

Testing using Robot Framework integration in Python or Java

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

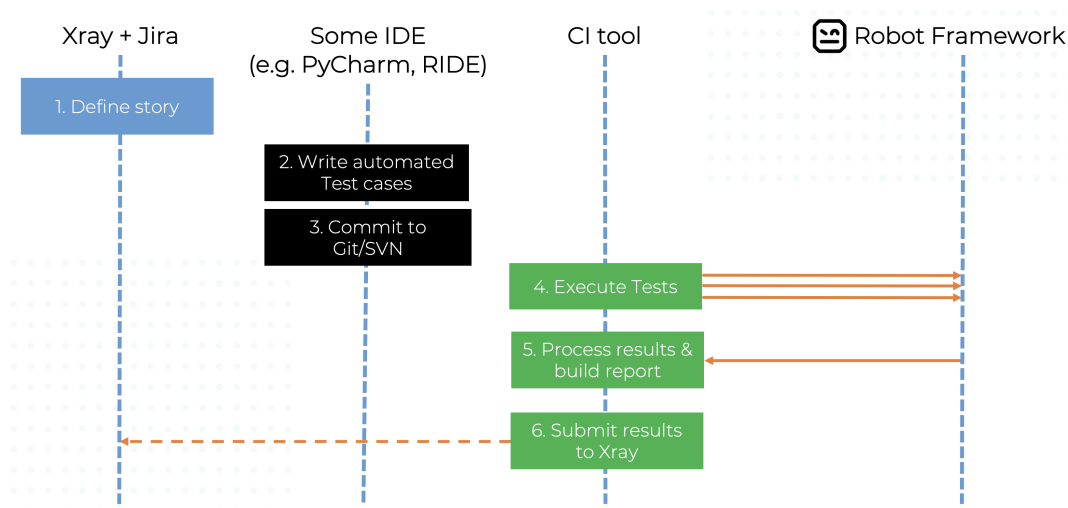
The full ATDD workflow

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).

Robot / ROB-11

As a user, I can login the web application

Edit
Comment
Assign
More
Start Progress
Resolve Issue
Close Issue
Admin

Details

Type: Story
 Priority: Major
 Affects Version/s: None
 Labels: None
 Sprint: Robot Sprint 1
 Requirement Status: UNCOVERED

Status: OPEN View Workflow
 Resolution: Unresolved
 Fix Version/s: None

Description


Test Coverage

Create Test
 + Link

No Tests were found testing the requirement.

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.


Robot / ROB-12

automated UI tests (RF)

Edit
Comment
Trigger Jenkins Build ...
More ▾
Stop Progress
Resolve Issue
Close Issue
Admin ▾

Details

Type: Test Plan

Priority: Major

Affects Version/s: None

Labels: None

Status: IN PROGRESS [\(View Workflow\)](#)

Resolution: Unresolved

Fix Version/s: None

Description

Tests

Test Plan 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.

Jira Software

Dashboards ▾
Projects ▾
Issues ▾
Boards ▾
Structure ▾
DbConsole
easyBI
Tests ▾
Create

Search

9 days remaining
Complete Sprint
Board ▾

Robot webinar

Robot Sprint 1

QUICK FILTERS: Only My Issues Recently Updated

TO DO

IN PROGRESS

DONE

ROB-11

As a user, I can login the web application

UNCOVERED

ROB-12

automated UI tests (RF)

No tests

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

*** 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 checked, 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.



Robot / ROB-18 Valid Login

[Edit](#) [Comment](#) [Assign](#) [More](#) [Start Progress](#) [Resolve Issue](#) [Close Issue](#) [Admin](#)

Details

Type: **Test** Status: **OPEN** ([View Workflow](#))
Affects Version/s: **None** Resolution: **Unresolved**
Labels: **UI** Fix Version/s: **None**

Description

Test Details

Type: **Generic**
Definition: **Login Tests.Valid Login.Valid Login**



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).



Robot / ROB-42 Execution results - output.xml - [1591121055062]

[Edit](#) [Comment](#) [Synchronize Tests from...](#) [More](#) [Close Issue](#) [Reopen Issue](#) [Admin](#)

Details

Type: **Test Execution** Status: **RESOLVED** ([View Workflow](#))
Priority: **Medium** Resolution: **Fixed**
Affects Version/s: **None** Fix Version/s: **None**
Labels: **None**
Revision: **12**
Test Environments: **headlessfirefox**
Test Plan: **ROB-12**

Description

Tests

[+ Add](#)

Overall Execution Status

8 PASS

Total Tests: 8

[Filter\(s\)](#)



Show **100** entries Columns

Rank	Key	Summary	Test Type	#Req	#Def	Assignee	Status
1	ROB-21	Invalid Username And Password	Generic	0	0	Administrator	PASS
2	ROB-20	Invalid Password	Generic	0	0	Administrator	PASS
3	ROB-19	Empty Username	Generic	0	0	Administrator	PASS
4	ROB-18	Valid Login	Generic	1	0	Administrator	PASS
5	ROB-23	Invalid Username	Generic	0	0	Administrator	PASS
6	ROB-22	Empty Username And Password	Generic	0	0	Administrator	PASS
7	ROB-14	Valid Login	Generic	0	0	Administrator	PASS
8	ROB-24	Empty Password	Generic	0	0	Administrator	PASS

Showing 1 to 8 of 8 entries

[First](#) [Previous](#) [1](#) [Next](#) [Last](#)

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.

The screenshot shows the 'Valid Login' test execution screen. At the top, the breadcrumb navigation indicates the path: Robot / Test Plan: ROB-12 / Test Execution: ROB-17 / Test: ROB-18. The execution status is 'PASS', and the test was started and finished on 14/May/20 at 4:42 PM. The test was assigned to and executed by the 'Administrator'. The screen includes sections for 'Execution Details', 'Test Description', 'Test Issue Links (1)', and 'Test Details'. The 'Test Details' section shows the test type as 'Generic' and the definition as 'Login Tests.Valid Login.Valid Login'. The 'Results' section displays a table of test steps with their durations and statuses.

Context	Output	Duration	Status
Open Browser To Login Page	-	3 sec	PASS
Input Username	-	24.000 ms	PASS
Input Password	-	22.000 ms	PASS
Submit Credentials	-	46.000 ms	PASS
Welcome Page Should Be Open	-	7.000 ms	PASS
Close Browser	-	1 sec	PASS

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:

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 headlessff safari)
PROJECT=CALC
TESTPLAN=CALC-6424

i=1
for browser in ${BROWSERS[@]}; do
    curl -H "Content-Type: multipart/form-data" -u admin:admin -F "file=@pabot_results/output$i.xml" \
    "http://jiraserver.example.com/rest/raven/1.0/import/execution/robot? \
    projectKey=$PROJECT&testPlanKey=$TESTPLAN&testEnvironments=$browser"
    i=$((i+1))
done
```

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.

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

The screenshot displays the Xray Test Plan interface for 'CALC-6424'. The top section shows the test plan details, including the title 'TP with automated tests (RF)' and a status of 'OPEN'. Below this, the 'Tests' section is expanded, showing a 'Test Plan Board' with a green bar indicating '2 PASS' results. A table below the board lists test cases with their keys, summaries, and environments. A red box highlights the first four test cases, which are the ones shown in the 'Test Executions' section below.

Key	Summary	Requirements	#Test Executions	Issue Assignee	Latest Status
CALC-6426	The user can search for flights		10	Administrator	PASS
CALC-7571	Execution results - output4.xml - [1593668051551]				PASS
CALC-7570	Execution results - output3.xml - [1593668047502]				PASS
CALC-7569	Execution results - output2.xml - [1593668043107]				PASS
CALC-7568	Execution results - output1.xml - [1593668038805]				PASS
CALC-6427	The search page presents valid options for searching		10	Administrator	PASS

The 'Test Executions' section shows a detailed view of the four test cases highlighted in the table above. It includes columns for Key, Summary, #Tests, Issue Assignee, Test Environments, Revision, Fix Version/s, and Status. The status for all four test cases is 'PASS'.

Key	Summary	#Tests	Issue Assignee	Test Environments	Revision	Fix Version/s	Status
CALC-7571	Execution results - output4.xml - [1593668051551]	2	Administrator	safari			PASS
CALC-7570	Execution results - output3.xml - [1593668047502]	2	Administrator	headlessff			PASS
CALC-7569	Execution results - output2.xml - [1593668043107]	2	Administrator	chrome			PASS
CALC-7568	Execution results - output1.xml - [1593668038805]	2	Administrator	firefox			PASS

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.

Robot / ROB-11

As a user, I can login the web application

EditCommentAssignMore

Start ProgressResolve IssueClose IssueAdmin

Details

Type:Story

Priority:Major

Affects Version/s:None

Labels:None

Sprint:Robot Sprint 1

Requirement Status:OK

Status:OPEN (View Workflow)

Resolution:Unresolved

Fix Version/s:None

Description

Test Coverage

Create Test+ Link

TEST COVERAGE FOR THE FOLLOWING ANALYSIS SCORE

Scope: Version; Version: None - latest execution; Environment: All Environments

OK

Filter(s)

Show 10 entriesColumns

P	Status	Resolution	Key	Summary	Test Runs	Test Status
	OPEN	Unresolved	CALC-6057	Valid Login	10	PASS

Showing 1 to 1 of 1 entries

FirstPreviousNextLast

On the Agile Board

On Agile Boards (e.g. Scrum boards), we can now assess the coverage of our user story taking into account the testing results.

We may also track the overall Test Plan consolidated progress on the Test Plan issue related card. Note that we could include Test Executions in the board if we wish so; however, in CI scenarios that could be counterproductive.

Robot webinar

Robot Sprint 1

9 day

QUICK FILTERS: Only My IssuesRecently Updated

TO DO

IN PROGRESS

DONE

ROB-11

As a user, I can login the web application

OK

ROB-12

automated UI tests (RF)

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).



Robot / ROB-12

automated UI tests (RF)

[Edit](#) [Comment](#) [Trigger Jenkins Build ...](#) [More ▾](#) [Stop Progress](#) [Resolve Issue](#) [Close Issue](#) [Admin ▾](#)

Details

Type: Test Plan
Priority: Major
Affects Version/s: None
Labels: None
Status: **IN PROGRESS** [\(View Workflow\)](#)
Resolution: Unresolved
Fix Version/s: None

Description

Tests

Test Plan Board

[+ Create Test Execution ▾](#)

[+ Add ▾](#)

Overall Execution Status

8 PASS

Total Tests: 8

Contains text: ROB-18



Show entries All Environments ▾ Columns ▾

	Key	Summary	Requirements	#Test Executions	Issue Assignee	Latest Status	
	ROB-18	Valid Login	ROB-11	1	Administrator	PASS	...

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](#)