**Lab Guide**

Bot Design Lab Guide

Nigel T. Crowther

ncrowther@uk.ibm.com

Hands-on Lab

Version 1.0 for General Availability



NOTICES

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive, MD-NC119

Armonk, NY 10504-1785

United States of America

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

**TRADEMARKS**

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a Registered Trade Mark of AXELOS Limited.

ITIL is a Registered Trade Mark of AXELOS Limited.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

© Copyright International Business Machines Corporation 2020.

This document may not be reproduced in whole or in part without the prior written permission of IBM.

US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

**Table of Contents**

[1 Introduction 4](#_Toc122008909)

[1.1 Prereqisites 4](#_Toc122008910)

[2 Unit Test Lab 5](#_Toc122008911)

[2.1 Setup Lab 5](#_Toc122008912)

[2.2 Open the Refunds Script 5](#_Toc122008913)

[2.3 Examine Code 6](#_Toc122008914)

[2.4 Examine Script Parameters 7](#_Toc122008915)

[2.5 Run the Template Unit Test 7](#_Toc122008916)

[2.6 Modifying the Template 9](#_Toc122008917)

[2.7 Build the Test Cases 10](#_Toc122008918)

[2.8 Run the bot 11](#_Toc122008919)

[3 Spreadsheet Driven Testing 12](#_Toc122008920)

[3.1 Why test with spreadsheets 12](#_Toc122008921)

[3.2 Open the test scenario 12](#_Toc122008922)

[3.3 Open the test script 12](#_Toc122008923)

[3.4 Run the bot script 13](#_Toc122008924)

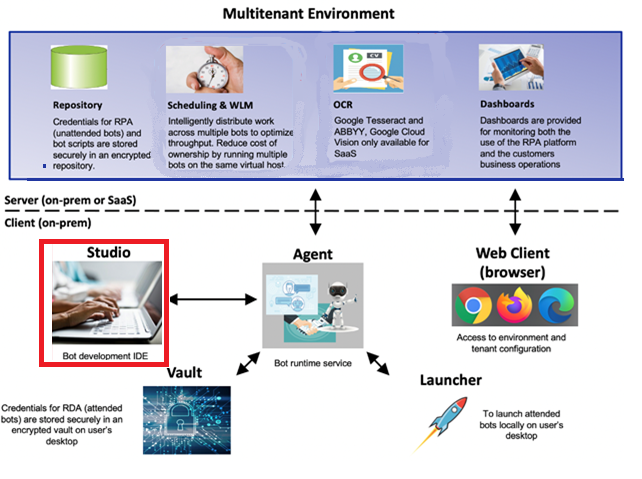
[4 Conclusion 14](#_Toc122008925)

# Introduction

Unit tests are an essential part of building a bot. Without unit tests, the developer cannot be certain that changes have not broken the bot. Unit tests give confidence the bot behaves as expected and prevents regression errors. Unit tests are also an essential part of any automated build tool chain.

This lab will examine the unit testing capabilities of IBM RPA by revisiting the customer refunds bot. We will build unit tests to check the main functions of the bot.

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



## Prereqisites

RPA 21.0.3 or later installed with Studio.

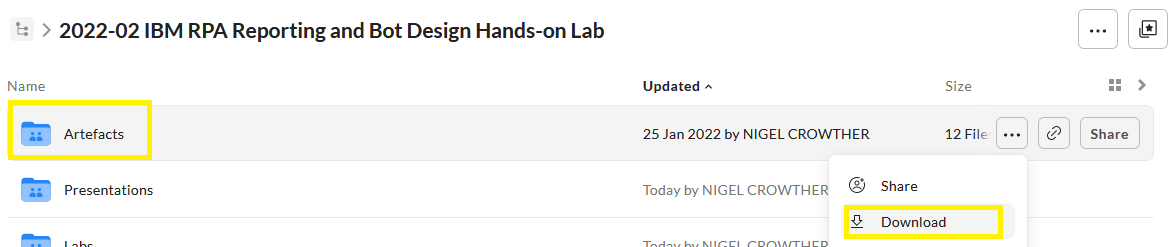
# Unit Test Lab

## Setup Lab

Go to the repo:

<https://ibm.ent.box.com/folder/154727284550?v=2022-02RPAReportingBotDesign>

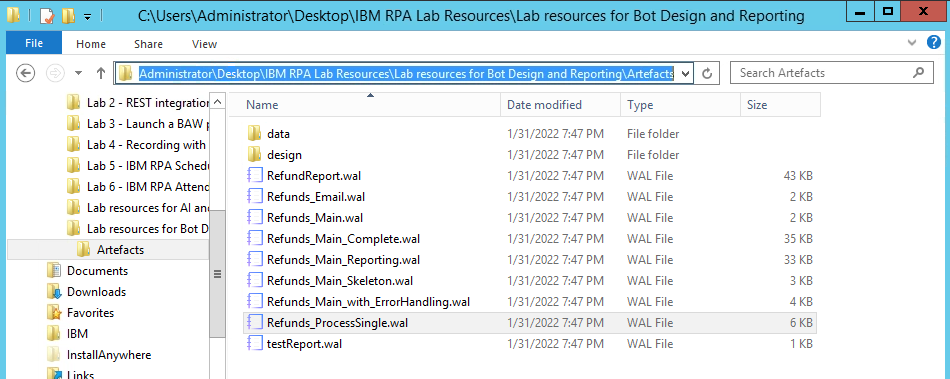
Download the *Artefacts* folder as a zip. See below:



The zip must be extracted to the following location:

C:\Users\Administrator\Desktop\IBM RPA Lab Resources\Lab resources for Bot Design and Reporting

**This location is important!** If you copy to a different folder, you will experience problems. You should see the contents downloaded to the following folder:



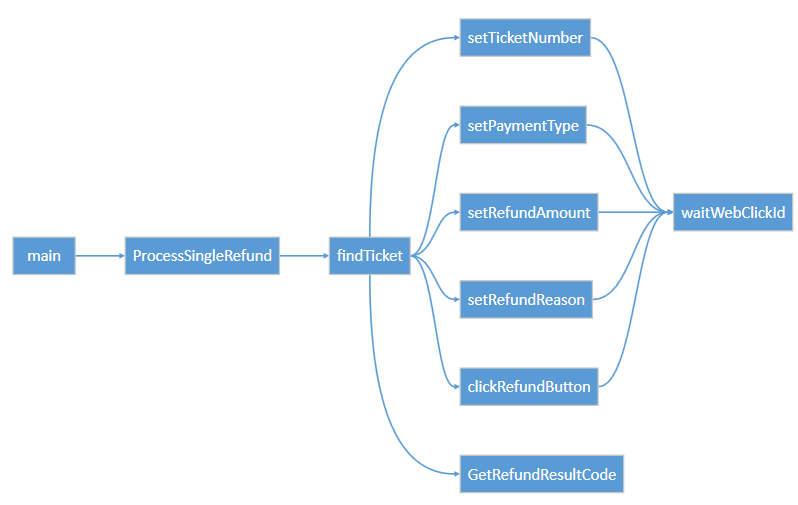
## Open the Refunds Script

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

[bot\_design\_lab]\Artefacts\Refunds\_ProcessSingle.wal

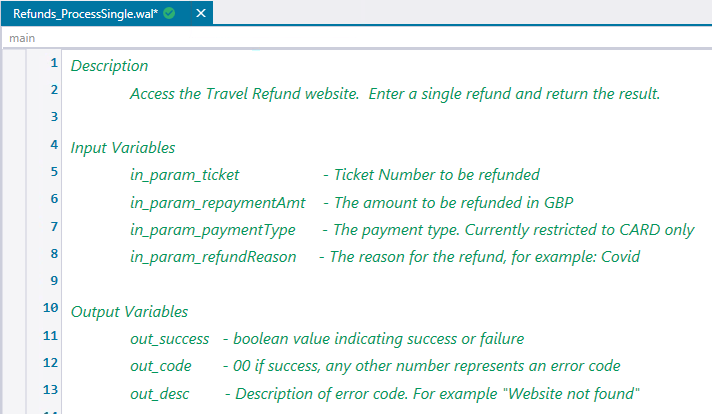
## Examine Code

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.



## Examine Script Parameters

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



The script has four input parameters. These 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 script contains three output parameters. These parameters are generic and not specific to refunds. These 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

## Run the Template Unit Test

When building bot unit tests, it is good practice to start with a working Template framework. Once this structure is in place, you can build out the bot tests.

**Start RPA Studio**

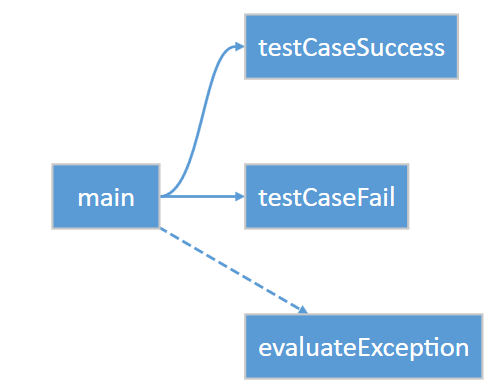
If not already started, launch IBM RPA Studio.

**Import Template code**

Open the bot script:

[bot\_design\_lab]\Artefacts\test\_template.wal

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

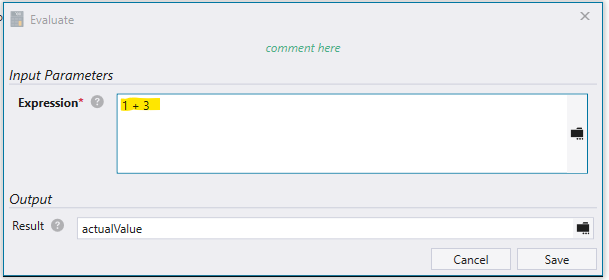


Now hit *Ctrl-F5* to run the bot without debugging (**important to run with no debug**). The output should be:

[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:



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

12/13/2022 3:09:42 PM - [Info] Starting Unit tests

12/13/2022 3:09:42 PM - [Info] Failed testCaseFail, Expected value: 3, actual value: 4.

Assert failed at line 34: testCaseFail

12/13/2022 3:09:43 PM - [Info] Unit tests completed with 1 failure

You can see the test captured the error and gave the reason for failure (the result should be 3 not 4). It also gave additional information to help you track down which test case failed.

## Modifying the Template

Now modify this bot script so that it tests our refund bot.

Save the *test\_skelton,wal* with a new name of *test\_refunds.wal*

At the top of the script, replace the existing variables with the following:

// Description  
//    Test cases for the Travel Refund website.    
//  
**defVar** --name ticket\_number --type String  
**defVar** --name payment\_type --type String  
**defVar** --name payment\_value --type String  
**defVar** --name out\_success --type String  
**defVar** --name out\_code --type String  
**defVar** --name out\_desc --type String  
**defVar** --name failureCount --type Numeric  
**defVar** --name testCaseName --type String  
**defVar** --name expectedValue --type String  
**defVar** --name actualValue --type String  
**defVar** --name botScriptPath --type String --value "[YOUR\_BOT\_SCRIPT\_PATH]\\artefacts\\bot"

## Build the Test Cases

Change the test cases so that they test 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

Start with the successful refund test case. Set three input parameters:

* ticket number: 567567
* payment type: Card
* payment value: 87

Execute the refund bot and check for return status 00 (success).

You can ‘cheat’ by copying & pasting the following code directly under the testcase (line 18):

**goSub** --label successfulCardRefund

**beginSub** --name successfulCardRefund  
      
**setVar** --name "${ticket\_number}" --value 567567  
**setVar** --name "${payment\_type}" --value Card  
**setVar** --name "${payment\_value}" --value 87  
      
**executeScript** --name "${botScriptPath}\\Refunds\_ProcessSingle.wal" --parameters "in\_param\_ticket=${ticket\_number},in\_param\_paymentType=${payment\_type},in\_param\_repaymentAmt=${payment\_value}" --output "out\_success=${out\_success},out\_code=${out\_code},out\_desc=${out\_desc}"  
      
**setVar** --name "${expectedValue}" --value 00  
**setVar** --name "${actualValue}" --value "${out\_code}"  
      
**assert** --message "${rpa:subName}" --left "${out\_code}" --operator "Equal\_To" --right "${expectedValue}"  
      
**endSub**

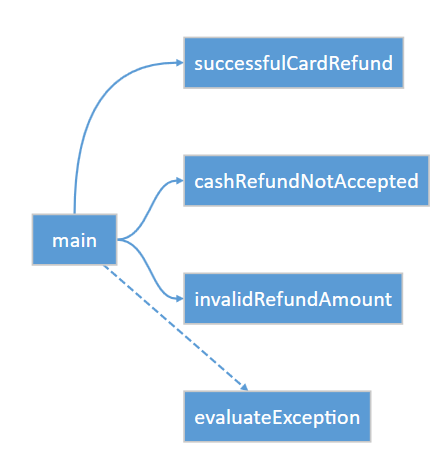
Create a similar test case called *cashRefundNotAccepted* which expects error code 02 from the following parameters:

**setVar** --name "${ticket\_number}" --value 453445  
**setVar** --name "${payment\_type}" --value Cash  
**setVar** --name "${payment\_value}" --value 8

And *invalidRefundAmount* which expects error code 03 from the following parameters:

**setVar** --name "${ticket\_number}" --value 087877  
**setVar** --name "${payment\_type}" --value Card  
**setVar** --name "${payment\_value}" --value 870000  
    

You should now see the following call graph:



## 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

Note that the website deliberately returns random errors to simulate real life, so you may see error code 98.

Compare your test bot to the completed example:

[bot\_design\_lab]\Artefacts\test\_refunds\_complete.wal

How did your bot compare to the completed bot?

# Spreadsheet Driven Testing

## Why test with spreadsheets

The tests built in section two were ‘hard coded’. In other word they were added directly to the bot code. A flexible alternative is to define your test scenarios and expected results in a spreadsheet. The bot read the spreadsheet to get the test data. The tests run and the actual results compared to the expected results. Results are written back to the spreadsheet where they can be verified.

Use this approach when:

* Business users need to create test scenarios.
* There are a lot of tests
* Tests are likely to be added frequently

Don’t use this approach when:

* If you have only one or two test cases
* The tests are technical

## Open the test scenario

Using RPA Studio, open the test scenario:

[YOUR\_PROJECT\_FOLDER]\refundBot\artefacts\data\refundsTestScenarios.xlsx

You should see the following:



The section in green is the test case which contains the test scenarios. The section in yellow contains the expected result. In the refunds test case, the expected result is a code and a status description.

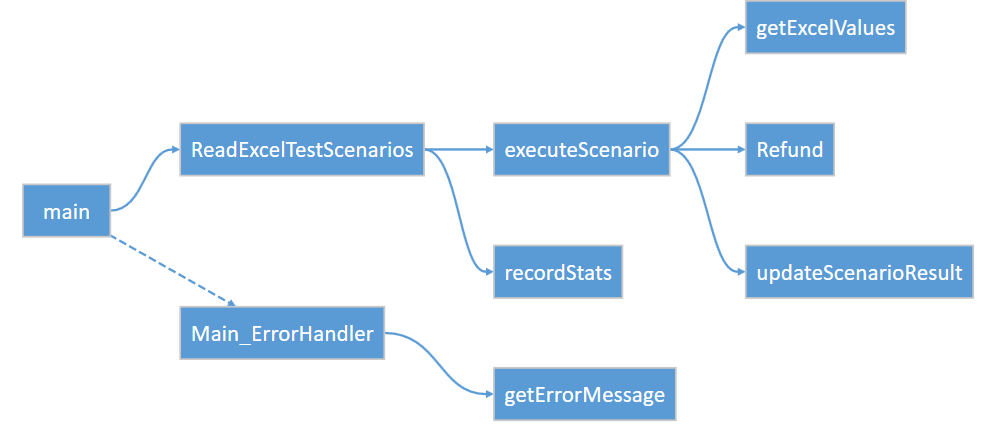
## Open the test script

Within RPA Studio, open:

[YOUR\_PROJECT\_FOLDER]\refundBot\artefacts\bot\UnitTestRefunds.xlsx

Examine the script flow

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



You can see that the excel scenario file is read, and for each scenario defined the refund is processed and the results updated back to excel.

## 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.

Now re-open the test scenario spreadsheet. You should see each scenario has been modified to reflect the result of the test and the time at which the test was run:



# Conclusion

In this lab we presented to ways to test your bots. The first way used the Assert method to build tests directly into bot code. The second way used a spreadsheet to define test scenarios. These scenarios were processed by the bot and the results written back to the spreadsheet. Use the first way if you have a few simple tests, use the spreadsheet way if you have scenarios that need to be specified and maintained by business users.

Nicely done! This concludes the lab.