

**Writing stable & maintainable RF Test Scripts**

**Roberto SP. Galman**

Document History

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Issue Date | Author | Changes |
| 1.00 | 8/27/2013 | Roberto Galman | Initial Document |
| 1.01 | 09/02/2013 | Roberto Galman | Added notes on wait usage, continuous integration using Jenkins, added useful keywords section, added input value override section |
| 1.02 | 09/04/2013 | Roberto Galman | Updated based from Review Feedback   * Added way to set implicit\_wait * Added new keyword – wait until page loaded * Added more useful keywords, built-in & Selenium |
|  |  |  |  |

**Contact Details:**

From: **NorthgateArinso**

Address: 4F Bldg 1. ETON Cyberpod

Corinthian Ortigas Ave. cor

EDSA, Quezon City

Telephone: +63 2 626 1200

Fax:

**Confidentiality/Disclaimer:**

This document and all information contained within is the property of NorthgateArinso. It is issued in confidence and must not be reproduced in whole or in part or used or given or communicated to any third party without the prior written consent of NorthgateArinso. This document does not form or constitute part of a contractual agreement, and its submission does not imply acceptance of any commercial terms. Any statements and recommendations made or advice given shall not in any circumstances be deemed to constitute a warranty by NorthgateArinso as to the accuracy of such statements, recommendations or advice. NorthgateArinso shall not be liable for any loss, expense, damage or claim arising from the statements made or omitted to be made, or advice given or not given in this document.

Table of Contents

[1. Introduction 4](#_Toc366073410)

[2. Test Script Implementation (Flow) , Structure & Execution 4](#_Toc366073411)

[2.1. Consistency & Accuracy 4](#_Toc366073412)

[2.2. Test Script Structure 6](#_Toc366073413)

[2.3. Writing maintainable test script using Template Flow & Keywords 6](#_Toc366073414)

[2.4. Batch Run 8](#_Toc366073415)

[3. Test Stability 10](#_Toc366073416)

[3.1 Element not found 10](#_Toc366073417)

[3.2 Field value not saved on record submit 12](#_Toc366073418)

[4. Use Input Value Overrides 13](#_Toc366073419)

[5. Performance 15](#_Toc366073420)

[5.1. Implicit Wait Setting 15](#_Toc366073421)

[5.2. Hardware 16](#_Toc366073422)

[6. Continuous Integration using Jenkins 17](#_Toc366073423)

[7. APPENDIX 18](#_Toc366073424)

[A. Useful Keywords 18](#_Toc366073425)

[B. References 19](#_Toc366073426)

List of Tables

[Table 1 Test Parameter - Value 4](#_Toc366073445)

[Table 2 Generic Keywords for Attendance Record 6](#_Toc366073446)

[Table 3 Implicit\_wait value per test run 15](#_Toc366073447)

List of Figures

[Figure 1 Attendance Request Form 5](#_Toc366073427)

[Figure 2 Overtime Request - Optional fields 5](#_Toc366073428)

[Figure 3 Test Flow using keywords 7](#_Toc366073429)

[Figure 4 Test Data using common flow 7](#_Toc366073430)

[Figure 5 rf\_batch\_runner usage 8](#_Toc366073431)

[Figure 6 Sample Test list 8](#_Toc366073432)

[Figure 7 generated zip package – test logs 8](#_Toc366073433)

[Figure 8 Batch test list organized by record type 9](#_Toc366073434)

[Figure 9 Progress Bar Pop-up 11](#_Toc366073435)

[Figure 10 Wait until element is visible 11](#_Toc366073436)

[Figure 11 Change focus by clicking another element 12](#_Toc366073437)

[Figure 12 Hardcoded date values 13](#_Toc366073438)

[Figure 13 Using Date override to create a record with FROM = Today & TO = 1 day from now 14](#_Toc366073439)

[Figure 14 Varying implicit\_wait results to shorter execution time 15](#_Toc366073440)

[Figure 15 Setting the implicit\_wait & timeout value 16](#_Toc366073441)

[Figure 16 Performance Comparison 16](#_Toc366073442)

[Figure 17 Sample Jenkins Project 17](#_Toc366073443)

[Figure 18 Robot Framework Plug-in for Jenkins 17](#_Toc366073444)

# Introduction

The purpose of this document is to discuss common errors encountered in the Robot Framework (RF) Automation test suite using Selenium Webdriver Library & to suggest techniques to solve these issues. The techniques & tips discussed in this document were based on the euHReka automation test execution but the general approach can also be applied to other projects. The primary objective is to create an accurate, stable & maintainable RF test suite.

# Test Script Implementation (Flow) , Structure & Execution

This section covers the following: consistency and accuracy in the test script implementation, Test case structure in a test suite, writing maintainable test script using keywords, and test execution techniques

## Consistency & Accuracy

The test script implementation should be consistent with the test requirements which can easily be determined from the test instance name.

For example, given the following test name:

*WA\_TMN\_TimeView(Monthly)\_Attendance (Work At Home-All Day)\_Display Create Change Delete\_by EMP*

Based from the test name, the test script should use the following test parameter & values:

|  |  |
| --- | --- |
| Test Parameter | Value |
| **Calendar View** | Monthly Calendar View |
| **Type** | Work at Home |
| **Schedule** | All Day |
| **Scenario (Test Cases)** | Create, Display, Change, Delete **to be implemented in Monthly View if possible** |

Table 1 Test Parameter - Value

At the minimum, the values listed in the table above should be set in the test script. However, in some instances, there are other fields available aside from the required field setting based on the test, and the mandatory fields as shown in **Figure 1** such as the *Comments* & *Cost Center* fields.

In this case, the question is whether we should also set these optional fields. Currently, we have no standard for field entries/selection and this is up to the tester’s judgment. As a guide in deciding whether to populate an optional field, you may consider the following:

* Is this covered in another test?
* Is there a default value set when record is saved?
* Is there a known issue related to this field?
* Is there an added value if this field is set? Statement/path coverage, State/Transition
* Is the field available only when another field is set to a specific value? Refer to **Figure 2** Overtime *Request - Optional fields*, the **Evaluation Type** field is visible only if **Type** is set to “Overtime”

Ideally, all fields should be set in order to increase test coverage. However, note that verification **SHOULD** be done on all fields regardless if it was set or not when the record was created.

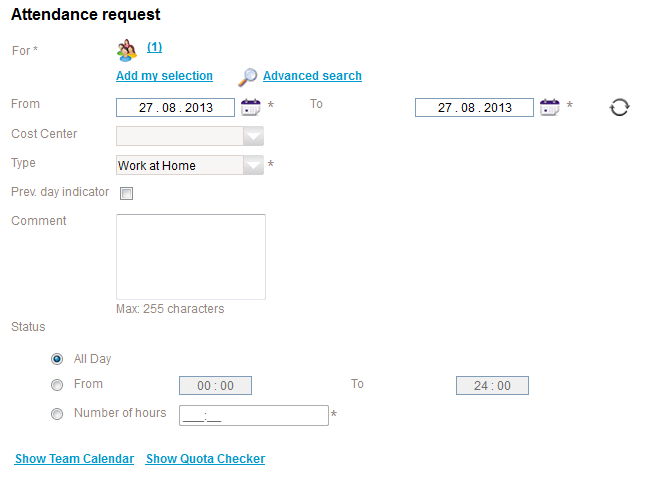
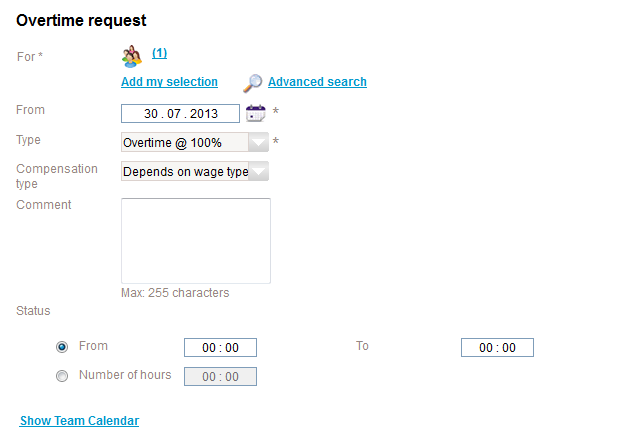


Figure 1 Attendance Request Form



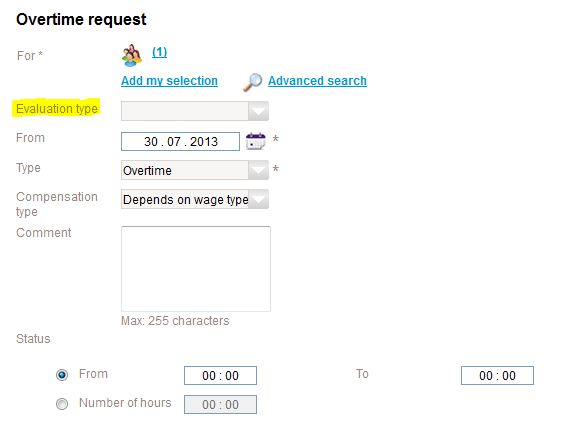


Figure 2 Overtime Request - Optional fields

## Test Script Structure

The code structure can be determined from the test name. In this context, the code structure refers to the number of Test Cases per Test Suite/Test Script.

In the case of “*WA\_TMN\_TimeView(Monthly)\_Attendance (Work At Home-All Day)\_****Display Create Change Delete****\_by EMP”,* the test script should have 4 test cases namely: Create, Verify, Change & Delete as shown in **Figure 3**.

Example 2:

*“ WA\_TMN\_Flexflows-TimeView(Monthly)\_Absence (Maternity Leave)\_Decline (Sent for Deletion Task)\_by MGR”*

Example 2 should be consists of a single test case – Decline Sent for Deletion Task.

## Writing maintainable test script using Template Flow & Keywords

For related tests, it is recommended to use common/generic keywords for consistency & ease of maintenance. For example in the case of Attendance record, you can create a generic keyword based on the common scenarios such as **Add, Verify, Change, and Delete**.

|  |  |  |  |
| --- | --- | --- | --- |
| Keyword | Attendance Type | Calendar View | Role |
| WA\_TMN\_ADD\_ATTENDANCE\_RECORD\_BY\_EMPLOYEE\* | Work at Home, Work-off site | List, Team, Monthly | Employee |
| WA\_TMN\_VERIFY\_ATTENDANCE\_RECORD\_DISP\_BY\_EMPLOYEE\* | Work at Home, Work-off site | List, Team, Monthly | Employee |
| WA\_TMN\_EDIT\_ATTENDANCE\_RECORD\_BY\_EMPLOYEE\* | Work at Home, Work-off site | List, Team, Monthly | Employee |

Table 2 Generic Keywords for Attendance Record

*\* Defined in EOD\_GL\_WA\_TM\_Time\_Views.txt*

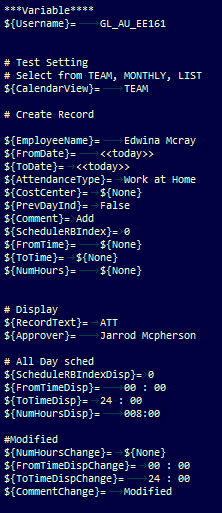
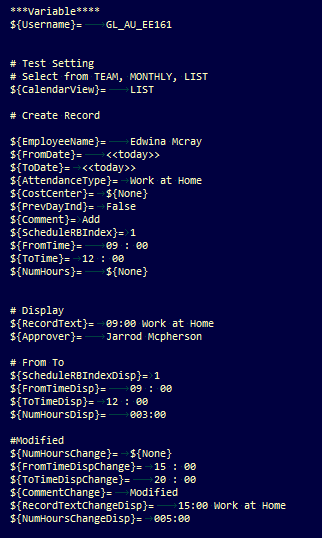
It is not necessary to create a keyword which can handle all cases as this will complicate the implementation. You can create common keywords per set of related tests- i.e. tests having the same flow & verification points. In Table 2, common keywords were created for Attendance record actions for Employee Role. These keywords can then be called in the test script to define the template workflow as shown in **Figure 3**.

The 3 keywords in Table 2 can be used in 6 tests (Role x Attendance Type x Calendar View – 1x2x3). Any change in the test requirements can be implemented in the keyword-level hence, no need to manually edit/update each test. In this way, tests can easily be maintained and no redundant keywords will be introduced.

## 

Figure 3 Test Flow using keywords

The test case flow can then be used as a template for similar tests. Only the test data will vary per test as shown in **Figure 4**.

*(a) Team View Test (b) List View Test*

Figure 4 Test Data using common flow

It may also be useful to have design documentation per sub-module. The documentation may include the test flow (flowchart), keywords, and test instances covered. Having a documentation of the implementation will present the following advantages:

* Accuracy – Focal points can easily review and verify the flow
* No redundant keywords
* Ease of Maintenance – well-planned structure & modular setup, high-reusability
* Reference to new Automation Engineers

## Batch Run

The rf\_batch\_runner.py is used in batch test runs. It executes tests included in the input test list. Compared to running test by tags, the batch runner automatically generates separate report per test in the list. It will generate a zip package which includes the test logs & screenshots. The generated zip can be uploaded in ALM as proof of regression test.

The filenames in the list should include the path to the file as shown in **Figure 6**. The directory can be an absolute or relative path to the file.

Sample usage:

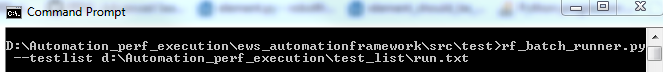


Figure 5 rf\_batch\_runner usage

run.txt:



Figure 6 Sample Test list

**Notes:**

* To exclude a test in a batch run, you may comment-out (#) the line in the test list.
* You can run a single test using rf\_batch\_runner.py by adding just one test in the list or by commenting-out the other entries
* Batch run automatically creates a zip package backup of the test logs – zip file is located in $TESTDIR/zip

*where:* $TESTDIR is the root folder (same level as euhreka-testing folder)

* The generated zip file is automatically named as follows:

***<test\_name>\_YYYY\_MM\_DD\_hh\_mm\_ss\_<PASS/FAIL>***

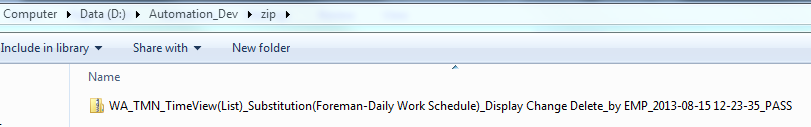


Figure 7 generated zip package – test logs

Using rf\_batch\_runner, we can group related tests and execute it in batches. Tests can be organized based on the workflow. Tests with similar flow & verification points can be grouped into one test list as shown in **Figure 8**.

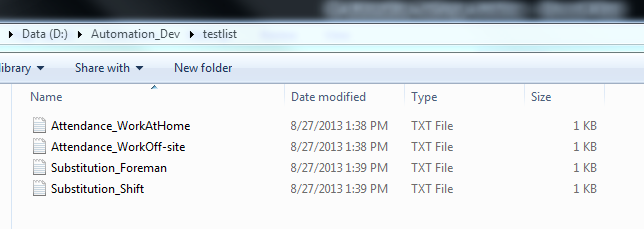


Figure 8 Batch test list organized by record type

1. **Addressing Batch run errors**

Batch run executes tests sequentially; hence, it is important that the previous test cleanly exits so as not to affect the succeeding tests. The active window from the previous run which failed to close due to errors can cause forced logout due to inactivity or eat-up resources affecting the performance.

“Close Browser” keyword is best handled by the “Test Teardown” since this will be executed at the end of each test case regardless if the test passed or failed. This will ensure that the previous test cleanly exit before the next one is triggered. However, we can’t implement this in all our test scripts because we have tests with multiple test cases. Note that if we define “close browser” in the “Test Teardown”, the browser window will be closed at the end of each test case. Hence, when the next test case is executed “No browser open” error will be encountered.

To solve this issue, we need to add cleanup steps at the end of each test. This can be as simple as deleting the created record and adding “euh EWS LogOff” at the end since this keyword also implements the “Close Browser” method. This will allow us to run the test independently & repeatedly.

# Test Stability

This section covers common errors encountered in the Automation tests & suggests ways to solve these issues

## Element not found

This is the most common error and is primarily due to timing issues. The test code is trying to access a WebElement which is not yet fully-loaded in the driver. This timing race can be handled by using waits – implicit & explicit waits.

* 1. **Implicit wait**

This is handled internally and in eWS the default value is 45 seconds. This will force WebDriver to poll the DOM for a specific amount of time to find an element if it’s not immediately available.

* 1. **Explicit wait**

When implemented, this will wait for a specific condition before proceeding to the next line of code. In eWS, explicit wait methods include:

* **wait\_until\_euh\_element\_is visible**
* **euh wait until loaded**
* **wait\_for\_manual\_step (default=30 seconds)**

It is recommended to wait for a specific element to exist in the DOM before proceeding with the verification. This will make the test more stable and resistant to errors due to slow application performance or timing race.

For example when creating/editing attendance record, you can add “wait until” the “Time entry screen” is visible. This will make sure that the page and fields to be processed are visible and can be interacted with. You can add this method at the start of the keyword prior to setting/verifying field values. It would be best to add this “wait” method after actions which trigger the **loading div** (page load) or **test\_popupCaption** (dummy class).

The pop-up prevents interaction with the elements underneath. For example in **Figure 9**, you need to wait for the pop-up to close. In this case since the pop-up will close by itself (no button to click), you need to wait for the main form underneath to be visible. You can add “wait until euh element is visible id =app\_timeEntryScreen” in the flow as shown in **Figure 10**. The main form id can be determined using Firebug.

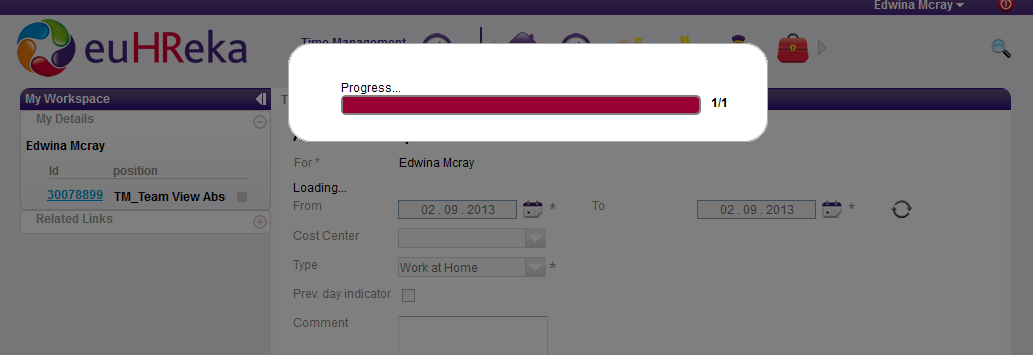


Figure 9 Progress Bar Pop-up

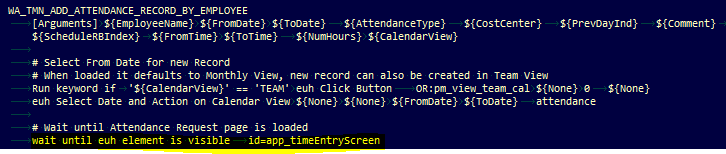


Figure 10 Wait until element is visible

You can use the “wait until euh element is visible” in the following format for faster execution:

**Using locator/s:**

Wait until euh element is visible <locator>

e.g. *Wait until euh element is visible id=content*

**Using Object Repository:**

Wait until euh element is visible OR:<object>

e.g. *Wait until euh element is visible OR:maindiv*

*Where:* **maindiv** is defined in object\_repository.py

If you will be using the “wait until euh element is visible” in multiple tests, you may opt to use the wait-OR format instead. This way you can assign a more descriptive name for the locator & if you need to change the locator you only need to update the OR definition.

Do **NOT** use “wait until euh element is visible” without any parameter because when the default value is used the function is just pass-through (no element wait).

**“euh wait until loaded”** is used to wait for page load to complete. This keyword waits until the loading div is cleared. This keyword can be used after actions which trigger page load.

It is advisable to poll for a certain condition instead of using thread sleep. **“Wait for manual step”** is an example of thread sleep. This pauses the script for “timeout” seconds (default is 30 seconds). This may seem to be the easiest solution to compensate for timing issues but this method is unreliable and fragile. More often than not, it “sleeps” the script longer than needed causing longer execution time.

The “wait for manual step” can be useful to stop the execution temporarily for debugging purposes.

You may use a combination of these wait keywords in your test. To start, you can code the test script without any “waits” then execute. If you encounter any issues related to timing, you can insert “waits” in the test flow.

## Field value not saved on record submit

There are instances where in the Automation fails because a specific field was not set, however, when you check the log & screenshots Selenium Webdriver was able to set a value but somehow the value did not reflect on record submit.

From Investigation, this may be related to onfocus or onblur events not triggered by Selenium Webdriver. To simulate onblur event, you can add an action which shifts the focus to another element prior to record submit.

For example, you observed that the “Comment” field of the attendance record is not ALWAYS being set. You can add an action which changes the focus from the Comment field to another element – e.g. main div.

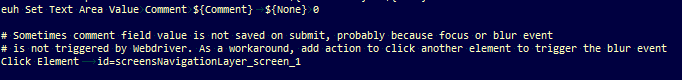


Figure 11 Change focus by clicking another element

In **Figure 11**, after setting the value for comment, I clicked another element to trigger onblur event.

Note that, another possible cause of this issue is the focus/blur event not being triggered if the window is not focused. You can try running the automation test on its own so that the window will not lose the focus state.

# Use Input Value Overrides

Input Value Override is a dictionary which specifies the text overrides that testers can pass for set and check keywords. The override object returns the appropriate handler based upon the text value of the variable – e.g. if val = <<TODAY>>, it will return the ‘today’ override handler.

To prevent hardcoded dates in the test script as shown in **Figure 12**, the following overrides can be used:

* <<TODAY>>
* <<DATEADD Y M D>>

Where:

<<TODAY>> override returns the current date based on server time

<<DATEADD Y M D>> override returns the result of Date math with +/- Year(Y) Month(M) Day(D) offsets respectively

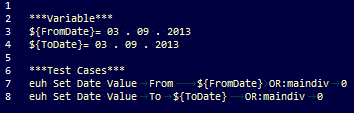


Figure 12 Hardcoded date values

For example,

To Get Date 5 days from now:

${future\_date}= <<DATEADD 0 0 5>>

Negative offset also works – i.e. select earlier date from reference date.

To Get Date 3 days ago:

${past\_date}= <<DATEADD 0 0 -3>>

Note that, the direct assignment to a variable in the RF script DOES NOT translate it to the expected date. The translation is implemented in the function/keyword used in the ews\_automationframework.

The offset or delta is measured from the current server date/time. If we want to define a start\_date, we need to add another parameter when using the override.

<<DATEADD Y M D YYYY-MM-DD>>

e.g. 2 Months & 1 Day from June 07, 2013

<<DATEADD 0 2 1 2013-06-07>>

Updating the test script in **Figure 12** to use date overrides,

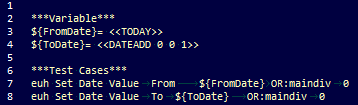


Figure 13 Using Date override to create a record with FROM = Today & TO = 1 day from now

# Performance

Currently, the performance has been a major issue in the Automation test execution. The following were identified as ways to improve performance:

## Implicit Wait Setting

The implicit wait in eWS is set to 45 seconds by default. If the WebElement is not located initially, it would wait 45 seconds for the element to appear. With the current improvement in the application & network performance (eop), implicit\_wait can be set to between 10-20 seconds to minimize wait/idle time. The 45 seconds wait time can add up especially if find element method is called repeatedly. This would result to longer execution time than necessary.

A high implicit\_wait value can significantly affect the execution time as shown in **Figure 12**. Setting implicit\_wait value from the default- 45 seconds (Build#10) to 10 seconds (Build#12) improves performance by 50%.

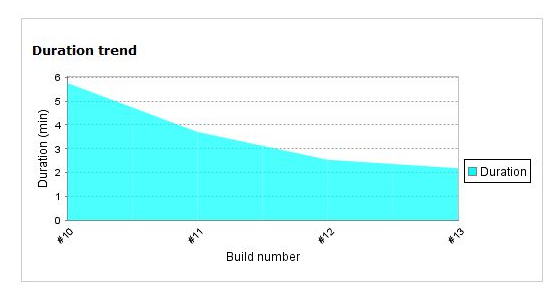


Figure 14 Varying implicit\_wait results to shorter execution time

|  |  |
| --- | --- |
| Build Number | Implicit\_wait value |
| #10 | 45 sec (default) |
| #11 | 20 sec |
| #12 | 10 sec |
| #13 | 5 sec |

Table 3 Implicit\_wait value per test run

Setting implicit\_wait to a much lower value can improve the performance but may affect the stability. Depending on the application or network performance, you can set the implicit\_wait to a reasonable value e.g. 10 seconds.

The implicit\_wait can be set in “Core-Flows\resources\EOD\_Master\_Test\_Flow.txt” as shown in **Figure 15**. The implicit\_wait is set to **45** seconds. You may set it between 10-20 seconds depending on the application or network performance.

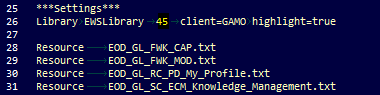


Figure 15 Setting the implicit\_wait & timeout value

## Hardware

Based from the Automation Performance test, the hardware has significant impact on execution time. It would be ideal to have a dedicated machine for running the Automation tests. This would have higher specs than the machine we are currently using to handle the memory-intensive Selenium WebDriver Processes.

The better performance on higher machine specs was consistently observed in our performance test execution in MNL, HYD and KL. The performance values would be more meaningful if tests were run on same machine specs to make sure that the difference in performance is due to the application and network performance.

1. Using company issued laptop:



1. Using PC with higher spec:

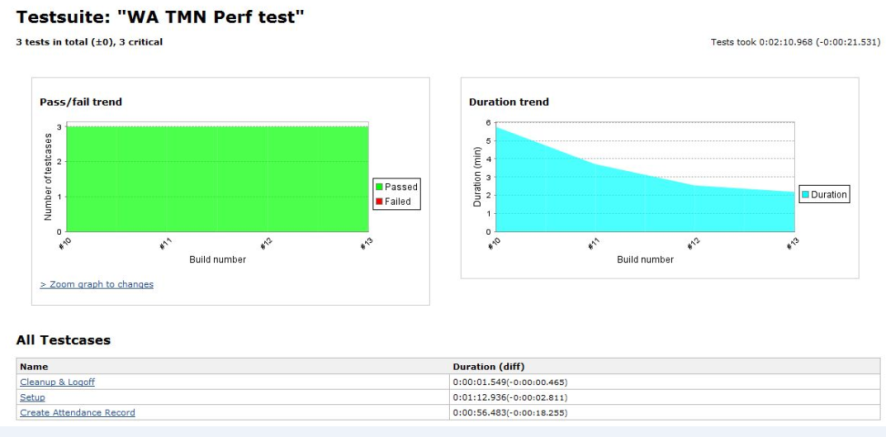


Figure 16 Performance Comparison

# Continuous Integration using Jenkins

To support continuous integration (CI), we can use [Jenkins](https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins) to run the test suite using predefined triggers – e.g. by Schedule, Code change. We can create Jenkins Jobs/Projects for the following:

* Dryrun
* Sub-modules (e.g. WA > TMN > TimeViews )
* Full euhreka-testing suite
* Suite for Performance test

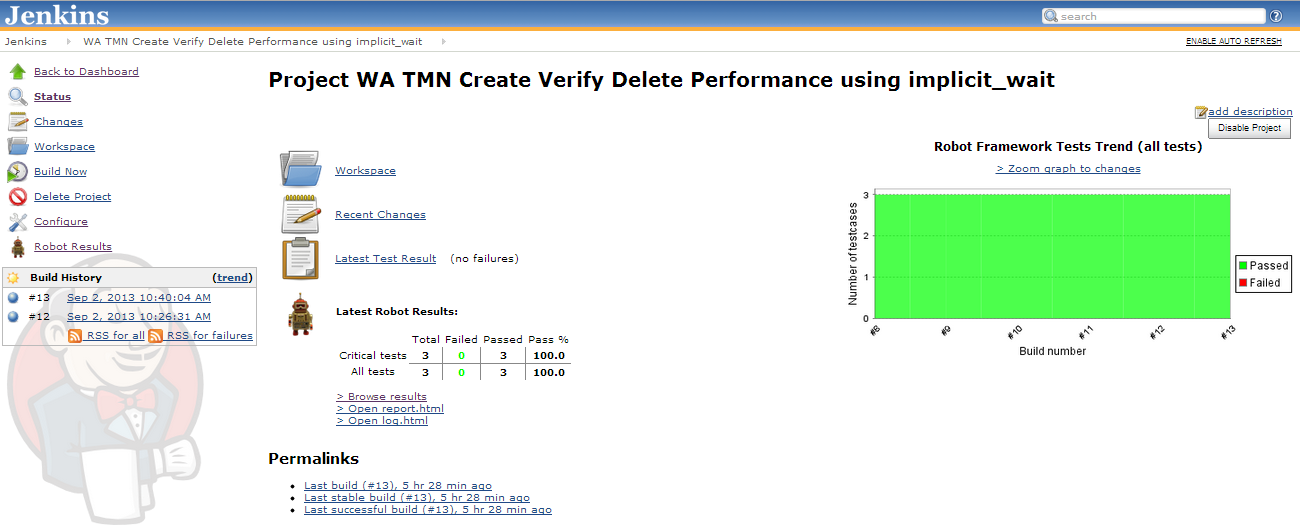


Figure 17 Sample Jenkins Project

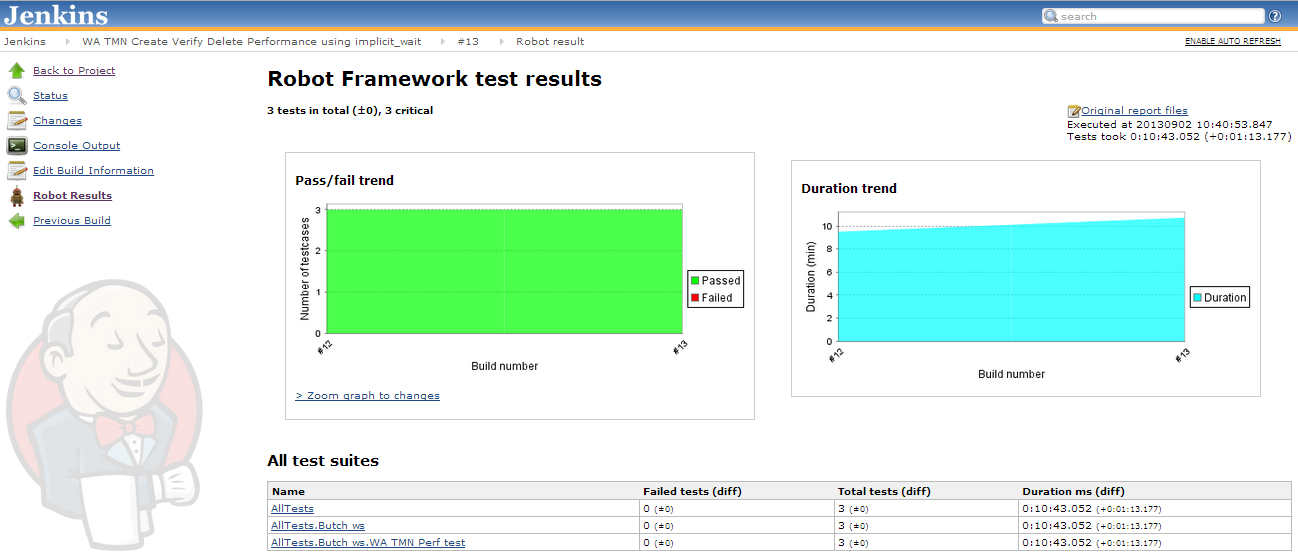


Figure 18 Robot Framework Plug-in for Jenkins

# APPENDIX

1. Useful Keywords

**1. Robot Framework** [**Built-in**](http://robotframework.googlecode.com/hg/doc/libraries/BuiltIn.html?r=2.6.3) **Keywords**

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Arguments** | **Description** |
| **Catenate** | *\*items* | Catenates the given items together and returns the resulted string. |
| **Get Time** | *format=timestamp, time=NOW* | Returns the given time in the requested format. |
| **Run Keyword And Expect Error** | *expected\_error, name, \*args* | Runs the keyword and checks that the expected error occurred. |
| **Run Keyword And Ignore Error** | *name, \*args* | Runs the given keyword with the given arguments and ignores possible error. |
| **Run Keyword If** | *condition, name, \*args* | Runs the given keyword with the given arguments, if *condition* is true. |

**2.** [**Selenium**](http://robotframework-seleniumlibrary.googlecode.com/hg/doc/SeleniumLibrary.html?r=2.8#introduction) **Library Keywords**

|  |  |  |
| --- | --- | --- |
| **Keyword** | **Arguments** | **Description** |
| **Page Should Contain** | *text, loglevel=INFO* | Verifies that current page contains *text*. |
| **Get Element Attribute** | *attribute\_locator* | Return value of element attribute. *attribute\_locator* consists of element locator followed by an @ sign and attribute name, for example "element\_id@class". |
| **Capture Page Screenshot** | *filename=None* | Takes a screenshot of the current page and embeds it into the log. |
| **Wait Until Page Contains** | *text, timeout=None, error=None* | Waits until *text* appears on current page. |

1. References

[1] **Test Automation with Robot Framework** . <http://www.virtuousprogrammer.com/?p=264>

[2] **Robot Framework User Guide** . <http://robotframework.googlecode.com/hg/doc/userguide/RobotFrameworkUserGuide.html> .

[3] **Ways to resolve unstable Webdriver tests** . <http://www.michaelthelin.se/?p=299> .

[4] **Meet Jenkins** . <https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins>

[5] **Selenium Library Keywords** . <http://robotframework-seleniumlibrary.googlecode.com/hg/doc/SeleniumLibrary.html?r=2.8#introduction>.

[6] **Robot Framework Built-in keywords** . <http://robotframework.googlecode.com/hg/doc/libraries/BuiltIn.html?r=2.6.3> .