Script

Christian Prior-Mamulyan

August 22, 2022

Table of Contents

# StarryArgon series

## Description

Inspire and demonstrate to follow-along the active adoption of the UiPath REFramework to the requirements of a RPA CoE Center of Excellence. Aimed ad intermediate RPA Developers and Head of CoE / Solution Architect / Lead Developers.

Demonstrate features in an end-to-end scenario resembling real-life projects and not just repeating a manual.

## Resources

### Code

* https://github.com/rpapub/REFramework-Excel
* https://gitlab.com/rpapub/tutorials/starry-argon

## Requirements to follow along

### Technology

* UiPath Studio
* Git CLI client, e.g. git for Windows from https://git-scm.com/download/win
* a terminal software with e.g. Powershell
* a GitHub account

### Skills

* intermediate UiPath RPA Development skills, e.g. on Advanced Developer level

### Time

* 1 hour for code changes
* 3 hours for additional research

## Recommended Reading / Bilbiography

* Studio Templates
* DocumentationDocumentation-EN.pdf
* https://git-scm.com/book/en/v2/Git-Tools-Rewriting-History

## Supplementary

### communication

* https://www.linkedin.com/in/cprima/
* issues at https://gitlab.com/rpapub/tutorials/starry-argon

# Adoption REFramework for input data in an Excel file

## metadata

* vendor: UiPath
* code location:
  + video location:
  + https://github.com/rpapub/REFramework-Excel
  + https://gitlab.com/rpapub/tutorials/starry-argon
* dependencies: none
* tested with: UiPath Studio v2022.10.3
* last updated at: 2022-12-12

## Intro

Any RPA CoE Center of Excellence with a RPA platform by UiPath needs several variants of the REFramework. And it is not hard to adopt this best-practises project template.

Typically such variants are required to work with

* input data from Excel files
* or input data in email inboxes

This video shows in 20 minutes

/ how to create a one-click template of the REFramework that works as a Performer with input data in an Excel file.

The estimated effort in real life would be 4 hours for

* research
* implementation
* documentation and
* publishing

of such a variant.

The code is loosely based on the industry-wide known usecase “rpachallenge dot com”.

Not only will be the actual adoption of the code in UiPath Studio shown, but also the Git and GitHub aspects of publishing a template.

To follow along you would need

* UiPath Studio
* Git CLI client, e.g. git for Windows from https://git-scm.com/download/win
* a terminal software with e.g. Powershell
* a GitHub account

This content is for intermediate RPA Developers and themembers of a RPA Center of Excellence.

## Initialize a GitHub repository

This is a UiPath Studio Community edition version 2022.10.3.

Start by crating a new REFramework project in a folder with a temporary name. The name is not important. This local repository will only be used to push the initial code to a remote location.

Put the project under version control:

* In the status bar click “Git Init”
* select the current folder

This will create a .git folder with various defaults.

In the status bar

* click in the pencil icon to open the Commit Changes dialog
* keep the Select All
* write a Commit Message, for example “Initial import REFramework v2022.10.3”
* and click Commit (only!).

This created in our local repository a branch named main. This branch will not be used for development. For development of the new feature create a new branch via the status bar:

* click on the branch symbol
* then Manage Branches
* click the plus sign
* enter a name for the feature branch like REFrameworkExcel
* and click Create branch from main
* Click save

This branch name “REFrameworkExcel” will be used repeatedly until the code template repository is published.

Go to your GitHub organisation and create a new repository. Give it the name you want to publish the code template under. Do not initialize the repository with a README file. Notice that no repository template exists yet.

Copy the link to the repository, here the HTTPS connection is demonstrated.

In UiPath Studio in the staus bar

* click Manage Remotes
* Click the plus sign
* add the URL
* and a name. Give the remote a descriptive name like github.

Then in the status bar

* click on the upload symbol to push to remote
* Select the remote in the popup dialog
* and click Save

At this time no GitHub authorization might exist and a warning popup instructs to install the UIiPath GitHub application.

* Do so by clicking on the link.
* A browser opens on the URLgithub dot com apps uipath installations new
* If your user is a member of many organizations select the right one.
* Here for demonstration purposes “Only select respitories” is clicked
* and from the dropdown menu the just created repository selected.
* Click Install

Back in Studio

* click Cancel
* and in the status bar click the upload symbol again to push to the github remote.

The code is now pushed to GitHub.

When prompted for credentials chose Sign in with GitHub. Here the dialog shows that UiPath Studio is already signed in, but as that was for another organization the permissions for the new repository do not exist yet. Alternatively a browser page might open.

On GitHub.com refresh the repository view to check if the feature branch “REFrameworkExcel” was pushed successfully.

### Changes to Config file and Process workflow location

Here a few changes to the default file structure are performed.

Create a parent folder to hold all Process-related files: In the UiPath Studio

* open the File Explorer
* and create a new folder named Process.
* Move Framework/Process.xaml into Process.

To prepare easy teastability in the same branch

* Open the file location of the Config.xlsx file
* make a copy of the file
* and name it Config\_Test.xlsx

In the file Main.xaml

* open the arguments panel
* and change in the state Initialization at InvokeWorkflow Framework/InitAllSettings.xaml the argument in\_ConfigFile to Data\_Test.xlsx

Future code in this series will build up on that change – but not now. Follow the related Issue on GitHub for updates: https://github.com/rpapub/REFramework-Excel/issues/1

Commit and Push the changes with a descriptive commit massage like “Adding Config\_Test.xlsx and moving Process.xaml into own folder.”

Currently

* the local repository has
  + the code base to implement the actual “adoption of the REFramework for input data in an Excel file” in
* and GitHub
  + has a repository with an empty initial branch
  + which will act as a central, shared location (Which is, by the way, only one of the many workflows Git supports.

## Adopt the code

To change from QueueItem to DataRow changes need to be made in

1. the config file
2. Framework/GetTransactionData.xaml
3. Framework/SetTransactionStatus.xaml
4. Main.xaml
5. Process.xaml

These changes will later be documented in the license file.

### Changes to Config file

To adopt the default config file

* Open the Config\_Test.xlsx file
* in the sheet Constants Change MaxRetryNumber to 2
* In settings delete the 2 rows OrchestratorQueueName and OrchestratorQueueFolder
* Change logF\_BusinessProcessName to e.g. REFrameworkTemplateExcel

Future forks from this code template should always change logF\_BusinessProcessName

The location of the input data file is made configurable by creating 2 config settings:

* key InputDataFilename with the value challenge.xlsx
* key InputDataFilePath with the value Data

To be able to execute the project a placeholder input data file shall be implemented. Here the file from the industry-wide know rpachallenge.com is implemented as a placeholder.

The resulting code will only need minor adjustments in future forks from this code template.

This concludes the changes in the 1st file out of 5

Commit and push the changes, for example with the commit message “Initial configuration in Config\_Test.xlsx”.

### Change Process xaml

In the file Process.xaml

* open the arguments panel
* and change the argument in\_TransactionItem type from QueueItem to DataRow

This is all that needs to be changed in Process.xaml

This concludes the changes in the second file out of 5.

### Change GetTransactionData xaml

From the Project panel

* Open GetTransactionData.xaml and
* change the argument in\_TransactionItem type from QueueItem to DataRow
* change the argument out\_TransactionItem type from QueueItem to DataRow

#### Read the input data file

To obtain a placeholder file to implement reading “input data in an Excel file” against

* Go to rpachallenge.com
* and click “Download Excel”

Here the the browser Edge is used and opens by default a Office Online version instead of downloading the file. To fix this

* click the Horizontal Ellipsis
* open Settings
* go to Downloads
* and set Open Office files in the browser to false.

While in the settingS take note of the download location.

Now the click on rpachallenge.com on the download Excel button will place the input data on the local filesystem.

Move the file to the folder as configured n the previous step.

Future forks from this code template will need to implement their own file retrieval.

To always have a reliable reference of the file, especially for use with Modern Design activities

* Make a folder in the same file hierarchy
* and name it ExampleFiles
* place a copy of the downloaded file in there

Its use will be shown in a minute.

Reading the input file shall only happen once per run. To implement this in the workflow GetTransac tionData.xaml

* delete the activity Gettransaction Item
* Drag an If activity in the same spot,
* with the condition io\_dt\_TransactionData Is Nothing

TransactionData is a default REFramework variable all the way up to Main.xaml

If its value is Nothing this means the execution is in the first pass into the GetTransaction state, so the input data file shall be read. To implement this

* drag into the Then condition
* an Excel Process Scope
* and drag a Use Excel file activity inside
* check the property Template File
* in the appearing file picker chose Data.xlsx
* Drag a read range into the Use ExcelFile
* Conveniently chose Choose Excel.Sheet(“Sheet1”) via the plus icon
* Optionally add dot Range A1
* The output is then saved into the variable io\_dt\_TransactionData

This is why a copy was stored in ExampleFiles. It does not need to contain production data, but only be in the same sheet and format. Great for bugfixes and debug runs in incident resolving.

The adoption of the REFramework for input data in an Excel file will simply loop over this datatable io\_dt\_TransactionData – which is a default datatable in REFramework.

To not just read the template file but actually the to-be-downloaded file

* point Use Excel File to the configured values.

Never, ever hardcode these values even in the earliest phases of development, always go via the Config dictionary.

Make the implementation a bit more robust against unwanted effects by

* unchecking Save changes
* and unchecking Create if not exists.

Remember that we read this file only on first pass, as identified by the If condition of io\_dt\_TransactionData being Nothing

#### Update out\_TransactionItem within each loop

Still the actual TranscationItem must be updated within each and every loop. Do implement this

* Drag another If activity underneath the Retry mechnism
* Make the condition If io\_dt\_TransactionData.Rows.Count >= in\_TransactionNumber

REFrameweork keeps track of the TransactionNumber by incrementing this integer value. So this integer can be used to access a row in the datatable.

This means the condition is true as long as the incrementing TransactionNumber has not reached the final row in io\_dt\_TransactionData.

* Then assign to out\_TransactionItem a single row with io\_dt\_TransactionData.Rows(in\_TransactionNumber - 1)

The minus one is to offset the TransactionNumber starting at value 1 with the datatable having a zero-based index.

* Else set out\_TransactionItem Nothing

REFramework does determine by that if it transitions into End state or not.

To handle Excel application in the depth of the RRFRamework files is not intuitive and likely hard to maintain. To refactor the just written code

* Go back to that Retry and add a (first) Log Message.
* Give the resulting sequence a descriptive name like Read Input File.
* right-click the sequence and extract as Workflow
* Place it in a newly created folder Process/Excel which will become a place for all Excel-application related code

Many other REFramework implementation will place application-level folder in the top folder, this template places them all underneath “Process”.

Reading the input data files needs some sanitation so open the newly created workflow ReadInputData

* Drag a Filter Data Table activity after read range,
* specify io\_dt\_TransactionData to use in the filtering,
* click configure the filter,
* and select Remove when Column 0 is Empty.

This should hold true in any future forks from this code template.

This concludes the changes in the third file out of 5

### Change SetTransactionStatus xaml

In the file SetTransactionStatus.xaml

* open the arguments panel
* and change argument in\_TransactionItem type from QueueItem to DataRow

Now code in the Flowchart’s nodes

* Success
* Business Exception
* System Exception

needs to be changed.

In Success

* delete the Set transaction status (Successful) activity.
* At this point simply replace it with a LogMessage.

Depending on the usecases future forks from this code template might want to implement – for example – writing into a status column of the input data file.

Repeat in the node Business Exception

* delete the Set transaction status (Business Exception) activity
* Replace it with a LogMessage

In the node SystemEception

* delete the If activity named If TransactionItem is a QueueItem (System Exception) which is located after the screenshot
* replace it with a Log Message

This concludes the changes in the 4th file out of 5.

### Change Main xaml

In the file Main.xaml

* open the variable panel
* and change the variable TransactionItem type from QueueItem to DataRow

Several warnings will appear with two types of fixes needed:

* the assign activities setting to Nothing do not automatically adopt, at least in this UiPath Studio version
  + Clear the fields
  + and re-enter, the error will disappear.
* Invoking workflows have their Import Arguments affected by the change from QueueItem to DataRow.
  + Open the Invoked workflow’s arguments and manually make sure all \*TransactionItems are of type DataRow
  + and have the correct variable passed in

This concludes the changes in the last file of 5.

Commit and Push the changes with a descriptive commit massage like “Adopting REFramework to work with input data in Excel file”

### Optional: Refinement

Currently ReadInputData is called every transaction loop, but nothing done inside this code. To make this a bit more performant

* Open GetTransacionData
* drag in a If activity
* and surround the Retried Invoke Workflow with the same condition io\_dt\_TransactionData Is Nothing

Commit and Push the changes with a descriptive commit massage like “Refactored GetTransactionData to only call ReadInputFile conditionally.”

## Preparing and pushing to the remote repository

Regarding the sourcecode management nothing has changed yet and currently

* the local repository has
  + the code base to implement the actual “adoption of the REFramework for input data in an Excel file” in
* and GitHub
  + the feature implementation up-to-date in the branch REFrameworkExcel

There is no .gitignore file yet, and the project.json is still the default one.

### Adding a .gitignore file

A good example for projects with Modern Design can be found on GitHub by impower-ai in the repository uipath-gitignore

From the Project panel

* Open the File Explorer
* add a text file named dot gitignore
* make sure to not accidentally add a txt file extension
* and place the content from impower-ai’s file inside

Commit and Push the changes with a descriptive commit massage like “Adding a .gitignore file”

### Updating project.json

In the projcet panel

* open the Project Settings
* and add a short description.

While there have a brief look at some of the defalt settings.

### Changes to LICENSE file

The REFramework is licensed with a MIT license.

It is customary in code like xaml files to have only one file in the top level hierarchy, as it is not possible to place header comments in every file.

Here I update just the LICENSE file in the top level folder so potential forks of my public repository on GitHub can evaluate the license conditions for my changes.

Commit and Push the changes with a descriptive commit massage like “Preparing a Release Candidate.”

### Creating a public master branch

Every user of Git should know the limitations of the respective Git GUI, and how to overcome these. To utilize the features of Git any Git GUI installation should be accompangied by a Git Commend Line Client – at minimum because tutorials on the internet will reflect the CLI commands.

Currently

* the local repository has
  + the original vendor code in the main branch
  + the feature implementation in the branch REFrameworkExcel
* the remote repository on GitHub has
  + the feature implementation up-to-date in the branch REFrameworkExcel

On GitHub the main branch should have the up-to-date implementation but without the history of commits and their commit messages – but rather a condensed version.

To ensure that the soon-to-be GitHub template repository is fully functional a fresh clone of the temporary development branch “REFRamework-Excel” is made, here shown with UiPath Studio:

* Get the HTTPS location to the repository on the github remote.
* Copy the remote URL from GitHub
* and clone the repostiroy in UiPath Studio.

One way to achieve the condensed history is to create a new master branch and before pushing to the shared remote repository rebase its commits to a condensed number of commits:

* change directory into the freshly cloned repo
* Have a look at the previous commits and their abbreviated commit hashes with git log –oneline
* Note the commit hashes, in this example the oldest starts with d9934be
* Create and switch into a branch names master git checkout -b master
* Start an interactive rebase with git rebase -i d9934be
  + All commits since that commit are shown as they were performed on top of d9934be
  + Long story short: Replace all but on the latest top line
    - the word pick
    - by squash
  + The resulting concatenation of commit messages in multiline format can then be rewritten, here I pick the conveniently phrased “Adopting REFramework to work with input data in Excel file”, but could have written anything on lines not starting with a hash sign.
* The branch is unkown on GitHub until now, so the push needs soem extra arguments git push –set-upstream github master

On GitHub there is now a master branch and the repository is a few clicks away from being a template repository:

* Go to the repostiroes settings, click branches
* Change the default banch – because this repository is new and nowhere else used a change of the default branch is uncritical
* click General and check the box “Template repostitory”

That is all!

## Testing the new template repository

Test the template by

* creating a new repository
* chose the newly created from the dropdown list
* typically only the default branch is needed
* here the codename StarryArgon is used as this the codename of my planned rpachallge.com implemetation in the near future
* Open UiPath Studio, clone the new derivative of the REFramework with input data from Excel files!