath	Result
`$..author`	All authors
`$.store.*`	All things, both books and bicycles
`$.store..price`	The price of everything
`$..book[2]`	The third book
`$..book[-2]`	The second to last book
`$..book[0,1]`	The first two books
`$..book[:2]`	All books from index 0 (inclusive) until index 2 (exclusive)
`$..book[1:2]`	All books from index 1 (inclusive) until index 2 (exclusive)
`$..book[-2:]`	Last two books
`$..book[2:]`	Book number two from tail
`$..book[?(@.isbn)]`	All books with an ISBN number
`$.store.book[?(@.price < 10)]`	All books in store cheaper than 10
`$..book[?(@.price <= $['expensive'])]`	All books in store that are not "expensive"
`$..book[?(@.author =~ /.*REES/i)]`	All books matching regex (ignore case)
`$..*`	Give me every thing
`$..book.length()`	The number of books

Regex Random String Generation Support

In addition to a simple way to generate random data, testRigor also supports a way to use Regular Expressions (Regex) for more complex scenarios. For instance, you can generate a random tag like this:

```
generate by regex "<(a|button)>data<\/\1>", then enter into "Tag data" and save as "generatedTag"
```

which generates `<a>data</a>` or `<button>data</button>`. Or you might want

```
generate by regex "https://(www\.)?([a-z][a-z0-9-]{2,30}[a-z0-9]\.){1,5}(com|edu|co\.uk|info|io)", then enter into "URL" and save as "generatedUrl"
```

which generates a unique URL in the format of "https://www.abc.de.com".

The rule of thumb to use ReGex is to avoid it as much as possible since it is very hard to read for anyone who is not familiar with it. Always try simple template first. One of the use cases where you

might be forced to use ReGex is if what you generate must have a variable length.

testRigor supports both generation of random strings based on Regex as well as validating and extracting data.

Regex Random Data Generation Cookbook

For simple scenarios like generating emails, phone numbers, credit card numbers, etc. please see simple template. The full command will look like this:

```
generate by regex "<YOUR_REGEX>", then enter into "Email" and save as "newEmail"
```

Regex	Use case
`[A-Z][a-z]{2,50}`	Random word starting from upper case like "Hello"
`[a-z]{2,50}`	Random word all lower case like "hi"
`[1-2][0-9]{0,8}`	Random number
`((1[0-2])|0[1-9])-((0[1-9])|([1-2][0-8]))-((20[0-2][0-9])|(19[0-9]{2}))`	Dates in `mm-dd-yyyy` format
`((20[0-2][0-9])|(19[0-9]{2}))-((1[0-2])|0[1-9])-((0[1-9])|([1-2][0-8]))`	Dates in `yyyy-mm-dd` format
`((1[0-2])|(0[0-9])):([0-5][0-9]) [AP]M`	Time in `11:34 AM` AM/PM format
`((2[0-3])|([0-1][0-9])):([0-5][0-9])`	Time in `14:25` 24hr format
`(www\.)?([a-z][a-z0-9-]{2,30}[a-z0-9]\.){1,5}[a-z]{2,3}`	Domain name
`https://(www\.)?([a-z][a-z0-9-]{2,30}[a-z0-9]\.){1,5}[a-z]{2,3}`	URL without path
`https?://(www\.)?([a-z][a-z0-9-]{2,30}[a-z0-9]\.){1,5}[a-z]{2,3}(/[a-z0-9-]{2,10}){0,5}/?(#[a-z0-9])?`	Full random URL
`((25[0-5]|2[0-4]\d|[01]?\d\d?)\.){3}(25[0-5]|2[0-4]\d|[01]?\d\d?)`	IP v4
`((\+[0-9][0-9]? [0-9]{3})|(\(0[0-9]{3}\))) [0-9]{3} [0-9]{4}`	International phone numbers in a standard E.123 format

Regex Random Generation Syntax Support

testRigor provides full support for Java-like regular expressions with POSIX character classes and full multibyte Unicode support for international characters. A simple example is:

Press **Ctrl+F** to search

LANGUAGE DOCUMENTATION

- Getting Started guide
- Basic Commands
- Reusable Rules (Subroutines)
- Referencing locations
- Using variables
- Loops
- Validations
- Visual Testing
- Working with Tables
- Conditional execution
- API Testing
- Uploading files
- Phone calls
- SMS messages (Phone Text)
- Login support
- Email testing
- Browser cookies, localStorage, sessionStorage, userAgent
- Comments
- Audio Testing
- Database Query
- Chrome Extension Testing
- Captcha resolution
- Scan QR Code

ADVANCED

AUTOMATE TESTING TABLES

JSONPATH OVERVIEW

REGEX SUPPORT

SIMPLE TEMPLATES

WHAT IS BOOLEAN LOGIC

```
generate by regex "[A-Z][a-z]{2,50}", then enter into "Name" and save as "generatedName"
```

Construct	Description	Example
`A text`	All regular text is copied as is. This is not changeable part. In the example on the right Lakshmi will be posted as is.	`generate by regex "Lakshmi", then enter into "Name" and save as "name"`
`|`	Select one option from another. testRigor will randomly choose from the options available. In the example on the right, it will use either Lakshmi or Meera	`generate by regex "Lakshmi|Meera", then enter into "Name" and save as "name"`
`()`	Grouping parts. In case if something needs to be executed on the whole rater than the last symbol. `Lakshmi+` will result in "Lakshmiiiiii", where as `(Lakshmi )+` will result in "Lakshmi Laskhmi Lakshmi ..."	`generate by regex "(Lakshmi )+", then enter into "Name" and save as "name"`
`?`	Add or do not add the part randomly. The example would result in "Lakshmi" about 50% or the time and empty string other 50%	`generate by regex "(Lakshmi)?", then enter into "Name" and save as "name"`
`+`	Repeat the part at least one and at most 100 times. The example on the right will have at least one Lakshmi but can also have up to 99 more.	`generate by regex "(Lakshmi )+", then enter into "Name" and save as "name"`
`*`	Repeat the part at least zero and at most 100 times. The example on the right could produce an empty string or up to 100 "Lakshmi" strings.	`generate by regex "(Lakshmi )*", then enter into "Name" and save as "name"`
`{N}`	Repeat the part precisely N times. In example on the right "Lakshmi " will be repeated twice like this: "Lakshmi Lakshmi ".	`generate by regex "(Lakshmi ){2}", then enter into "Name" and save as "name"`
`{N,M}`	Repeat the part between N and M times. In example on the right "Lakshmi " can be repeated either once or twice.	`generate by regex "(Lakshmi ){1,2}", then enter into "Name" and save as "name"`
`[abc]`	Select just one of the characters specified. In the example only a, b, or c will be selected randomly.	`generate by regex "[abc]", then enter into "Name" and save as "name"`
`[a-c]`	Select just one of the characters in the range specified. In the example only a, b, or c will be selected randomly.	`generate by regex "[a-c]", then enter into "Name" and save as "name"`

Construct	Description	Example
`[^abc]`	If the first symbol in square brackets is `^`, then testRigor will use visible symbols except the ones provided. In the example, it could be any character like 'A', '7', 'd', '&', etc. This reversal applies universally and can be used with ranges as well, like `[^a-c]`, which in this case, will have the same effect.	`generate by regex "[^abc]", then enter into "Name" and save as "name"`
`\s`	One character of any whitespace like space or tab.	`generate by regex "\s", then enter into "Name" and save as "name"`
`\w`	One character of lower and upper case Latin letters, digits and '_'. Equivalent to `[a-zA-Z0-9_]`.	`generate by regex "\w", then enter into "Name" and save as "name"`
`\d`	One digit character. Equivalent to `[0-9]`.	`generate by regex "\d", then enter into "Name" and save as "name"`
`.`	Any character. testRigor will generate characters that can be typed on a US keyboard randomly.	`generate by regex "\d", then enter into "Name" and save as "name"`
`\`	Escapes the following special character. For example, if you'd like to include '[' and ']' into the set of characters you use, you could specify `[\[\]]`	`generate by regex "[\[\]]", then enter into "Name" and save as "name"`
`\n`	Just one new line character.	`generate by regex "\n", then enter into "Name" and save as "name"`
`\t`	Just one tab character.	`generate by regex "\t", then enter into "Name" and save as "name"`
`\N`	A reference to the previously generated Nth group. For example, `<(a|button)>data<\/\1>` will generate the opening and closing of the same tag ("a" or "button" in this case).	`generate by regex "<(a|button)>data<\/\1>", then enter into "Code" and save as "code"`

Advanced Topics

When we were creating the Regex generator we wanted the syntax to be as close as it is technically possible to Java version of Regex (Pattern). You might find a lot of additional quirks and features to be similar to Java's version. However, the parser was built from scratch and had to be generation-compatible, so there are some differences. For instance, we included support for almost all POSIX character classes. However, we emulated how Java works with backreferences and group numbering. Also, we tried to build parser in a way that would produce the most readable and understandable error messages we could do.

Question	Answer
How does testRigor generates random?	testRigor generates random numbers with pseudo-random sequence seeded with nanoseconds.
Is there a bound for unbounded operations like `*` ?	Yes, default bound is 100
What is I need more than 100 symbols generated?	You can use range, and specify any number as upper bound. testRigor will repeat up to your specified number or 10,000, whichever is smaller.
Does testRigor support intersections like `[a&&[b]]` ?	Yes for validation/search. Not yet for random data generation.
Does testRigor support multicharacter Unicode symbols?	Yes, naturally out of the box. You can specify `😀{3}` to get "😀😀😀"
Does testRigor support non-capturing groups?	Yes for validation/search. For generation positive non-capturing groups like `(?<=group)` will generate data, but negative groups like `(?!group)` will be ignored.
Does testRigor support flags like `(?i)` ?	Yes for validation/search. Not yet for random data generation.

More simple use cases like phone numbers, passport numbers, SSNs, emails, etc. see simple template.

Regex for validations and search

You can use Regex for search/validation in testRigor. testRigor will use Java 11's Regex Pattern to complile/process the Regex in this case. See the full supported syntax here. Examples of usages are:

```
check that page has regex "Current time: [0-2][0-9]:[0-5][0-9]"
check that page doesn't have regex "fail|crash|500"
check that URL matches regex "https://mycompany\.com/list/item/[a-z0-9]{2,30}"
check that text below "Current time" matches regex "[0-2][0-9]:[0-5][0-9]"
grab value of regex "[0-2][0-9]:[0-5][0-9]" and save it as "currentTime"
```

How to use simple templates to generate unique data

testRigor provides as a way to help you to easily generate unique data. For instance, you can generate unique data like phone numbers like this:

```
generate from template "###-555-####", then enter into "Phone" and save as
"generatedPhone"
```

Which would generate a unique phone number like `752-555-0912` . To do that testRigor uses 4 different symbols to indicate types of data you want to generate: `#$%*` .

Unique data generating symbols

Symbol	Description
#	Generates a random number in range 0-9.
$	Generates a random lower case letter in range a-z.
%	Generates a random upper case letter in range A-Z.
*	Generates a random alphanumeric symbol, either number, or lower or upper case letter (0-9, a-z, A-Z).
\	If the following symbol is one of the above or another \ then this \ symbol is removed and the following symbol is returned verbatim instead of generating a random symbol.

Examples

Keep in mind, that testRigor has already a built-in words to generate unique email or unique name. You do not need to provide a template for them. Those would work like so:

```
generate unique email, then enter into "Email" and save as "newEmail" generate unique name, then enter into "Name" and save as "generatedName"
```

Example	Description
`generate unique email, then enter into "Email" and save as "newEmail"`	Generates a random email in testrigor-mail.com domain.
`generate unique name, then enter into "Name" and save as "generatedName"`	Generates a random name.
`generate from template "$**********@testrigor-mail.com", then enter into "Email" and save as "newEmail"`	Generates a random email in a custom domain.
`generate from template "###-###-####", then enter into "Phone" and save as "generatedPhone"`	Generates a random phone number.
`generate from template "811-###-####", then enter into "Phone" and save as "generatedPhone"`	Generates a random phone number in a 811 area code.
`generate from template "000-##-####", then enter into "SSN" and save as "generatedSsn"`	Generates a random SSN number.
`generate from template "4###-####-####-####", then enter into "card" and save as "generatedCard"`	Generates a random VISA credit card.
`generate from template "https://$$$$$$$$$$$$$$$$$$$$$$$$$$$$$.info", then enter into "url" and save as "generatedUrl"`	Generates a random URL.

Press **Ctrl+F** to search

LANGUAGE DOCUMENTATION

Getting Started guide

Basic Commands

Reusable Rules (Subroutines)

Referencing locations

Using variables

Loops

Validations

Visual Testing

Working with Tables

Conditional execution

API Testing

Uploading files

Phone calls

SMS messages (Phone Text)

Login support

Email testing

Browser cookies, localStorage, sessionStorage, userAgent

Comments

Audio Testing

Database Query

Chrome Extension Testing

Captcha resolution

Scan QR Code

ADVANCED

AUTOMATE TESTING TABLES

JSONPATH OVERVIEW

REGEX SUPPORT

SIMPLE TEMPLATES

WHAT IS BOOLEAN LOGIC

Example	Description
`generate from template "%******************************", then enter into "data" and save as "generatedData"`	Generates a random alphanumeric data.
`generate from template by string with escaped parameters "${nowDateTimeIso}-**********", then enter into "Data" and save as "generatedData"`	Generates a random alphanumeric data starting with date-time in ISO format.
`generate from template "%$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$ $$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$.", then enter into "description" and save as "generatedDescription"`	Generates a random text description.

More complex use cases in some cases might require ReGex and can be found here.

What is Boolean Logic

Boolean Logic in the context of testRigor is a way to combine boolean conditions like `"Status" contains "Success"` . You can use three main operators to join these conditions:

- AND
- OR
- NOT
- Brackets ()

Here are some examples:

AND operator

AND operator helps to join conditions which both must be true for the result to be true. For example, `"Status" contains "Success" and "City" is equal to "Paris"` will only be true when the selected row contains "Success" in column "Status" and has string "Paris" in column "City" at the same time.

OR operator

OR operator helps to find a row containing one or both conditions. For example, `"Status" contains "Success" or "City" is equal to "Paris"` will only be true when the selected row either contains "Success" in column "Status" or has string "Paris" in column "City" or both.

NOT operator

NOT operator helps calculate the negative of the condition. For example, `not "Status" contains "Success"` will only be true when the selected row does not contain "Success" in column "Status".

Brackets ()

Brackets () help to group conditions together. For example, `("Status" contains "Success" or "Status" contains "In progress") and "City" is equal to "Paris"` will only be true when the selected row contains "Success" or "In progress" in column "Status" and has string "Paris" in column "City" at the same time. In this example "Status" column must contain either "Success" or "In progress", and column "City" must contains "Paris".