Lab Guide

Unit Testing Bots Lab Guide

Nigel T. Crowther
ncrowther@uk.ibm.com

Hands-on Lab

Version 1.0 for General Availability

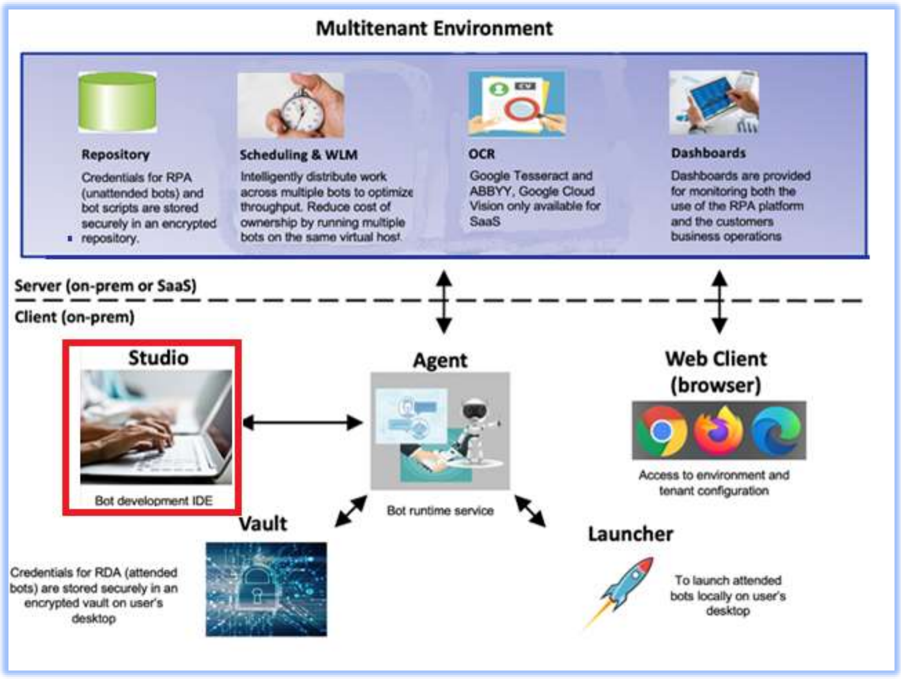
Table of Contents

1 Introduction

Unit tests are essential for building production-quality bots. Unit tests give confidence that the bot behaves as expected and prevents regression. If you are automating bot builds, unit tests are a component of the build toolchain.

This lab will examine unit testing by revisiting the customer refunds bot covered in an earlier lab. We will first build ‘hard coded’ unit tests. Then we will look at extracting tests into a test scenario spreadsheet.

For the context of this lab see the highlighted area below.


Multitenant Environment
Repository
Credentials for RPA (unattended bots) and bot scripts are stored securely in an encrypted repository.
Scheduling & WLM
Intelligently distribute work across multiple bots to optimize throughput. Reduce cost of ownership by running multiple bots on the same virtual host
OCR
Google Tesseract and ABBYY, Google Cloud Vision only available for SaaS
Dashboards
Dashboards are provided for monitoring both the use of the RPA platform and the customers business operations
Server (on-prem or SaaS)
Client (on-prem)
Studio
Bot development IDE
Agent
Bot runtime service
Web Client (browser)
Access to environment and tenant configuration
Vault
Credentials for RDA (attended bots) are stored securely in an encrypted vault on user's desktop
Launcher
To launch attended bots locally on user's desktop

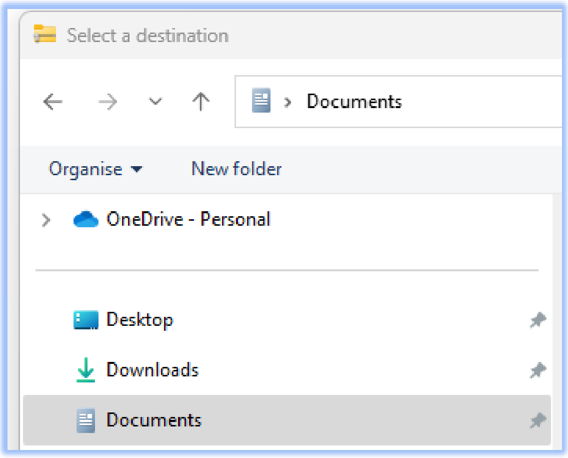
1.1 Prereqisites

RPA 21.0.2.5 or later.

2 Unit Test Lab

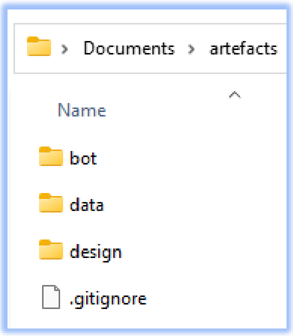
2.1 Setup Lab

Download the *Artefacts*.zip. Extract the zip to the *documents* folder.



This location is important! If you copy to a different folder, the supplied scripts won't run

You should see the contents downloaded:



2.2 Run the Unit Test Template

We will start with a simple unit test to demonstrate principles. Then we will build on these principles to test the refund bot.

Start RPA Studio

If not already started, launch IBM RPA Studio.

Tip. If you are using the IBM lab environment, your credentials are:

Username: admin@ibmdba.com

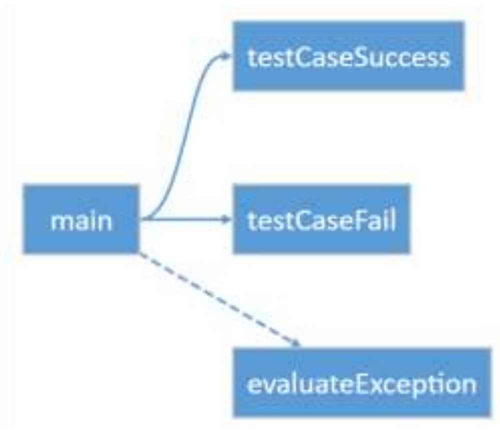
Password: passw0rd

Import Template code

Open the bot script:

```
[Documents]\Artefacts\bot\test_template.wal
```

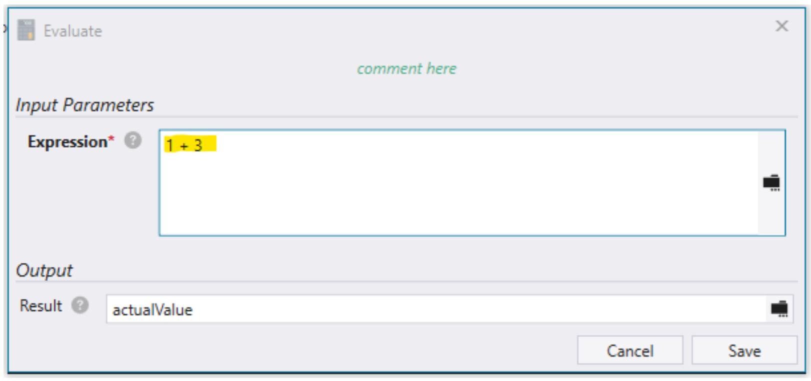
Within RPA Studio, press the call graph tab. You should see the following:



Now hit *Ctrl-F5* to run the bot without debugging (**important**). The output should be:

```
[Info] Starting Unit tests
[Info] Unit tests completed with 0 failures
```

Now edit the evaluate expression on line 31 so that the test case will fail:



Input Parameters

Expression* : 1 + 3

Output

Result : actualValue

Save and rerun. This time you will see the following test failure:

```
[Info] Starting Unit tests

[Info] Failed testCaseFail, Expected value: 3, actual value: 4.
Assert failed at line 34: testCaseFail

[Info] Unit tests completed with 1 failure
```

You can see the test captured the error and gave the reason. It also gave additional information to help diagnose the failure.

2.3 Open the Refunds Script

Let's revisit the refund bot. Open the following script in RPA Studio:

```
Dopcuments\artefacts\bot\Refunds_ProcessSingle.wal
```

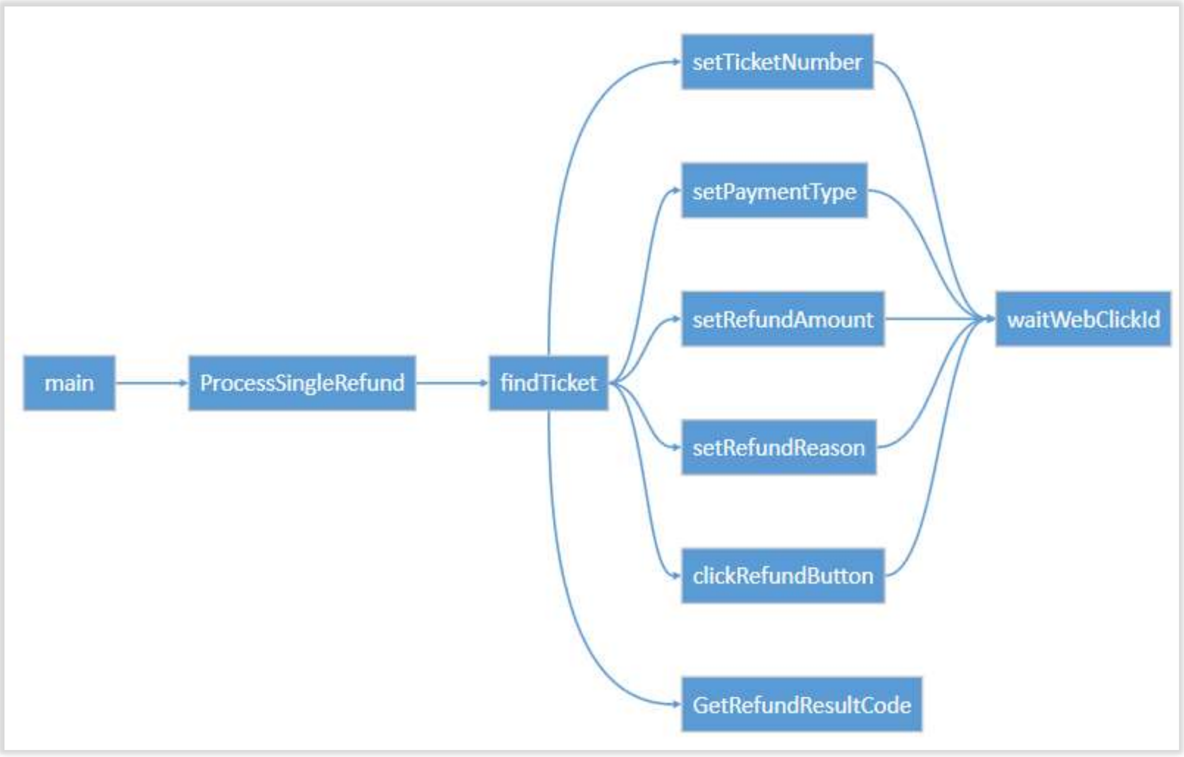
2.4 Run the bot

Press *Ctrl+F5* to run the bot. See how it issues a single refund on a web site, and then returns a status code and description.

2.5 Examine Code

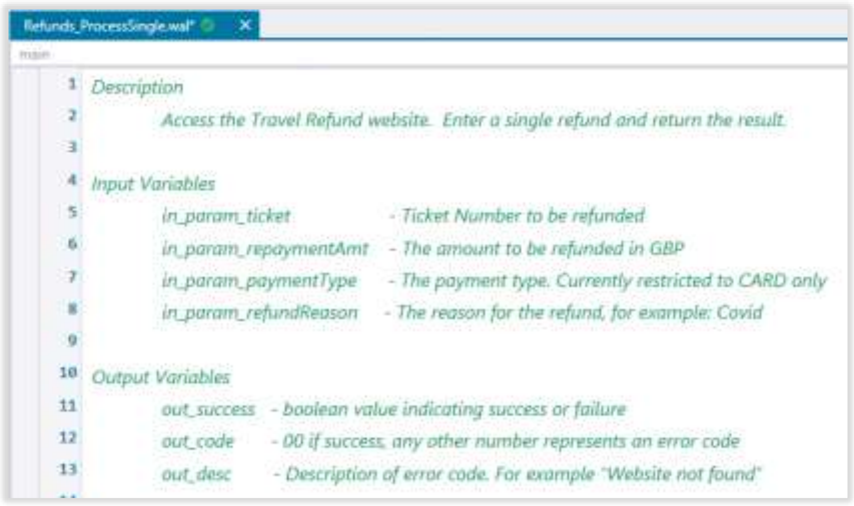
Press the *Call Graph* tab to view the bot structure. The script enters refund information into a web site. It then clicks the *Refund* button and returns the result. The result will be on of:

00 – A successful refund
02 – A Failed refund due to invalid payment type
03 – A failed refund due to a too large refund amount
98 –Processing error (this is randomly generated)
99 –Website error



2.6 Examine Script Parameters

Select the *Designer* tab. Examine the input and output variables**:**



```
Description
        Access the Travel Refund website.  Enter a single refund and return the result.

Input Variables
        in_param_ticket            - Ticket Number to be refunded
        in_param_repaymentAmt      - The amount to be refunded in GBP
        in_param_paymentType       - The payment type. Currently restricted to CARD only
        in_param_refundReason      - The reason for the refund, for example: Covid

Output Variables
        out_success   - boolean value indicating success or failure
        out_code      - 00 if success, any other number represents an error code
        out_desc      - Description of error code. For example "Website not found"
```

The input parameters are:

In_param_ticket – the customer ticket number
In_param_repaymentAmt – the amount to refund. Amounts over 10000 are not valid.
In_param_payment_type – the payment type. Only type *Card* is accepted.
In_param_refundReason – the reason for the refund

The output parameters are:

Out_success – a flag indicating returning whether the script ran successfully
Out_code – A number between **00** and **99**. **00** indicates success, any other number is an error
Out_desc – The description of the error code

2.7 Testing the Refund bot

Now let’s create a test script for the refund bot.

In RPA Studio, open *test_refunds_start.wal*

2.8 Build the Test Cases

We will now create test cases for the following scenarios:

00 – A successful refund
02 – A Failed refund due to invalid payment type
03 – A failed refund due to a too large refund amount

2.9 Successful Refund Test Case

Let’s start with successful refund. Copy & paste the following code directly under the start of the *successfulCardRefund* subroutine (from line 32):

```
    setVar --name "${ticket_number}" --value 567567
    setVar --name "${payment_type}" --value Card
    setVar --name "${payment_value}" --value 87
    setVar --name "${expectedValue}" --value 00
```

2.10 Cash Refund Test Case

Now add a test case for cash refunds. This refund should return error code 02.   Copy & paste the following code directly under the start of the *cashRefundNotAccepted* subroutine:

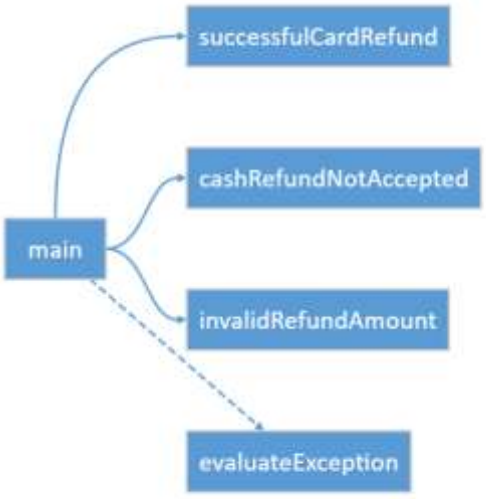
```
    setVar --name "${ticket_number}" --value 453445
    setVar --name "${payment_type}" --value Cash
    setVar --name "${payment_value}" --value 8
    setVar --name "${expectedValue}" --value 02
```

2.11 Invalid Amount Test Case

Now add a test case for an invalid refund amount. In this test case, the refund amount is too large and should return error code 03.   Copy & paste the following code directly under the start of the *invalidRefundAmount* subroutine:

```
    setVar --name "${ticket_number}" --value 087877
    setVar --name "${payment_type}" --value Card
    setVar --name "${payment_value}" --value 870000
    setVar --name "${expectedValue}" --value 03
```

You should now see the following call graph:



2.12 Run the bot

Run the bot without debug (Ctrl+F5)

You should see the following output:

```
[Info] Finding ticket 567567

[Info] Finding ticket 453445

[Info] Finding ticket 087877

[Info] Unit tests completed with 0 failures
```

3 Spreadsheet Driven Testing

In this section we will demonstrate how to test bots with spreadsheets.

3.1 Why test with spreadsheets

The tests built in the previous section were 'hard coded'. In other words, they were coded directly into the bot. It is more agile to define test scenarios in a spreadsheet. The bot runs against spreadsheet data and results are written back to the spreadsheet.

Use test spreadsheets when:

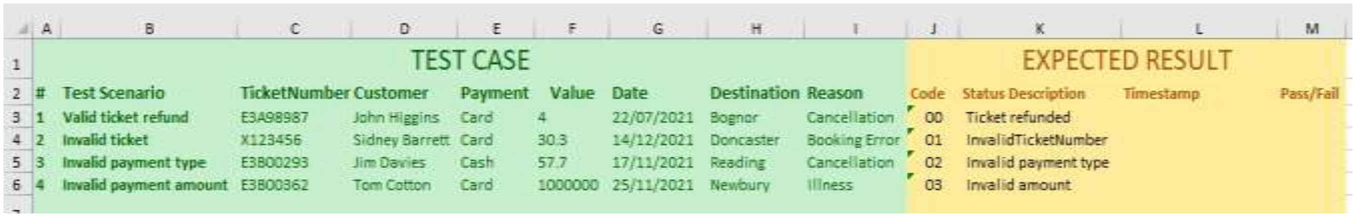
- Business users need to create tests.
- There are many tests
- Tests are changed frequently

3.2 Open the test scenario

Using RPA Studio, open the test scenario:

`[Documents]`\artefacts\data\refundTestScenarios.xlsx

You should see the following:



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	TEST CASE									EXPECTED RESULT			
2	#	Test Scenario	TicketNumber	Customer	Payment	Value	Date	Destination	Reason	Code	Status Description	Timestamp	Pass/Fail
3	1	Valid ticket refund	E3A98987	John Higgins	Card	4	22/07/2021	Bognor	Cancellation	00	Ticket refunded		
4	2	Invalid ticket	X123456	Sidney Barrett	Card	30.3	14/12/2021	Doncaster	Booking Error	01	InvalidTicketNumber		
5	3	Invalid payment type	E3B00293	Jim Davies	Cash	57.7	17/11/2021	Reading	Cancellation	02	Invalid payment type		
6	4	Invalid payment amount	E3B00362	Tom Cotton	Card	1000000	25/11/2021	Newbury	Illness	03	Invalid amount		

The green area defines the test scenarios. The yellow area contains the expected results. In this case, the expected result attributes are status code and status description.

Close the spreadsheet without making changes.

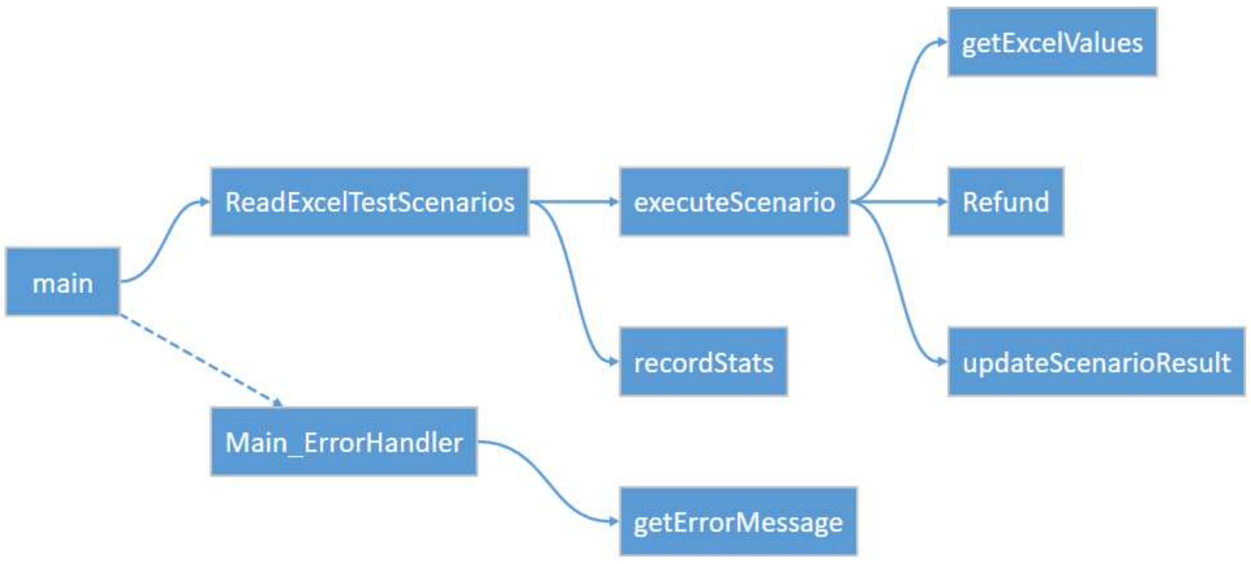
3.3 Open the test script

Within RPA Studio, open:

[Documents]\artefacts\bot\refundTestScenarios.wal

3.4 Examine the script flow

Go to the Call Graph tab. You should see the following:



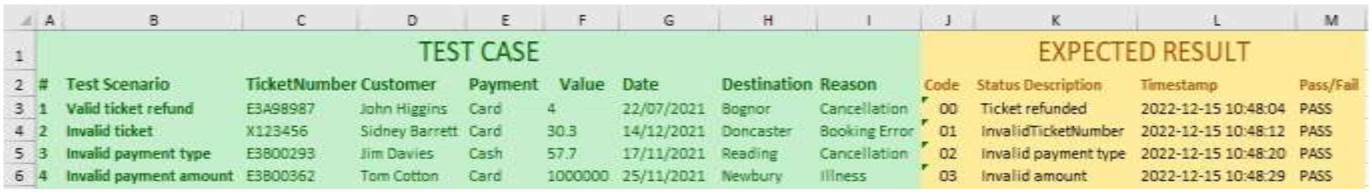
You can see the excel scenarios are read, each scenario executed, and the results written back to excel.

3.5 Run the bot script

Hit *Ctrl+F5* to run the bot without debugging. After a while you should see the Refund website pop up four times, once for each test scenario defined in the spreadsheet. When finished, you should see the following in the console.

```
12/15/2022 3:22:56 PM - [Info] Refunds Unit Tests Started

12/15/2022 3:22:56 PM - [Info] Excel Path - ...\artefacts\data\refundTestScenarios.xlsx

12/15/2022 3:22:56 PM - [Info] Begin tests

12/15/2022 3:22:56 PM - [Info]
Ticket Number - E3A98987
Status -
Status Code - 00
Payment Type - Card
Payment Value - 4


12/15/2022 3:22:59 PM - [Info] Finding ticket E3A98987

12/15/2022 3:23:06 PM - [Info]
Ticket Number - X123456
Status -
Status Code - 01
Payment Type - Card
Payment Value - 30.3


12/15/2022 3:23:08 PM - [Info] Finding ticket X123456

12/15/2022 3:23:15 PM - [Info]
Ticket Number - E3B00293
Status -
Status Code - 02
Payment Type - Cash
Payment Value - 57.7


12/15/2022 3:23:17 PM - [Info] Finding ticket E3B00293

12/15/2022 3:23:24 PM - [Info]
Ticket Number - E3B00362
Status -
Status Code - 03
Payment Type - Card
Payment Value - 1000000


12/15/2022 3:23:25 PM - [Info] Finding ticket E3B00362

12/15/2022 3:23:32 PM - [Info] Robot Refunds Unit Tests ran at host XXXP52
Processed File: …\artefacts\data\refundTestScenarios.xlsx
Total items processed - 4
Robot took 0 Minute(s) 36 Second(s) 605 Millisecond(s)
```

Now re-open the test scenario spreadsheet. You should see each scenario has changed to reflect the result of the test:



TEST CASE									EXPECTED RESULT			
#	Test Scenario	TicketNumber	Customer	Payment	Value	Date	Destination	Reason	Code	Status Description	Timestamp	Pass/Fail
1	Valid ticket refund	E3A98987	John Higgins	Card	4	22/07/2021	Bognor	Cancellation	00	Ticket refunded	2022-12-15 10:48:04	PASS
2	Invalid ticket	X123456	Sidney Barrett	Card	30.3	14/12/2021	Doncaster	Booking Error	01	InvalidTicketNumber	2022-12-15 10:48:12	PASS
3	Invalid payment type	E3B00293	Jim Davies	Cash	57.7	17/11/2021	Reading	Cancellation	02	Invalid payment type	2022-12-15 10:48:20	PASS
4	Invalid payment amount	E3B00362	Tom Cotton	Card	1000000	25/11/2021	Newbury	Illness	03	Invalid amount	2022-12-15 10:48:29	PASS

Verify that all tests passed.

4 Conclusion

In this lab we presented two methods for unit testing bots. The first used the *Assert* method to unit test inside bot scripts. The second method extracted tests to a spreadsheet for more flexible testing.

Use the first method if you want to keep your test suite simple and have just a handful of tests. Use the spreadsheet if you have many test scenarios that may need to be maintained by the business.

Nicely done! This concludes the lab.