

Testing using Robot Framework integration in Python or Java

- [Overview](#)
- [Common requirements](#)
- [Examples](#)
 - [The full ATDD workflow](#)
 - [Attaching screenshots](#)
 - [Running tests in parallel, against different environments](#)
- [Tracking automation results](#)
 - [On the user story issue screen](#)
 - [On the Test Plan](#)
- [References](#)

Overview

Robot Framework is a tool [used by teams adopting ATDD \(Acceptance Test Driven Development\)](#).

Broadly speaking, it can be used to automate acceptance “test cases” (i.e. scripts) no matter the moment you decide to do so or the practices your team follows even though it's preferable to do it at start, involving the whole team in order to pursue shared understanding.

In this article, we will specify some tests using [Robot Framework](#) and see how we can have visibility of the corresponding results in Jira, using Xray.

This tutorial explores the [specific integration Xray provides for Robot Framework XML reports](#).

Common requirements

- Robot Framework
- SeleniumLibrary
- Java (if using the Java variant of the "Robot Framework")

Examples

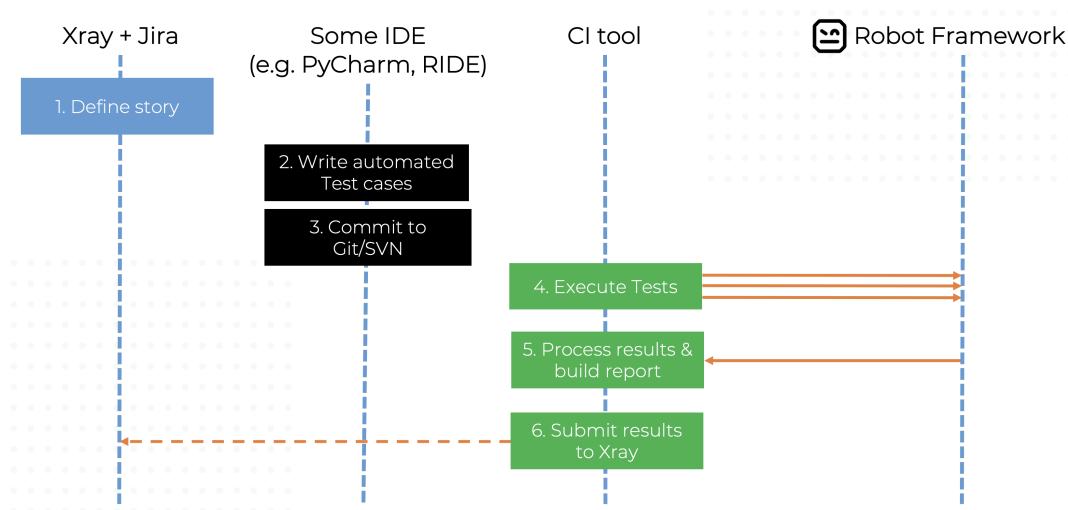
The full ATDD workflow

In this example we're going to validate a dummy website (provided in the [GitHub repository](#)), checking for valid and invalid logins.



You may find the full source for this example in this [GitHub repository](#), which corresponds in essence to previous work by Pekka Klärck from the Robot Framework Foundation.

If the team is adopting ATDD and working collaboratively in order to have a shared understanding of what is going to be developed, why and some concrete examples of usage, then the flow would be something similar to the following diagram.



All starts with a user story or some sort of “requirement” that you wish to validate. This is materialized as a Jira issue and identified by the corresponding issue key (e.g. ROB-11).

Projects / Robot Framework / ROB-11

As a user, I can login the web application

Attach Create subtask Link issue Test Coverage

Description

As a user, I can login the web application

Test Coverage

...

Create new Sub Test Execution

Create new Test

No Tests are associated with this issue.

UNCOVERED

We can promptly check that it is “UNCOVERED” (i.e. that it has no tests covering it, no matter their type/approach).

A Test Plan can be created to define the scope of the testing that we aim to perform, group, and consolidate the corresponding results. Besides the user story, we may also add the Test Plan to the Board and assign it explicitly to a sprint. This will increase visibility of testing progress and help closing the gap between dev<>testers.

Test Plan for automated UI tests (RF)


 Attach

 Create subtask

 Link issue



 Tests

 Test Executions



Description

Add a description...

Tests

 View on Board

...

 Add



This Test Plan is not associated with Tests yet.

Test Executions

Add Test Executions

...

This Test Plan is not associated with Test Executions yet.

A tester/SDET could simply focus on implementing the automated test cases:

- The tester would write one or more test suites and corresponding test cases, using his/her favorite tool/IDE
- Each test case could be linked to the corresponding requirement/user story in Jira by adding its key as a tag
- Tests could then be run locally, or from the CI pipeline
- Unique, non-duplicating, Test entities would be auto-provisioned in Xray, corresponding to each test case; tester could also, optionally, enforce the result to an existing Test entity by specifying its issue key as a tag

Let's take the following .robot file as an example, which acts as a suite containing one test case.

login_tests/valid_login.robot

```
*** Settings ***
Documentation      A test suite with a single test for valid login.
...
...               This test has a workflow that is created using keywords in
...               the imported resource file.
Resource           resource.robot

*** Test Cases ***
Valid Login
    [Tags]          ROB-11  UI
    Open Browser To Login Page
    Input Username   demo
    Input Password   mode
    Submit Credentials
    Welcome Page Should Be Open
    [Teardown]      Close Browser
```

The previous Robot file uses a common resource that contains some generic variables and some reusable "keywords" (i.e., steps).

login_tests/resource.robot

```
*** Settings ***
Documentation      A resource file with reusable keywords and variables.
...
...               The system specific keywords created here form our own
...               domain specific language. They utilize keywords provided
...               by the imported SeleniumLibrary.
Library           SeleniumLibrary          run_on_failure=Capture Page Screenshot
screenshot_root_directory=EMBED

*** Variables ***
${SERVER}         192.168.56.1:7272
${BROWSER}        Firefox
${DELAY}          0
${VALID USER}     demo
${VALID PASSWORD} mode
${LOGIN URL}       http://${SERVER}/
${WELCOME URL}     http://${SERVER}/welcome.html
${ERROR URL}       http://${SERVER}/error.html

*** Keywords ***
Open Browser To Login Page
    Open Browser    ${LOGIN URL}    ${BROWSER}
    Maximize Browser Window
    Set Selenium Speed    ${DELAY}
    Login Page Should Be Open

Login Page Should Be Open
    Title Should Be    Login Page

Go To Login Page
    Go To    ${LOGIN URL}
    Login Page Should Be Open

Input Username
    [Arguments]    ${username}
    Input Text    username_field    ${username}

Input Password
    [Arguments]    ${password}
    Input Text    password_field    ${password}

Submit Credentials
    Click Button    login_button

Welcome Page Should Be Open
    Location Should Be    ${WELCOME URL}
    Title Should Be    Welcome Page
```

Running the tests can be done from the command line or from within Jenkins (or any other CI tool); this will produce a XML based report (e.g. [output.xml](#)).

Build

Execute shell

Command

robot --variable BROWSER:\${BROWSER} --variable SERVER:\${SERVER} login_tests

See [the list of available environment variables](#)

Advanced...

Importing results is as easy as submitting them to the [REST API](#) with a POST request (e.g. curl), or by using one of the CI plugins available for free (e.g. [Xray Jenkins plugin](#)).

Post-build Actions

Xray: Results Import Task

JIRA Instance

xray-vm

Format

Robot XML

Parameters

Import to Same Test Execution☒

When this option is check, if you are importing multiple execution report files using a glob expression, the results will be imported to the same Test Execution

Execution Report File (file path with file name)

output.xml

Project Key

ROB

Test Execution Key

Test Plan Key

ROB-12

Test Environments

Revision

\${BUILD_NUMBER}

Fix Version

Examples of running tests from the command line

Running tests is primarily done using the "robot" utility which provides many options that allow you to define which tests to run, the output directory and [more](#).

You may also specify some variables and their values.

Next follows some different usage examples.

If you're using Python:

```
robot -d output --variable BROWSER:Firefox login_tests
```

If you're using Java:

```
java -jar robotframework-3.0.jar login_tests
```

An unstructured (i.e. "Generic") Test issue will be auto-provisioned the first time you import the results, based on the name of the test case and of the corresponding test suites.

If you maintain the test case name and the respective test suites, the Test will be reused on subsequent result imports. You may always enforce the results to be reported against an existing Test, if you wish so: just specify its issue key as a tag.

Tags can also be used to cover an existing requirement/user story (e.g. "ROB-11"): when a requirement issue key is given, a link between the test and the requirement is created during the results import process.

Otherwise, tags are mapped as labels on the corresponding Test issue.

Valid Login



Description

Add a description...

Test Details

Generic

Definition

Login Tests.Valid Login.Valid Login

To Do

Assignee

Unassigned

Reporter

Sérgio Freire

Labels

UI

Priority

Medium

Test Repository



Please note

Note that Robot Framework considers the base folder of the project as the first test suite. The way you run your tests also affects Robot's XML; so, if you execute the file from somewhere else or you execute the file directly by passing it as an argument, the test suite's information will potentially be different.

A Test Execution will be created containing results for all test cases executed. In this case, you can see that it is also linked back to an existing Test Plan where you can track the consolidated results from multiple "iterations" (i.e. Test Executions).

Execution results [1594735455364]



Description

Add a description...

Tests

Create Test

+ Add

Overall Execution Status

TOTAL TESTS: 8

8 PASSED

Rank	Key	Summary	Test Type	Status	Actions
<input type="checkbox"/>	1	ROB-14 Valid Login	Generic	PASSED	...
<input type="checkbox"/>	2	ROB-15 Invalid Username	Generic	PASSED	...
<input type="checkbox"/>	3	ROB-16 Invalid Password	Generic	PASSED	...
<input type="checkbox"/>	4	ROB-17 Invalid Username And Password	Generic	PASSED	...
<input type="checkbox"/>	5	ROB-18 Empty Username	Generic	PASSED	...
<input type="checkbox"/>	6	ROB-19 Empty Password	Generic	PASSED	...
<input type="checkbox"/>	7	ROB-20 Empty Username And Password	Generic	PASSED	...
<input type="checkbox"/>	8	ROB-21 Valid Login	Generic	PASSED	...

Test Plans

Associated Test Plans

ROB-12

Within the execution screen details, accessible from each row, you can look at the Test Run details which include the overall result and also specifics about each keyword, including duration and status.

Execution Status

PASSED

Timer

▶ 00:00:00 ◀

No time logged

Started On

19/Sep/2024 10:53 AM

Assignee

Cristiano Cunha

Versions

-

Test Version

v1

Finished On

19/Sep/2024 10:53 AM

Executed By

Cristiano Cunha

Revision

-

Test Environments

-

> Findings +

Test details

GENERIC

Results 15

Context	Output	Duration	Status
KW: Given browser is opened to login page	-	4ms	PASSED
KW: Open Browser To Login Page	-	4ms	PASSED
KW: Open Browser (\${LOGIN_URL}, \${BROWSER})	Opening browser 'Firefox' to base url 'http://127.0.0.1:7272/'	4ms	PASSED
KW: Maximize Browser Window	-	0ms	PASSED
KW: Set Selenium Speed (\${DELAY})	-	0ms	PASSED
KW: Login Page Should Be Open	-	0ms	PASSED
KW: Title Should Be (Login Page)	Page title is 'Login Page'.	0ms	PASSED
KW: When user "demo" logs in with password "mode"	-	0ms	PASSED
KW: Input Username (\${username})	-	0ms	PASSED
KW: Input Text (username_field, \${username})	Typing text 'demo' into text field 'username_field'.	0ms	PASSED
KW: Input Password (\${password})	-	0ms	PASSED
KW: Input Text (password_field, \${password})	Typing text 'mode' into text field 'password_field'.	0ms	PASSED
KW: Submit Credentials	-	0ms	PASSED
KW: Click Button (login_button)	Clicking button 'login_button'.	0ms	PASSED
KW: Then welcome page should be open	-	0ms	PASSED
KW: Location Should Be (\${WELCOME_URL})	Current location is 'http://127.0.0.1:7272/welcome.html'.	0ms	PASSED
KW: Title Should Be (Welcome Page)	Page title is 'Welcome Page'.	0ms	PASSED
KW: Close Browser	-	0ms	PASSED

Attaching screenshots

Attaching screenshots at the step level is possible by using the [SeleniumLibrary](#) RF library. A [configuration](#) must be provided to embed the screenshots on the output.xml report; it can also be configured to take screenshots automatically on failed steps.

Example of including and initializing the SeleniumLibrary:

```
Library SeleniumLibrary run_on_failure=Capture Page Screenshot screenshot_root_directory=EMBED
```

In the [GitHub repository](#), there's a [buggy web server](#) implementation. If tests are run against it, two of them will fail (i.e., the ones related with valid login).

Execution results [1650275017336]

[Attach](#) [Create subtask](#) [Link issue](#) [Tests](#) [...](#)

Description

Add a description...

Tests

[Add Tests](#)

Overall Execution Status

6 PASSED **2** FAILED

TOTAL TESTS: 8

Rank	Key	Summary	Test Type	TestRun Assignee	Status	Actions
1	ROB-14	Valid Login	Generic	Sérgio Freire	FAILED	...
2	CALC-1264	Invalid Username	Generic	Sérgio Freire	PASSED	...
3	CALC-1265	Invalid Password	Generic	Sérgio Freire	PASSED	...
4	CALC-1266	Invalid Username And Password	Generic	Sérgio Freire	PASSED	...
5	CALC-1267	Empty Username	Generic	Sérgio Freire	PASSED	...
6	CALC-1268	Empty Password	Generic	Sérgio Freire	PASSED	...
7	CALC-1269	Empty Username And Password	Generic	Sérgio Freire	PASSED	...
8	CALC-1270	Valid Login	Generic	Sérgio Freire	FAILED	...

Prev 1 Next

Total 8 issues

After importing the generated test report, we can see the screenshot in the Test Run details, in this case on the failed step.

Valid Login

[Execute with Exploratory App](#)[Dataset](#)[Import Execution Results](#)

Execution Status

Timer

Started On

Assignee

Versions

Test Version

FAILED

00:00:00

19/Sep/2024 09:59 AM

Cristiano Cunha

-

v1

No time logged

19/Sep/2024 09:59 AM

Cristiano Cunha

-

-

Findings

COMMENT

+

Test details

GENERIC

Results 18

Context	Output	Duration	Status
KW: Given browser is opened to login page	-	9ms	PASSED
KW: Open Browser To Login Page	-	9ms	PASSED
KW: Open Browser (\${LOGIN URL}, \${BROWSER})	Opening browser 'Firefox' to base url 'http://127.0.0.1:7272'.	9ms	PASSED
KW: Maximize Browser Window	-	0ms	PASSED
KW: Set Selenium Speed (\${DELAY})	-	0ms	PASSED
KW: Login Page Should Be Open	-	0ms	PASSED
KW: Title Should Be (Login Page)	Page title is 'Login Page'.	0ms	PASSED
KW: When user "demo" logs in with password "mode"	-	0ms	PASSED
KW: Input Username (\${username})	-	0ms	PASSED
KW: Input Text (username_field, \${username})	Typing text 'demo' into text field 'username_field'.	0ms	PASSED
KW: Input Password (\${password})	-	0ms	PASSED
KW: Input Text (password_field, \${password})	Typing text 'mode' into text field 'password_field'.	0ms	PASSED
KW: Submit Credentials	-	0ms	PASSED
KW: Click Button (login_button)	Clicking button 'login_button'.	0ms	PASSED
KW: Then welcome page should be open	-	0ms	FAILED
KW: Location Should Be (\${WELCOME URL})	Location should have been 'http://127.0.0.1:7272/welcome.html' but was 'http://127.0.0.1:7272/error.html'.	0ms	FAILED

Running tests in parallel, against different environments

In this distinct and more evolved example we're going to run tests in parallel using "pabot"; we'll also take advantage of the [Test Environments](#) concept provided by Xray.

This example uses a [fake travel agency site](#) (kindly provided by BlazeMeter) as the testing target.

Welcome to the Simple Travel Agency!

This is a sample site you can test with BlazeMeter!

Check out our [destination of the week! The Beach!](#)

Choose your departure city:

Choose your destination city:

Find Flights

We have two tests that use low-level keywords (note: this is not a good practice; it's just for simplicity) and one of those keywords is defined within a SeleniumLibrary plugin (i.e. it extends the keywords provided by SeleniumLibrary).

search_flights.robot

```
*** Settings ***
Library SeleniumLibrary    plugins=${CURDIR}/MyPlugin.py
Library Collections

Suite Setup      Open browser    ${URL}    ${BROWSER}
Suite Teardown   Close All Browsers

*** Variables ***
${URL}           http://blazedemo.com/
${BROWSER}       Chrome
@{allowed_destinations}  Buenos Aires    Rome    London    Berlin    New York    Dublin    Cairo

*** Test Cases ***
The search page presents valid options for searching
    [Tags]    1
    Go To     ${URL}
    Title Should Be    BlazeDemo
    Element Should Be Visible    css:input[type='submit']
    Wait Until Element Is Enabled    css:input[type='submit']
    Wait Until Element Is Clickable    input[type='submit']
    ${values}=    Get List Items    xpath://select[@name='fromPort']    values=True
    Log    ${values}
    ${allowed_departures}=    Create List    Paris    Philadelphia    Boston    Portland    San Diego    Mexico City    São
    Paolo
    Lists Should Be Equal    ${allowed_departures}    ${values}
    ${values}=    Get List Items    xpath://select[@name='toPort']    values=True
    Log    ${values}
    Should Be Equal    ${allowed_destinations}    ${values}

The user can search for flights
    [Tags]    search_flights
    Go to     ${URL}
    Select From List By Value    xpath://select[@name='fromPort']    Paris
    Select From List by Value    xpath://select[@name='toPort']    London
    Click Button    css:input[type='submit']
    @{flights}=    Get WebElements    css:table[class='table']>tbody tr
    Should Not Be Empty    ${flights}
```

MyPlugin.py

```
from robot.api import logger

from SeleniumLibrary.base import LibraryComponent, keyword
from SeleniumLibrary.locators import ElementFinder

from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support.expected_conditions import presence_of_element_located
from selenium.webdriver.support.expected_conditions import element_to_be_clickable
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By

class MyPlugin(LibraryComponent):

    def __init__(self, ctx):
        LibraryComponent.__init__(self, ctx)

    @keyword
    def wait_until_element_is_clickable(self, selector):
        """Adding new keyword: Wait Until Element Is Clickable."""
        self.info('Wait Until Element Is Clickable')
        wait = WebDriverWait(self.driver, 10)

        my_elem = self.element_finder.find("css:"+selector)
        print(my_elem)
        first_result = wait.until(element_to_be_clickable((By.CSS_SELECTOR, selector)))
        return first_result
```

Running the tests in parallel is possible using [pabot](#).

Tests can be parallelized in different ways; we'll split them for running on a test basis.

We can also specify some variables; in this case, we'll use it to specify the "BROWSER" variable which is passed to the SeleniumLibrary.

chromebrowser.txt

```
--variable BROWSER:Chrome
```

```
pabot --argumentfile1 ffbrowser.txt --argumentfile2 chromebrowser.txt --argumentfile3 headlessffbrowsers.txt --
argumentfile4 safaribrowsers.txt --testlevelsplits 0_basic/search_flights.robot
```

Running these tests will produce a report per each "argumentfileX" parameter (i.e. per each browser). We can then submit those to Xray (e.g. using "curl" and the REST API), and assign it to distinct Test Executions where each one is in turn assigned to a specific Test Environment identifying the browser.

run_parallel_and_import.sh

```
#!/bin/bash

BROWSERS=(firefox chrome headlessfirefox safari)
PROJECT=ROB
TESTPLAN=ROB-22

token=$(curl -H "Content-Type: application/json" -X POST --data @"cloud_auth_prod.json" https://xray.cloud.getxray.app/api/v2/authenticate| tr -d ' ')

i=1
for browser in ${BROWSERS[@]}; do
  curl -H "Content-Type: application/xml" -X POST -H "Authorization: Bearer $token" --data @"pabot_results/output$i.xml" "https://xray.cloud.getxray.app/api/v2/import/execution/robot?projectKey=$PROJECT&testPlanKey=$TESTPLAN&testEnvironments=$browser"

  i=$((i+1))
done
```

example of cloud_auth_prod.json

```
{ "client_id": "0000215FFD69FE4644728C72182E0000", "client_secret":
"1c00f8f22f500006a8684d7c18cd6147ce2787d95e4da9f3bfb0af8f02ec0000" }
```

Projects / Robot Framework / ROB-23

Execution results [1594738115133]

Attach Create subtask Link issue Tests

Description

Add a description...

Tests

Create Test Add

Overall Execution Status

TOTAL TESTS: 2

2 PASSED

		Filters		10	Columns		
Rank	Key	Summary		Test Type	Status	Actions	
<input type="checkbox"/>	1	ROB-24	The search page presents valid options for searching	Generic	PASSED		...
<input type="checkbox"/>	2	ROB-25	The user can search for flights	Generic	PASSED		...
Prev	1	Next	Total 2 issues				

Test Environments

Associated Test Environments

firefox

In Xray, at the Test Plan-level we can see the consolidated results and for each test case we may drill-down and see all the runs performed and in which environment/browser.

Projects / Robot Framework / ROB-22

Test Runs of Test ROB-24

ROB-24

The search page presents valid options for searching

Latest Status

PASSED

All Environments, final status

10

Columns





Key	Summary	Test Environment	Status	Actions
<input type="checkbox"/> ROB-28	Execution results [1594738129645]	SAFARI	PASSED	...
<input type="checkbox"/> ROB-27	Execution results [1594738124786]	HEADLESSFIREFOX	PASSED	...
<input type="checkbox"/> ROB-26	Execution results [1594738119724]	CHROME	PASSED	...
<input type="checkbox"/> ROB-23	Execution results [1594738115133]	FIREFOX	PASSED	...

Prev 1 Next

Total 4 issues

In this case, we have the total of 4 Test Executions (i.e. for safari, headlessfirefox, chrome, firefox).

Test Plan for automated UI tests (RF)

 Attach  Create subtask  Link issue  Tests  Test Executions 

Description

Add a description...

Tests

 View on Board

All Environments, final status 

 Create Test Execution 

 Add 

Overall Execution Status

TOTAL TESTS: 2

2 PASSED

Filters 

10 

Columns 

Key	Summary	Assignee	#Test Executions	Latest Status	Actions
<input type="checkbox"/> ROB-24	The search page presents valid options for searching		4	 PASSED	 
<input type="checkbox"/> ROB-25	The user can search for flights		4	 PASSED	 
Prev 1 Next	Total 2 issues				

Test Executions

 Add Test Executions

10 

Columns 

Key	Summary	Assignee	#Tests	Status	Actions
<input type="checkbox"/> ROB-28	Execution results [1594738129645]	Sérgio Freire	2		
<input type="checkbox"/> ROB-27	Execution results [1594738124786]	Sérgio Freire	2		
<input type="checkbox"/> ROB-26	Execution results [1594738119724]	Sérgio Freire	2		
<input type="checkbox"/> ROB-23	Execution results [1594738115133]	Sérgio Freire	2		
Prev 1 Next	Total 4 issues				

Tracking automation results

Besides tracking automation results on the Test Execution issues themselves, it's also possible to track in different places so the team gets fully aware of them.

On the user story issue screen

Right from within the user story issue screen, we now see one test (i.e. automated script) covering it. We can also see its latest result and how it impacts the overall coverage calculation for the user story; if the user story shows as “OK”, you know that all tests covering it passed, accordingly with the latest results obtained for each one of them.

Projects / Robot Framework / ROB-11

As a user, I can login the web application

Attach

Create subtask

Link issue

Test Coverage

Description

As a user, I can login the web application

Linked issues

is tested by

ROB-21 Valid Login

↑

TO DO

Test Coverage

Calculate the Test Coverage for the following scopes.

Latest

Version

Test Plan

Create new Sub Test Execution

Create new Test

Test Environment

All Environments

OK

Final statuses have precedence over non-final.

Status	Key	Summary	Test Status
↑.. TO DO	ROB-21	Valid Login	PASSED

Prev 1 Next

On the Test Plan

At the Test Plan-level, the entity that defines the scope of testing and tracks its progress, we can quickly assess the latest consolidated test results (i.e. the latest result obtained for each Test being tracked).

Test Plan for automated UI tests (RF)

Attach

Create subtask

Link issue

Tests

Test Executions

...

Description

Add a description...

Tests

View on Board

All Environments, final status

Create Test Execution

Add

Overall Execution Status

TOTAL TESTS: 8

8 PASSED

Filters

10

Columns

Key	Summary	Assignee	#Test Executions	Latest Status	Actions	
<input type="checkbox"/> ROB-14	Valid Login		1	<div></div> PASSED	<div></div>	...
<input type="checkbox"/> ROB-15	Invalid Username		1	<div></div> PASSED	<div></div>	...
<input type="checkbox"/> ROB-16	Invalid Password		1	<div></div> PASSED	<div></div>	...
<input type="checkbox"/> ROB-17	Invalid Username And Password		1	<div></div> PASSED	<div></div>	...
<input type="checkbox"/> ROB-18	Empty Username		1	<div></div> PASSED	<div></div>	...
<input type="checkbox"/> ROB-19	Empty Password		1	<div></div> PASSED	<div></div>	...
<input type="checkbox"/> ROB-20	Empty Username And Password		1	<div></div> PASSED	<div></div>	...
<input type="checkbox"/> ROB-21	Valid Login		1	<div></div> PASSED	<div></div>	...

References

- Robot Framework
- Awesome Robot Framework (curated list of resources)
- Code used in the first example
- Integration capabilities that Xray provides for Robot Framework XML reports
- pabot