**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](#_Toc94550497)

[1.1 What is good design? 4](#_Toc94550498)

[1.2 Prerequisites 4](#_Toc94550499)

[1.3 Scenario Description 5](#_Toc94550500)

[2 Best Practices 6](#_Toc94550501)

[2.1 Best Practice – Start with a Design 6](#_Toc94550502)

[2.2 Best Practice – Build an Initial Framework 11](#_Toc94550503)

[2.3 Best Practice – Use Error Handlers 12](#_Toc94550504)

[2.4 Best Practice – Separate Activities into Separate Scripts 14](#_Toc94550505)

[2.5 Best Practice – Recover from non-fatal errors 16](#_Toc94550506)

[2.6 Best Practice – Use Dashboard Counters 19](#_Toc94550507)

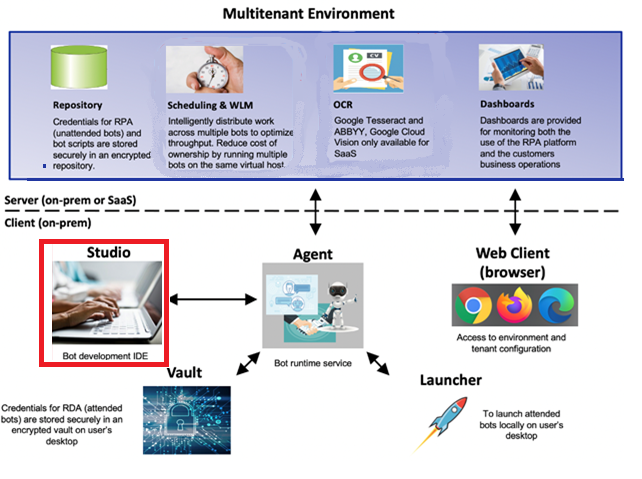
[3 Advanced Exercise 21](#_Toc94550508)

# Introduction

This lab will examine bot design and best practices.

We first design a bot and then build it. Each lab will enhance its successor until we have a bot that implements all best practices.

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



## What is good design?

The difference between a well-designed bot and a bad one is maintainability. A well-designed bot is understandable by others. A bad bot design is the opposite. It is hard to understand by others and maybe even by the author. This makes it hard to maintain and results in a short shelf life.

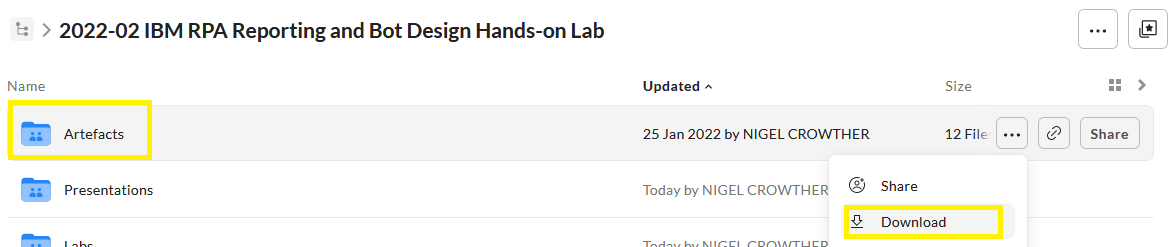
## Prerequisites

RPA 21.0.2 or later installed with Studio.

**IMPORTANT NOTE FOR SKYTAP USERS:** Within your VM, open Firefox, navigate to the lab folder:

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

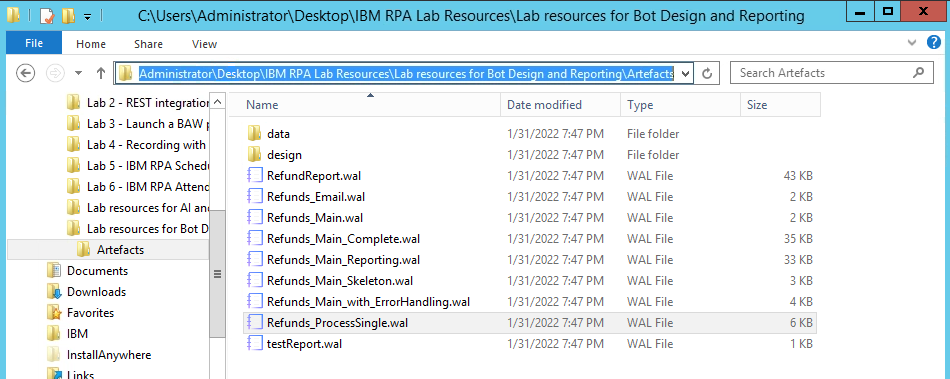
Download the *Artefacts* folder as a zip. See below:



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:



## Scenario Description

You are an RPA developer working for a successful travel company. You have been approached by the customer service team. The team is working night and day cancelling customers after a pandemic. The team keep a list of customers in a spreadsheet. They manually refund each customer by entering details into a web site. Once the customer has been refunded, they update the spreadsheet and send a remittance to the customer. There is no API into the refund web site, and it randomly fails. Your task is to design and build a production-ready bot that refunds customers.

# Best Practices

In this lab we will build a bot using the best practices found in the IBM RPA guidelines:

<https://www.ibm.com/docs/en/rpa/21.0?topic=development-script-guidelines>

## Best Practice – Start with a Design

IBM RPA Studio has a built-in **workflow editor** to design and create bots. With this tool you can create diagrams that help structure your bot and share your understanding to team members.

Note that the **workflow editor** can be used to build executable workflows. We will *not* be using this feature in this lab

**Start RPA Studio**

On your desktop find the IBM RPA Studio icon and launch it.

A picture containing text

Description automatically generated

**Log In**

Login to RPA Studio with your credentials.

**Tip:** If you are using SkyTap, the credentials are:

User: *admin@ibmdba.com*

Password: *passw0rd*

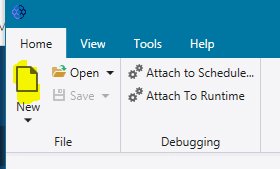
Graphical user interface, application

Description automatically generated

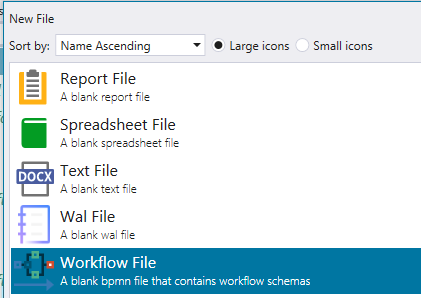
**Create a Workflow**

Once in RPA studio, select the *New* icon.

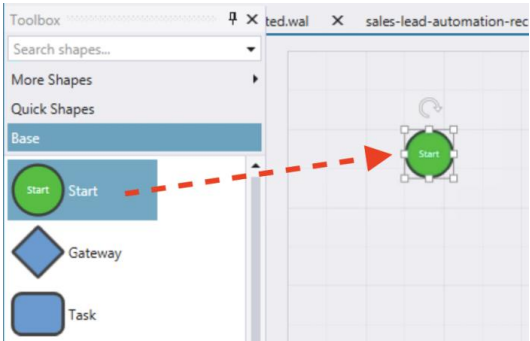
***Tip:*** *Select the upper part of the icon inside the blank page.* See below:



Select *Workflow File* and press *Create*:



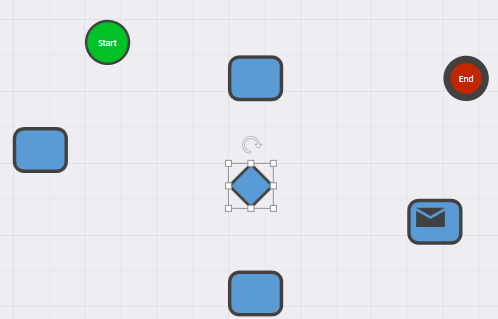
Drag and drop a *Start* element to your empty canvas.



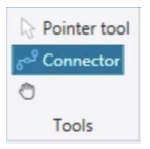
Add the following components to the canvas:

* Three *Empty* *Task* elements
* One *Send* *Task*
* One *Gateway*
* One *End*

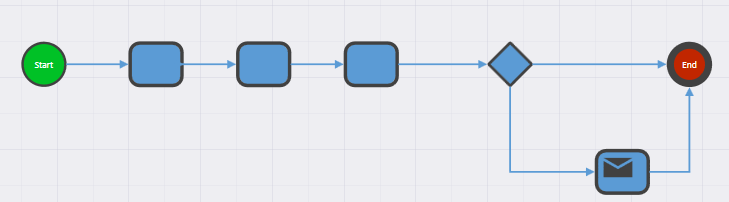
You should now have something like this:



Click the Connector from the top toolbar to select the connector tool.



Reorder the elements and then connect them to achieve the flow presented below.

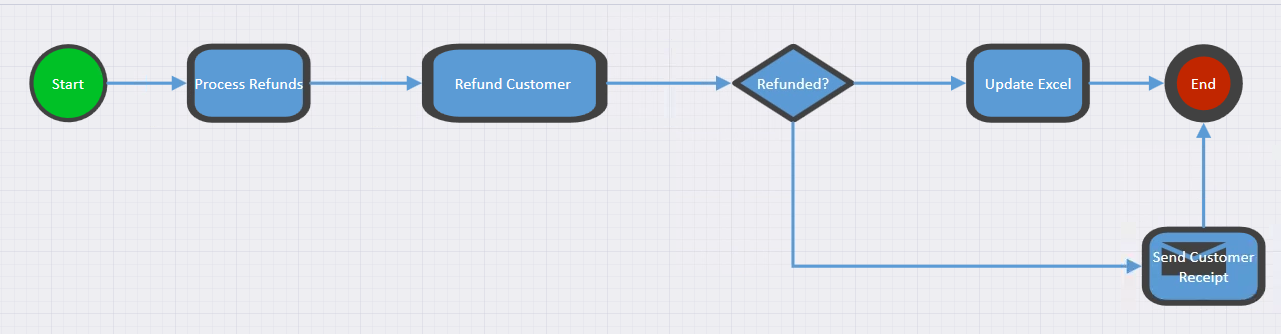


Deselect the Connector tool by selecting the *Pointer* tool just above the *Connector* in the *Tools* section of the top toolbar.

Next, we want to name our elements. For each element, right-click and select *Edit text.* Type the following names

* Process Refunds
* Refund Customer
* Refunded?
* Update Excel
* Send Customer Receipt

Once done, you should have the design below:



Now name your workflow. Click *Save* and then save as *refundBotDesign.bpmn*

In the next section we will build the bot based on this design.

**Lesson learnt**

Before you dive into build, consider creating a design first. This helps you understand the problem, structure your bot, and communicate your understanding to the stakeholders.

## Best Practice – Build an Initial Framework

When building a bot, it is good practice to start with a framework. By framework we mean the subroutines, conditional branching, and sub-scripts within the bot. Once this structure is in place, you can build out the bot code.

**Start RPA Studio**

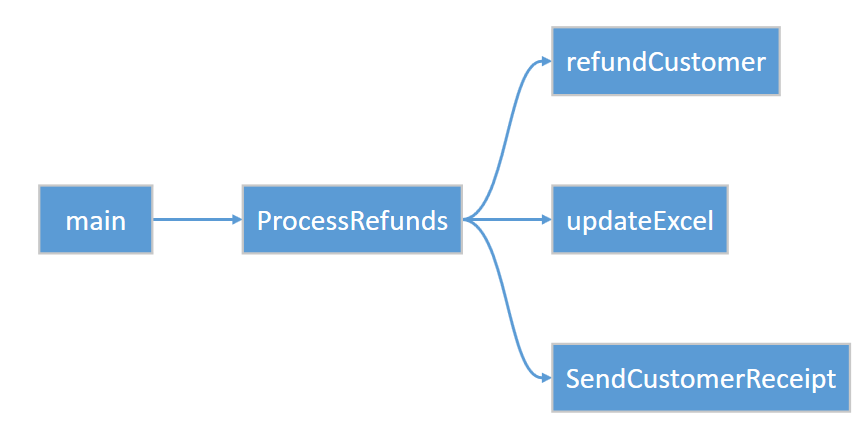
If not already started, launch IBM RPA Studio.

**Import skeleton code**

Open the bot script:

[bot\_design\_lab]\Artefacts\Refunds\_Main\_skeleton.wal

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



Compare the call graph to the BPMN bot design created in the previous exercise. The BPMN tasks have become subroutines. The decision point is implemented as an *if/then* condition in the *processRefunds* subroutine on line 40. Verify this in RPA Studio.

Now hit *Ctrl-F5* to run the bot. The log output should be:

[Info] Starting refunds. Read excel file containing customers to be refunded

[Info] Excel Path …\refunds.xlsx

[Info] Refund individual customer using refund website

[Info] Update excel with the result of the refund operation

[Info] If refund was successful, send the customer a refund receipt

**Note:** If the script fails, make sure the project path is set correctly. See prerequisites at the start of this lab.

**Lesson learnt**

Start by building a framework to get the structure right. This helps readability and maintainability of your bot.

## Best Practice – Use Error Handlers

Bots should have error handling. There are many reasons why a bot can fail, and when problems occur, administrators need to resolve them quickly. In this lab, we touch upon error handling practices that you should consider.

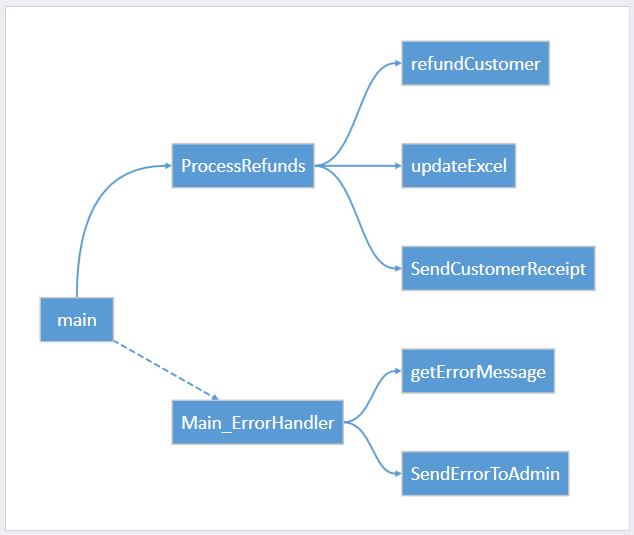
**Tip**: We use just one error handler in this example, but you could have an error handler for each subroutine to handle errors differently in each part of your script.

**Examine Error Handler Routine**

Open the following script in RPA Studio:

[bot\_design\_lab]\Refunds\_Main\_with\_ErrorHandling.wal

Examine the call graph. Note the error handler:



Double-click the routines *Main\_ErrorHandler* and *getErrorMessage* and *SendErrorToAdmin* to view the source code.

**Test Error Handling**

Test the error handler. Hit *Ctrl-F5* to run the bot without debug.

**IMPORTANT**: Do not run with debug (*F5*). If you do, the exception handler will not run.

The log output should be as follows:

Windows Host: IBMBAW

Error Message: Error reading excel

Subroutine: ProcessRefunds

Line Number: 72

[Info] ------------------------------------------------------------------------

Error executing command at line 31: failTest --message "Error reading excel"

The error handler has caught the exception and created a message containing:

* Machine name
* Bot name
* Error message
* Line number of the error

In a real-world implementation, this message could be emailed to an administrator to resolve.

**Task:** Using the error information, fix the problem.

**Hint:** The project path for the excel file does not exist on this machine. View the source of the error and work backwards to fix the path.

**Lesson learnt**

Adding error handling to your bot saves time finding problems, both in development and in production.

## Best Practice – Separate Activities into Separate Scripts

If your bot performs distinct activities, it is good practice to split the bot into smaller scripts. This facilitates delegation of development between several developers and also allows for script re-use. Lastly, each script will be fully tested before it is integrated with the main script, leading to a more resilient and maintainable bot.

In our Refund bot, we are going to separate the web site interaction into a separate script. This script will be invoked from the main script.

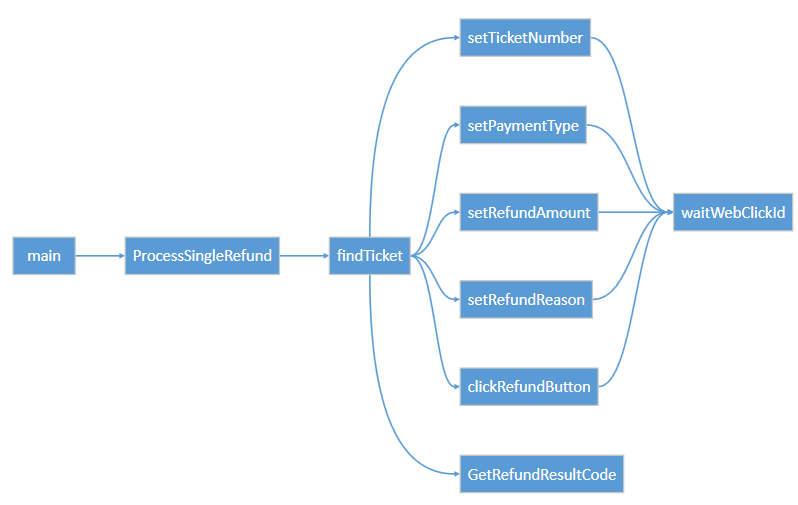
**Open the Web Interaction Script**

Open the following script in RPA Studio:

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

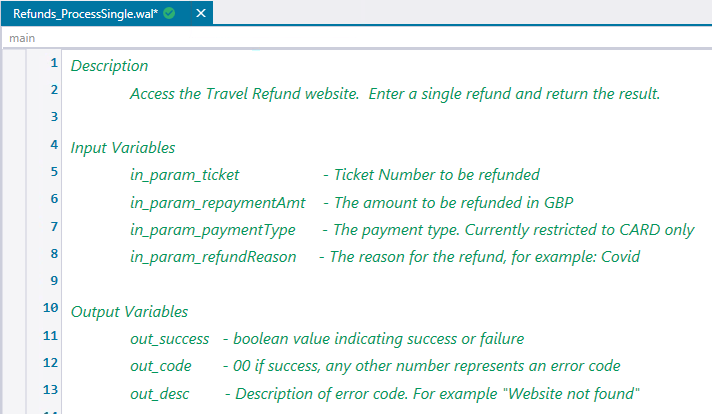
**Examine Code**

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. The bot uses the function **webWaitElement**whichuses a time out rather than a delay to wait for HTML elements. Compare this to using the **Delay** function, which will always pause even if the element is available immediately.



**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

**In\_param\_payment\_type** – the payment type. Only type *Card* is accepted.

**In\_param\_refundReason** – the reason for the refund

The values of the input parameters are defined with dummy test data. This allows us to test the script. When you come to call this script from another script, these parameters are overridden.

The script contains three output parameters. These parameters are generic and not specific to refunds. These are:

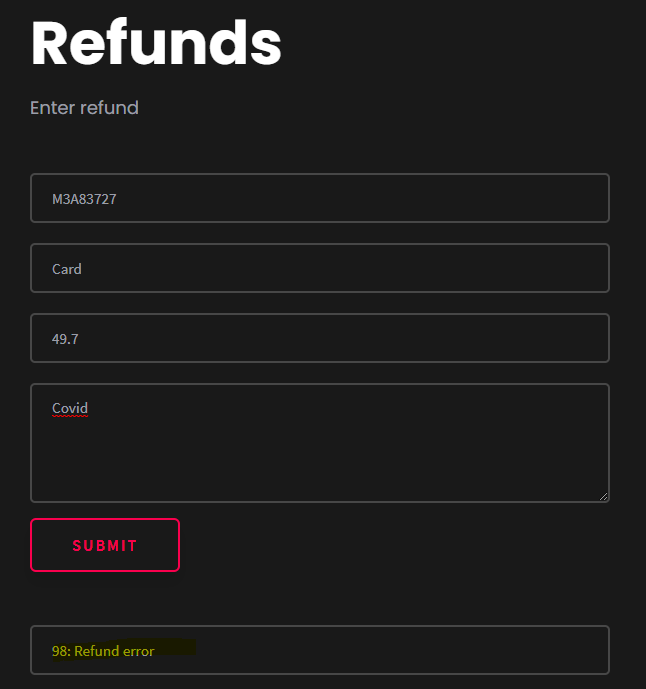
**Out\_success** – a flag indicating returning whether the script ran successfully

**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 script**

Run the bot with *Ctrl+F5*. You should see the following in the Refund web site:



**Lesson learnt**

Separating your bot into smaller scripts allows for script re-use and independent testing of scripts.

## Best Practice – Recover from non-fatal errors

The error code **98** was returned when running the refund bot above. This error simulates the real world where web sites don’t always work as expected. Error code **98** is a recoverable error - it is an intermittent problem with the web site. It is good practice to make a distinction between recoverable errors and non-recoverable errors such as a missing file. Recoverable errors should be retried, whereas non-recoverable errors should throw an exception and stop.

In this section we will show how to handle recoverable exceptions and retry them.

**Examine the completed script**

Open the following script in RPA Studio:

[bot\_design\_lab]\Artefacts\Refunds\_Main\_Complete.wal

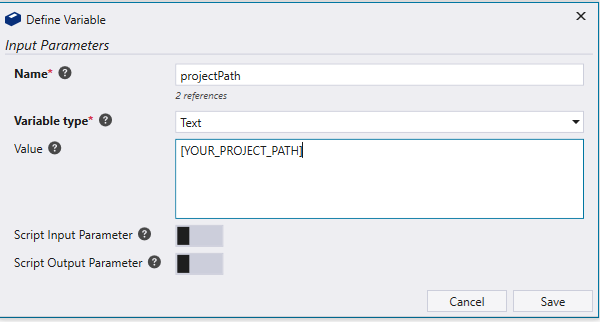
This script contains the complete implementation of the refunds bot. Take a while to examine the implementation. Note how each row in the excel file is read and then passed to Refunds\_ProcessingSingle.wal using the *ExecuteScript* command. See below:



The response is written back to the excel file. If the customer refund status code is **98** or **99** then the bot will retry the customer on a subsequent run.

**Set project path**

In RPA Studio, edit the *projectPath* variable to match your project folder (if not already):



Hit *Ctrl-F5* to run the bot. You should see each customer is read, the web site starts, and the customer refund entered to the website. The script returns with a status code and excel is updated with the result. All eleven rows in the spreadsheet are processed. On completion, the log output should look like this:

[Info] Robot IBM RPA Refunds Robot ran at host NIGELSP52

Processed File: ..\refunds.xlsx

Total items processed - 10

Refunds Issued - 5

Refunds not issued - 5

Robot took 1 Minute(s) 41 Second(s) 219 Millisecond(s)

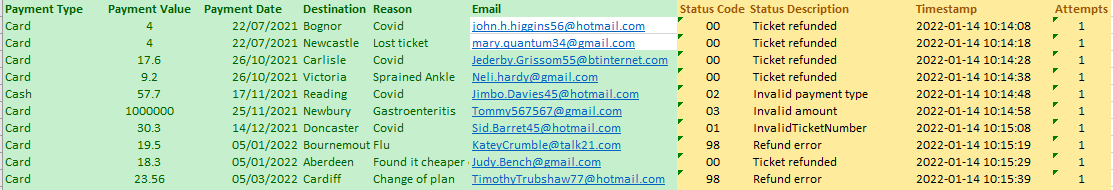
**Examine Excel Data**

Open the excel file containing the refunds:

[PROJECT\_FOLDER]/Artefacts/Data/refunds.xls

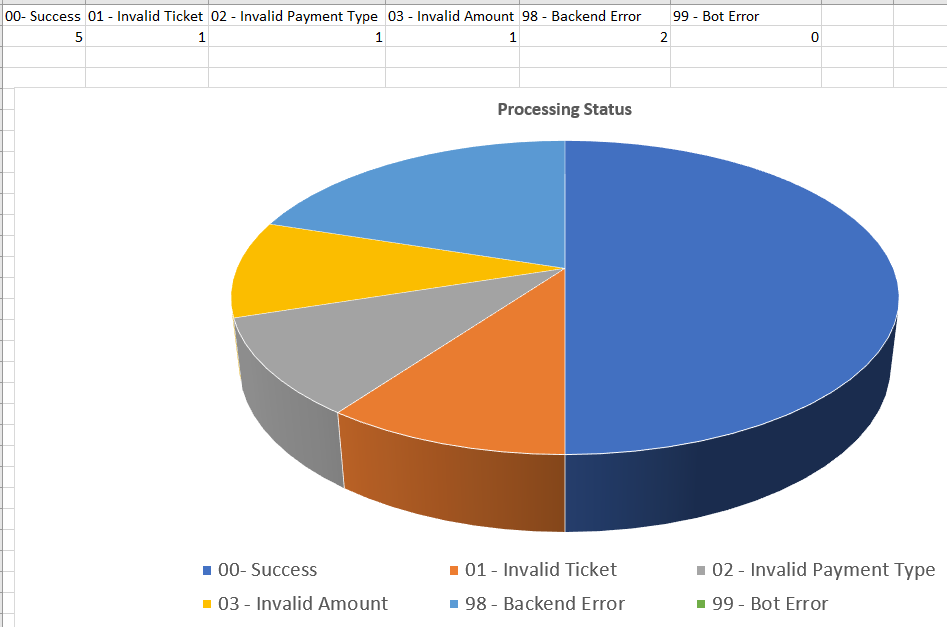
**Tip:** You can open Excel files directly in RPA Studio.

You should see the following:



**Note:** Your response data may be different as the website generates random errors.

Still in Excel, click on the *Processing Status* tab. You should see the following chart:



The chart has two Backend errors (both error code 98). If you run the bot again, these rows will be reattempted until they disappear. The bot will *not* retry rows with non-recoverable errors such as *Invalid Payment Type*.

**Lesson learnt**

Making your bot retry non-fatal errors adds resiliency.

## Best Practice – Use Dashboard Counters

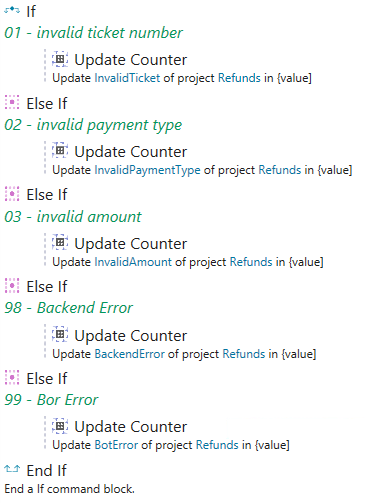
The Dashboard is essential for monitoring the health of your bot. Not only can the Dashboard show status of bots, but it can also show the inner workings of bots in real time. Each time your bot updates its counters they can be displayed in on the dashboard. This allows RPA administrators to monitor the internal state of your bot and react to problems.

**Create counters in the Bot**

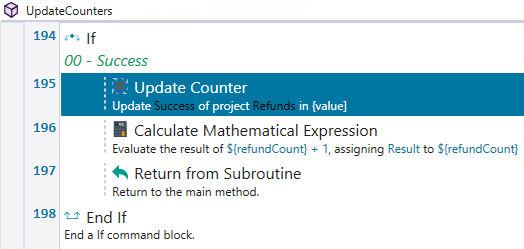
Open the following script in RPA Studio:

[bot\_design\_lab]\Artefacts\Refunds\_Main\_Complete.wal

Navigate to the *updateCounters* subroutine and comment out the *updateCounter* calls. When done you should see the following:



Make sure you uncomment the success counter as well:



This code updates counters in your dashboard so that the refund count can be seen in real time.

There is one crucial step missing. We need to define the counters and the dashboard in the tenant. We will do this in the next guide, Dashboards in IBM RPA. Please refer to this guide to proceed.

**Lesson learnt**

Adding counters allow administrators to see the internal workings of your bot in real time and react to issues quickly.

# Advanced Exercise

Examine the best practices identified in the RPA script guidelines:

<https://www.ibm.com/docs/en/rpa/21.0?topic=development-script-guidelines>

What are the best practices *not* followed in the refund bot? Identify and rectify.

Nicely done! This concludes the lab.