POWER AUTOMATE DESKTOP

SUCCINCTLY®

BY ED FREITAS

SUCCINCTLY® EBOOK SERIES

Syncfusion®

Power Automate Desktop Succinctly

Ed Freitas

Foreword by Daniel Jebaraj

2501 Aerial Center Parkway
Suite 111
Morrisville, NC 27560
USA

ISBN: 978-1-64200-238-6

Technical Reviewer: James McCaffrey

Copy Editor: Courtney Wright

Acquisitions Coordinator: Tres Watkins, VP of content, Syncfusion, Inc.

Proofreader: Jacqueline Bieringer, content producer, Syncfusion, Inc.

Table of Contents

The *Succinctly*® Series of Books

Daniel Jebaraj
CEO of Syncfusion®, Inc.

When we published our first *Succinctly*® series book in 2012, *jQuery Succinctly*, our goal was to produce a series of concise technical books targeted at software developers working primarily on the Microsoft platform. We firmly believed then, as we do now, that most topics of interest can be translated into books that are about 100 pages in length.

We have since published over 200 books that have been downloaded millions of times. Reaching more than 1.7 million readers around the world, we have more than 70 authors who now cover a wider range of topics, such as Blazor, machine learning, and big data.

Each author is carefully chosen from a pool of talented experts who share our vision. The book before you and the others in this series are the result of our authors' tireless work. Within these pages, you will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

We are absolutely thrilled with the enthusiastic reception of our books. We believe the *Succinctly*® series is the largest library of free technical books being actively published today. Truly exciting!

Our goal is to keep the information free and easily available so that anyone with a computing device and internet access can obtain concise information and benefit from it. The books will always be free. Any updates we publish will also be free.

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctlyseries@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on social media and help us spread the word about the *Succinctly*® series!



About the Author

Ed Freitas is a business process automation consultant, technologist, and solutions architect focused on customer success. He likes technology and enjoys learning, playing soccer, running, traveling, and being around his family.

Ed is available at https://edfreitas.me.

Acknowledgments

Thank you to the fantastic Syncfusion® team that helped this book become a reality—especially Jacqueline Bieringer, Tres Watkins, Graham High, and Daniel Jebaraj.

The manuscript managers and technical editor thoroughly reviewed the book's organization, code quality, and overall accuracy—Jacqueline Bieringer, Tres Watkins, Graham High from Syncfusion®, and James McCaffrey from Microsoft Research. Thank you all.

I dedicate this book to A.C., E.R., and J.D. May your journeys be blessed.

Introduction

The recent explosion of information and services has added a significant workload to business users, who face increasing challenges when performing their roles. Part of those challenges involves performing repetitive tasks on a computer, whether updating a CRM application, processing PDF invoices, or following up with customers through email.

To alleviate this burden on business users and focus on tasks that add value to their core business, the field of robotic process automation (RPA) has emerged as a crucial aspect of digital transformation within many organizations.

Power Automate Desktop is an RPA tool developed by Microsoft, part of the broader Power Automate platform, which includes cloud-based services for automating workflows and integrating various applications and services.

Power Automate Desktop specifically focuses on automating repetitive tasks on a user's desktop environment. These tasks may involve interacting with desktop applications, websites, files, and other user interface elements.

Power Automate Desktop provides a user-friendly, drag-and-drop interface that lets users design automation workflows without extensive programming knowledge.

It seamlessly integrates with the Microsoft Power Platform, enabling users to leverage other Power Platform services, such as Power BI and Power Apps, for comprehensive automation and data analysis. It offers a broad range of connectors to interact with various applications and services, both Microsoft and third-party, facilitating integration and automation across diverse systems.

Power Automate Desktop is crucial for automating routine and time-consuming tasks. By automating these tasks, organizations can significantly increase operational efficiency and reduce errors associated with manual processes.

RPA tools like Power Automate Desktop contribute to cost reduction by automating tasks requiring human intervention. This is especially beneficial for repetitive tasks that do not require complex decision-making.

Automation reduces the chance of human errors, leading to more accurate and consistent results—this is particularly important in industries where precision is critical, such as finance or data entry. By offloading repetitive tasks to RPA, business users can focus on more value-added activities, boosting overall productivity and job satisfaction.

Power Automate Desktop can automate data entry tasks and validation processes, reducing manual effort and errors. It can also generate reports by automating data extraction and formatting from various sources and email-related tasks, such as sending alerts, processing emails, and extracting information from emails or files.

Power Automate Desktop interacts with and automates tasks within legacy systems that lack modern APIs and integrates with other Microsoft products, creating a unified automation and data management ecosystem.

Power Automate Desktop is a powerful RPA tool vital in automating desktop-based tasks, contributing to increased efficiency, reduced costs, and improved accuracy in business processes. Its integration with the broader Power Platform and Microsoft ecosystem positions it as a versatile solution for organizations seeking comprehensive automation capabilities.

Throughout this book, we'll explore how Power Automate Desktop (the free version) can be used to automate some tedious business processes. By doing that, we'll learn how to leverage some of its essential features and automate repetitive and dull manual processes.

GitHub repo

You can download the workflows for this book from the GitHub repo.

> ***Tip: If you want to save time, avoid mistakes, and follow along more quickly (and don't want to recreate each workflow manually), please check the Appendix section at the end of the book. This way, you can see how to import the finished and ready-to-use workflows easily from the GitHub repo into Power Automate Desktop—this will save you a lot of time.***

Chapter 1 Getting Started

Quick intro

Getting started with Power Automate Desktop is straightforward. To begin, you'll need to have either Windows 10 or 11.

We need to install the Power Automate Desktop application, and to do that, we need to either use the MSI installer or get the app from the Microsoft Store. In my case, I'll do it using the MSI installer. I suggest you do the same so you can follow along easily. Once downloaded, we can click on the file to install the application.

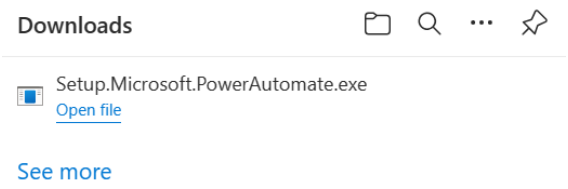

Figure 1-a: The Power Automate Desktop MSI Installer

Once you execute the installer, typically, the installation process runs in the background. When it's done, you should see something similar to the following screen.

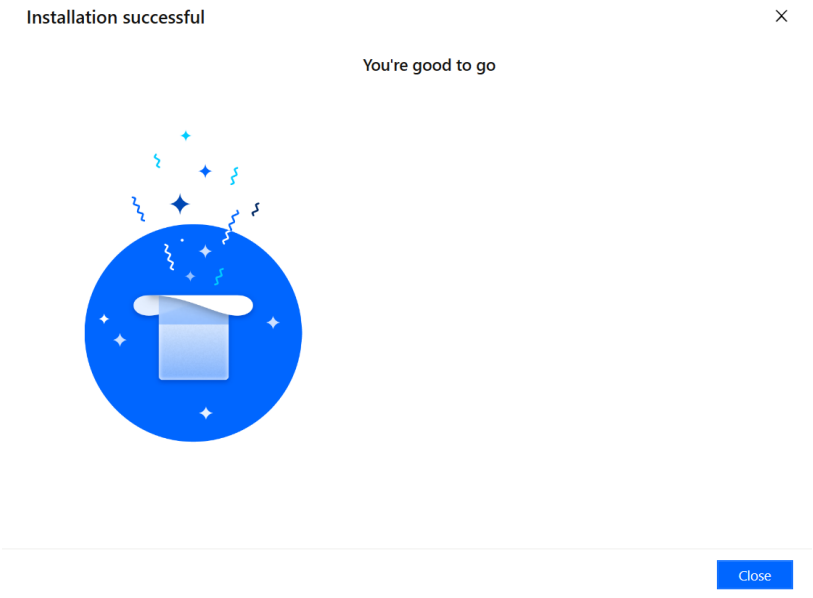

Figure 1-b: The Power Automate Desktop Installation Process Finished

That was easy! Let's run the app by going to the Windows menu and clicking the **Power Automate** app menu option.

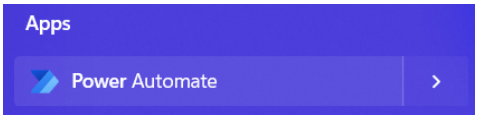

Figure 1-c: The Power Automate Desktop App Option (Windows 11 Menu)

When launching Power Automate Desktop, you must sign in and set default parameters such as your country or region.

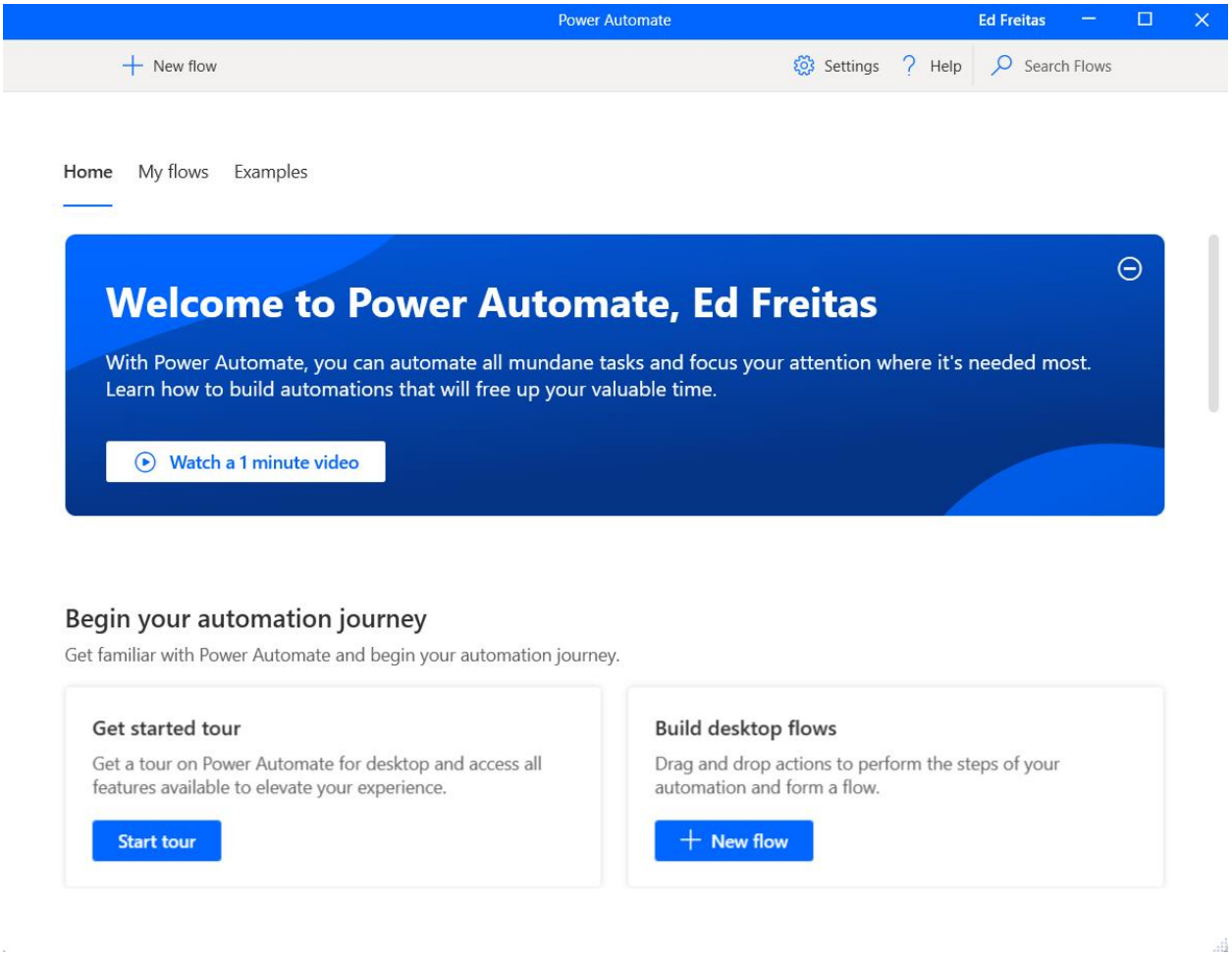

Figure 1-d: Welcome Screen—Power Automate Desktop

This welcome screen includes three options to introduce us to this technology. We can watch a one-minute video, tour the product, or create a new flow.

I suggest you watch the video and later take the tour, which I'll skip here and instead go straight into the action.

Creating a new flow

Automation with Power Automate Desktop is all about creating and using flows, which are made up of actions. Let's go ahead and create our first flow by clicking **+ New flow**. Once done, you'll see a screen similar to the following one.

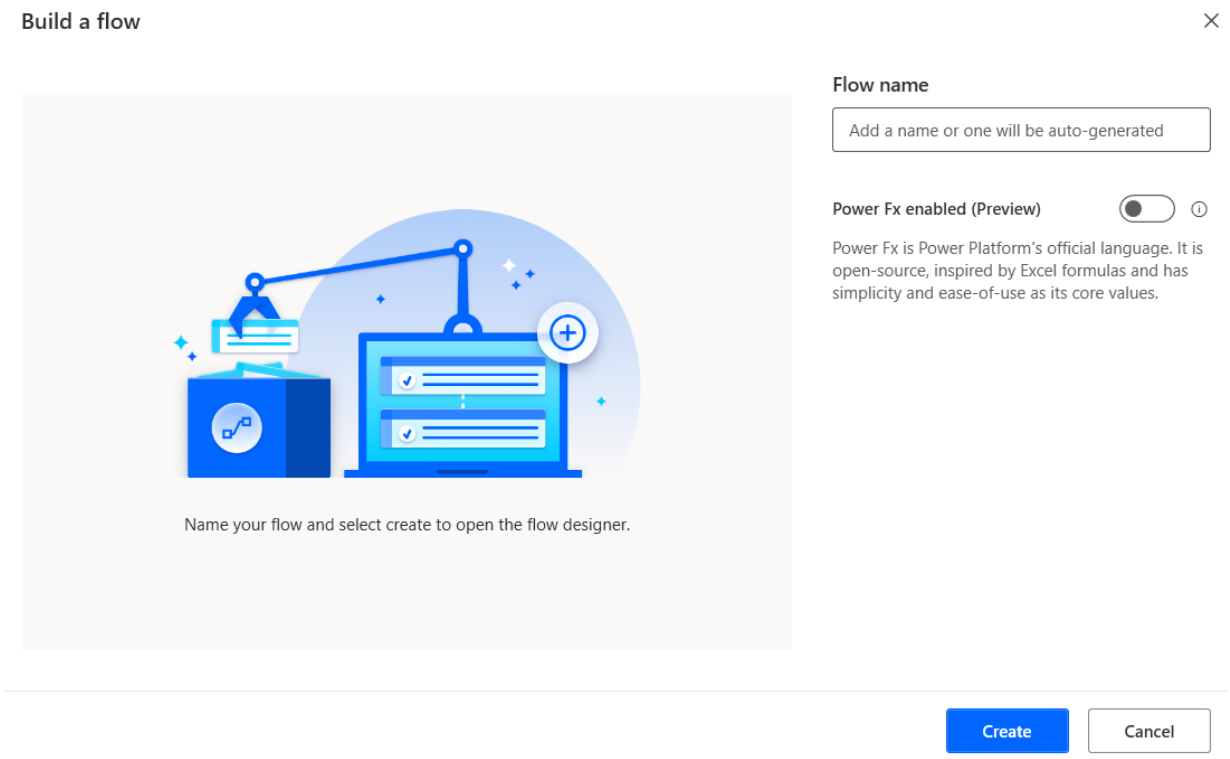

Figure 1-e: Power Automate Desktop (Build a flow Screen)

All we need to do to create a flow is give it a name and click **Create**. Let's name this flow **InitialFlow**.

Once we do that, we should see the following screen.

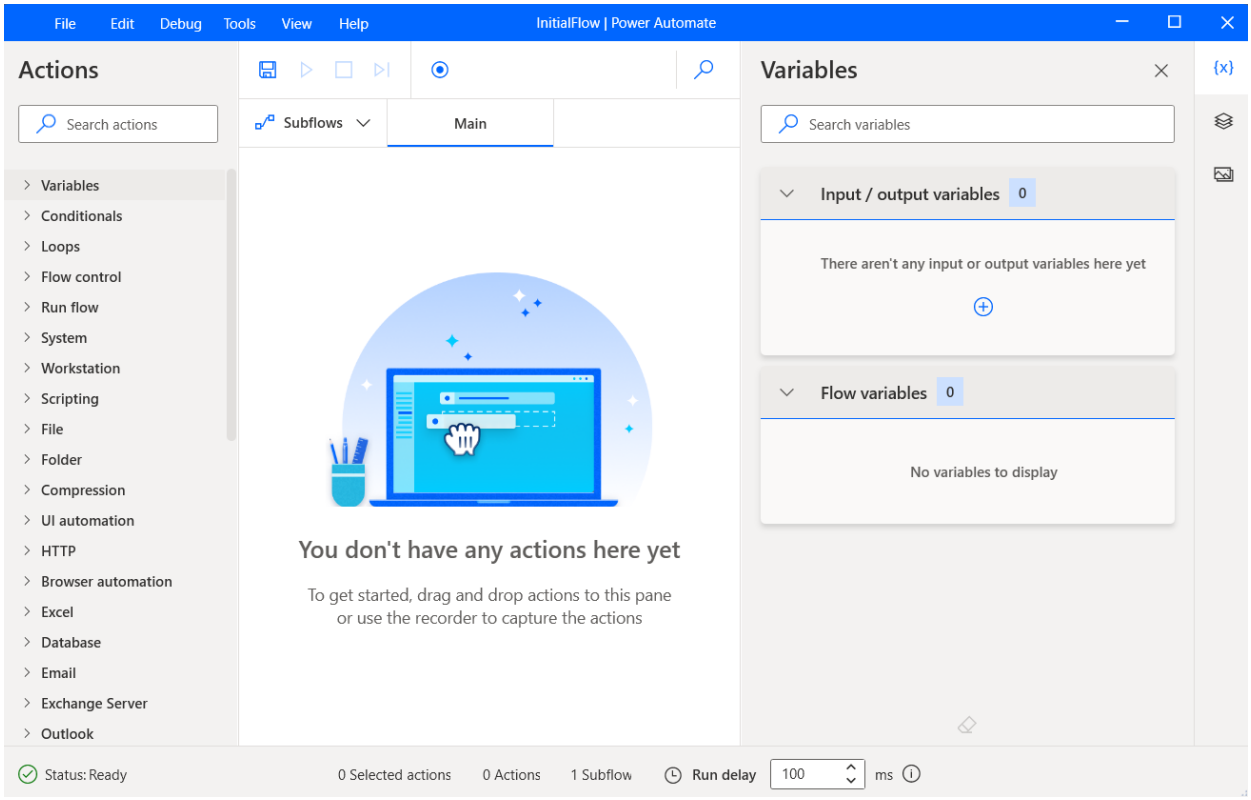

Figure 1-f: Power Automate Desktop (InitialFlow)

This screen contains the Actions pane on the left, the Subflows/Main pane in the middle, and the Variables pane on the right.

The magic of Power Automate Desktop is contained within the Actions pane, which includes diverse actions that perform all sorts of operations.

Actions are grouped into categories, such as actions that allow you to work with variables, conditionals, loops, flow control, run flow, systems, files, folders, browser automation, databases, email, and Outlook.

Designing the flow

Now that we have created the flow, let's start with the basics and explore how to work with variables using Power Automate Desktop. Figure 1-g shows what the finished flow looks like.

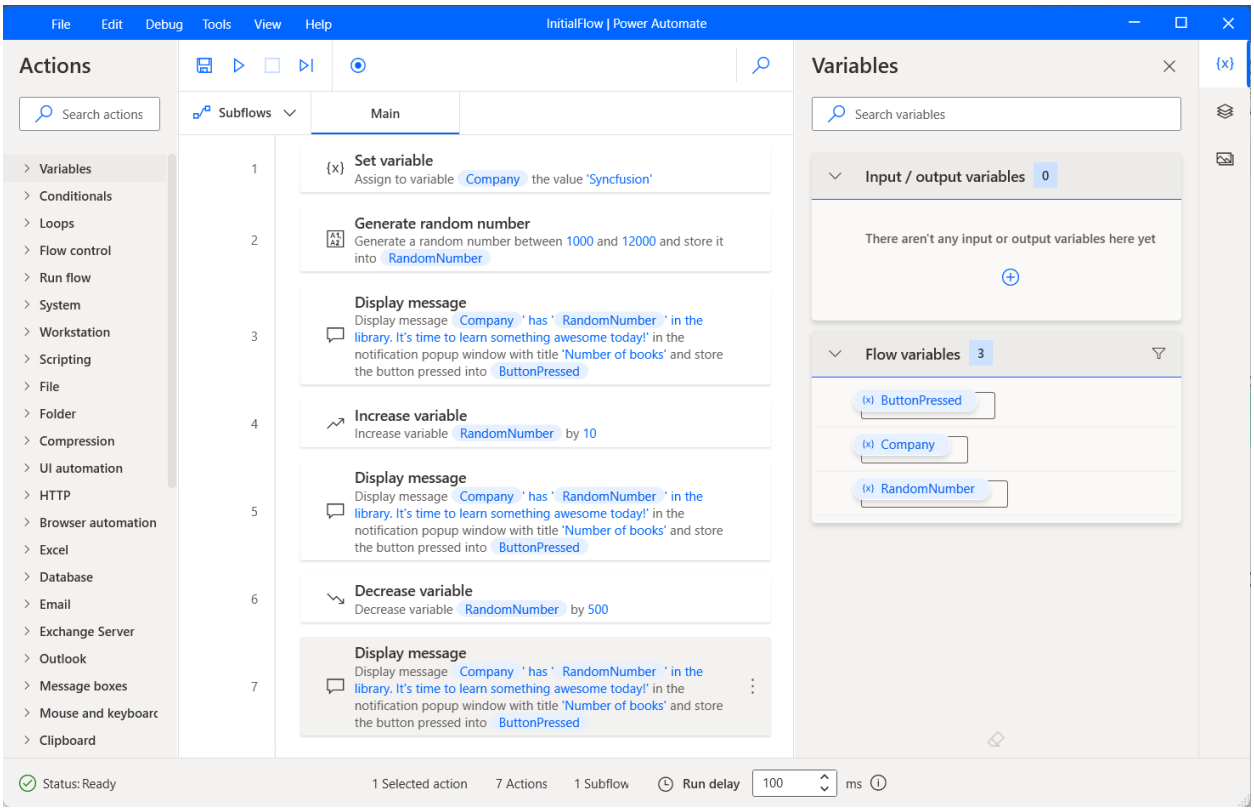

Figure 1-g: Power Automate Desktop (The Finished InitialFlow)

To begin, expand the **Variables** category, click the **Set variable** action, and drag it to the **Main** section within the middle pane.

Once this action has been dragged to the Main section, we will see the following dialog.

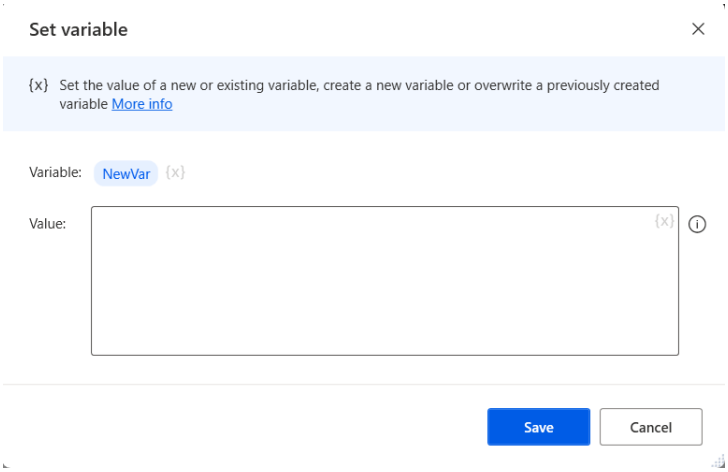

Figure 1-h: Power Automate Desktop (The Set variable Action—Default Details)

By default, this variable is called NewVar. Let's click on the **NewVar** name and rename it **Company**.

Note: When clicking on the NewVar variable, you'll see %NewVar%. This is how variables are specified in Power Automate Desktop. Variables within Power Automate Desktop are contained within the %% chars. So, when changing the name of the variable Company, keep the %% chars. Therefore, the variable name should be %Company%. Power Automate Desktop adds them if you don't explicitly add the percent (%%) characters.

After changing the variable name to **Company**, let's also set **Syncfusion** as the variable's value.

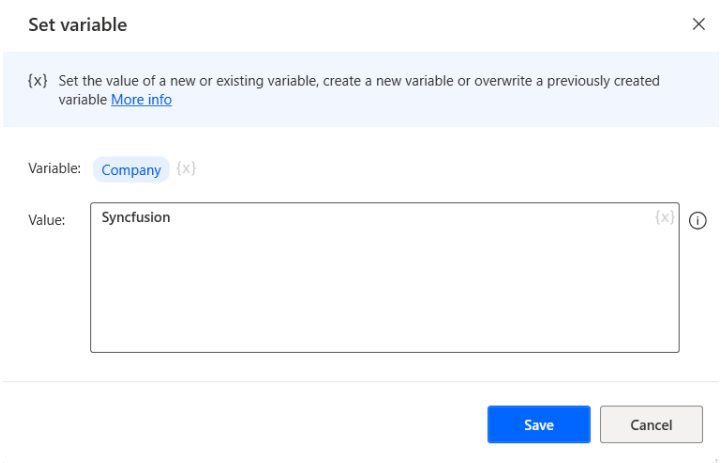

Figure 1-i: Power Automate Desktop (The Set variable Action—Changed Details)

Once done, let's click **Save**. By doing that, we'll have added the first action to the flow, which will now look as follows.

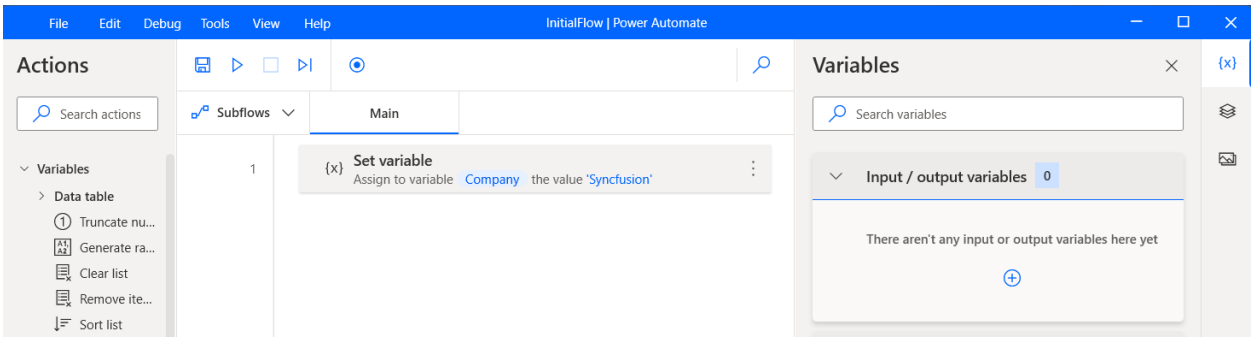

Figure 1-j: Power Automate Desktop (The First Action Added)

Adding Generate random number

Now that we have added the first action to the flow, let's add additional ones. The goal of this flow is not to automate anything specifically but to show you the basics of what any flow uses, which are variables.

So, let's go ahead and add the **Generate random number** action to the flow by dragging it to the main window, and you'll see the following screen. This action can be found under Variables.

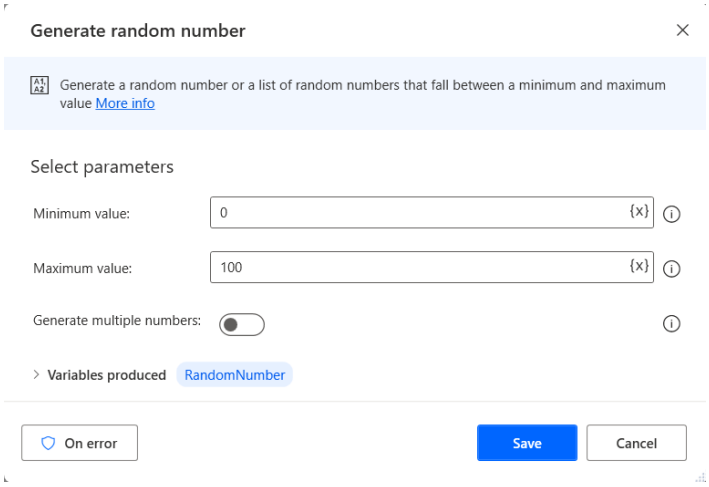

Figure 1-k: Power Automate Desktop (The Generate random number Action—Default Details)

Let's change the default values to **1000** as the **Minimum value** and **12000** as the **Maximum value**.

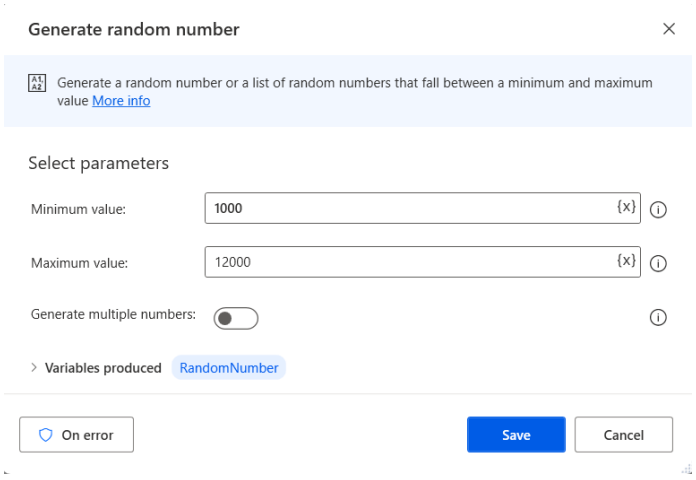

Figure 1-l: Power Automate Desktop (The Generate random number Action—Changed Details)

With the values changed, let's click **Save**.

Adding Display message

Now, the next thing we want to do is to add the **Display message** action to the flow. We can do this by dragging the action to the flow. This action can be found under **Message boxes**. We can easily find it by typing “display” in the Actions search bar.

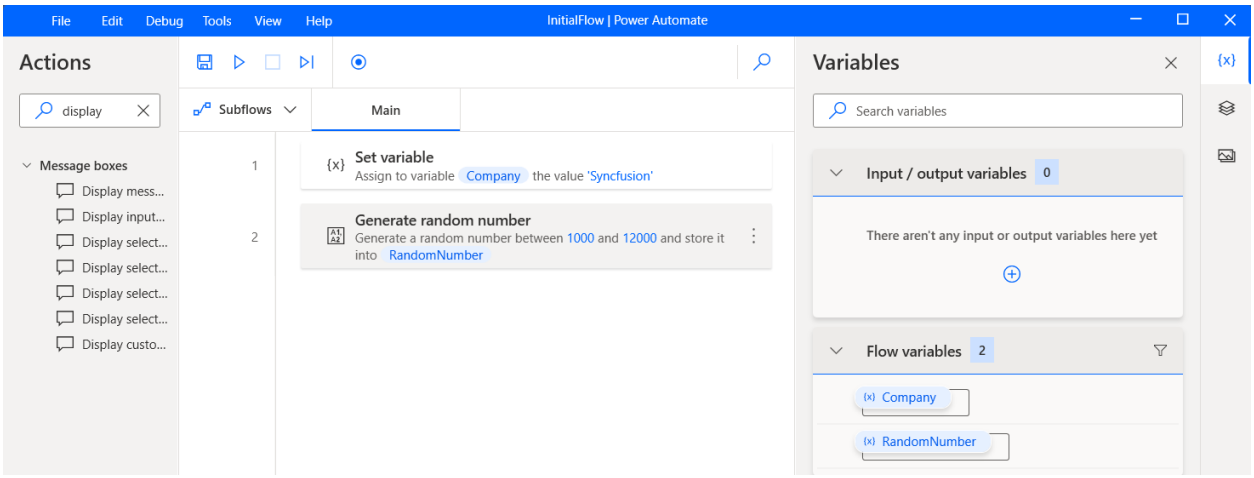

Figure 1-m: Power Automate Desktop (Searching for the Display message Action)

We can set the following values for the Message box title and Message to display fields once the action has been added to the flow.

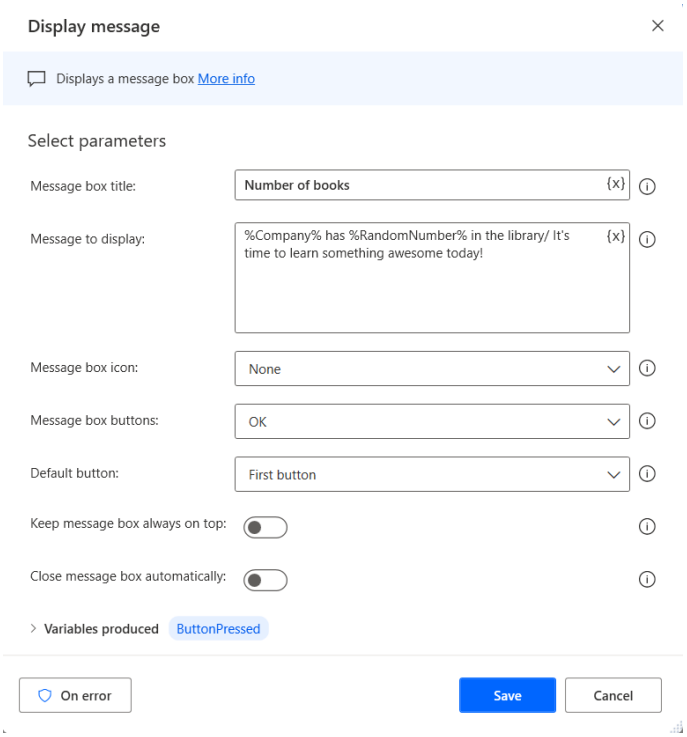

Figure 1-n: Power Automate Desktop (The Display message Action—Changed Details)

Notice how, within the Message to display field, we include the values of the Company and RandomNumber variables—%Company% and %RandomNumber%. The rest of the action properties are left with their default values. Finally, we can click **Save**.

At this stage, the flow does only three things so far. It assigns to the Company variable *Syncfusion* as a value, generates a random number, and displays a message with the values of the company name and the random number generated.

Adding Increase variable

The next thing we want to do is add the Increase variable action to the flow. The idea is to increase the value of the RandomNumber variable by 10.

We can find the Increase variable action under Variables. So, drag it to add it to the flow and place it after the Display message action we previously added.

As the variable name, we can indicate that we want to increment the RandomNumber variable, so it must be specified using the %% chars—**%RandomNumber%**. We can increment its value by **10**.

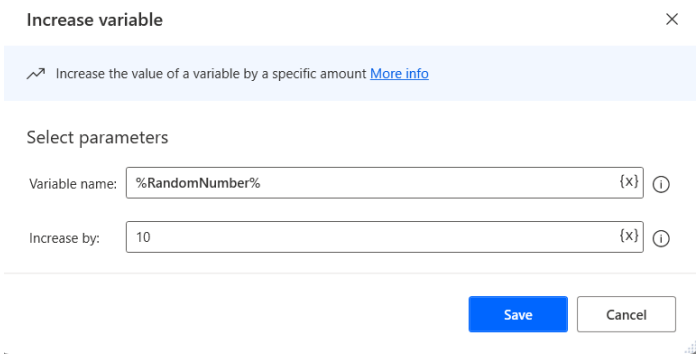

Figure 1-o: Power Automate Desktop (The Increase variable Action—Changed Details)

We can click **Save** once we have specified these field values.

Copying Display message

Next, we want to show the value of the RandomNumber variable after increasing it. To do this, we must select the previous **Display message** action, press **Ctrl+C** to copy it, and then press **Ctrl+V** to paste it.

Once done, drag the copied **Display message** action right after the Increase variable action.

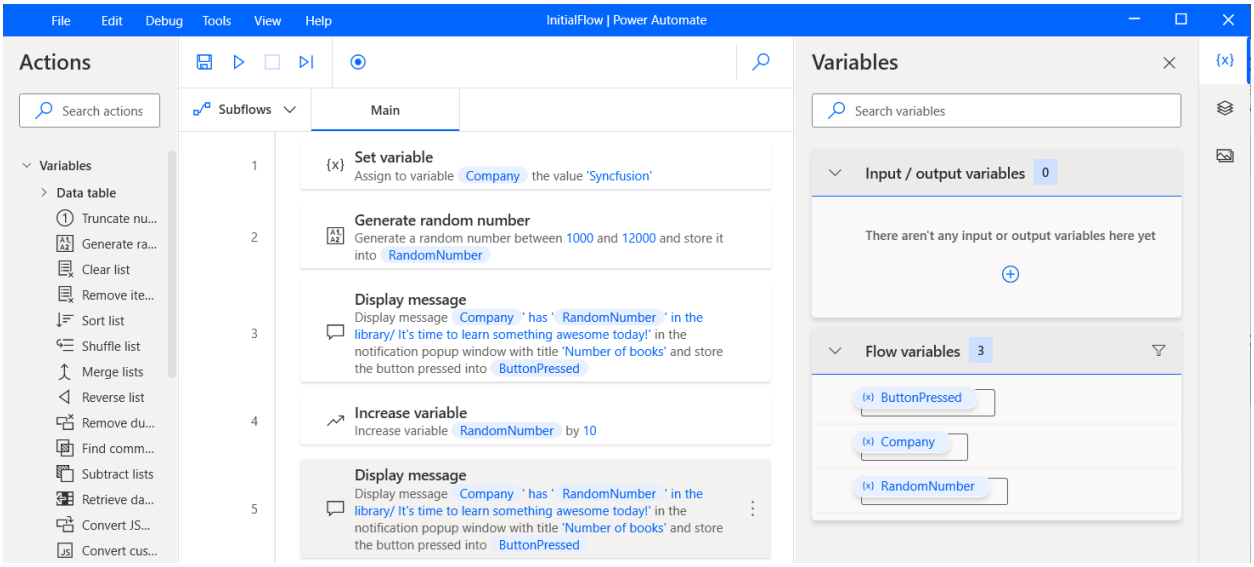

Figure 1-p: Power Automate Desktop (The Updated InitialFlow)

Adding Decrease variable

Let's add the **Decrease variable** action to the flow and place it after the second Display message action. The Decrease action variable can be found under Variables.

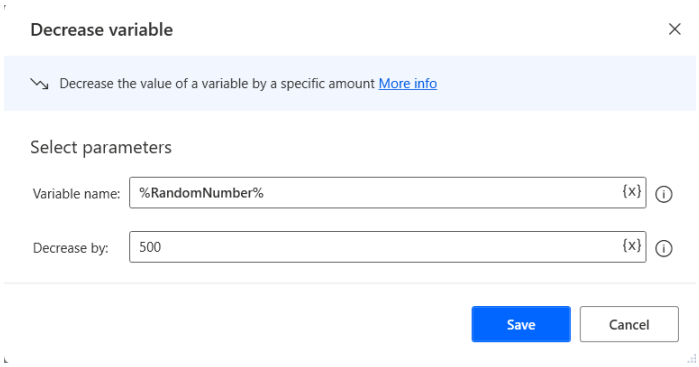

Figure 1-q: Power Automate Desktop (The Decrease variable Action—Changed Details)

As the variable name, we indicate that we want to decrease the value of the RandomNumber variable, which is why we write it as %RandomNumber%. Let's reduce its value by **500**.

Note: When specifying a variable, you must keep the %% chars—in this case, the variable name will be written as %RandomNumber%.

Once we're done, we can click **Save**.

Copying Display message

Next, we want to show the value of the RandomNumber variable after decreasing it. To do this, we can select the previous **Display message** action, press **Ctrl+C** to copy it, and then press **Ctrl+V** to paste it.

Once done, drag the copied **Display message** action right after the Decrease variable action.

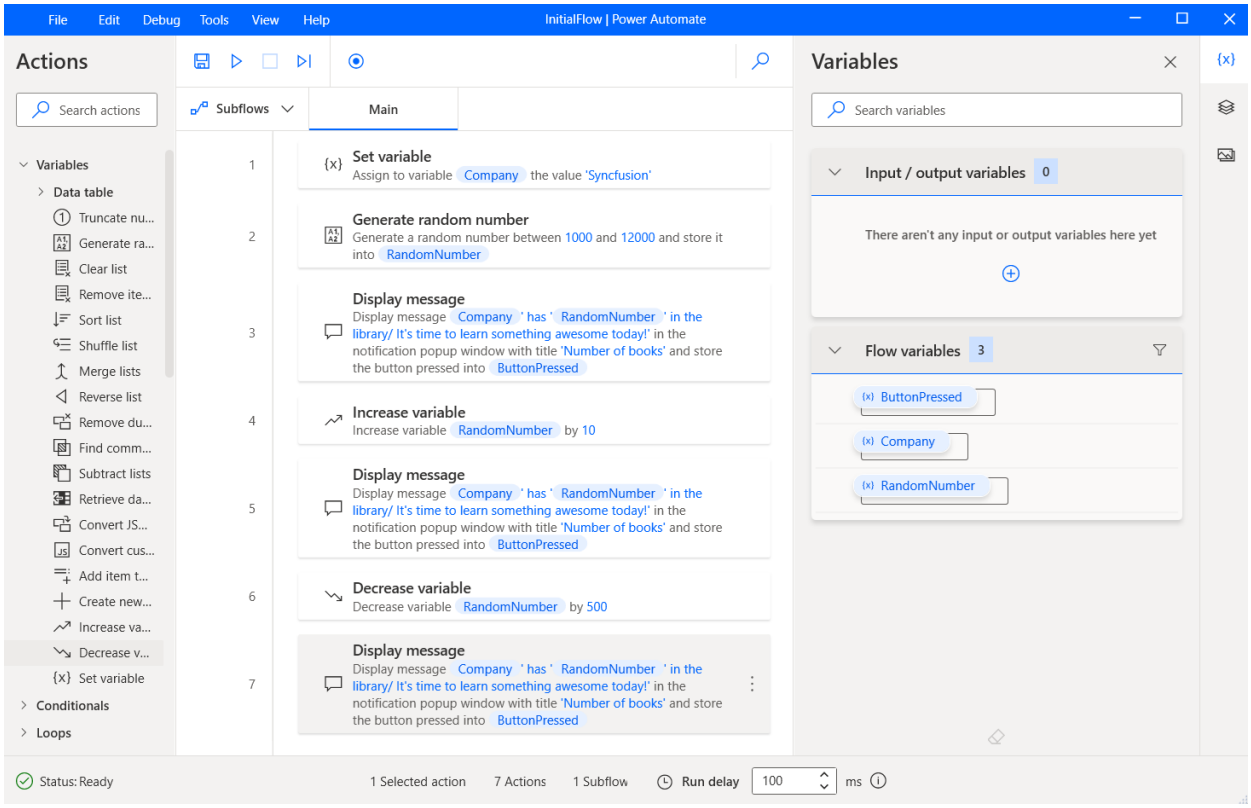

Figure 1-r: Power Automate Desktop (The Updated InitialFlow)

So now, we can display the value of the RandomNumber variable three times: after the random number has been generated, after its value has been increased, and after its value has been decreased.

By doing this, we see how to work with variables, crucial to building automated workflows using Power Automate Desktop.

Saving and running the flow

Let's save the flow and run it to see what we have created in action. First, click the save-flow icon, and then the arrow icon next to it to run the flow.

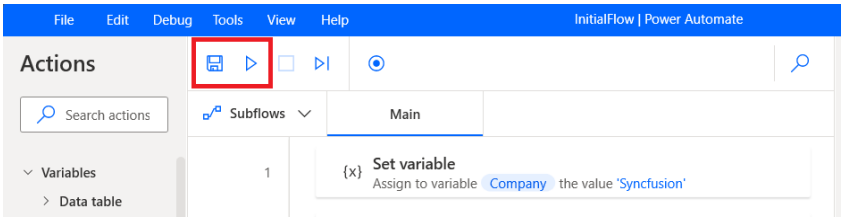

Figure 1-s: Power Automate Desktop (The Save and Run Flow Buttons)

The flow execution begins, and we should see three message boxes being displayed—for example, here's the first one.

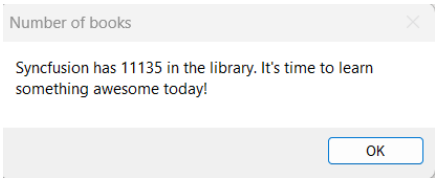

Figure 1-t: Power Automate Desktop (The Display message Action Running)

Each Display message action will show a different value and will most likely differ from the ones you'll see on your machine, given that it's a randomly generated value.

Quick recap

Awesome! We've now seen and gone through the process of building our first automated flow using Power Automate Desktop.

Although the flow technically doesn't do much and doesn't automate anything, we've explored how to add actions to the flow and work with variables, which is crucial for building more complex workflows.

We'll explore and build some practical automated workflows in the following chapters.

Chapter 2 Working with Files

Quick intro

Now that we've seen the basic concepts behind Power Automate Desktop, it's time to create a more realistic automated workflow and explore how to work with files.

To do that, let's create a new flow and name it **WorkingWithFiles**. Click **Create** to complete the flow.

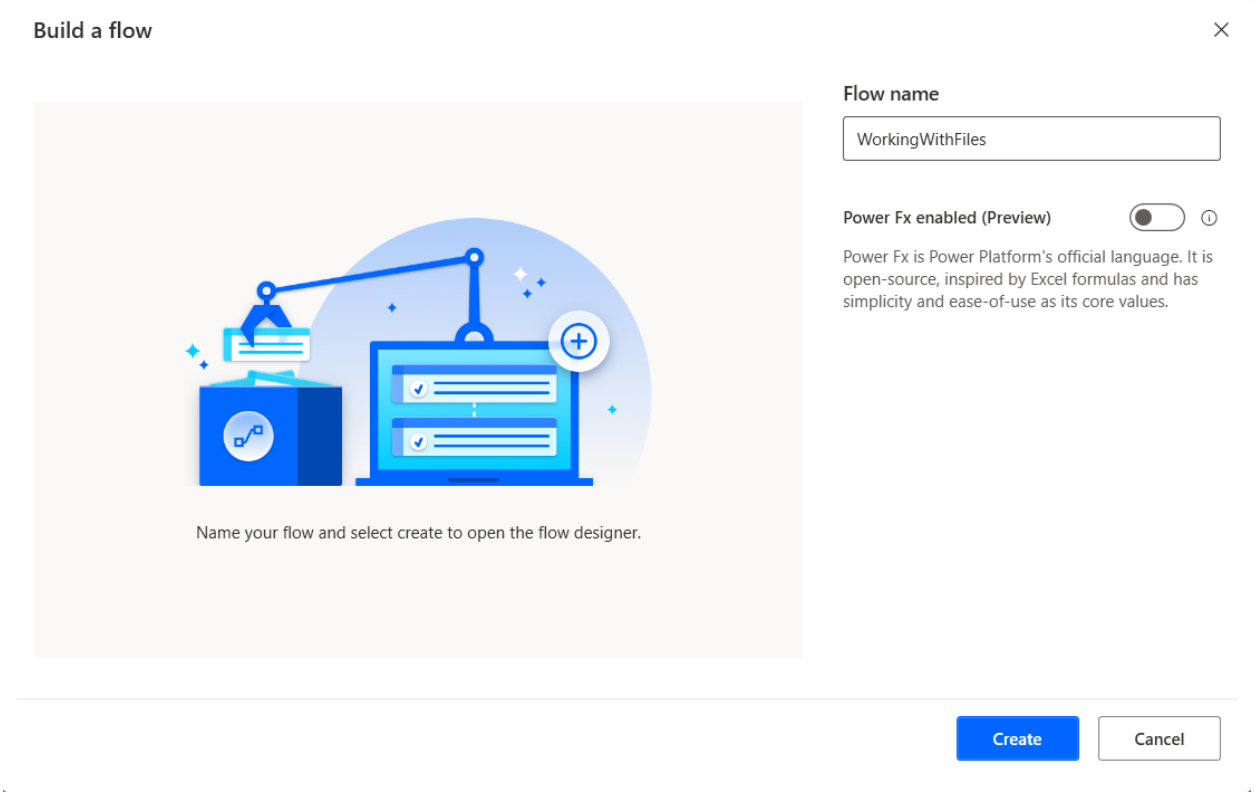

Figure 2-a: Power Automate Desktop (Creating a New Flow—WorkingWithFiles)

Flow functionality

We will build a flow that copies two files from an origin folder to a destination folder, zips those files into an archive, renames the copied files, deletes the renamed files, and finally, unzips the original files. By doing this, we'll get all the necessary knowledge to work with files using Power Automate Desktop.

Checking for files

To begin, the first thing we need to do is check if the files we intend to work with exist. To check whether a specific file exists, we need to add an **If file exists** action to the flow—so let's begin by doing that. The If file exists action can be found under File**.**

After you drag the action to the flow, within the file path text box, specify the name of the file whose existence you want to check. In my case, I'm checking if the C:\Tmp\Test\origin\index.html file exists.

I highly suggest you use the same file path and name. You may choose another file, but be consistent and use the same file name throughout the flow.

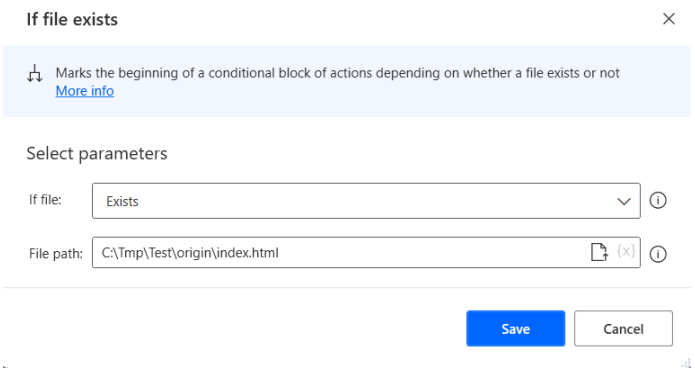

Figure 2-b: Power Automate Desktop (The First If file exists Action—Changed Details)

Once you have specified the file name you want to check, click **Save**. The flow should now look as follows.

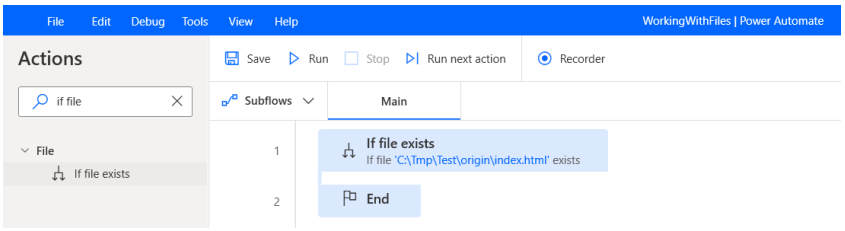

Figure 2-c: Power Automate Desktop (The WorkingWithFiles Flow—Step 1)

Furthermore, there's a second file to process, and I also want to check if it exists. To do that, I will add another **If file exists** action to the flow by dragging and placing it within the first If file exists action.

Then, indicate the file name to check—in my case: C:\Tmp\Test\origin\missing.html. You may choose another file—but again, be consistent and use the same file name throughout the flow.

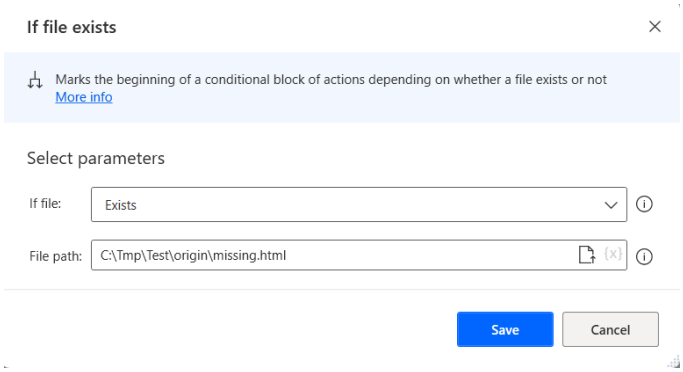

Figure 2-d: Power Automate Desktop (The Second If file exists Action—Changed Details)

Once you have specified the file name you want to check, click **Save**. The flow should now look as follows.

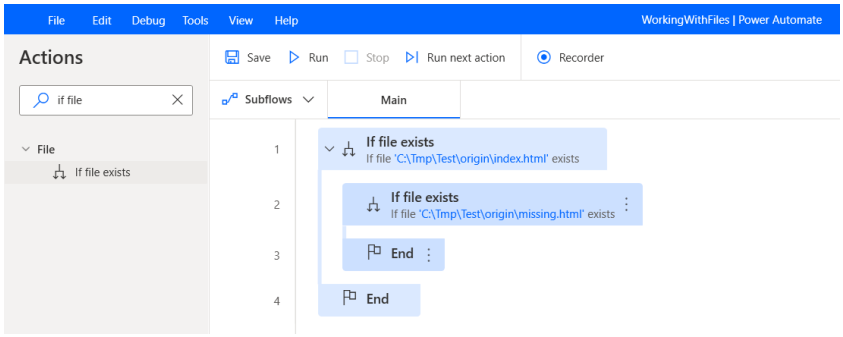

Figure 2-e: Power Automate Desktop (The WorkingWithFiles Flow—Step 2)

Copying the files

Now that we have checked if the files we will process throughout the flow exist, we can copy these files to the destination folder. We can add the **Copy file(s)** action to the flow and place this action within the second If file exists action. The Copy file(s) action is under File within the Actions pane.

When the Copy file(s) action dialog displays, within the File(s) to copy text box, add the following values:

%['C:\\Tmp\\Test\\origin\\index.html', 'C:\\Tmp\\Test\\origin\\missing.html']%.

So, we are creating a variable (**%%**) that is a list (**[]**), which contains two elements, where each element is a string (**''**). Each element indicates the **file name** of a file to copy.

Because we are specifying both file names within a list of strings, and as a Power Automate Desktop variable, we must escape the (****) character to indicate that it represents a file path, so we write it twice (****).

Note: If you used different file paths, use those instead within the previous list variable (%[…]%).

As for the value of the Destination folder, we can indicate the folder path where we want to copy the files. In my case, **C:\Tmp\Test\destination**. We don't need to use double backslashes (****) because this is not a variable, but a hard-coded value.

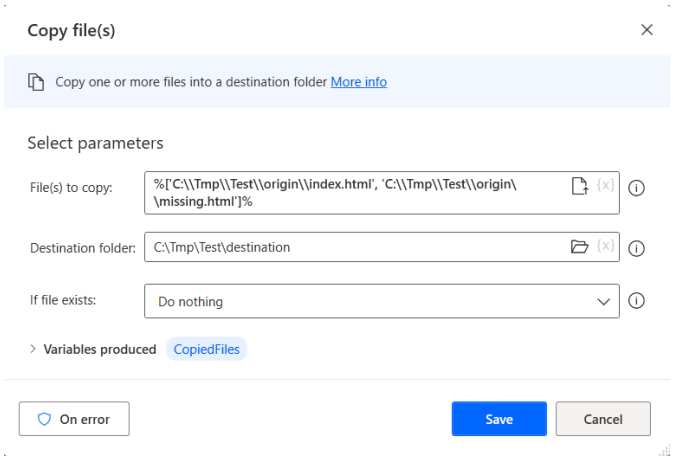

Figure 2-f: Power Automate Desktop (The Copy file(s) Action—Changed Details)

Once you have specified the values, click **Save** to continue.

Zipping the files

With the original files copied to the destination folder, we can next compress those files into a zip archive. We can add the **Zip files** action to the flow and place it after the Copy file(s) action.

The Zip files action can be found under Compression within the Actions pane. Within the Zip files action, we need to indicate the **Archive path**, which is the file path of the compressed file that will be created, and the **File(s) to zip**, which will be the value of the CopiedFiles variable (%CopiedFiles%) from the previous action.

The Zip files action will produce a ZipFile variable, which can be used in subsequent flow actions.

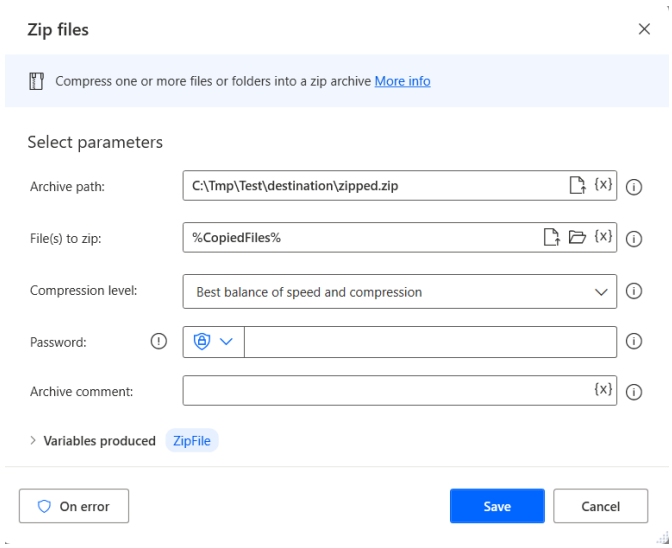

Figure 2-g: Power Automate Desktop (The Zip file Action—Changed Details)

Once you have indicated these values, click **Save** to add this action to the flow. At this stage, the flow looks as follows.

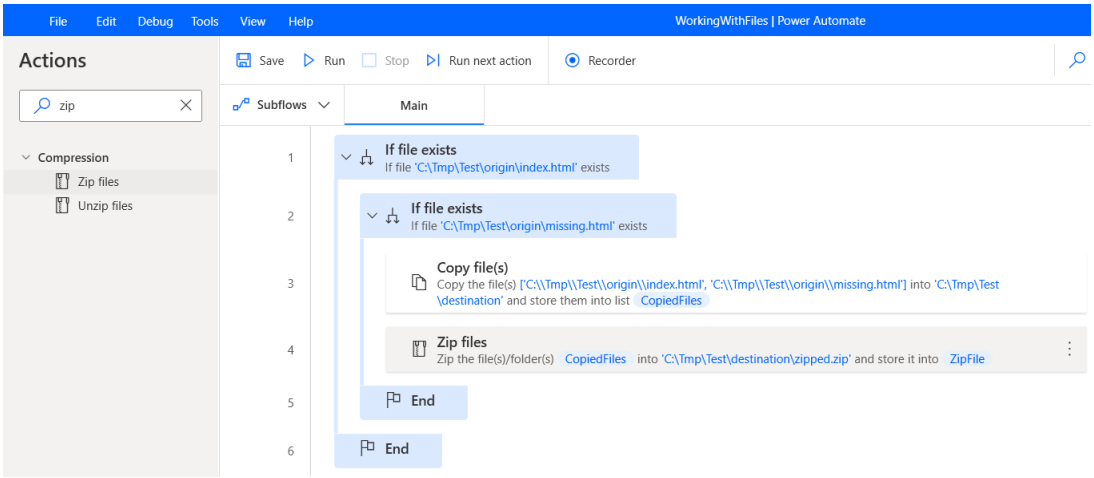

Figure 2-h: Power Automate Desktop (The Updated WorkingWithFiles Flow)

Displaying a message

With the flow as is, we can copy and compress files. Now, let's add a **Display message** action to the flow and display the value of the CopiedFiles variable (%CopiedFiles%).

Let's add the Display message action after the Zip files action. The Display message action can be found under Message boxes within the Actions pane.

To the **Message box title** text box, we can assign the **Files** string. To the **Message to display** text box, we can assign the value of the CopiedFiles variable (**%CopiedFiles%**).

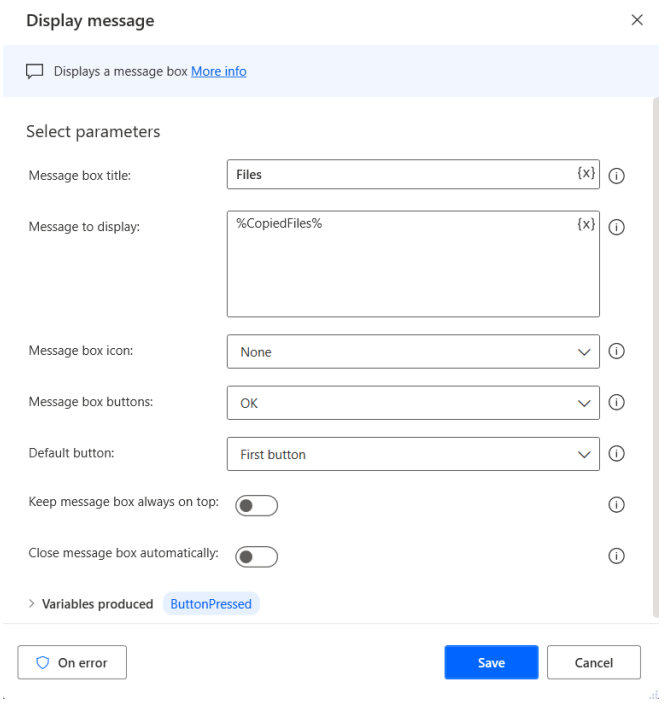

Figure 2-i: Power Automate Desktop (The Display message Action)

Once done, we can click **Save**. By doing this, we can see the file paths of the copied files.

Renaming files

The next thing we can do is rename the copied files. To do this, we need to loop through the list of copied files by using the **For each** action.

The For each action is found under Loops within the Actions pane. So, let's drag this action to the flow and place it after the Display message action.

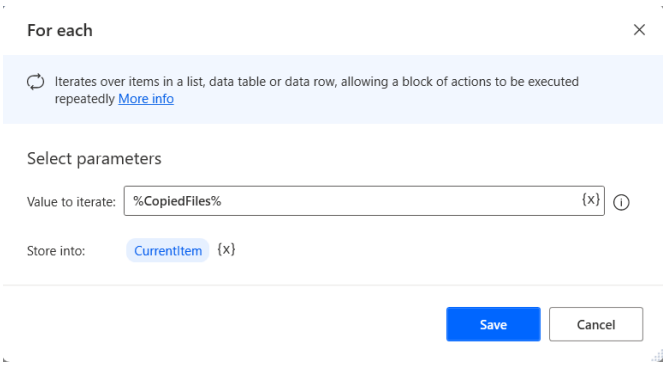

Figure 2-j: Power Automate Desktop (The For each Action)

Within the **Value to iterate** text box, let's assign the value of the CopiedFiles variable (**%CopiedFiles%**). Once done, click **Save**.

Let's add a **Rename file(s)** action within the loop. We can find this action under File within the Actions pane.

To the **File to rename** text box, we can assign the value of the CurrentItem variable (**%CurrentItem%**), which represents each copied file.

As for the **Rename scheme**, we need to choose the **Add text** option explicitly, and then set the value of the **Text to add** text box as **renamed-**.

We also need to choose the **Before name** option as the value of the **Add text** dropdown.

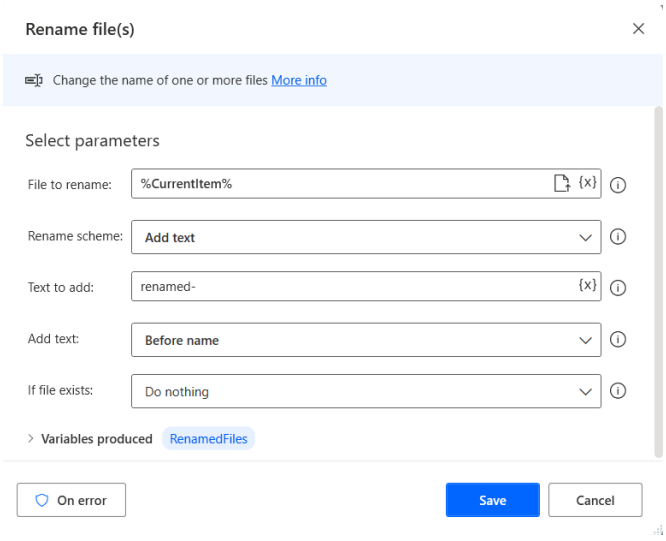

Figure 2-k: Power Automate Desktop (The Rename file(s) Action—Changed Details)

By selecting these options, we can prepend the **renamed-** string prefix to the file name of each copied file. So, in essence, if the name of one of the copied files is C:\Tmp\Test\origin\index.html, the renamed file will be C:\Tmp\Test\origin\renamed-index.html.

Once those options have been selected, click **Save**. Once done, the flow will look as follows.

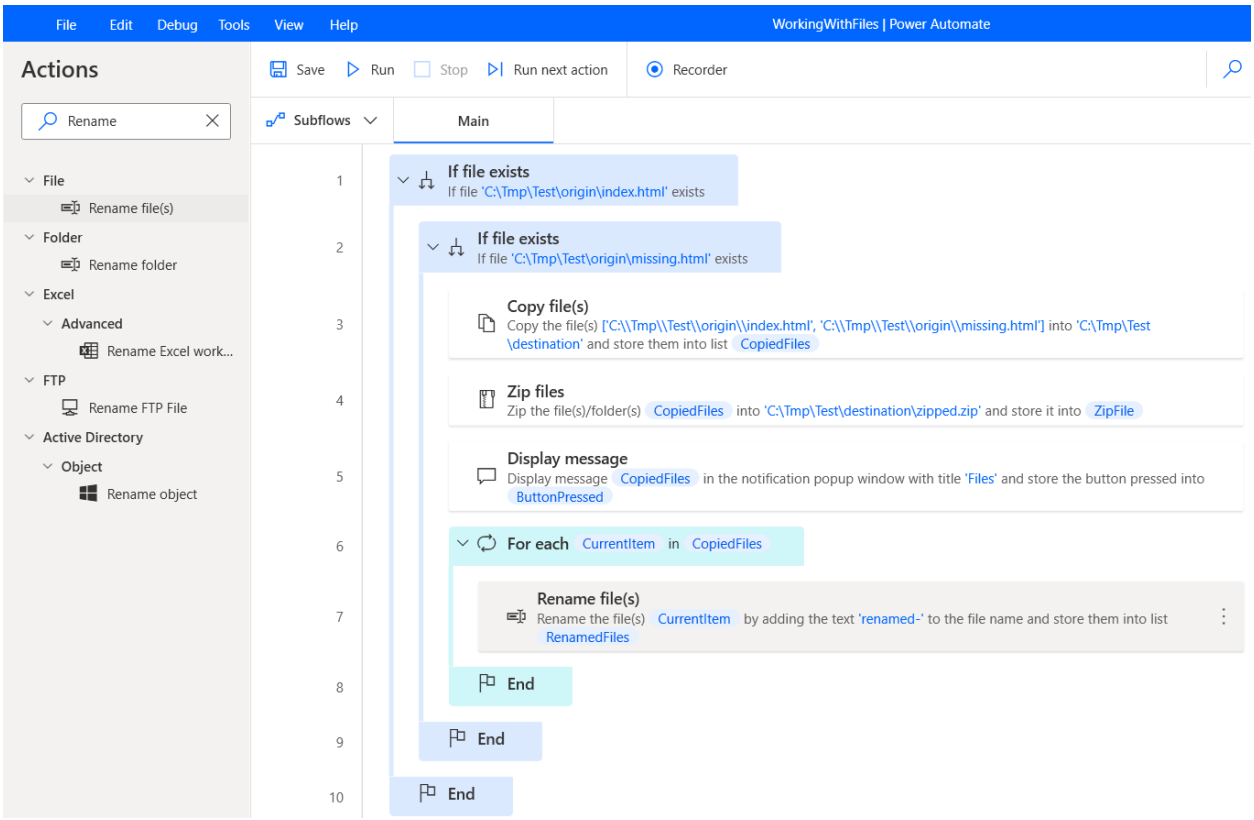

Figure 2-I: Power Automate Desktop (The Updated WorkingWithFiles Flow)

Displaying another message

Now that we have renamed the copied files, let's add another **Display message** action to the flow after the For each action to see the names of the files. As you know, the Display message action is under Message boxes within the Actions pane.

We can assign the **Files** string to the **Message box title** text box and assign the value of the CopiedFiles variable (**%CopiedFiles%**) to the **Message to display** text box.

Once done, we can click **Save**.

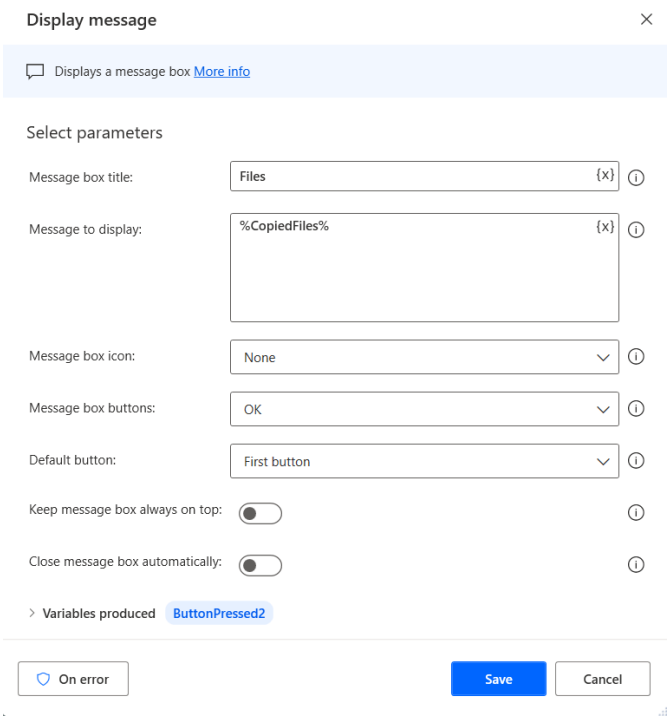

Figure 2-m: Power Automate Desktop (The Display message Action—Changed Details)

After these changes, the flow should now look as follows.

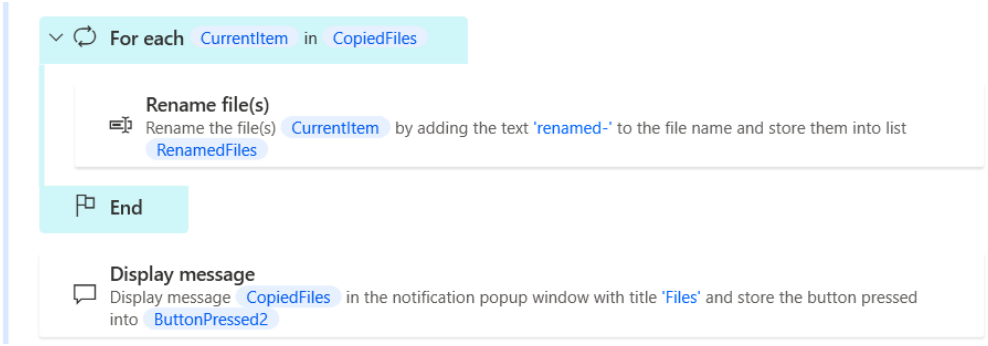

Figure 2-n: Power Automate Desktop (The Updated WorkingWithFiles Flow—Partial View)

Deleting files

After renaming the files, the next thing we want to do is delete them. To achieve that, we'll have to loop through the list of copied files, get the file path of each file we want to delete, and then delete each file in each iteration.

So, let's add a **For each** action to the flow after the Display message action we just added. The For each action is under Loops within the Actions pane.

Let's assign the value of the CopiedFiles variable (**%CopiedFiles%**) to the **Value to iterate** text box. We'll store these details in the **CurrentItem** (%CurrentItem%) variable.

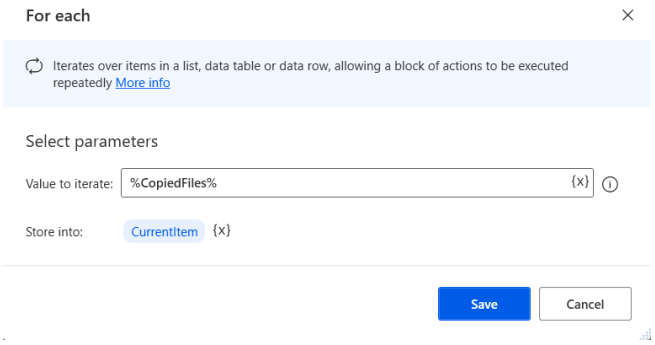

Figure 2-o: Power Automate Desktop (The For each Action—Changed Details)

Once done, we can click **Save**. Following that, the flow should look as follows.

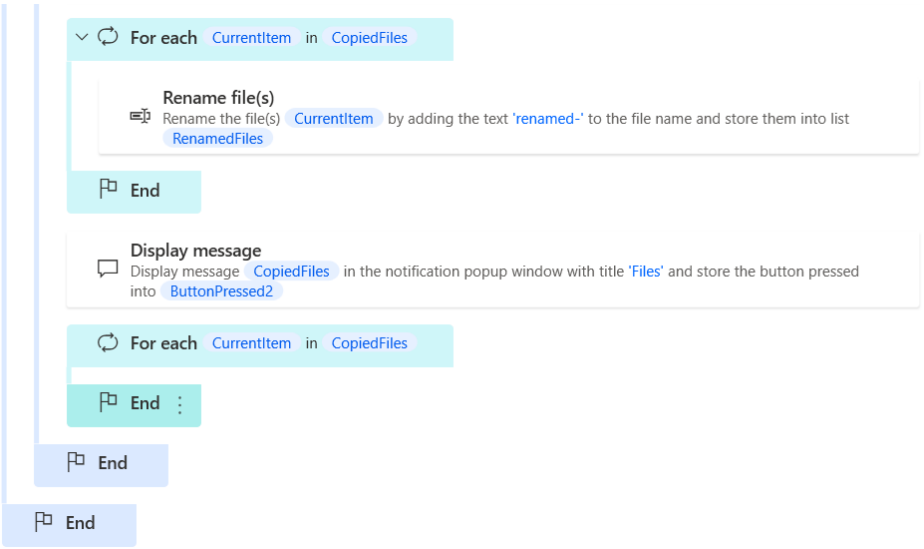

Figure 2-p: Power Automate Desktop (The Updated WorkingWithFiles Flow—Partial View)

Get file path

To delete the renamed files, inside the For each action we just added, we need to add a **Get file path part** action, allowing us to get the exact file name for each file we want to delete. The Get file path part action is under File within the Actions pane. So, let's drag this action and place it within the **For each** action we just added.

Let's assign the value of the CurrentItem (**%CurrentItem%**) variable to the **File path** text box. The CurrentItem variable represents the name of each file in the copied file list.

Once done, click **Save**.

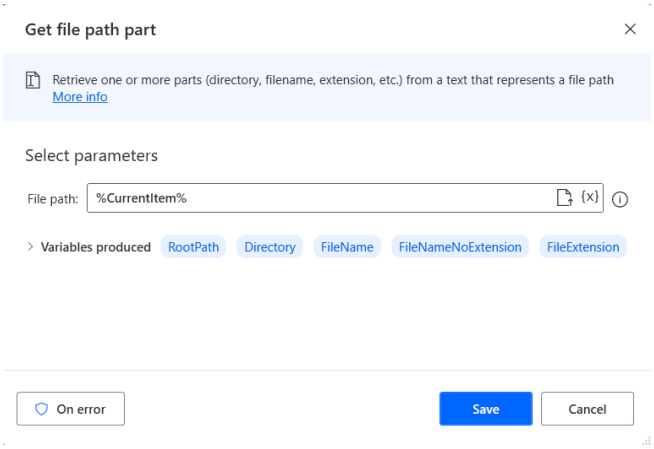

Figure 2-q: Power Automate Desktop (The Get file path part Action—Changed Details)

The Get file path part action returns a few variables we'll use in the following action, notably the Directory and FileName variables.

Deleting files

Now that we have the partial file path for each renamed file we want to delete, we can use the Delete file(s) action. Let's add the **Delete file(s)** action to the flow and place it after the Get file path part action that we just added. The Delete file(s) action is under File under the Actions pane.

To the **File(s) to delete** text box, we can assign the following value:

%Directory%\renamed-%FileName%

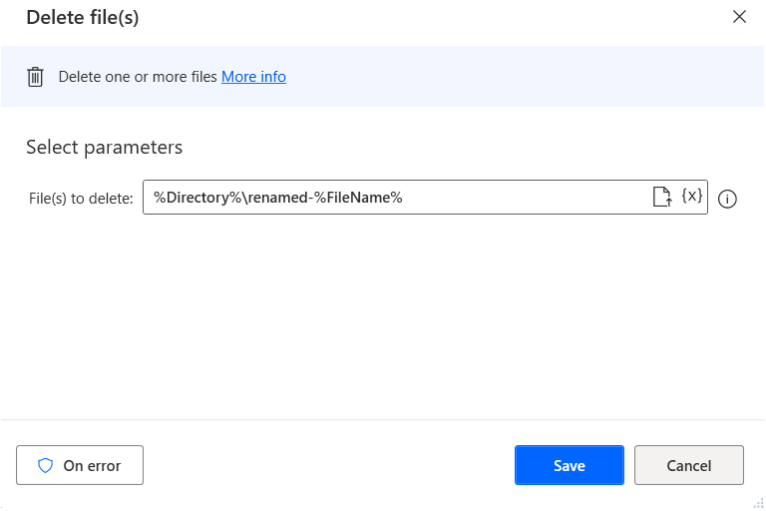

Figure 2-r: Power Automate Desktop (The Delete file(s) Action—Changed Details)

This means we can get the complete file path of each renamed file we want to delete by concatenating the folder name (**%Directory%**) where the file is contained, plus the **\renamed-** prefix and the name of the file itself to delete (**%FileName%**).

Once done, click **Save**. The flow should now look as follows.

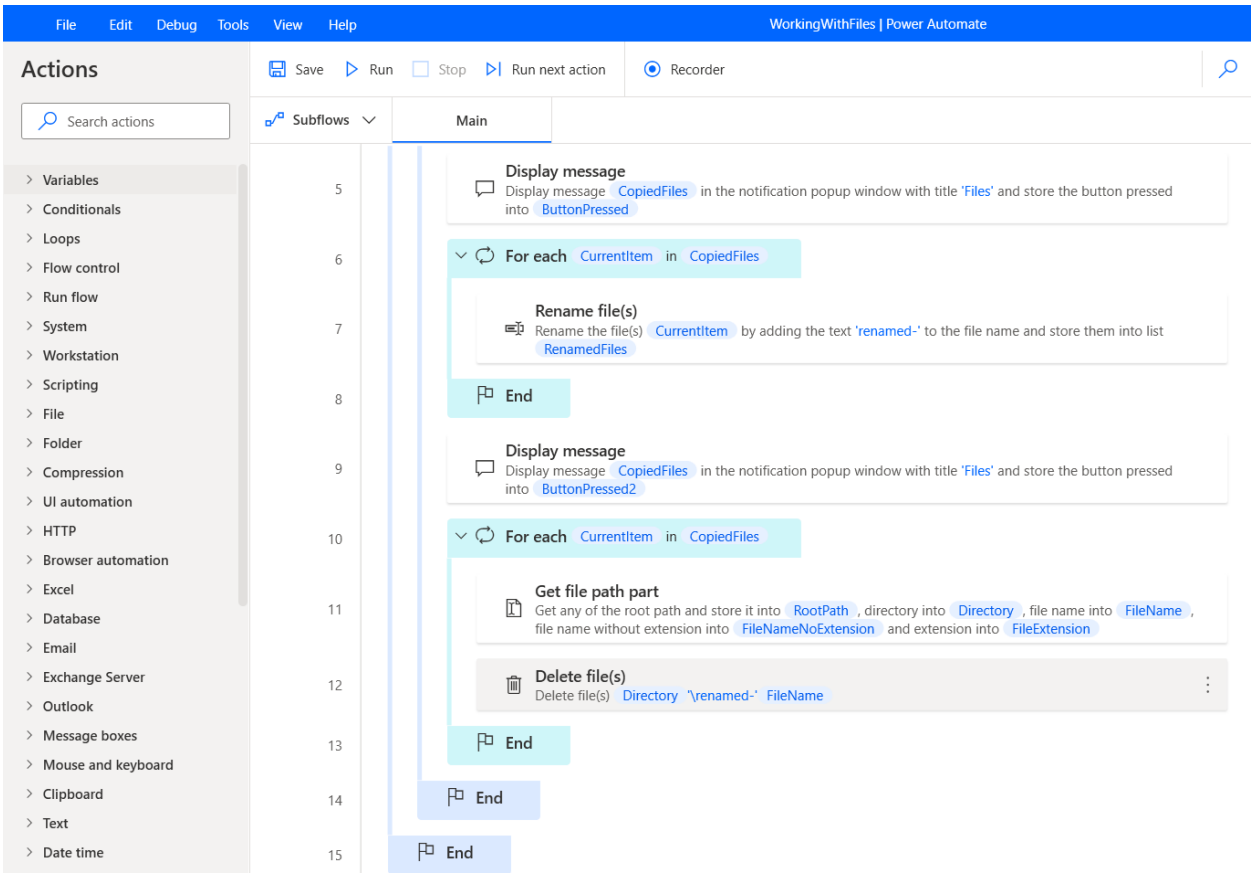

Figure 2-s: Power Automate Desktop (The Updated WorkingWithFiles Flow—Partial View)

Unzipping files

Now that we have added the functionality to delete files, let's unzip the copied files that we previously compressed.

To do that, let's add the **Unzip files** action to the flow after the end of the most recent For each action we added. The Unzip files action can be found under Compression in the Actions pane.

Let's assign the file path of the zip file we created using a previous action to the **Archive path** text box. In my case, it is **C:\Tmp\Test\destination\zipped.zip**.

Let's also assign the path of the destination folder where the zip file's contents will be unzipped. In my case, it is **C:\Tmp\Test\destination\unzipped**.

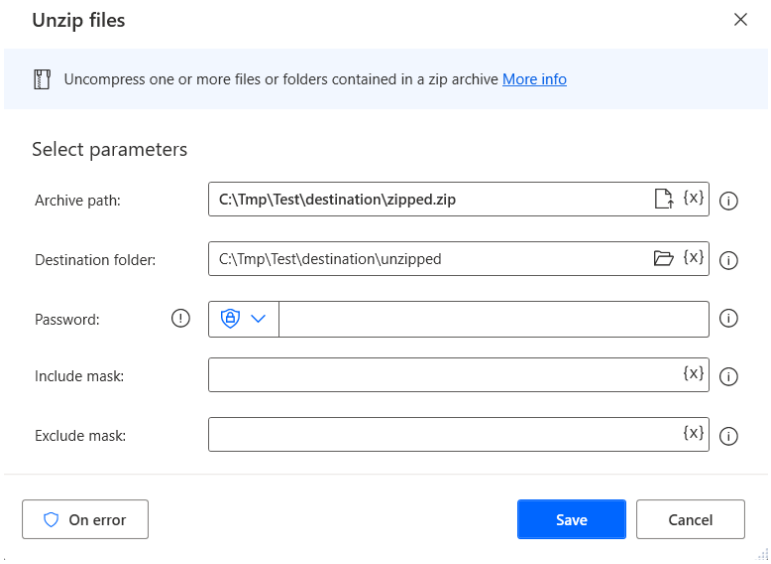

Figure 2-t: Power Automate Desktop (The Unzip files Action—Changed Details)

Once done, click **Save**. The flow should now look as follows.

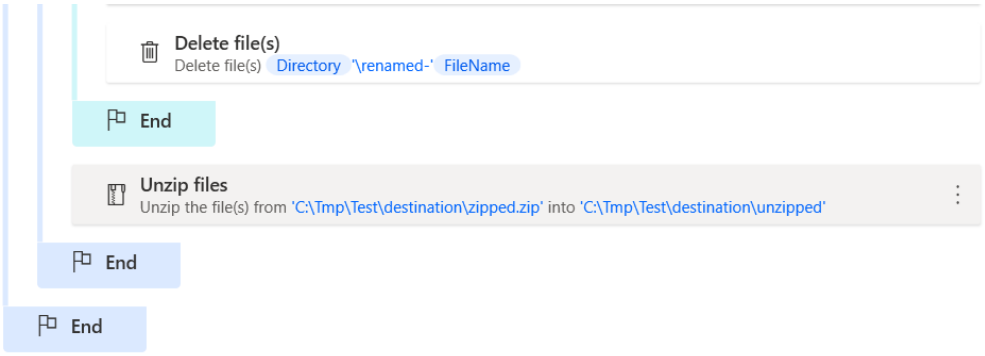

Figure 2-u: Power Automate Desktop (The Updated WorkingWithFiles Flow—Partial View)

Great, we now have the flow ready!

Running the flow

It's time to run the flow to see it in action. But before we do that, open a Windows Explorer window pointing to the C:\Tmp\Test\destination folder (if that's the destination folder you used throughout the flow) to see what the flow does during execution.

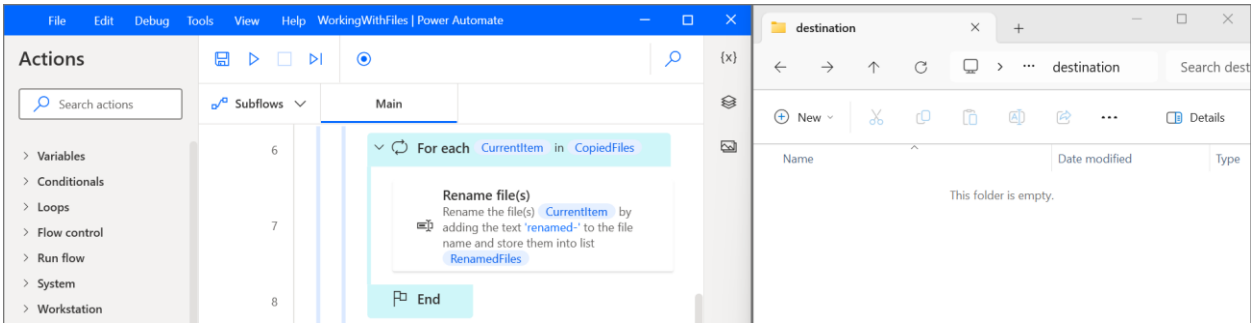

Figure 2-v: Power Automate Desktop (The Finished WorkingWithFiles Flow—Partial View) and Destination Folder, Side-by-Side

To begin the execution, click the Run icon (which is to the right of the Save icon). Shortly after the execution starts, we can see the message with the names of the copied files and how these files have been copied and zipped to the destination folder.

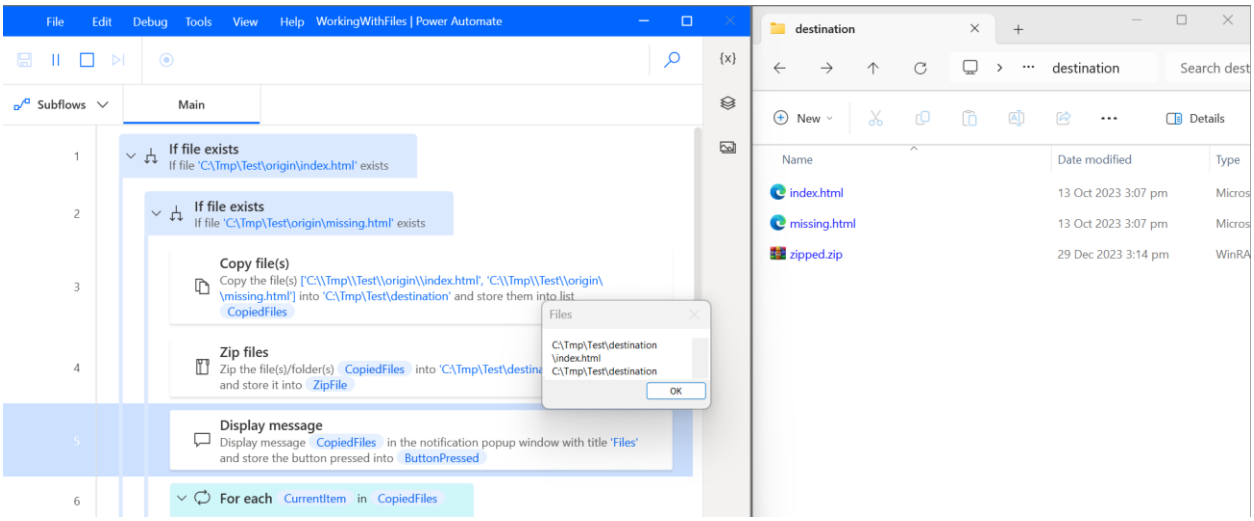

Figure 2-w: Power Automate Desktop (WorkingWithFiles Flow Execution—Partial View) and Destination Folder (During Execution—Part 1)

Let's click **OK** to continue the flow execution. Shortly after, we can see that the copied files have been renamed.

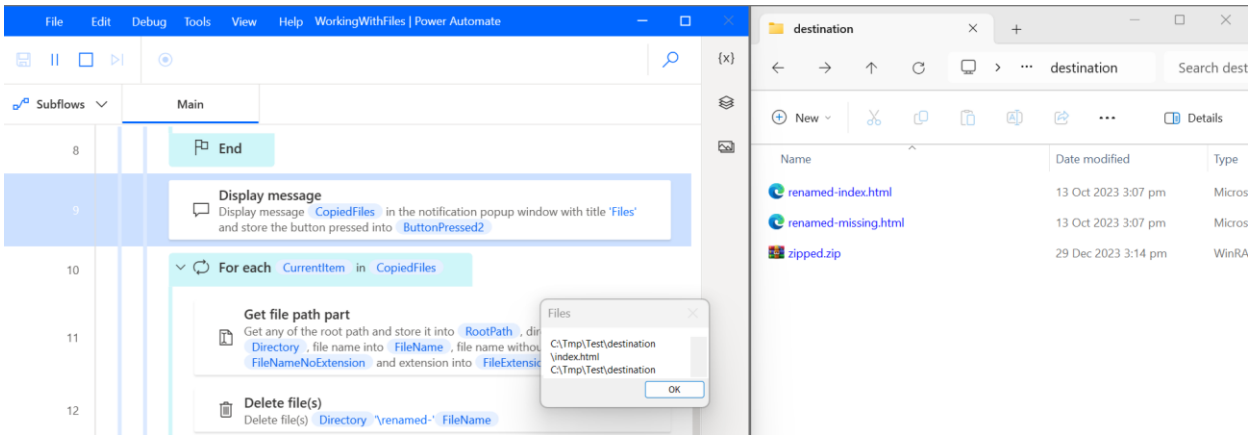

Figure 2-x: Power Automate Desktop (WorkingWithFiles Flow Execution—Partial View) and Destination Folder (During Execution—Part 2)

Let's click **OK** again to continue the flow execution. Shortly after, we will see that the renamed files have been deleted, and the compressed file has been unzipped.

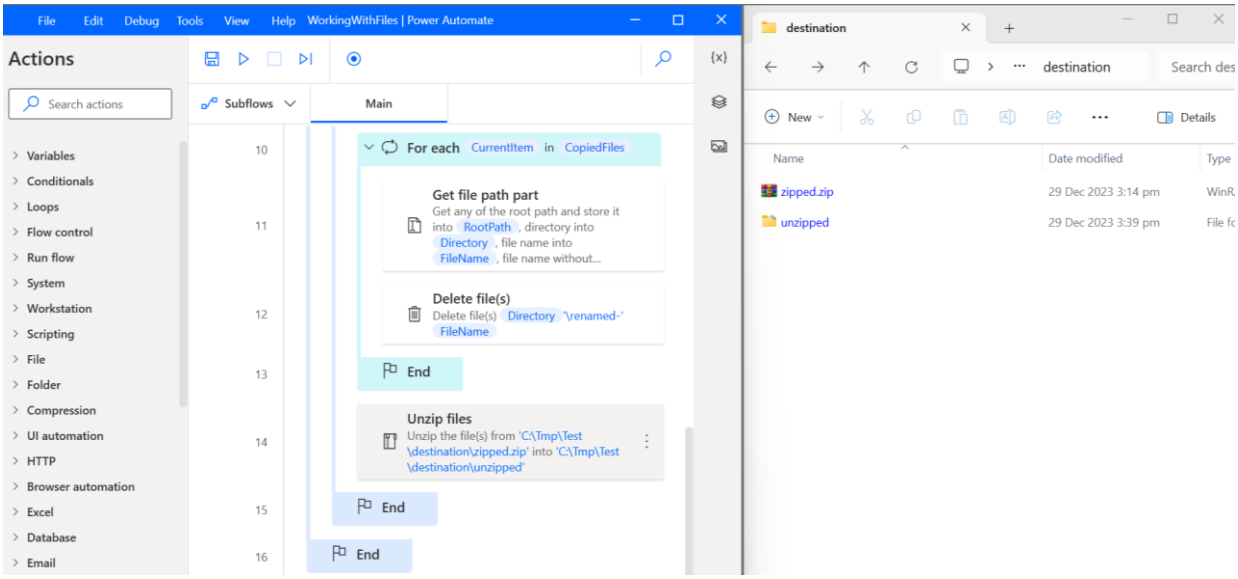

Figure 2-y: Power Automate Desktop (WorkingWithFiles Flow Execution—Partial View) and Destination Folder (During Execution—Part 3)

Ctrl+A/Ctrl+C trick

Now, here's a trick you were probably not expecting to learn. You can select all the flow actions if you press **Ctrl+A** within Power Automate Desktop.

Once all the actions are selected, press **Ctrl+C** to copy them. You can then open your text editor and paste the details copied using **Ctrl+V**.

Here is what those copied details look like.

Code Listing 2-a: Pseudo-Code Representation of the Flow

```
FUNCTION Main_copy GLOBAL
    IF (File.IfFile.Exists File: $'''C:\\Tmp\\Test\\origin\\index.html''')
THEN
        IF (File.IfFile.Exists File:
$'''C:\\Tmp\\Test\\origin\\missing.html''') THEN
            File.Copy Files: ['C:\\Tmp\\Test\\origin\\index.html',
'C:\\Tmp\\Test\\origin\\missing.html'] Destination:
$'''C:\\Tmp\\Test\\destination''' IfFileExists: File.IfExists.DoNothing
CopiedFiles=> CopiedFiles
            Compression.ZipFiles ArchivePath:
$'''C:\\Tmp\\Test\\destination\\zipped.zip''' FilesOrFoldersToZip:
CopiedFiles CompressionLevel:
Compression.CompressionLevel.BestBalanceOfSpeedAndCompression
ArchiveComment: $'''''' ZipFile=> ZipFile
            Display.ShowMessageDialog.ShowMessage Title: $'''Files'''
Message: CopiedFiles Icon: Display.Icon.None Buttons: Display.Buttons.OK
DefaultButton: Display.DefaultButton.Button1 IsTopMost: False
ButtonPressed=> ButtonPressed
            LOOP FOREACH CurrentItem IN CopiedFiles
                File.RenameFiles.RenameAddText Files: CurrentItem
TextToAdd: $'''renamed-''' TextPosition: File.AddTextPosition.BeforeName
IfFileExists: File.IfExists.Overwrite RenamedFiles=> RenamedFiles
            END
            Display.ShowMessageDialog.ShowMessage Title: $'''Files'''
Message: CopiedFiles Icon: Display.Icon.None Buttons: Display.Buttons.OK
DefaultButton: Display.DefaultButton.Button1 IsTopMost: False
ButtonPressed=> ButtonPressed2
            LOOP FOREACH CurrentItem IN CopiedFiles
                File.GetPathPart File: CurrentItem RootPath=> RootPath
Directory=> Directory FileName=> FileName FileNameWithoutExtension=>
FileNameNoExtension Extension=> FileExtension
                File.Delete Files: $'''%Directory%\\renamed-%FileName%'''
            END
            Compression.UnzipFiles ArchivePath:
$'''C:\\Tmp\\Test\\destination\\zipped.zip''' DestinationFolder:
$'''C:\\Tmp\\Test\\destination\\unzipped'''
        END
    END
END FUNCTION
```

Recap

Great! We have now seen how to work with files using Power Automate Desktop and copy, rename, delete, zip, and unzip files.

Behind the scenes, Power Automate Desktop uses a lightweight programming language called Microsoft Power Fx.

Chapter 3 Automated Data Entry

Quick intro

Now, imagine the following scenario. Your organization has set up a Google Form, which users cannot fill in themselves, and you have been tasked with filling in this form for them.

If you have to fill in this form two or three times, it's not an issue. However, it's a different story if you must manually fill in this form 100 times. That's where Power Automate Desktop can be a lifesaver.

Throughout this chapter, we will create an automated flow that can automatically submit the details of a Google Form multiple times based on information read from a text file with various records.

Customer Feedback form

Most people have a Gmail account—it's free to sign up for one if you don't have one. The Google Forms template we'll use is Customer Feedback. If you don't see the Customer Feedback form, click the **Template gallery** dropdown in the upper-right corner.

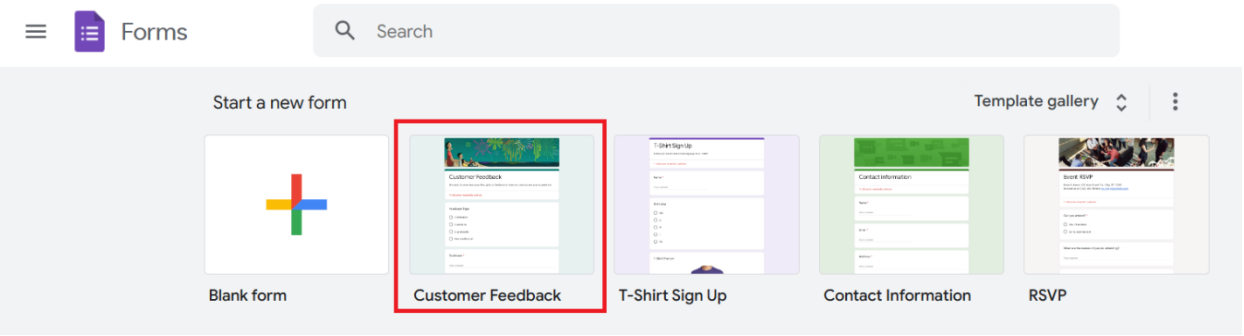

Figure 3-a: Google Forms—The Customer Feedback Form (Template Highlighted)

Click the **Customer Feedback** template to create the form. Once the form has been created, we need to get the form's URL, which we can use within Power Automate Desktop to automate the process.

To access the URL that we can use within Power Automate Desktop to automate the data entry of this form, we first need to click the **Send** button at the top of the form.

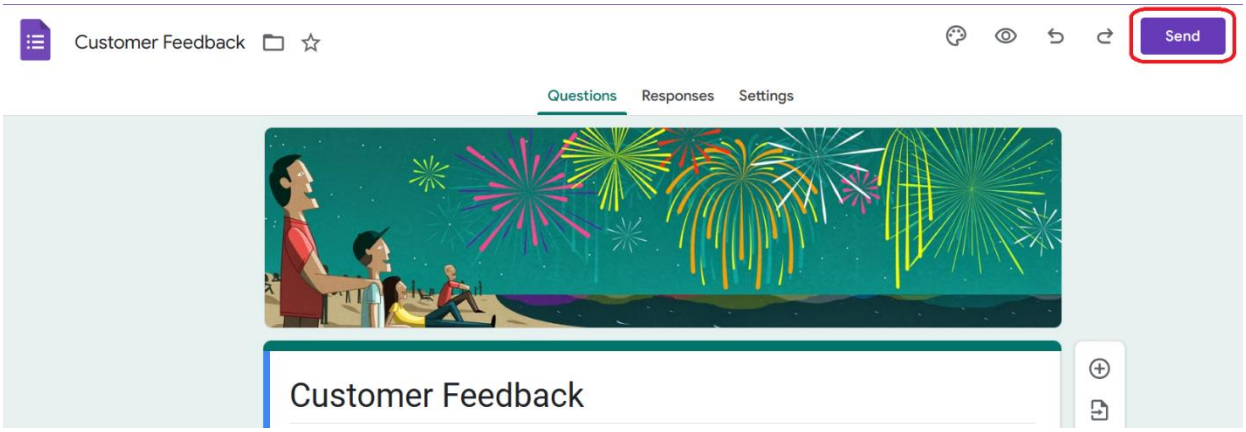

Figure 3-b: Google Forms—The Customer Feedback Form (Send Button Highlighted)

Once we click the Send button, we'll see a pop-up window; let's click the tab with the link icon to see the URL. Click **Copy** to copy the URL.

Once copied, paste it into Notepad and save it as a text file, as we'll need to use this URL later.

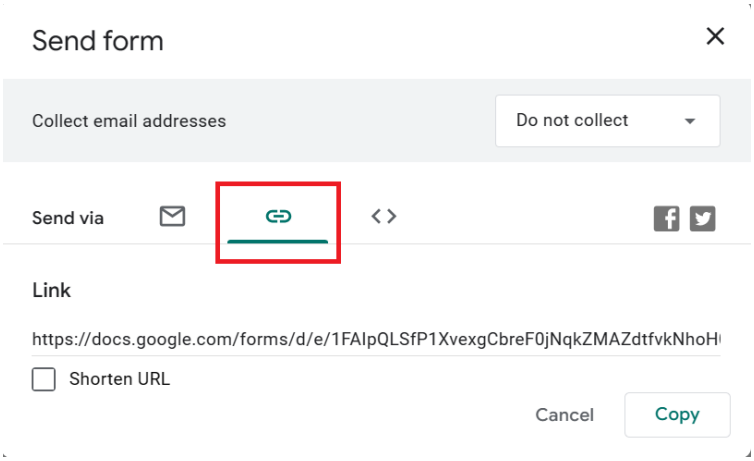

Figure 3-c: Google Forms—Send form Pop-up (Copy Link Tab Highlighted)

Now that we have copied the form's URL, let's look at the fields we need to populate. In my case, I've used the Shorten URL checkbox option and obtained the following URL:

https://forms.gle/JYhWMhc3bSNjxiQ98

Note: I strongly suggest you use the URL that your Customer Feedback form provides you rather than mine, as at some point, I will remove this form, and it is possible that by the time you are reading this book, that form and URL might not work any longer.

Form fields

Our Power Automate Desktop flow will need several input fields to populate automatically. The first is the Feedback Type field, which contains four options, as seen in the following figure.

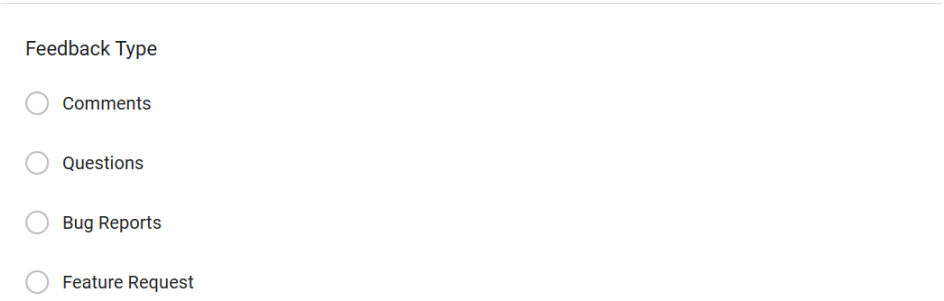

Figure 3-d: The Feedback Type Field—Customer Feedback Form

Next, we have the Feedback, Suggestions for improvement, and Name fields.

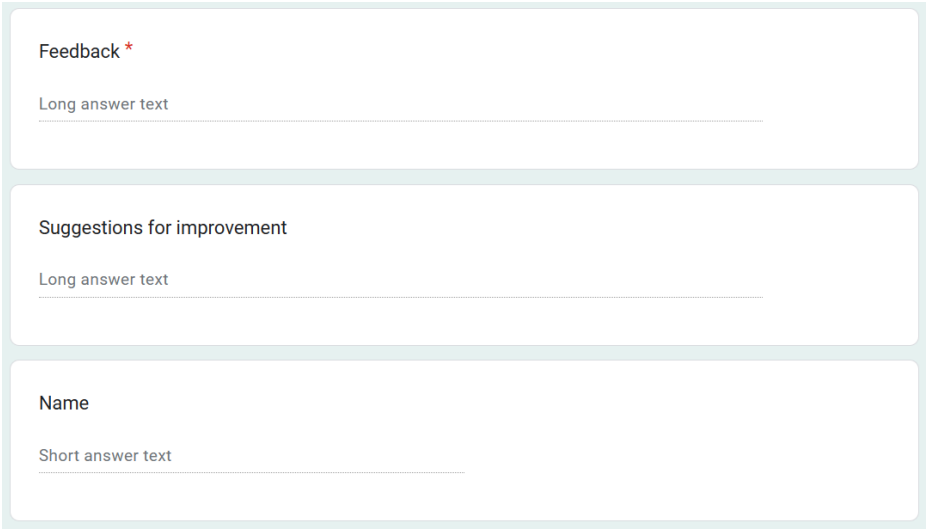

Figure 3-e: The Feedback, Suggestions for improvement, and Name Fields—Customer Feedback Form

Following these fields, we also need to populate the Email field.

Figure 3-f: The Email Field—Customer Feedback Form

Those are all the fields we need to populate with Power Automate Desktop automatically.

Creating the flow

Now that we know what fields to populate for the online form, let's create a new workflow. Let's call this flow **PopulateOnlineForm**.

Note: We've previously explored creating a new flow, so we won't cover how to do this again.

Setting the form URL

The first action we need to add to the flow is one that automatically sets the URL to the online form we want to populate. The action that does this is the Set variable action. We can find this action under Variables within the Actions pane. So, let's drag that action to the flow to add it.

Then, click on the default variable name, **NewVar** (%NewVar%), and change it to **Url** (%Url%). Let's also assign the online form URL to the **Value** text box, which you previously copied to Notepad and saved as a text file.

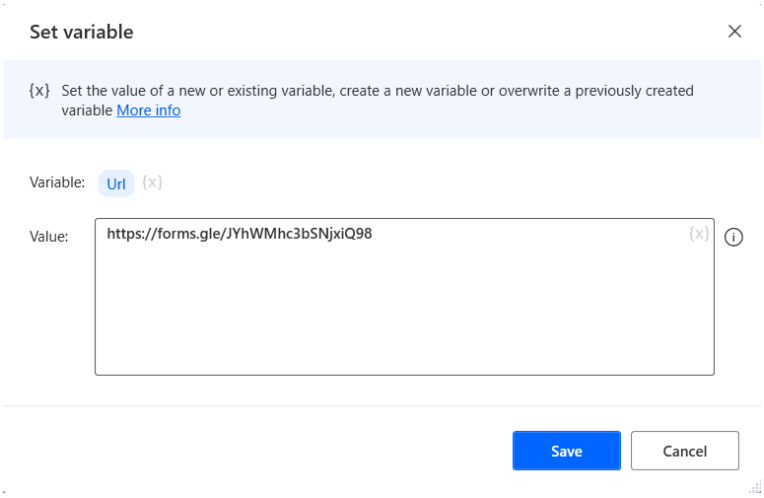

Figure 3-g: The Set variable Action—(Changed Details)

Once you have changed the default values, click **Save**.

Launch a browser instance

Now that we have set the URL for the online form that we will populate, let's launch a browser instance.

To do that, let's add the **Launch new Chrome** action to the flow, which we can find under Browser automation within the Actions pane. Let's assign the Url variable (**%Url%**) to the **Initial URL** text box and set the **Window state** property as **Maximized**.

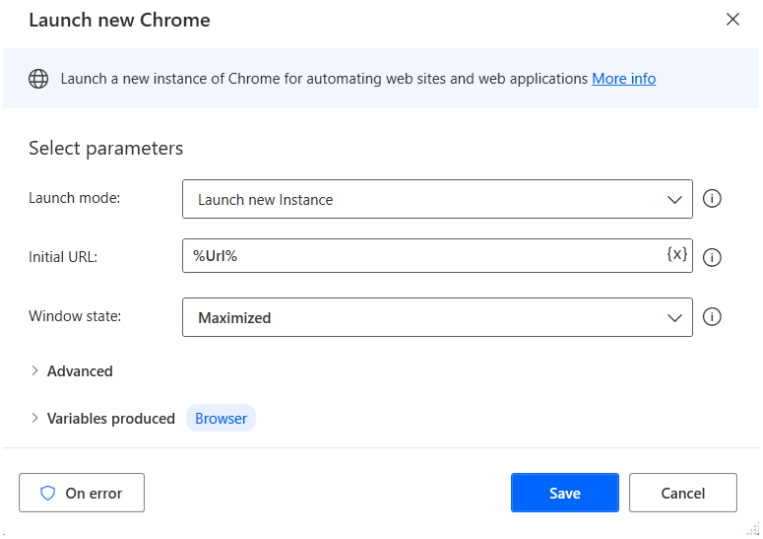

Figure 3-h: The Launch new Chrome Action—(Changed Details)

Once done, click **Save** to continue.

Note: Under Browser automation, you'll also find the Launch new Internet Explorer, Launch new Firefox, and Launch Microsoft Edge actions. In my case, I'm used to working with Chrome, so that's why I've chosen the Launch new Chrome action instead.

Loading the data to populate

To populate the online form, we need some data. I've prepared a feedback.txt file with three test records, which we can read and use to populate the form:

```
Comments|This is a comment|A great suggestion|John Diego|john.diego@learner.com
BugReports|This is a bug report|Get rid of all the bugs, please|Pepe Trueno|pepe.trueno@learner.com
FeatureRequest|This is a feature request|Make an amazing product|Juan Josh|juan.josh@learner.com
```

So, to read and load this file, we need to use the Read text from file action. We can find this action under File within the Actions pane.

Let's assign the file path of the feedback.txt file (where the data resides) to the **File path** text box.

Let's also set the value of **Store content as** to **List (each is a list item)**. We can leave the **Encoding** property with the default value of **UTF-8**.

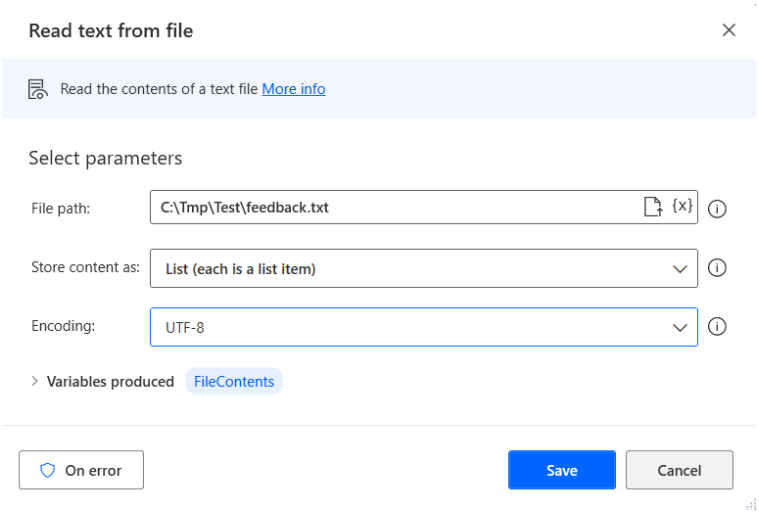

Figure 3-i: The Read text from file Action—(Changed Details)

Once done, let's click **Save**. Doing this allows us to read and load all the data from the feedback.txt file, which we'll use to populate the online form.

Iterating through the data loaded

Now that the data is loaded, we need to iterate through it. We can do this using a **For each** action. We can find the For each action under Loops within the Actions pane. So, let's add it to the flow.

Let's assign the FileContents variable (**%FileContents%**) to the **Value to iterate** text box, which contains the data loaded from the previous action.

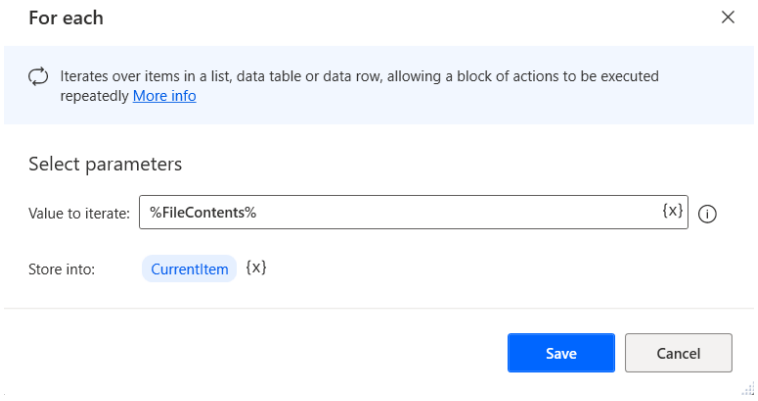

Figure 3-j: The For each Action—(Changed Details)

Each loaded record will be available within the CurrentItem (%CurrentItem%) variable. Once done, click **Save**.

Splitting each record

Looking closely at each record, we see a pipe (|) separator character between each data item. To access each data item, we need to separate each record using every occurrence of the pipe (|) character.

Comments|This is a comment|A great suggestion|John Diego|john.diego@learner.com

To do that, we need to use the **Split text** action, which we can find under Text within the Actions pane. Let's add this action to the flow.

Once the action has been added, let's assign the CurrentItem variable (**%CurrentItem%**) to the Text to split text box.

To the **Delimiter type** property, we must assign the **Custom** option, and as the **Custom delimiter**, we need to indicate the pipe (|) character.

Let's also change the **Variables produced** from the default name to **LineParts**.

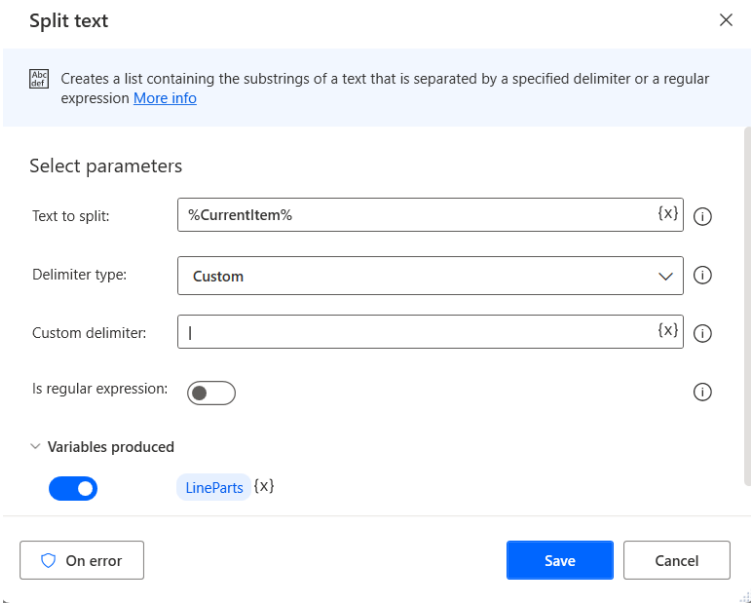

Figure 3-k: The Split text Action—(Changed Details)

Once done, click **Save**. After this action executes, we'll go from this:

Comments|This is a comment|A great suggestion|John Diego|john.diego@learner.com

To this (contained within the LineParts variable (%LineParts%)):

Comments
This is a comment
A great suggestion
John Diego

We are splitting each record into parts because each corresponds to a field within the online form we must populate.

Looping through fields

With the fields now available as LineParts (%LineParts%), we must iterate through them to determine how to assign each value to its counterpart on the online form.

To do that, we'll need to add another For each action to the flow after the Split text action we previously added. But before we add this new For each action to the flow, we need to add a Set variable action to keep track of each iteration.

Let's add the **Set variable** action to the flow. We can find this action under Variables within the Actions pane. Let's rename the **Variable** name to **Index** rather than use the default name. Let's set the **Value** to **0**.

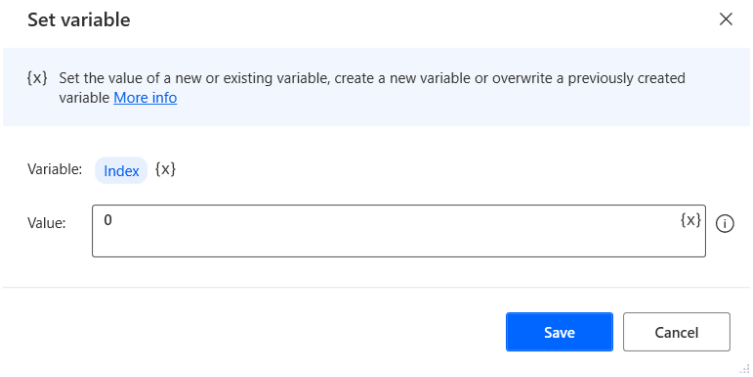

Figure 3-I: The Set variable Action—(Changed Details)

Once these values have been set, let's click **Save**. We'll use the Index variable (%Index%) later within the For each action that we will add next.

For each action

Let's add the **For each** action to the flow. We can find it under Loops within the Actions pane. We can assign the LineParts variable (**%LineParts%**) to the **Value to iterate** text box.

Let's also rename the name of the **Store into** property to **Field** instead of the default value.

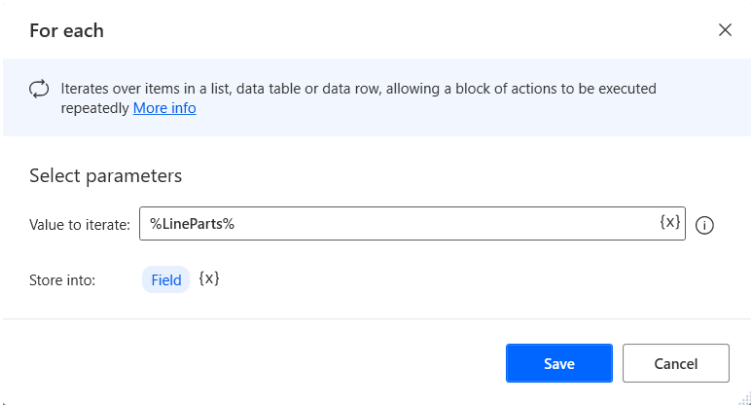

Figure 3-m: The For each Action—(Changed Details)

Once those values have been changed, click **Save**. The flow should now be as follows.

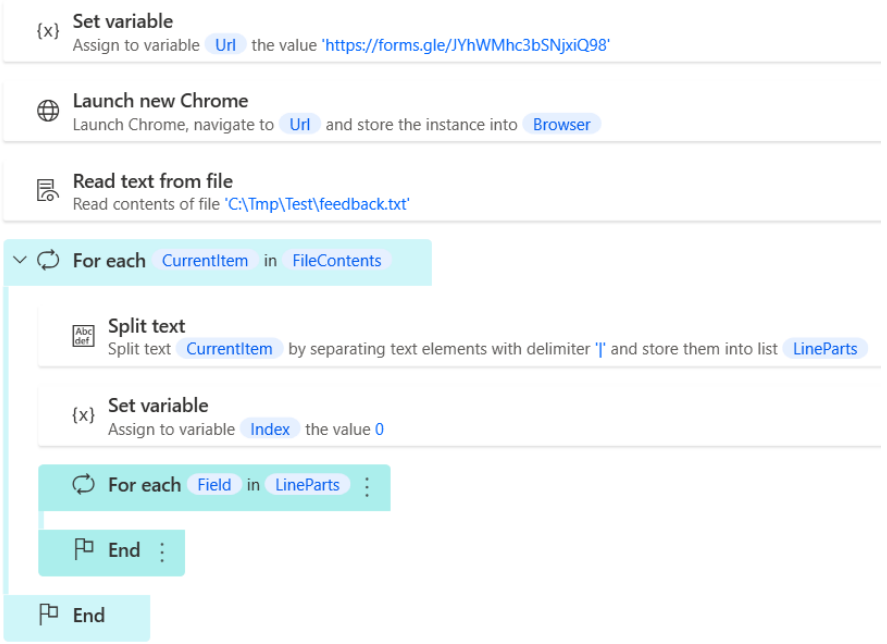

Figure 3-n: The Current Flow So Far

UI Elements tab

Now that we can iterate through each field, we must determine how to assign those values to their respective fields on the online form.

To do that, we must instruct Power Automate Desktop to find those fields on the online form and then use If and Else if actions to determine how they are populated. To identify the field on the form, we need to add the UI elements that correspond to them. We can do this by accessing the **UI elements** tab.

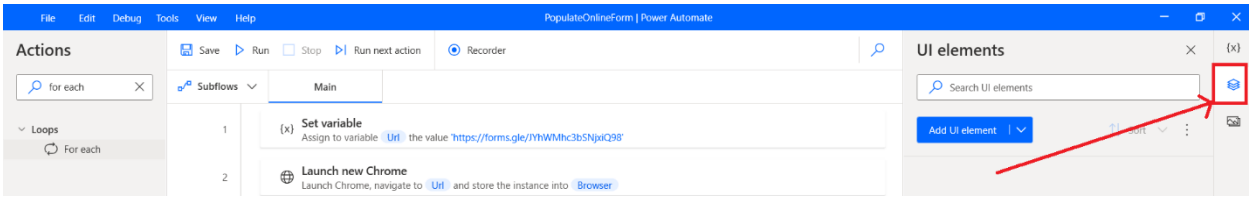

Figure 3-o: The UI elements Tab

In principle, we'll need to add at least one UI element for every text field of the form, and in the case of the multiple-choice field, we'll need to add one UI element per choice.

Online form (design mode vs. runtime mode)

Before we add any UI elements, there's one crucial thing you need to know. The online form is, by default, shown in design mode.

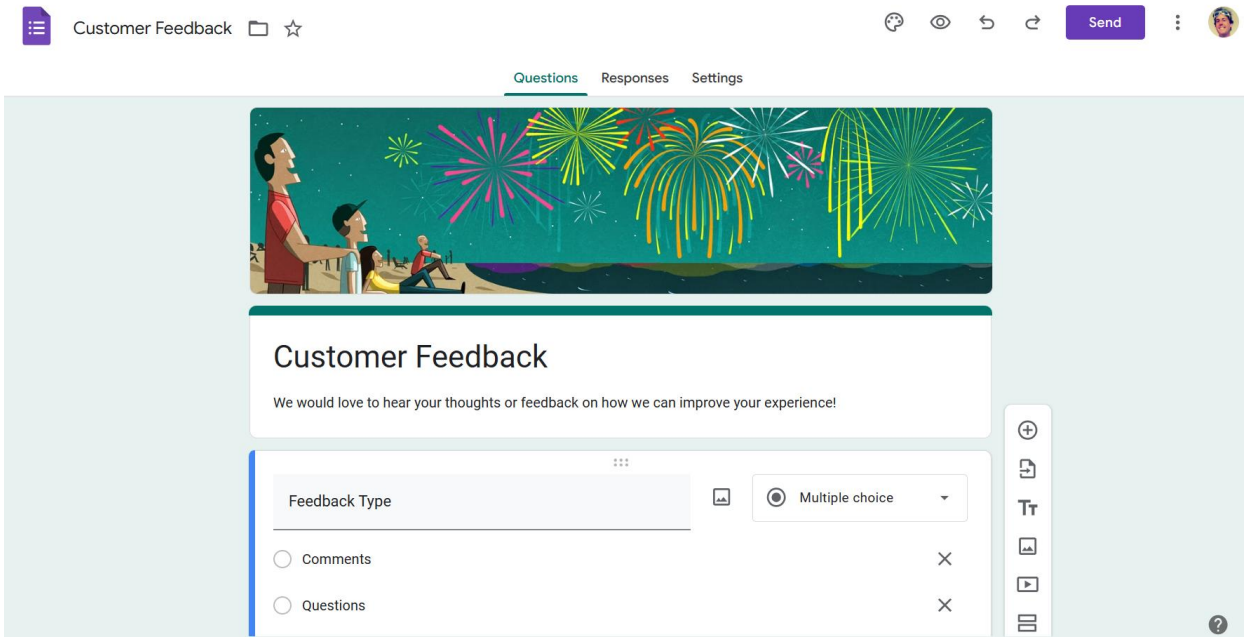

Figure 3-p: The Online Form (Design Mode)

When inspecting and adding UI elements, we never add those elements from the online form in design mode. We add them instead from the online form in runtime mode.

To access the online form in runtime mode, we must click the **Send** button and then retrieve the URL of the form, which, in the real world, we would provide to users to fill in the form manually.

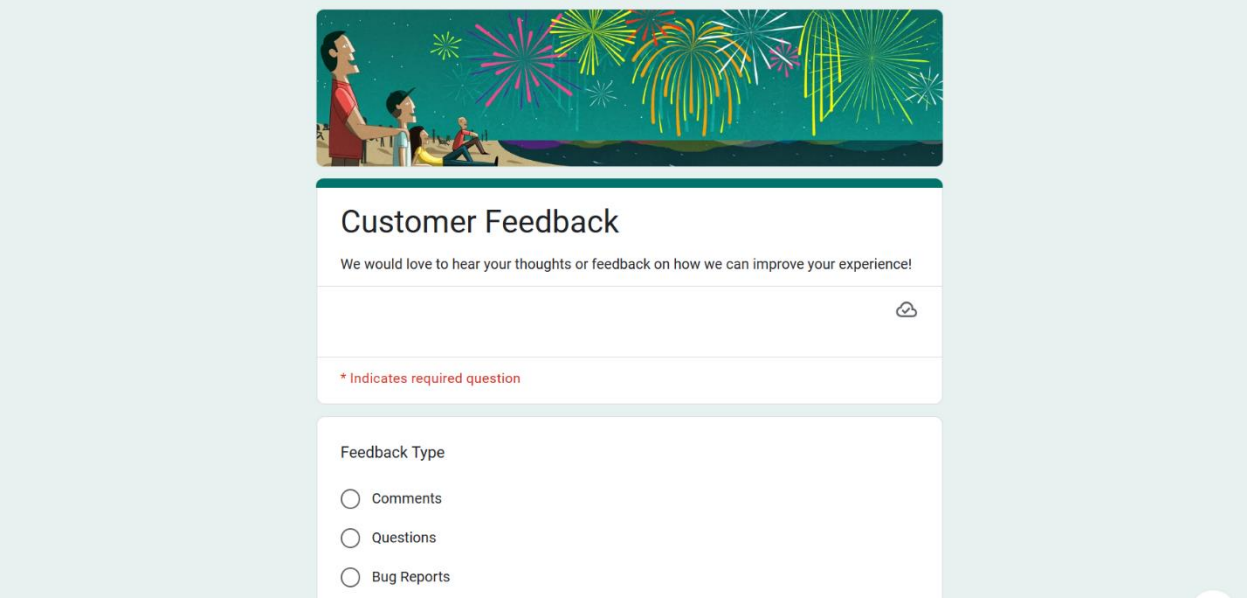

Figure 3-q: The Online Form (Runtime Mode)

We previously saw how to get this URL, so I won't cover that again. In any case, you can get this URL from the flow's initial Set variable action.

Feedback Type UI elements

The Feedback Type is a multiple-choice field with four possible answers, each representing a UI element: Comments, Questions, Bug Reports, and Feature Requests. We need to add a UI element for each choice.

First, let's ensure the online form (in runtime mode) is visible. So, using the same form URL you added to the flow's initial Set variable action, manually open a new browser instance and navigate to that URL. You should now see the runtime version of the online form that we'll be populating.

> ***Note: You must manually open the online form using the URL you added to the flow's initial Set variable action, because you must inspect the UI elements of the online form in runtime mode (when users would manually fill in the form). You mustn't check the UI elements of the online form in design mode.***

Within Power Automate Desktop, we need to click the downward-facing arrow to the right of the Add UI element button, and then click **Inspect UI elements**.

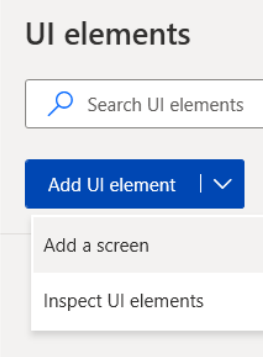

Figure 3-r: The Add UI element Options

The following UI element picker window will appear once you have clicked the Inspect UI elements option.

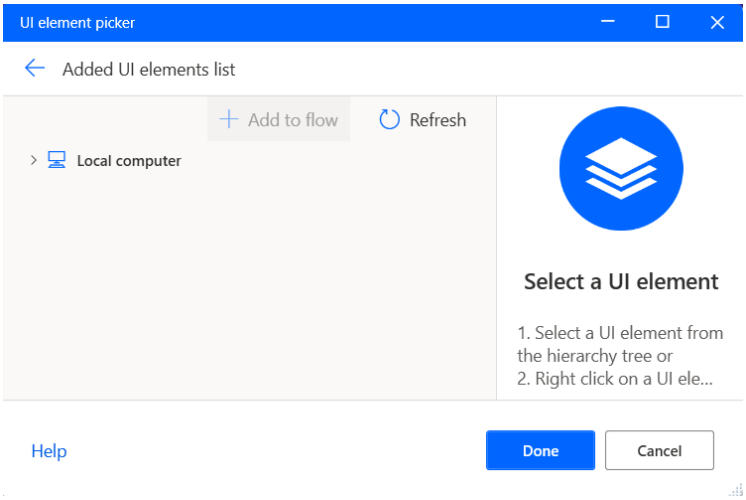

Figure 3-s: The UI element picker Window

The UI element that you'll add will be shown within this UI element picker window. So, to capture the Comments UI element, we need to hover over the **Comments** radio button, press the Ctrl key, and simultaneously do a mouse click.

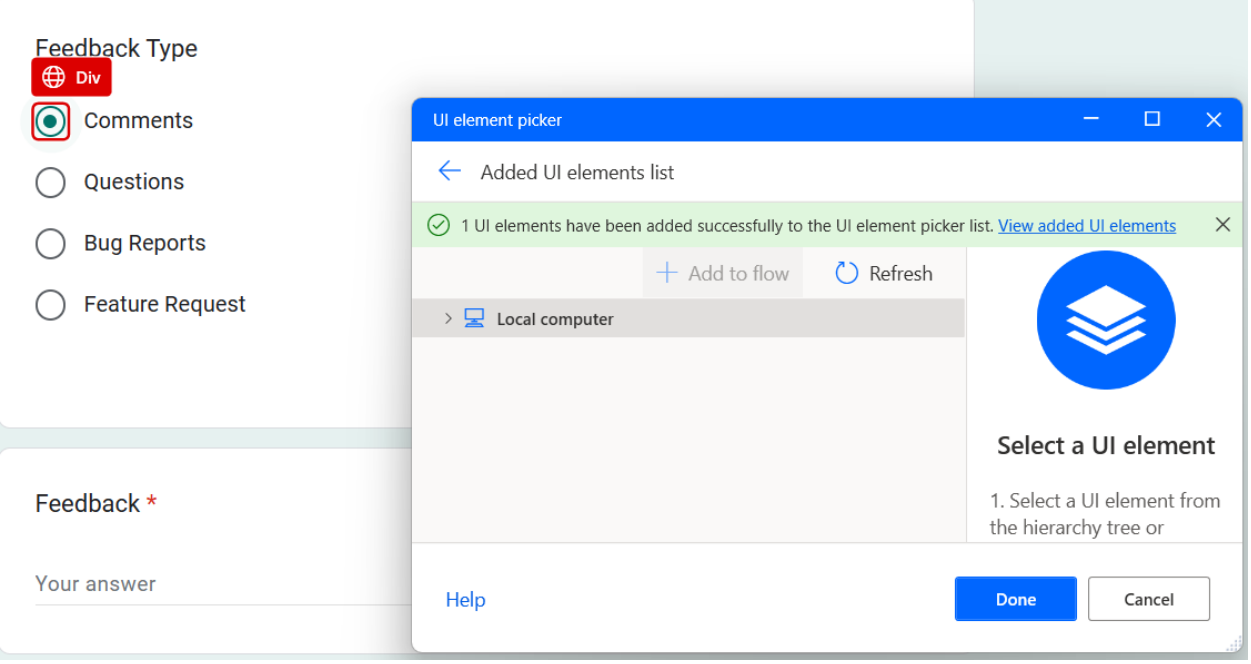

Figure 3-t: Selecting the Comments UI Element

Once done, you'll see that the UI element picker will show a message indicating that the UI element has been added to the UI element picker list. To finalize the process, click **Done**.

We'll then see the UI element on the list within the Power Automate Desktop flow window.

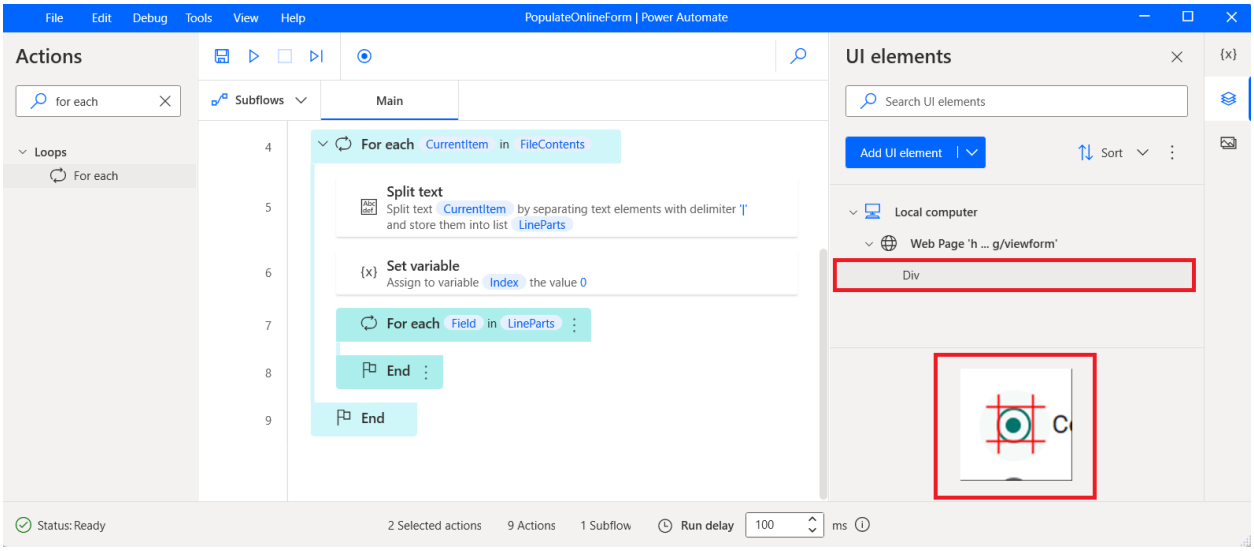

Figure 3-u: The UI Element Added (Flow Windows—UI elements Section)

So far, we've just added the Comments option as a UI element, but we still have to add the Questions, Bug Reports, and Feature Request options.

So, let's repeat the process. Click the downward-facing arrow to the right of the Add UI element button, and then click **Inspect UI elements**.

When the UI element picker appears, select the radio button corresponding to the Questions UI element, press Ctrl, and click the mouse to add the element. Then, click **Done** so the UI element appears in the list of UI elements.

Repeat those steps to add the Bug Reports and Feature Request UI elements.

Other input UI elements

Now that we have added the four UI elements corresponding to the Feedback Type field, we need to add the UI element to the Feedback field. The process is essentially the same. Click the downward-facing arrow to the right of the Add UI element button, and then click **Inspect UI elements**.

When the **UI element picker** appears, select the text box corresponding to the **Feedback** field, press the Ctrl key, and click the mouse to add the element.

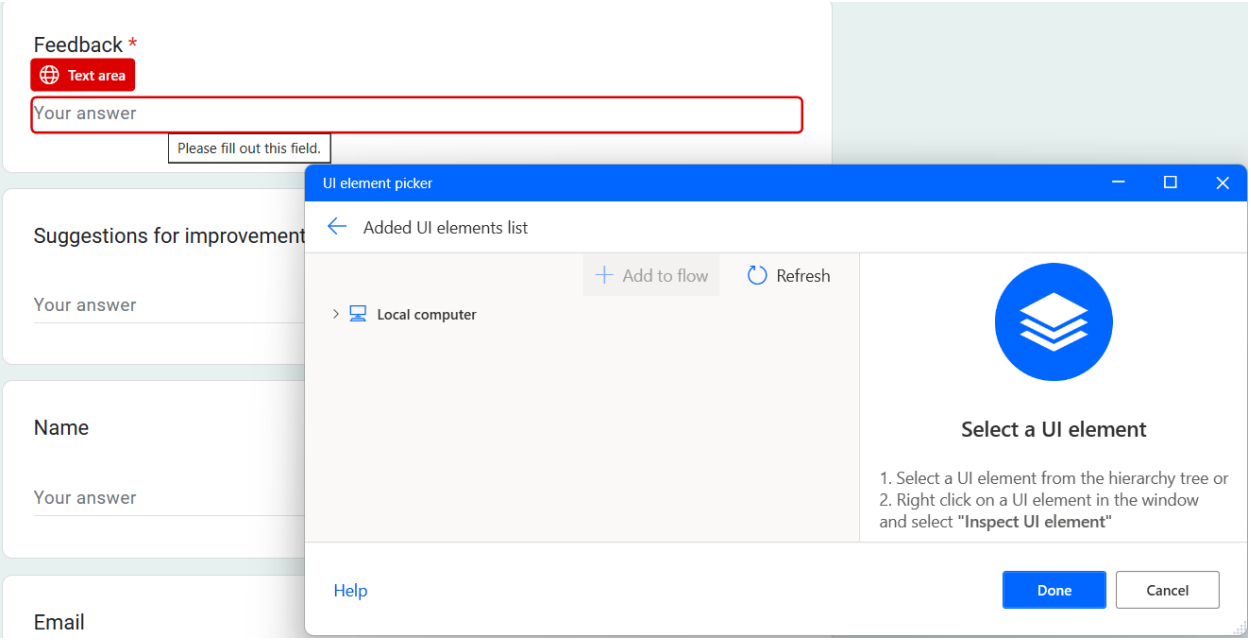

Figure 3-v: Adding the Feedback UI Element (Before Ctrl + Mouse Click)

Then, click **Done** so the UI element appears in the list of UI elements on the Power Automate Desktop flow window.

Complete the same steps to add the UI elements for the Suggestions for improvement, Name, and Email fields.

The Submit button UI element

The final UI element that we need to add is the Submit button. The steps are the same. Click the downward-facing arrow to the right of the Add UI element button and then click **Inspect UI elements**.

When the UI element picker appears, select the **Submit** button, press the Ctrl key, and click the mouse to add the element.

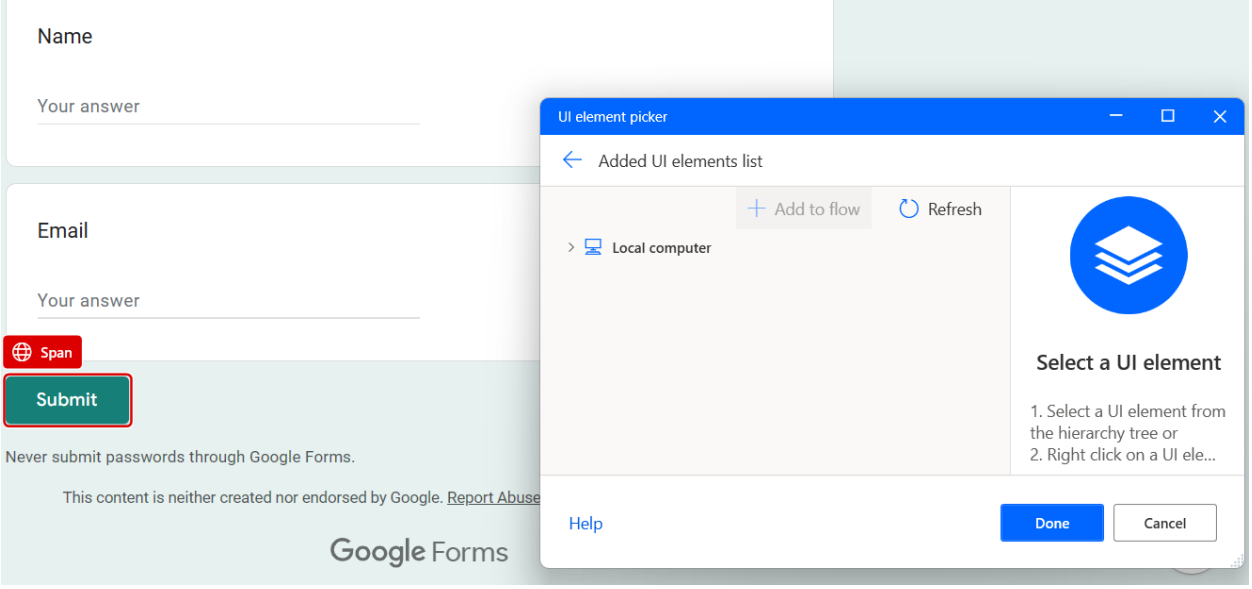

Figure 3-w: Adding the Submit Button Element (Before Ctrl + Mouse Click)

Then, click **Done** so the UI element appears in the list of UI elements on the Power Automate Desktop flow window.

Awesome! We now have all the UI elements identified, and can focus on adding the rest of the flow's functionality.

Determining the input values

Previously, we added a Set variable action to the flow, which we called Index, and assigned a value of zero to it. The purpose of this variable is to determine what input field will be automated when entering data into the form.

If the Index variable has a value of 0, we can assume that we are automating the data entry of the Feedback Type field of the form, as it's the first field that appears on the form.

If the Index variable has a value of 1, we can assume that we are automating the data entry of the Feedback field, as it's the second field that appears on the form.

If the Index variable has a value of 2, we can assume that we are automating the data entry of the Suggestions for improvements field, as it's the third field that appears on the form.

If the Index variable has a value of 3, we can assume that we are automating the data entry of the Name field, as it's the fourth field that appears on the form.

If the Index variable has a value of 4, we can assume that we are automating the data entry of the Email field, as it's the fifth field that appears on the form.

Regarding the Feedback Type field, we must evaluate if the selected UI element corresponds to the Comments, Questions, Bug Reports, or Feature Request option. So, based on this logic, we need to add two levels of conditionals to the flow when looping through the LineParts variable (%LineParts%).

The outer level evaluates the value of the Index variable, and the inner level only applies to the Feedback Type field and checks what type of option was selected for the Feedback Type.

Outer-level conditions

Let's add an **If** action to the flow and place it within the second **For each** action that iterates through the LineParts variable (%LineParts%). The If action is under Conditionals within the Actions pane.

We can assign the Index variable's (**%Index%**) value to the **First operand** and **0** to the **Second operand**. Once done, click **Save**. This If action evaluates whether we are on the Feedback Type field.

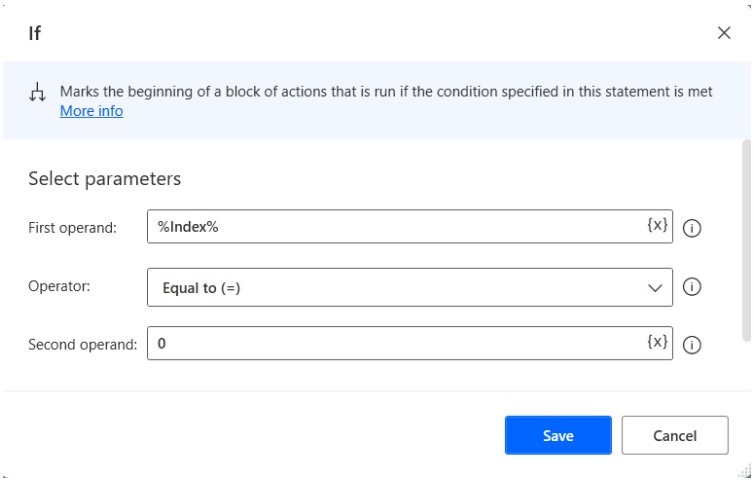

Figure 3-x: If Action (Changed Details)

Next, let's add an **Else if** action and place it after the If action we just added. The Else if action is also under Conditionals within the Actions pane.

We can also assign the Index variable's (**%Index%**) value to the **First operand** and **1** to the **Second operand**. Once done, click **Save**.

This If action evaluates whether we are on the Feedback field.

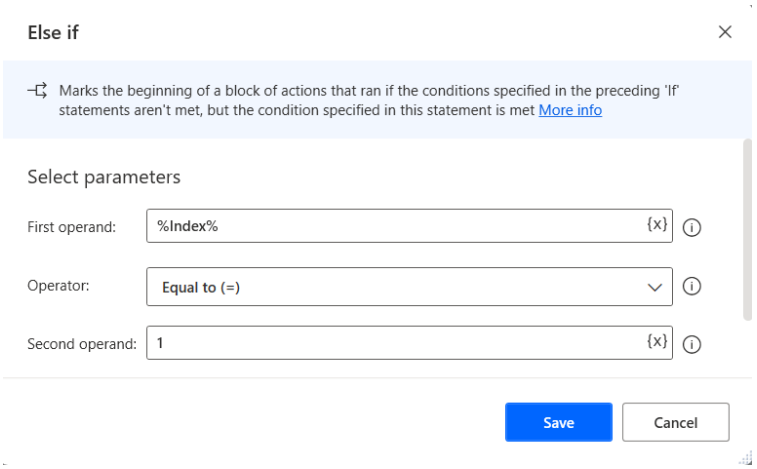

Figure 3-y: Else if Action (Changed Details)

Next, let's add another **Else if** action and place it after the Else if action we just added. We can also assign the Index variable's (**%Index%**) value to the **First operand** and **2** to the **Second operand**. Once done, click **Save**.

This Else if action evaluates whether we are on the Suggestions for improvement field.

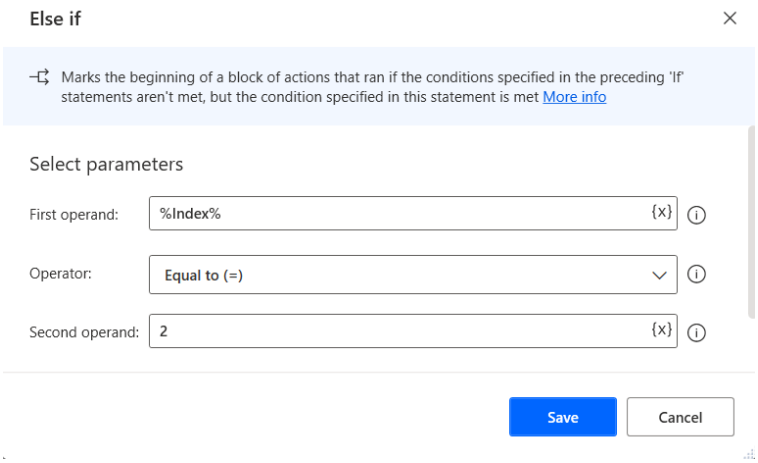

Figure 3-z: Another Else if Action (Changed Details)

Next, let's add another **Else if** action and place it after the Else if action we just added. We can also assign the Index variable's (**%Index%**) value to the **First operand** and **3** to the **Second operand**. Once done, click **Save**.

This Else if action evaluates whether we are on the Name field.

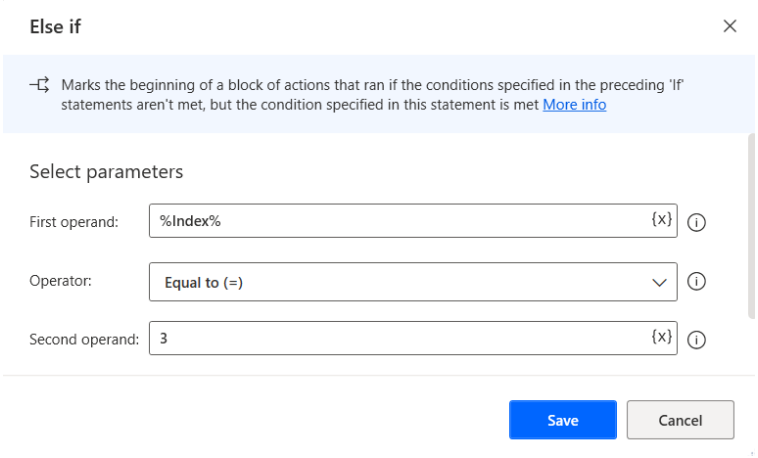

Figure 3-aa: Another Else if Action (Changed Details)

Finally, let's add another **Else if** action and place it after the Else if action we just added. We can also assign the Index variable's (**%Index%**) value to the **First operand** and **4** to the **Second operand**. Once done, click **Save**.

This Else if action evaluates whether we are on the Email field.

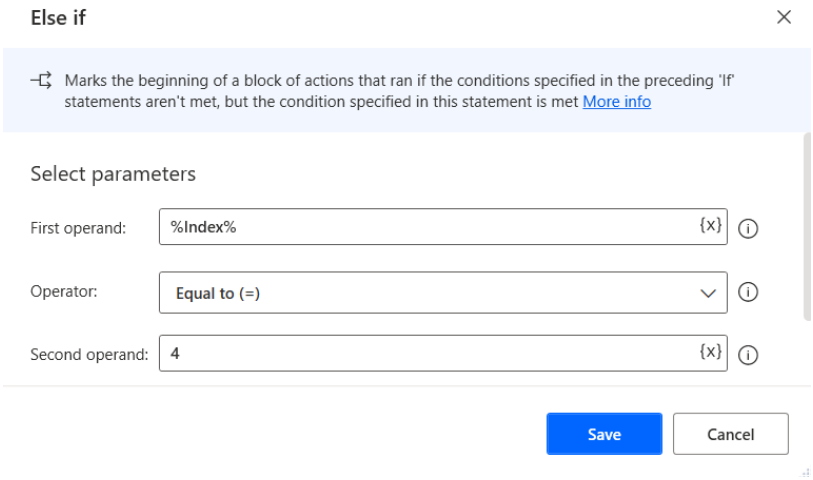

Figure 3-ab: Another Else if Action (Changed Details)

After adding these conditions, the flow should now look as follows.

Figure 3-ac: The Updated Flow (Outer-Level Conditions)—Partial View

Now that we have the outer-level conditions defined, let's add the conditions required to evaluate inner-level conditions, which will be used to determine the choice for the Feedback Type field.

Inner-level conditions

Let's add another **If** action, and this time, place it within the **If** action where we check if the Index variable (%Index%) equals zero (0).

Let's assign the Field variable (**%Field%**) to the **First operand** and **Comments** to the **Second operand**. By doing this, we are checking whether the Feedback Type field has the Comments option as the selected choice. Once done, click **Save**.

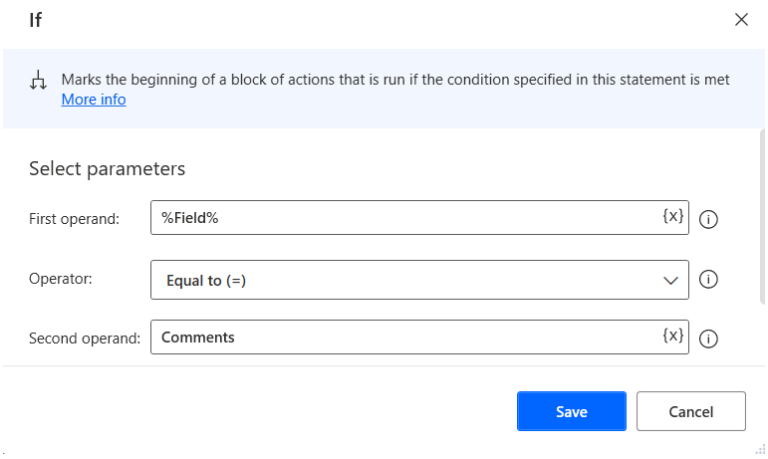

Figure 3-ad: Inner-Level If Action (Changed Details)

Now, let's add an **Else if** action right after the If action we just added. This action will be used to check whether the choice for the Feedback Type field is the Questions option.

Let's assign the value of the Field variable (**%Field%**) to the **First operand** and **Questions** to the **Second operand**. Once done, click **Save**.

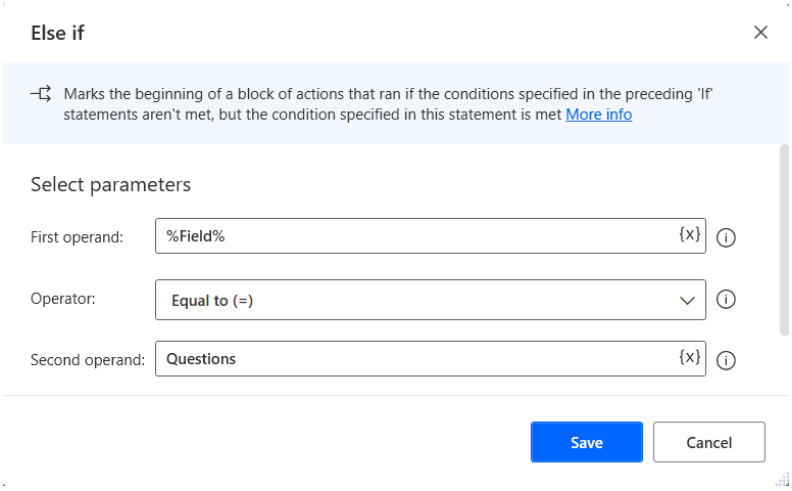

Figure 3-ae: Inner-Level Else if Action (Changed Details)

Moving on, let's add another **Else if** action to the flow after the Else if action we just added. This action will be used to check whether the choice for the Feedback Type field is the Bug Reports option.

Let's assign the value of the Field variable (**%Field%**) to the **First operand** and **BugReports** to the **Second operand**. Once done, click **Save**.

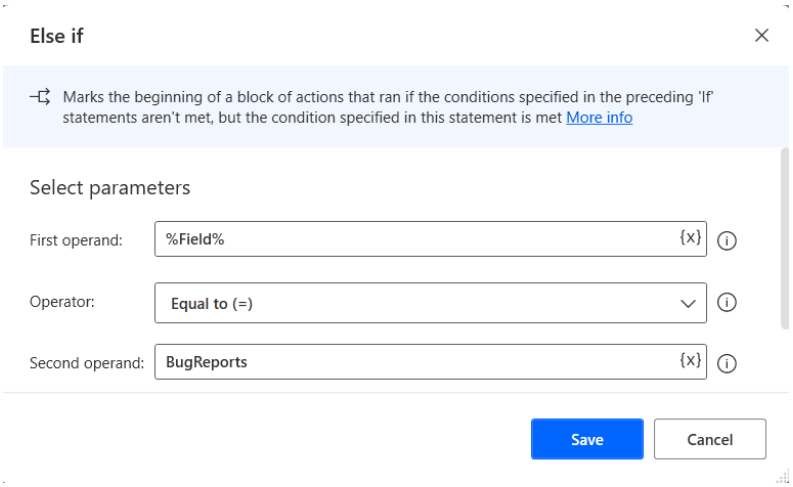

Figure 3-af: Inner-Level Else if Action (Changed Details)

Next, let's again add another **Else if** action to the flow after the Else if action we just added. This action will be used to check whether the choice for the Feedback Type field is the Feature Request option.

Let's assign the value of the Field variable (**%Field%**) to the **First operand** and **FeatureRequest** to the **Second operand**. Once done, click **Save**. If we now look at the flow, it should be as follows.

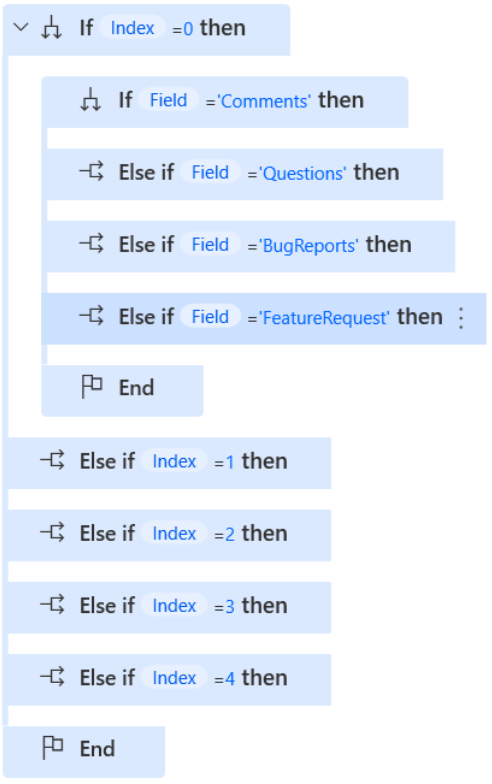

Figure 3-ag: Updated Flow—Partial View

Populating the Feedback Type field

Now that we have defined the logic required to automate how the form fields should get populated, let's add the actions that populate those fields. We'll get started by populating the Feedback Type field.

Add the **Select radio button on web page** action to the flow to do that. This action is under Browser automation and Web form filling within the Actions pane. Let's place it within the **If** action that evaluates if the Field variable equals Comments.

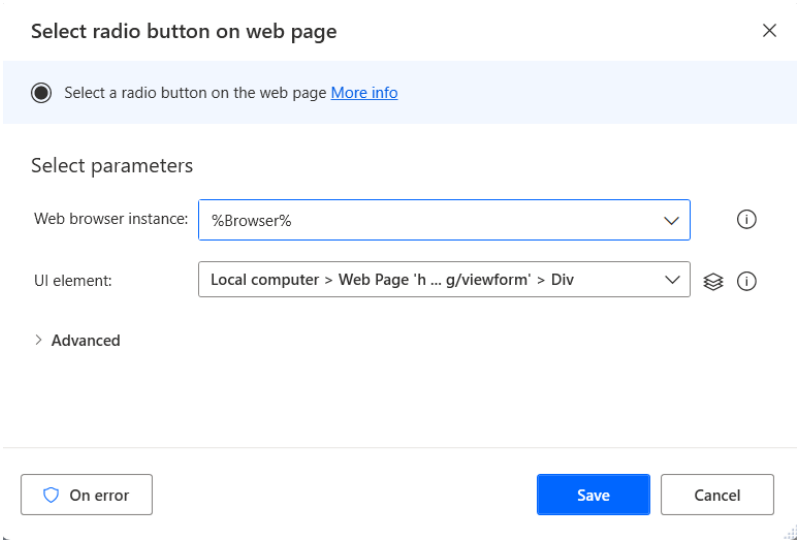

Figure 3-ah: The Select radio button on web page Action (Comments Option)

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

However, the UI element property is not populated by default. You'll need to click the downward arrow on the dropdown and select the UI element corresponding to the radio button (option) we want to populate automatically. In this case, we want to set the UI element corresponding to the Comments option (from the list of available UI elements), which, in my case, is the **Div** element highlighted in Figure 3-ai.

Note: The names of your UI elements might differ from mine. For example, the Div element highlighted in the following figure might have a completely different name in your case. Although Power Automate Desktop takes the names of the UI elements from the names of the web form elements, those UI element names might not always match the names of the web form elements.

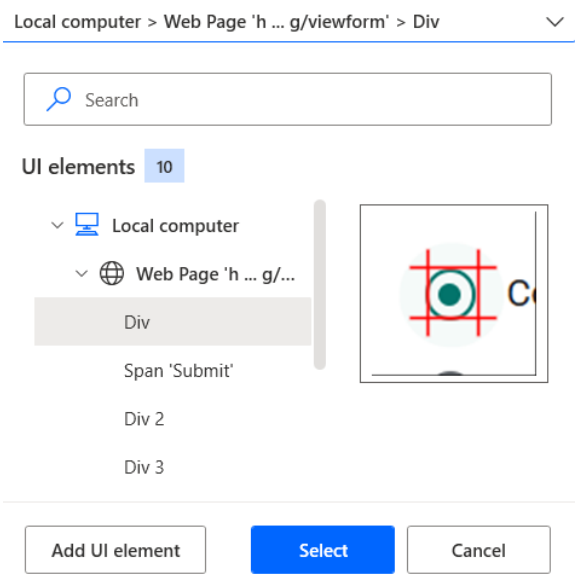

Figure 3-ai: The List of Available UI Elements (When Setting the UI element property)

Now that we have added the Select radio button on web page action corresponding to the Comments option, we need to add another **Select radio button on web page** action to match it to the **Questions** option. Let's place it within the **Else if** action that evaluates if the Field variable equals Questions.

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

However, the UI element property is not populated by default. You'll need to click the downward arrow on the dropdown and select the UI element corresponding to the Questions option, which, in my case, is the **Div 2** element highlighted in the following figure.

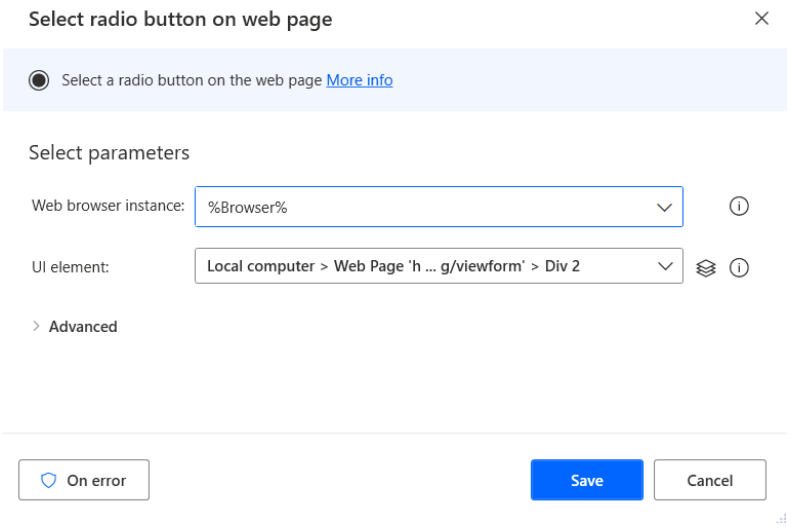

Figure 3-aj: The Select radio button on web page Action (Questions Option)

Let's add another **Select radio button on web page** action to match it to the **Bug Reports** option. Let's place it within the **Else if** action that evaluates if the Field variable equals BugReports.

Note: The reason the Else if action checks if the Field variable equals BugReports and not Bug Reports is because, within the text file that contains the data that we will use to populate the online form, the value is stored as BugReports, whereas Bug Reports refers to the label of the field on the online form.

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

The UI element property is not populated by default. You'll need to click the downward arrow on the dropdown and select the UI element corresponding to the Bug Reports option, which, in my case, is the **Div 3** element.

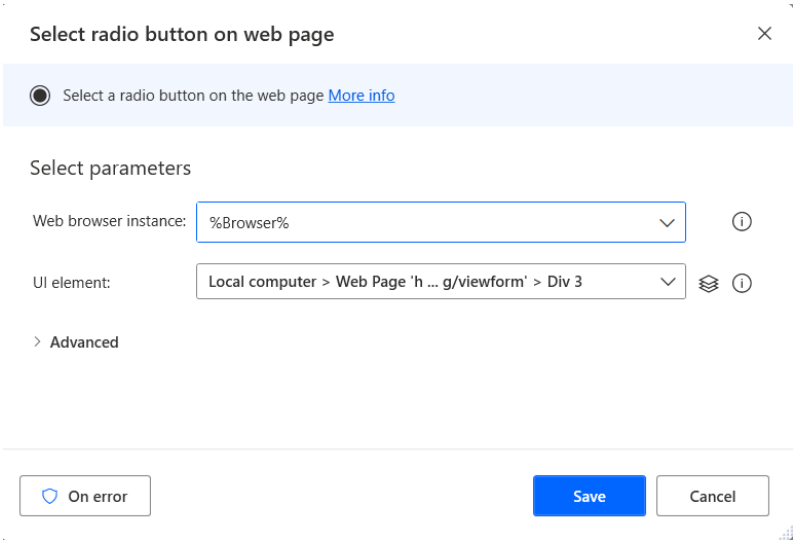

Figure 3-ak: The Select radio button on web page Action (Bug Reports Option)

Finally, let's add another **Select radio button on web page** action to match it to the **Feature Request** option. Let's place it within the **Else if** action that evaluates if the Field variable equals FeatureRequest.

Note: The reason the Else if action checks if the Field variable equals FeatureRequest and not Feature Request is because, within the text file that contains the data that we will use to populate the online form, the value is stored as FeatureRequest. Feature Request refers to the label of the field on the online form.

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

The UI element property is not populated by default. You'll need to click the downward arrow on the dropdown and select the UI element corresponding to the Feature Request option, which, in my case, is the **Div 4** element.

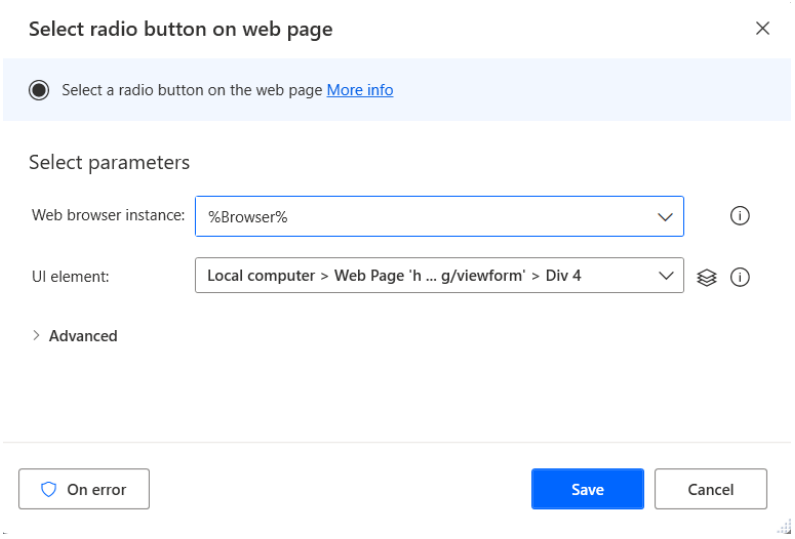

Figure 3-al: The Select radio button on web page Action (Feature Request Option)

Here's what the conditions for the Feedback Type field look like.

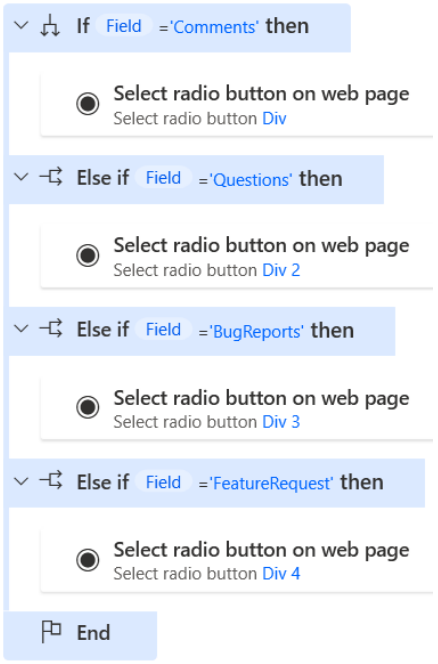

Figure 3-am: The Conditions for Populating the Feedback Type Field

Populating the Feedback field

Now, let's add a **Populate text field on web page action** within the **Else if** action that checks if the value of Index equals **1**. We can find this action under Browser automation and Web form filling within the Actions pane.

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

The UI element property is not populated by default. You must click the downward arrow on the dropdown and select the UI element corresponding to the Feedback text box.

To the **Text** property, we assign the value of the Field variable (**%Field%**). Once done, click **Save**.

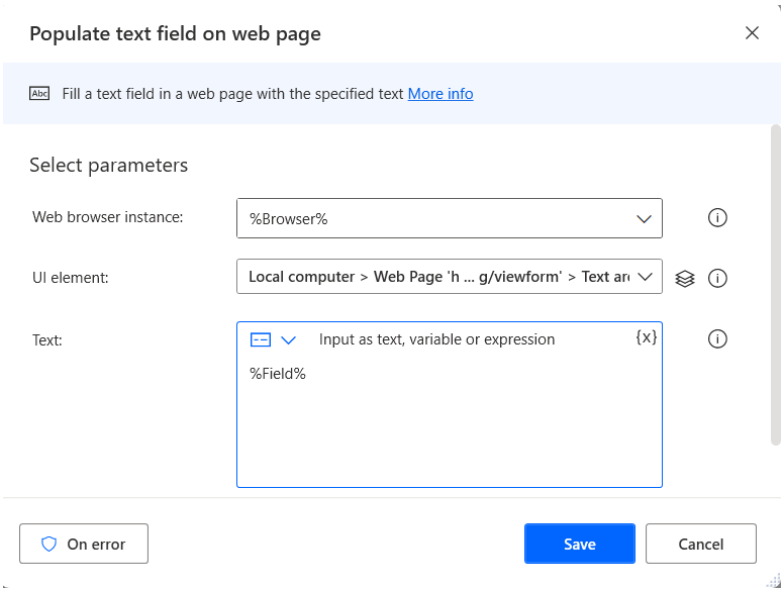

Figure 3-an: The Populate text field on web page Action (Feedback Field)

Populating the Suggestions field

The next thing we must do is to populate the Suggestions for improvement field of the online form. To do that, we must add a **Populate text field on web page action** within the **Else if** action that checks if the value of Index equals 2. We can find this action under Browser automation and Web form filling within the Actions pane.

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

The UI element property is not populated by default. You must click the downward arrow on the dropdown and select the UI element corresponding to the Suggestions for improvement text box.

To the **Text** property, we assign the value of the Field variable (**%Field%**). Once done, click **Save**.

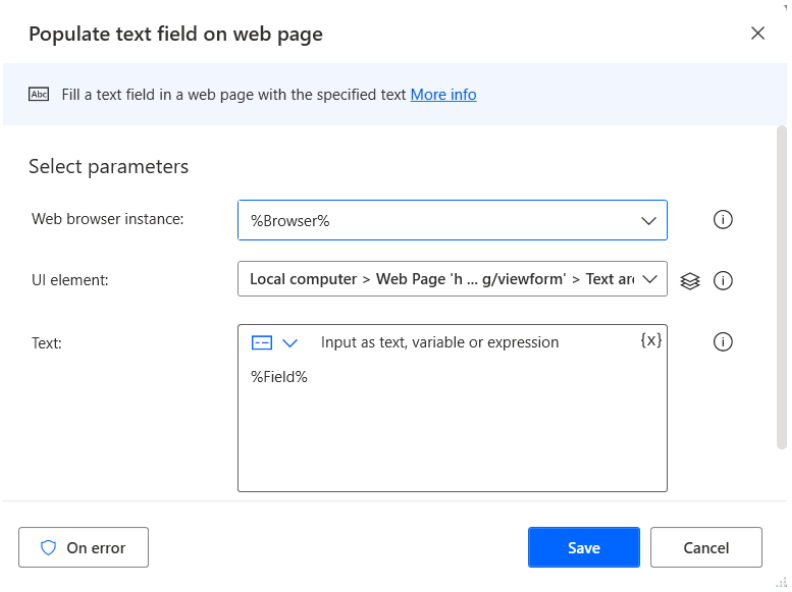

Figure 3-ao: The Populate text field on web page Action (Suggestions for improvement Field)

Populating the Name field

Next, we must populate the Name field on the online form. We can do this by adding another **Populate text field on web page action** within the **Else if** action that checks if the value of Index equals 3.

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

The UI element property is not populated by default. You must click the downward arrow on the dropdown and select the UI element corresponding to the Name text box.

To the **Text** property, we assign the value of the Field variable (**%Field%**). Once done, click **Save**.

Populating the Email field

The last form field we must populate is the Email field. We can do this by adding another **Populate text field on web page action** within the **Else if** action that checks if the value of Index equals 4.

Again, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

The UI element property is not populated by default. You must click the downward arrow on the dropdown and select the UI element corresponding to the Email text box.

To the **Text** property, we assign the value of the Field variable (**%Field%**). Once done, click **Save**.

Following that, these conditions should now appear on the flow as follows.

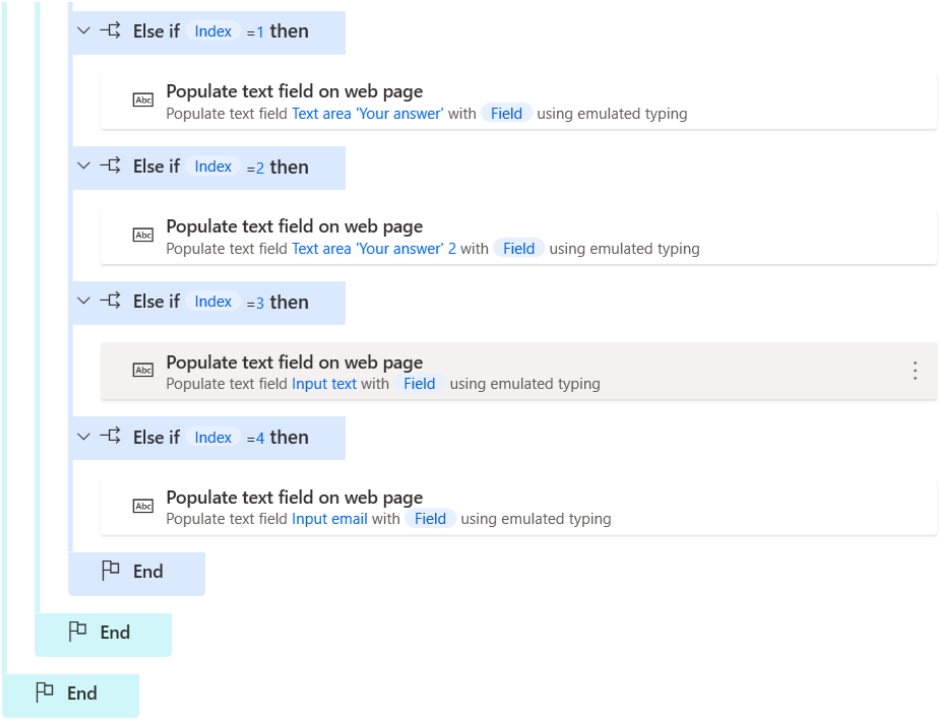

Figure 3-ap: The Conditions for Populating the Remaining Fields

Increasing the Index variable

During each iteration, we can only populate one online form field. So, after every iteration, we must increase the value of the Index variable. We do this so that the following field on the online form can be populated in the next iteration.

So, let's add the **Increase variable** action to the flow and place it after the End that belongs to the last Else if action we added to the flow. The Increase variable action can be found under Variables within the Actions pane.

Let's assign the value of the Index variable (**%Index%**) to the **Variable name** property and set the value of the **Increase by** property to **1**. Once done, click **Save**.

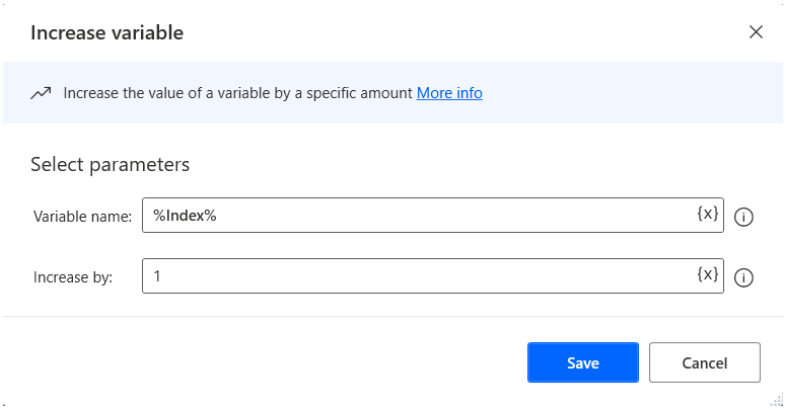

Figure 3-aq: The Increase variable Action (Changed Details)

After adding this action, the flow should now look as follows.

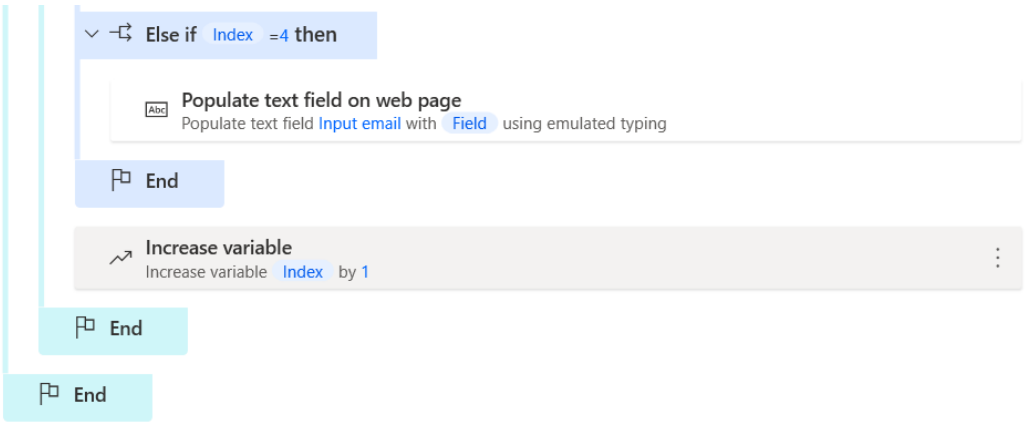

Figure 3-ar: The Updated Flow (Partial View)

Submitting the form

Now that we have populated all the online form fields, we must submit the form. To do that, let's add the **Press button on web page** action to the flow. The Press button on web page action can be found under Browser automation and Web form filling within the Actions pane.

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property.

The UI element property is not populated by default. You must click the downward arrow on the dropdown and select the UI element corresponding to the Submit button. Once done, click **Save**.

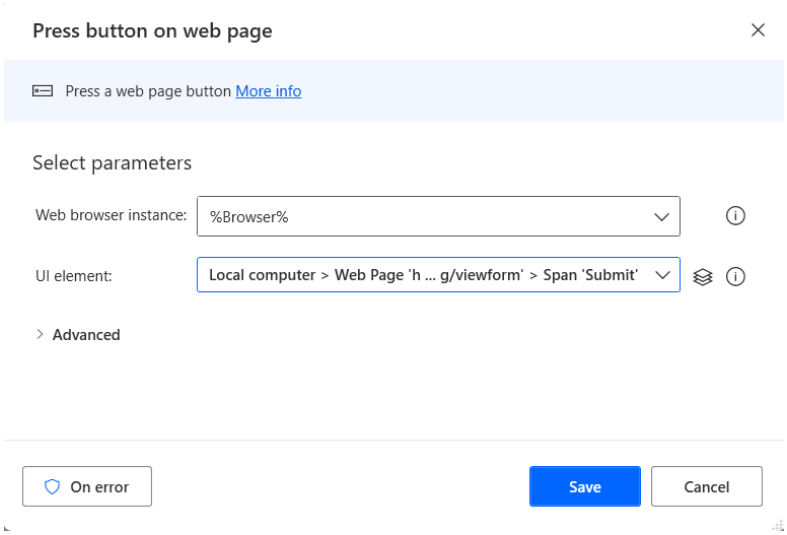

Figure 3-as: The Press button on web page Action

Moving to the next record

At this stage, we have populated all the online form fields; however, we've only managed to do it for one record. So, how can we replicate this automated process for other records? The answer is more straightforward than you would expect.

To move on to the next record and populate the online form with new data, we must add the **Go to web page** action to the flow after the End of the inner (second) For each loop.

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property. The Navigate property also has a default value, which is To Url. This means that this action, by default, will navigate to any URL we specify.

So, to the **URL** property, we can assign the value of the Url variable (**%Url%**)—this is the URL of the online form in runtime mode, which was added initially to the flow using the Set variable action. Once done, click **Save**.

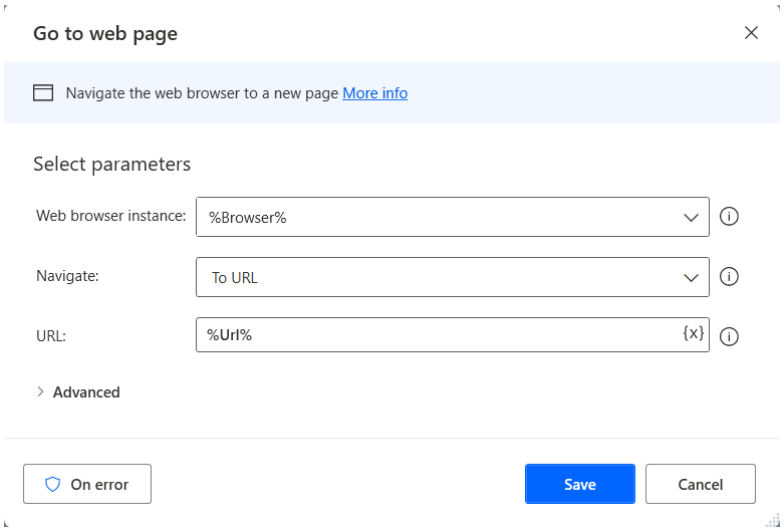

Figure 3-at: The Go to web page Action

After adding this action, the flow now looks as follows.

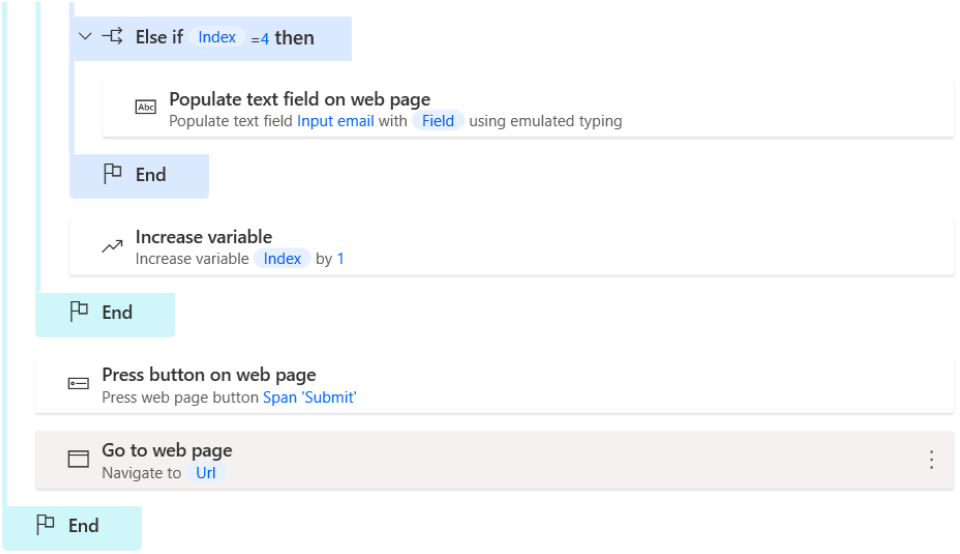

Figure 3-au: The Updated Flow (Partial View)

We can now populate the online form for multiple records using this Go to web page action.

Closing the Browser instance

One final thing we must do before finishing the flow is to close the browser instance we opened at the beginning. To do that, let's add the **Close web browser** action to the end of the flow.

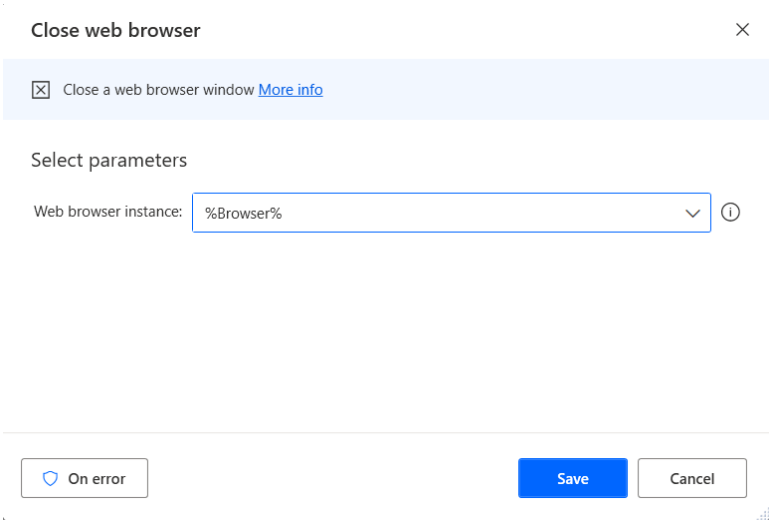

Figure 3-av: The Close web browser Action

By default, when the action opens, the value of the Browser variable (%Browser%) is assigned to the Web browser instance property. Once done, click **Save**.

Well done! We have finished the flow, and this is how it looks.

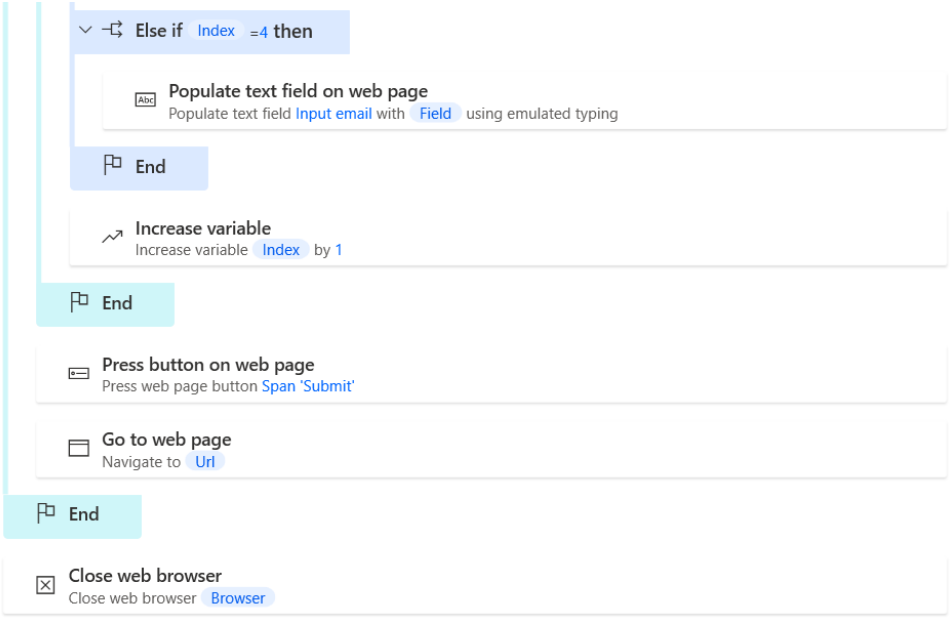

Figure 3-aw: The Updated Flow (Partial View)

Running the flow

Now that we have the flow ready, let's run the flow. The flow will read the information in the feedback.txt file and use that to populate the online form for each row found within the feedback.txt file. When the flow execution begins, a new browser window will be opened, showing the online form, and the data will be automatically added.

Before running the flow, let me show you the admin view of the customer feedback form. Notice how it doesn't contain any data or responses.

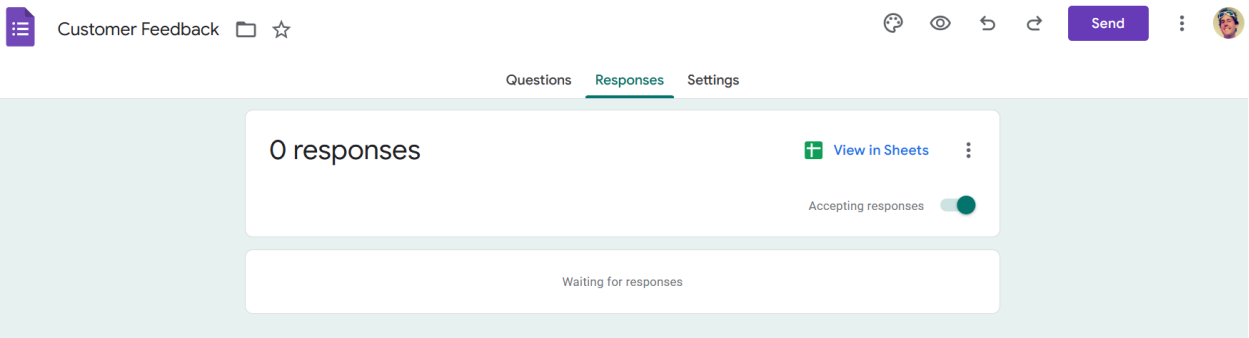

Figure 3-ax: The Customer Feedback Form (Admin View—No Responses)

Once the flow executes, the admin view will show the data entered automatically. Let's run the flow by clicking the **Run** button within Power Automate Desktop.

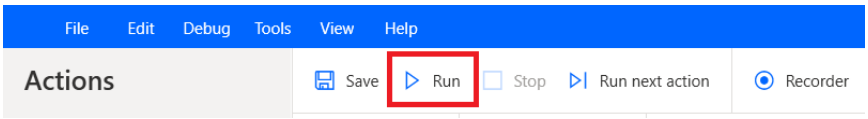

Figure 3-ay: The Run Button (Power Automate Desktop)

Shortly after this, you should see that the browser opens, and the data is automatically entered into the online form. After the execution, you should see the results in the online form admin view.

Figure 3-az: The Customer Feedback Form (Admin View—3 Responses)

Recap

So, throughout this chapter, we've created an excellent and helpful flow, which takes rows from a text file and automatically enters them into an online form. A similar logic could be used to enter data from an Excel file into an online CRM, such as Salesforce or another system.

Chapter 4 PDF Data Extraction

Quick intro

One helpful technique is extracting data from a PDF and using that data elsewhere. PDFs are incredibly common everywhere, and there's a wealth of business data in PDFs, so extracting data from them is crucial for business operations, such as invoice processing.

In this chapter, I will show you a flow that I use to extract total amounts from PDF invoices and add those amounts together.

As we have gone through the steps of creating a step-by-step flow in the previous chapter, rather than creating a flow from scratch in this chapter, let's review the steps for a PDF processing flow I already have. It reads total amounts from invoices, adds these values, and saves the total amount to an Excel file.

Flow overview

Let's look at the PDF flow to understand what activities it includes.

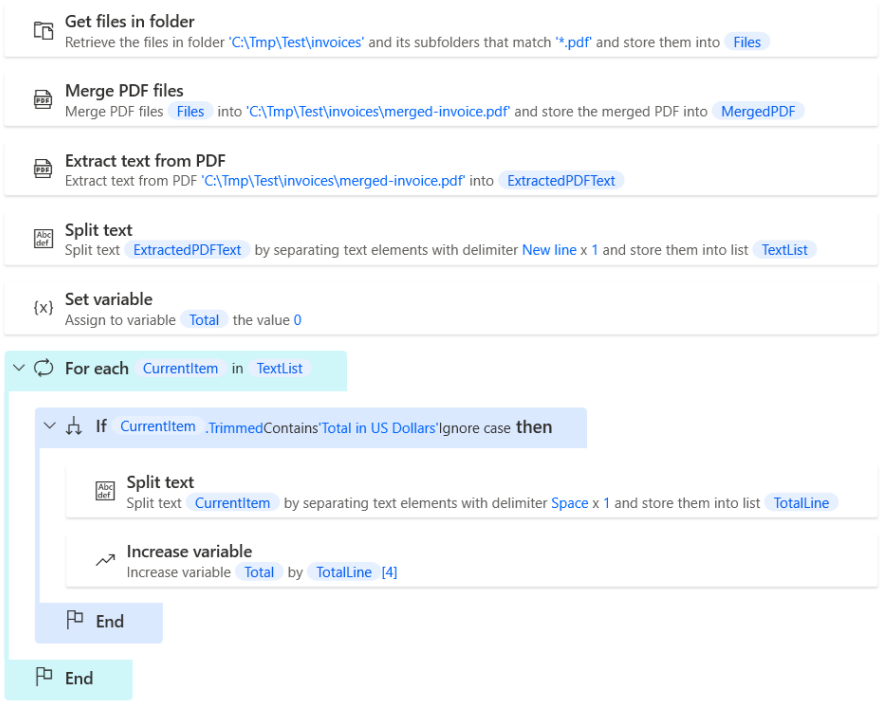

Figure 4-a: The PDF Flow (Part 1)

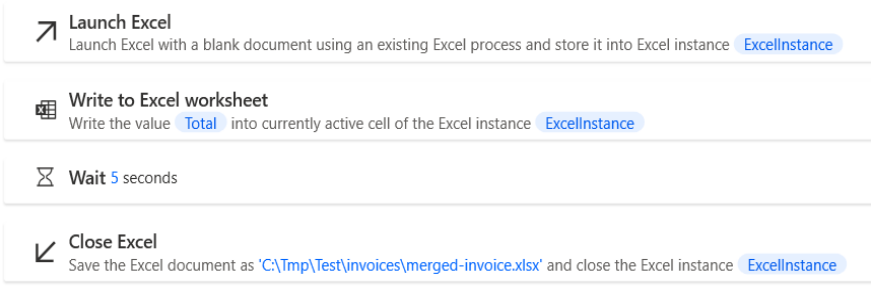

Figure 4-b: The PDF Flow (Part 2)

Let's break this flow down into its parts. The flow begins with the Get files in folder action, and this action retrieves all the PDF files within the origin folder, as we can see in the following figure.

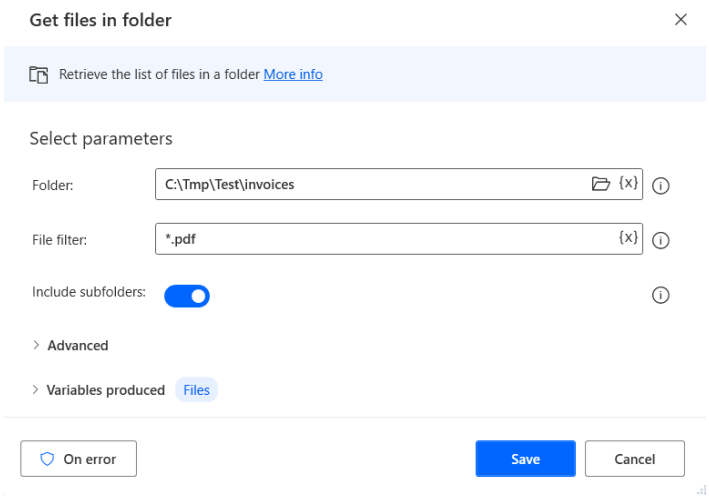

Figure 4-c: The Get files in folder Action

Once we have gathered and read all the PDF files from the origin folder, the flow then merges the PDF files into a single PDF—which is what the Merge PDF files action does. Let's have a look at this action.

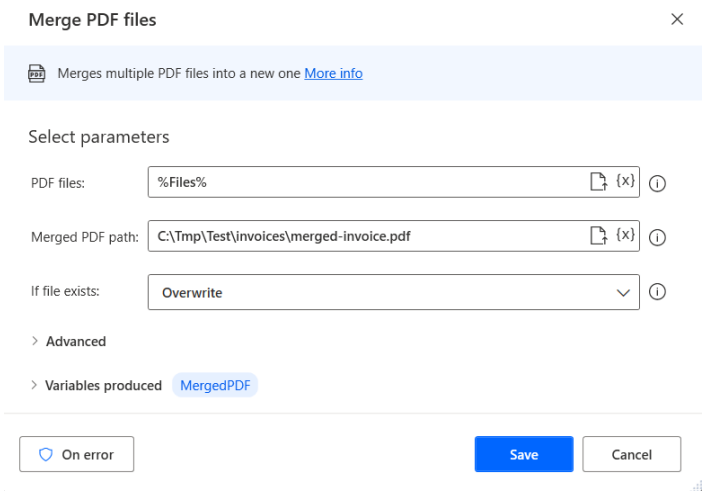

Figure 4-d: The Merge PDF files Action

We can see that the Merge PDF files action loops through all the PDF files read by the previous action (%Files%), and it merges those files into a single file called merged-invoice.pdf. If the merged file exists, the Merge PDF files action will overwrite that file.

Next, the flow extracts the text from the merged PDF file to find the total of each invoice.

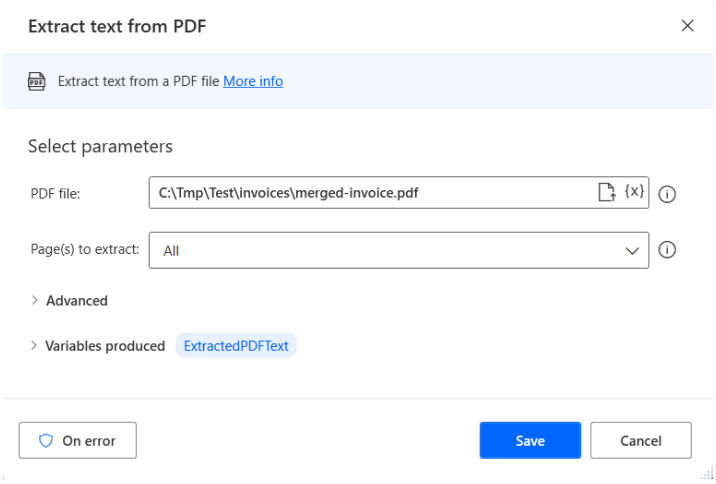

Figure 4-e: The Extract text from PDF Action

The action reads all the pages of the merged PDF file and returns all the text read as the ExtractedPDFText variable.

Now that we have read all the text from the merged file, we need to split the text read into lines to analyze each line and find those lines that contain each invoice's total—this is what the Split text action does.

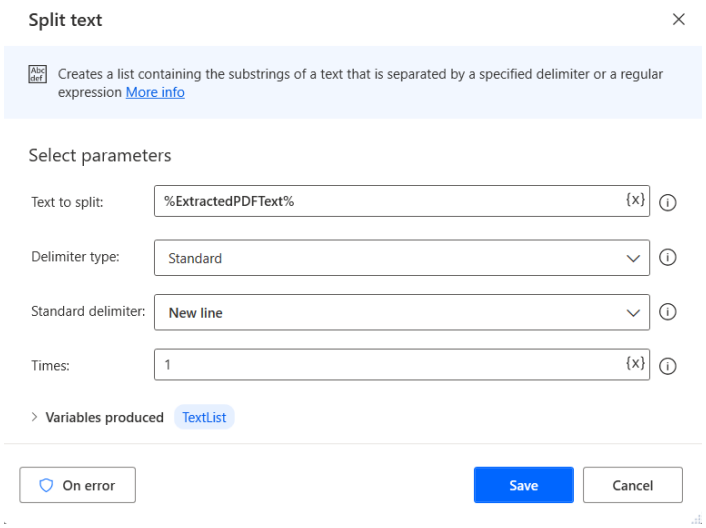

Figure 4-f: The Split text Action (For All the Text Lines)

So, the Split text action receives an input, the text to split, which is the value of the ExtractedPDFText variable. Then, the text is divided into lines when a new line is found. By doing this, we'll get lines of text—which is the variable produced by this action (%TextList%).

With the lines of text, the flow loops through each line, and then it only considers the lines with the **Total in US Dollars** substring.

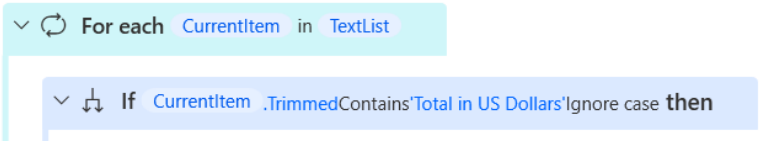

Figure 4-g: The For each and If Actions

Then, the line text is split for each line with the Total in US Dollars substring. In this case, the text line is divided when a space is found.

Once the Split text action is executed, a line containing the Total in US Dollars substring would look as follows.

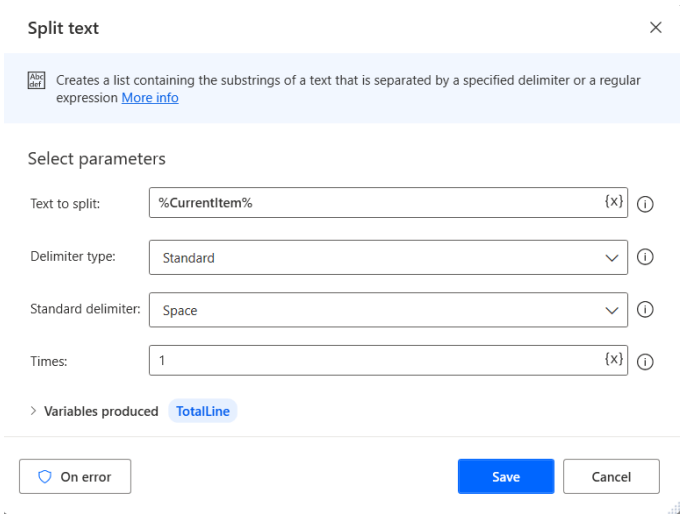

Figure 4-h: The Split text Action (For Each Line)

To understand this better, let's look at one of the PDF invoices we are merging.

Cool Company, LLC.

123 Your Street
Your Town
Address Line 3

(123) 456 789
email@yourcompany.com

INVOICE

13-November-2023
Invoice #2334889
PO 456001200

Att: Ms. Jane Doe
Client Company Name

Dear Ms. Jane Doe,

Please find below a cost-breakdown for the recent work completed. Please make payment at your earliest convenience, and do not hesitate to contact me with any questions.

Many thanks,
Your Name

#	Item Description	Quantity	Unit price ($)	Total
1	Supporting of in-house project (hours worked) - Part 1	40	125.00	5000.00
2				-
3				-
4				-
5				-
6				-
7				-
8				-
Subtotal				**5000.00**
Sales Tax (20%)				**1000.00**
Total in US Dollars				**6000.00**

Many thanks for your custom! I look forward to doing business with you again in due course.

Figure 4-i: PDF Invoice

For each invoice, we are interested in extracting the total amount from the Total is US Dollars line, which is why the line is split.

The merged file (of all the invoices) has various lines with total amounts (as we can see in the following figure), which is why we do this within the For each action to access each line.

Total in US Dollars	**4650.00**
Total in US Dollars	**3750.00**
Total in US Dollars	**12000.00**
Total in US Dollars	**6000.00**

Figure 4-j: Merged PDF Invoices (Total Amount Lines)

For each of these total lines, the split is done to extract the total amount value. The split occurs when a space is found within the line. The total line contains four spaces (**Total in US Dollars 4650.00**), which would be split as follows.

```
0 -> [Total]
1 -> [in]
2 -> [US]
3 -> [Dollars]
4 -> [4650.00]
```

Figure 4-k: Total Line Split

Therefore, the total amount can be found within the fourth (4) position of the parts after splitting the line.

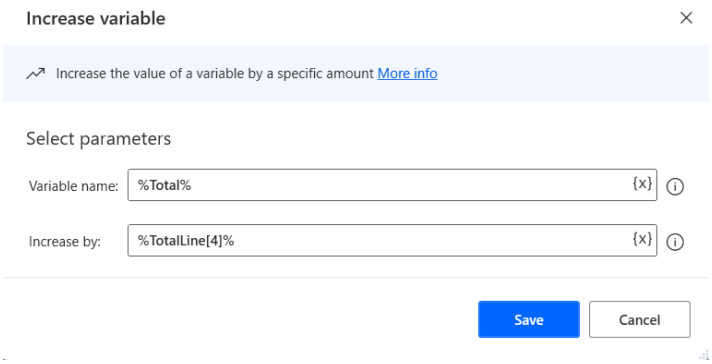

Figure 4-l: The Increase variable Action

By using the Increase variable action, we are adding the total value for each line. We achieve this by increasing the value of the Total (%Total%) variable by the value of the TotalLine[4] (%TotalLine[4]%) variable.

Once the total amounts have been added, the loop is finalized, and the flow runs the Launch Excel action, opening a blank Excel document.

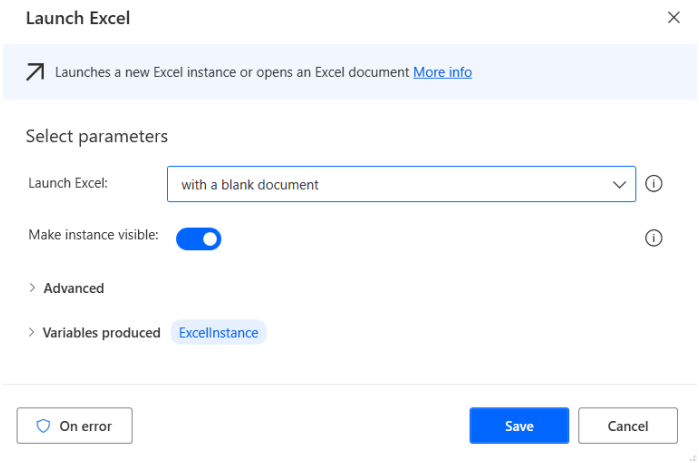

Figure 4-m: The Launch Excel Action

Immediately after, the Write to Excel worksheet action is executed, which is responsible for writing the total amount value of all the invoices (%Total%) to the Excel worksheet.

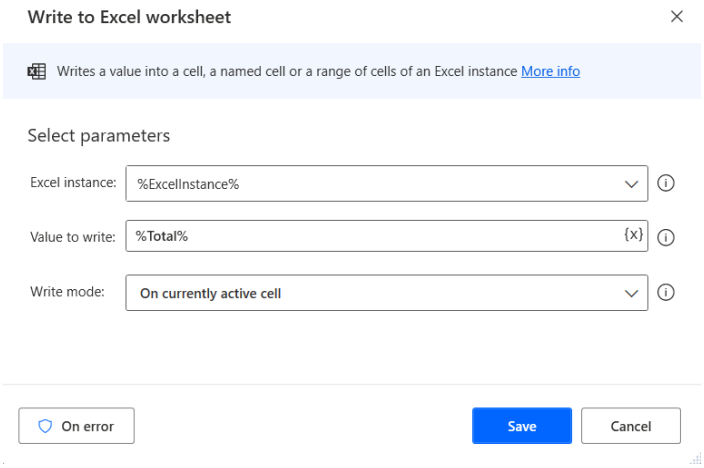

Figure 4-n: The Write to Excel worksheet Action

Finally, the flow waits for five seconds, the Excel instance is closed, and the total amount result is saved to the merged-invoice.xlsx file.

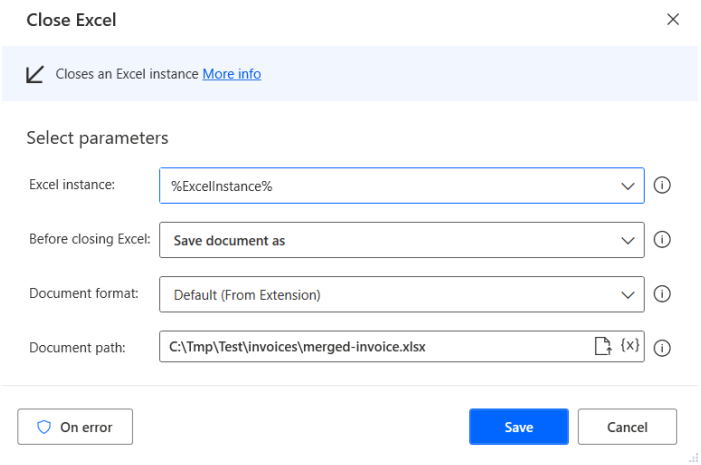

Figure 4-o: The Close Excel Action

Before running the flow

To run the flow successfully, you must have the invoice PDF files in the C:\Tmp\Test\invoices folder. The flow will then create the C:\Tmp\Test\invoices\merged-invoice.pdf file and then extract the text from this file to add the total amounts and generate the C:\Tmp\Test\invoices\merged-invoice.xlsx file.

You can change the folder path from C:\Tmp\Test\invoices to something else; ensure consistency and change it everywhere, where applicable, within the flow.

Running the flow

Place the invoice PDF files within the C:\Tmp\Test\invoices folder (if that's the folder you're using), and after clicking the **Run** button, you'll see that the merged-invoice.pdf will be created, and shortly after, the merged-invoice.xlsx file, with the total value of all the invoices.

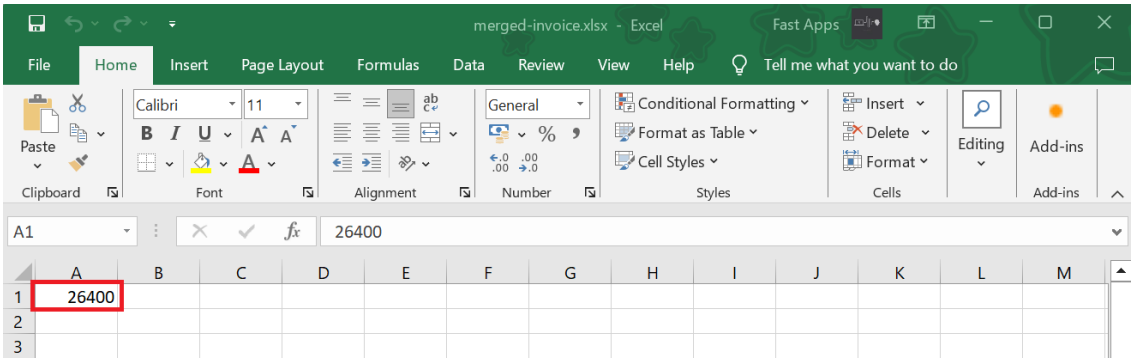

Figure 4-p: The merged-invoice.xlsx File

Pseudo-code

Code Listing 4-a: Pseudo-code Representation of the Flow

```
Folder.GetFiles Folder: $'''C:\\Tmp\\Test\\invoices''' FileFilter:
$'''*.pdf''' IncludeSubfolders: True FailOnAccessDenied: True SortBy1:
Folder.SortBy.NoSort SortDescending1: False SortBy2: Folder.SortBy.NoSort
SortDescending2: False SortBy3: Folder.SortBy.NoSort SortDescending3: False
Files=> Files
Pdf.MergeFiles PDFFiles: Files MergedPDFPath:
$'''C:\\Tmp\\Test\\invoices\\merged-invoice.pdf''' IfFileExists:
Pdf.IfFileExists.Overwrite PasswordDelimiter: $''','''' MergedPDF=>
MergedPDF
Pdf.ExtractTextFromPDF.ExtractText PDFFile:
$'''C:\\Tmp\\Test\\invoices\\merged-invoice.pdf''' DetectLayout: False
ExtractedText=> ExtractedPDFText
Text.SplitText.Split Text: ExtractedPDFText StandardDelimiter:
Text.StandardDelimiter.NewLine DelimiterTimes: 1 Result=> TextList
SET Total TO 0
LOOP FOREACH CurrentItem IN TextList
    IF Contains(CurrentItem.Trimmed, $'''Total in US Dollars''', True) THEN
        Text.SplitText.Split Text: CurrentItem StandardDelimiter:
Text.StandardDelimiter.Space DelimiterTimes: 1 Result=> TotalLine
        Variables.IncreaseVariable Value: Total IncrementValue:
TotalLine[4]
    END
END
Excel.LaunchExcel.LaunchUnderExistingProcess Visible: True Instance=>
ExcelInstance
Excel.WriteToExcel.Write Instance: ExcelInstance Value: Total
WAIT 5
Excel.CloseExcel.CloseAndSaveAs Instance: ExcelInstance DocumentFormat:
Excel.ExcelFormat.FromExtension DocumentPath:
$'''C:\\Tmp\\Test\\invoices\\merged-invoice.xlsx'''
```

Final thoughts

Well done for making it so far! We've just explored how to combine multiple PDF invoices into a single (merged) PDF and extract the total invoice values into an Excel file.

Similar challenges are present in everyday business operations, and this flow provides the necessary foundations to tackle similar scenarios of extracting data from PDFs, which you can adapt to your work processes or business requirements.

As you have seen, working with Power Automate Desktop is not that difficult. It requires some basic understanding of how logical workflows work, a bit of imagination, and experimenting with the different types of available actions.

Although it's impossible to cover within an ebook as short as this one all the actions that Power Automate Desktop includes out of the box, most of the actions that you'll need have been covered throughout these chapters, such as working with variables, conditionals, loops, Excel, PDFs, or web browsers.

I've also included an appendix with ready-to-use flow examples, which you can copy and paste into your existing Power Automate Desktop application on your system.

Power Automate Desktop is a fantastic tool that empowers anyone to automate mundane and everyday repetitive processes quickly and easily. I hope you have enjoyed some of the tips in the book and can leverage this tool to make your work life more productive.

Appendix

GitHub repo

All the flows covered throughout this book (and additional ones) can be found on this GitHub repo.

Importing the flows

Open the flow as a text file within any text editor you choose. Then, copy all the file's content, paste it inside an empty flow within Power Automate Desktop, and save the flow.

Open the flow text file in your favorite text editor. Select all its content.

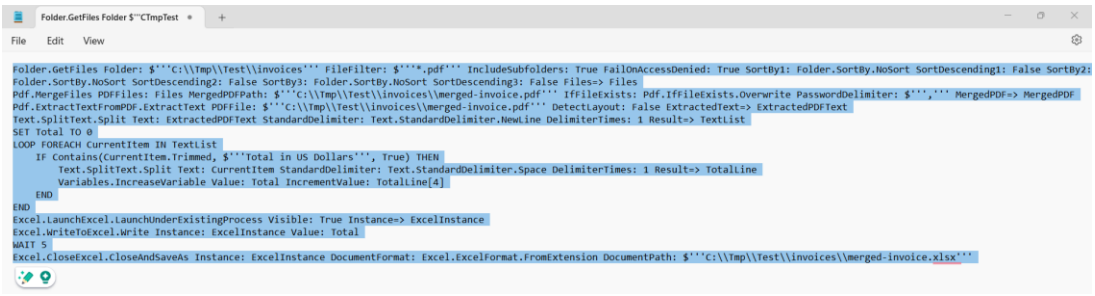

Figure Appendix-a: A Power Automate Desktop Flow (Seen as a Text File)

Then, paste it within an empty flow in Power Automate Desktop.

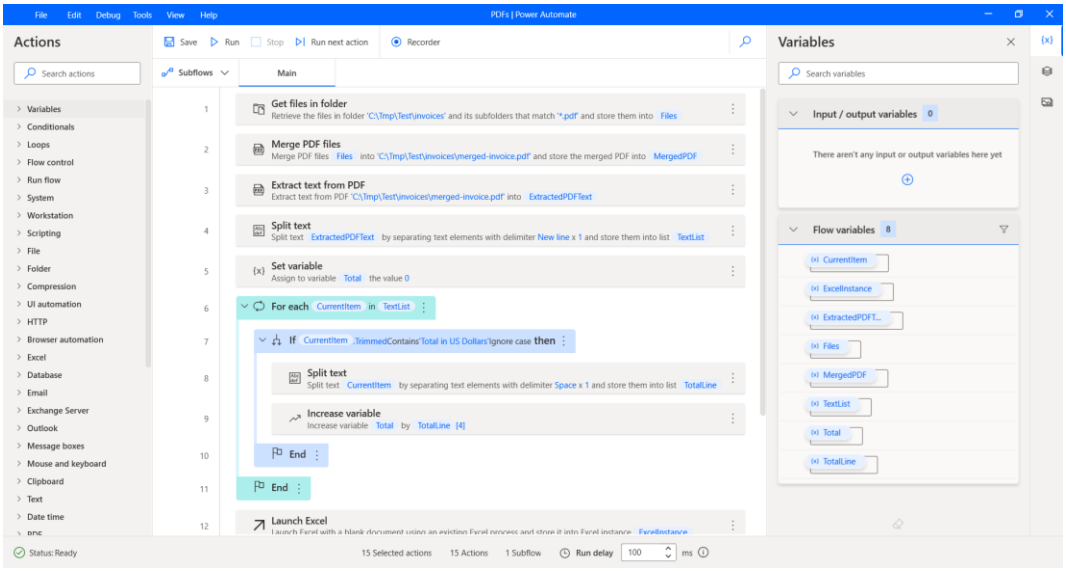

Figure Appendix-b: A Power Automate Desktop Flow (Copied and Pasted from the Text File)

GitHub repo flows

The GitHub repo contains the following flows:

- **Variables**: Shows how to use variables within Power Automate Desktop.
- **Lists**: Describes how to use lists.
- **Conditionals**: Shows how to use conditional statements.
- **Loops**: Explains how to use loops and iterations.
- **ReadWriteFiles**: Demonstrates how to read and write files.
- **File Actions**: Describes how to use various file actions.
- **Folder Actions**: Explains how to work with different folder actions.
- **CleanText**: Demonstrates how to clean text.
- **AppendText**: Shows how to append text.
- **SplitJoinText**: Shows how to split and join text.
- **ReplaceParseText**: Describes how to parse and replace text.
- **ConvertText**: Explains how to convert text to other types and vice versa.
- **PopulateOnlineForm**: Describes how to fill in a Google Form automatically from data in a text file.
- **PDFs**: Shows how to merge various PDF invoices into a single PDF, add the total amount values for each invoice, and output that total value to an Excel file.

GitHub repo test files

You can find the test files under the Tmp/Test folder within the GitHub repo.