NEW Innovation Management

UiPath Best practices	
Revision number: 2.1	Date: 2018-11-30
	Pages: 14

Revision history

Version	Description	Date	Author
0.1	Creation.	2017-06-02	Susanna Hellström Gefwert
1.0	First version.	2017-10-05	Susanna Hellström Gefwert
2.0	Revision of complete document.	2018-07-10	Susanna Hellström Gefwert
2.1	**Update the naming of arguments and added a better description of the glossary.**	**2018-11-29**	**Susanna Hellström Gefwert**

1. Table of Contents

Glossary

PDD - Process Definition Document.
SLA - Service Level Agreement.

Development Process – New's development process.

Project – A UiPath project that can be compiled into a nuget package.

Workflow – A set of activities that together creates a .xaml file.

Script – A UiPath workflow file.

Routines – A workflow that has one function and can be shared between projects or clients.

Templates – A set of activities that are used as a base to functionality and needs to be customized in order to work.

3. Introduction

This document explains the best practices of UiPath development at New Innovation Management. All automation projects developed in UiPath must follow these guidelines. There are multiple benefits of having all projects following the same standard:

- Easier to understand the code
- Consistency among the automation projects
- Enables cooperation among developers
- Easier to maintain
- Security standards are met
- Faster development

1. High Level Design

1.2 Pre-Code Design

Before determining the design of the workflow, make sure to have oversight of the process and good understanding of the aims of the process. All automations must follow the development process, independent of the size and goal of the automation. Even if the code is not directly intended for production, all the steps needs to be followed. A business process flowchart is created to determine the requirements of the process. This will considerably reduce the development lead time as well as increase the likelihood that all the aims and requirements are fulfilled upon completion.

A UiPath project is not necessarily the same thing as a business process. The designer of the automation project must take into consideration whether the business process might be broken up into several projects. This can make the code more readable and easier to maintain, as the projects are kept at a reasonable size and have reduced complexity. Each individual project must be independent in the sense that they cannot depend on the successful execution of the other. The obvious exception to this rule is when linking projects with Orchestrator queues.

3.2 Flowcharts

Flowcharts allows for great oversight of the flow of the process and visualizes the business logic. If there are a lot of nestled if-statements, use a flowchart.

3.3 Sequences

A sequence is an activity used for bundling up workflows. It is the most suitable for a series of activities such as Ui Interactions or multiple assign-activities. They are however unsuitable for more advanced decision logic and loops. Avoid using nested if-statements in sequences and turn instead to flowcharts to achieve a better visualization of the logic.

3.4 State Machines

A state machine is a more complex representation of a workflow. While it does not clearly show the flow of the process, it handles advanced business logic and multiple specific business rules particularly well. The State Machine makes it easy to re-use code for example when dealing with error handling.

New has three different templates that should be used when creating automation projects with UiPath. All of them are found on SVN as separate repositories. A part of the design is to carefully choose the most appropriate template for the task at hand. The three are described below in the order they preferably should be used.

The design of the projects should follow the principle of Dispatcher and Performer. These are explained below.

3.5 Dispatcher

A dispatcher is a script specifically designed to upload work for later processing. This process usually entails a filtering of specific data to be used. Using a dispatcher enables one to easily scale up the workload and balance between different robots and processes.

3.6 Performer

The performer processes transactions uploaded by the dispatcher. This should handle the work within the business process. Multiple performers can be used to obtain the complete workload of one business process.

3.6.1 Queues Template

The queue template is based on transaction process flows. This means that as soon as there is a repetitive process flow that is used for all transactions, this template should be used. In general, this is the majority of all the projects that are automated. The template utilizes Orchestrator queues to get a full reporting functionality and traceability of the transactions. To utilize this template Orchestrator must be available during development and testing. If for some reason Orchestrator is not available but there is a good reason it is at a later stage, one should try to make the code as compatible as possible with a strategy using queues.

3.6.2 Non-Queues Template

Non-Queues template is very similar to the Queues template with the difference of not using Orchestrator queues. The basis is still transactions but retained from a data table where each transaction item is a data row. This should be used in the cases where Orchestrator is not used in production. To get the same reporting as using the queues a standard reporting document is created upon runtime.

3.6.3 Non-Repetitive Template

The Non-repetitive template differs the most from the other two. This should be used when the business process is non-repetitive, that is, not based on independent transactions processed in the same way.

4. Project Structure

One of the most important programming principles is to make sure that the code is as modular as possible. A good project structure is split into multiple smaller parts making use of the activity Invoke Workflow. This allows for multiple people to work on the project simultaneously as well as allowing for individual parts be separately tested. Extracted workflows that solve general problems are preferably put into a common library. Therefore, make sure to check the common library for the functionality needed in advance.

4.1 Workflows

For a workflow to be a suitable candidate for extracted workflow file it should solve a well-defined problem. It is advisable to check the *Isolated*-box in an invoke workflow activity to run it in a separate windows process. This makes the script less susceptible to memory-related issues.

4.2 Queues

Preferably all automation projects should use Orchestrator queues. Unique reference should be used for the automation projects where it is important that the same transaction is processed only once.

4.3 Variables

All variables should be declared in the smallest possible scope to optimize memory utilization. It also minimizes the risk of a variable from a larger scope corrupting the local namespace. It also keeps the variable panel nice and tidy, making it more understandable.

4.4 Arguments

Information passed to and from an invoked workflow file is called an argument. Make sure to use the right" direction" of your arguments such that in-data is declared in UiPath as in and not in/out. If the workflow is set to isolated, only serializable types of arguments can be used.

4.5 Folders

All types of local folders used in the process should be created by the robot. This allows the robot to execute in different environments without the need of code-changes. For the same reason, absolute search paths should be obtained through environment variables. Shared folder paths should be set in among the configuration values.

4.6 Starting Scripts

The first part of the scripts should be designated to environmental control. The script should make sure all necessary demands for execution are met, for instance opening/closing applications, and that required folders exist.

4.7 Stopping Scripts

Just as important as starting scripts in the right way is stopping scripts graciously. If, for instance, a script opens an application, the same script must make sure that the application is closed by the end of the execution.

If the process was unable to perform in a correct way, e.g. the applications used could not be opened, it is important to give a notice to Orchestrator. This is done by throwing an exception in the very end of the process which makes the job status set to *failed*.

4.8 The Project Folder

Keeping the folder structure clean in the project folder is crucial for the readability of the code. Therefore, all *xaml* files that are used for a specific system or purpose should be gathered in one folder. The folder should be named as the system or purpose. Make sure to remove all unused files before publishing the project.

By default, the screen shots and report file are saved in the project folder. Make sure to change the path before committing the project as these will not be reachable, and the project size has a risk of being too large and prevented from being published.

4.9 Use of special folders

Special folders exist in all computers and can be used within the automation projects. As stated in the chapter 7.3 the script should use the activity *environment folders* to get the absolute path of these folders.

5. Naming Conventions

5.1 Names

The reason for adhering to name convention is partly to make the code more readable for other people, and partly to save the developer from getting lost in his or her own code. The basic principle is that the name should reflect its function. This is not only important when naming variables, but also when naming files, folders, activities, etc. The natural exception to this rule is that the main script of a project always should be called *Main* as it is the starting node of execution. In UiPath, and VB.NET in general, variables are not case sensitive. Arguments, however, are. If nothing else is specifically agreed upon, the code should be written in English.

5.1.1 Version Format

The standard format of Major.Minor.Patch is used at New.

a. MAJOR version when you make incompatible changes,

b. MINOR version when you add functionality in a backwards-compatible manner, and

c. PATCH version when you make backwards-compatible bug fixes.

5.1.2 Workflow names

Shared system components: Shall be named according to the following: <System name>_<task>_<version>

<version> - indicates the version of the system.

Process specific system components: Shall be named according to the following: <System name>_<task>

If a workflow does not interact with the system, the naming is as follows:

Shared components: Shall be named according to the following: <task>_<version>

<version> - indicates the version of the workflow file. These should be version handled as the format described in chapter 5.1.1

Process specific components: Shall be named according to the following: <task>

There are a few standards regarding the naming of the <task>

Naming	Description
NavigateFrom_[part]_To_[part]	Navigating within a system to one point to another. Example: System1_NavigateFrom_Dashboard_To_WorkItems.xaml
NavigateTo_[part]	For general navigations from any point in the system. Example: System1_NavigateTo_Dashboard.xaml
Download_[type]	Download a specific file. Example: Acme_Download_YearlyReport.xaml
FilterFile_[type]	Filter one specific file. Example: System1_FilterFile_YearlyReport.xaml
Close	Close an application. Example: Excel_Close.xaml
Logout	Logout from an application. Example: System1_Logout.xaml

Login	Login, using credentials. Example: System1_Login.xaml
Open	Open an application. Example: Excel_Open.xaml
Upload_[type]	Uploads a file of a specific type. Example: System1_Upload_YearlyReport.xaml
SendEmail_[information]	Sends an email. Example: Outlook_SendEmail_SuccessfulTransactions.xaml
Update_[type]	Updates information in the system. Example: System1_Update_ClientInformation.xaml
Extract_[information]	Gets information from a system. Example: System1_Extract_Address.xaml
PopUp_[popup name]	Handles popup. Example: System1_PopUp_WrongEmail.xaml
Delete_[type]	Deletes information within a system. Example: System1_Delete_ClientInformation.xaml
Create_[information]	Creates information in the system. Example: System1_Create_Client.xaml

5.1.3 Naming Activities

Activities should be named according the general principle that the name shall reflect the function. For instance, an activity clicking a" Save"-button should not be named" Click" but rather" Click Save". The name should transmit as much information as possible in the form: less specific -> more specific.

5.1.4 Naming Variables

Just like in a .NET program, variables should be named according to camelCase, i.e. written into one word with a small letter in the first word and a capital letter on every following word. The variables should include the type of the variable within the name.

Ex. str_orderData, dt_customerInformation

The most common types are abbreviated as shown in Table 1.

Type	Abbreviation
String	str
Int	int
Data table	dt
Boolean	bool
Data column	dclm
Double	dbl
Date Time	date
Array	arr

Table 1 Most common variable types and abbreviation

5.1.5 Naming Arguments

Arguments are to be named in the same way as variables with the difference that the direction of the argument is detailed in the name. Note that, in contrast to variables, arguments are case-sensitive.

Ex. in_str_orderData, out_bool_customerExist, io_int_orderNumber

5.1.6 Naming Processes

To separate the different processes in Orchestrator, the naming must be unique.

5.1.7 Naming Assets

Asset names should be unique. Assets can be general or process specific.

General: <AssetName>

Process specific: <Process name>_<AssetName>

5.1.8 Naming Queues

Queues can be general or process specific.

General: <QueueName>

Process specific: <Process name>_<Queue Name>

6. Comments

There can never be too many comments and/or annotations. Think about what information is essential for understanding the script. In general, comments should always contain high-level information about what the program is doing. There should also be information relating to how the program is to be used. If the program for instance is an extracted workflow, make sure to specify arguments and the returned state (for instance if applications are opened and left open, excel documents, etc.). If there are non-mandatory arguments it should be clear what happens if these are left untouched.

The following comments are mandatory for all workflows:

a. Short description of the functionality

b. Pre-condition

c. Post-condition

d. Arguments (optional or mandatory)

e. Namespace imports

7. Logging

When a script is put into production via Orchestrator the output of the execution can be found in the logs. Because logs constitute our principle way of robot diagnosis, it is vital that the information is detailed and clear. Whenever there is an error, information regarding where in the process the error occurred and why should be easily found. This is not only vital in figuring out how to fix the problem, but also in understanding how serious the problem is from a business perspective. The logs, together with supporting documentation, should be enough for judging whether processes need to be stopped and if customer contact is necessary. The robot developer controls how the logging is done. This is implemented through the log message-activity where a level of severity is set according to Table 2.

Level	Description
Trace	Detailed description for debugging, only for the developer. This should be used for all developer information that is used during development.
Info	Information points. Must reflect business process information and is

	used debugging in production. The info logs help the support team talk to the client. BRE should be handled at info-level.
Warning	Problems that are not serious enough for ending the process. These can be transaction item exceptions.
Error	Critical error that must be dealt with before the process can be continued. E.g. not able to login to systems.

Table 2 Logging levels description

For reporting purpose there are a few logs that needs to be incorporated in the code. The name of these are fixed and are shown in the table below, Table 3 . Easiest to incorporate the log information is to use the activity *Add log fields*. These log fields should only come once for each transaction, and if the process does not have a transaction, only once for the complete process.

Log Name	Description
BusinessProcessName	Name of the business process.
TransactionTime	Time that the transaction was executed.
TransactionStatus	The outcome of the transaction.
TransactionType	The type of the transaction. This information is process specific.
TransactionNumber	Number of transactions that are processed

Table 3 Log information that needs to be incorporated within all processes

Worth noticing is that logging is a sensitive part of an SLA. Always try to send as little information as possible regarding information that can be classified as company secrets. By default, all activities in UiPath generate logging data. Sometimes it is necessary to shut of this function for sensitive parts. This is done by checking the private box in the properties-panel of relevant activities.

8. Error Handling

Error handling will inexorably take a large portion of a developer's time. It is therefore a well worth investment to make the script as stable as possible early in the process.

Errors will, however, occur. To handle *run time*-errors, *try/catch*-blocks are used. Try/catch allows the robot to try to execute a block of code. If the execution fails, an *exception* is thrown, which can be *catched* and subsequently handled. Depending on the nature of the error, alternative routes for the continued execution can be identified. In practice, try/catch-blocks are used to end the script and ready the environment for a potential retry, as well as providing the developer with as much information as possible to solve the problem. It is therefore essential to generate well thought out log messages when a problem occurs. Absolute minimum for these error messages is to provide the same information as in exception.Message and exception.Source. As to ending processes; make sure to clean up the environment. Sometimes a half-done transaction needs to be reverted.

Not all exceptions are thrown from the different activities, the developer can also throw exceptions in appropriate places. An exception should be thrown in the code when the circumstances are not according to the design, and the transaction or process needs to be handled manually before continuing. Important to note is that exceptions are errors and should always be treated manually. This means that exceptions are not thrown to navigate through the workflow. For this there are if-statements and decisions-activities.

There is a variety of exceptions to choose from other then the mentioned below. The developer needs to choose the appropriate exception for the situation and think about how the different exceptions are to be handled.

8.2 Business Rule Exceptions

Business rule exceptions (abbreviated BRE or BE) are used to indicate that the input data was incorrect and could not be processed. This means that there is no point in retrying the item, the outcome will be the same. Business rule exceptions are thrown by the developer.

8.3 Application Exception

Another exception that can be thrown by the developer is application exception. This indicates that there is an error with the application which makes it impossible to continue the process. These can be when a password has expired, or for other reasons the robot is unable to login to an application.

8.4 System Exception

System Exception is the mother of all exceptions. This means that if an exception is thrown but not catched in a more detailed exception, System Exception deals with the exception. In general, System Exception deals with selector issues or other timeout problem which makes it impossible to continue the process.

8.5 Should Stop

To be able to end a job from Orchestrator with the *Cancel* button, the Should Stop activity needs to be incorporated in the project. The activity registers that someone wants to end the job and close all programs. This is needed to adhere to the *transactional* requirement that each transaction should never be interrupted in the middle.

9. Managing account information and preferences

7.1 Credentials

Essentially, all sensitive information should be stored in the Orchestrator for encryption. In situations where credentials must be stored on the local machine, it must be in Windows Credential Manager, for example under development stages when Orchestrator is not available.

7.2 Local settings on robot

To make the robot script as portable as possible – so that it can run on any machine – all developers should assume that the machine is a standard machine and not assume that any local settings or configurations are made. Any deviations from this principle must be stated in the robot design documentation, RDD.

7.3 Configurations

The processes should be designed so that no changes in the script is necessary for the script to run in a different environment (test or prod). This means that all configurations should be stored in Orchestrator for production and test environments. If Orchestrator isn't available during development, a replica of the configurations (except credentials, see chapter 7.1) should be stored in the configuration file.

Necessary run-time resources such as lists, files, dates etc. that should be configurable for the process owner should always be stored on network shares that are tightly role-controlled.

7.3.1 Test

If a target application does not have a working test-environment, it is recommended to add controls allowing for a different workflow path during testing.

7.3.2 Assets

Assets are configurations that are set from Orchestrator. These can be set per robot or for several robots (Value per robot).

By default, assets should be robot specific as this enables configurations to be divided between different robots as well as environments.

7.4 Emails

The client might be inexperienced with emails sent from robot. Therefore, when sending email to the robot or business administrator it is important to include a notice stating that the email is sent from robot and that receiver cannot reply to this email.

8. GDPR

To be compliant with the new EU regulations on personal data, remove all personal data from the code. This means that all print screens taken as informative screenshots for activities as well as personal information within the code needs to be removed.

9. Utilize API:s

The script should mainly be implemented in a way that simulates a human. Direct integration with API is not recommended if a graphic interface is available. As this creates a dependency on third party. Information regarding third party systems must be documented in the RDD.

10. Documentation

The business process is comprehensively documented in the PDD, while the implementation specific documentation is found in the RDD. The RDD should be in the *Documentation* folder in the UiPath project.

11. During Development

11.1 Backups

All automation projects should use SVN with the release code in the trunk and the different published versions as tags. The branches are used for further improvements and bug-fixing.

12. Shared library

In general, all code should be shared between all colleagues at New. Only very processes specific workflows – that cannot in anyway be of public interest – are not shared. In SVN there is a shared library that consist of two different parts Templates and Routines.

In the shared library each system has its own folder. System or application independent routines and templates have their own folders.

12.1 Templates

A set of activities that can be shared with some smaller modifications are called templates. These have the purpose of helping the developer to get started but doesn't solve the complete problem. Before templates are added to the shared library they need to follow the guidelines and have annotations stating:

a. Description regarding the functionality

b. How it should be modified to be used

c. Dependencies

d. Author

e. Date updated

f. Namespace imports

12.2 Routines

Workflows that solve a problem and can be shared between projects without modifications are called Routines. These can be easily added to the project by copying the workflow file into the project folder and used with the *invoke workflow file* activity. Before they are shared with everyone else they need to follow the guidelines and have an annotation stating:

a. Description regarding the functionality
b. Pre-condition
c. Post-condition
d. Arguments (optional or mandatory)
e. Dependencies
f. Namespace imports
g. Author
h. Date updated

13. End of development

13.1 Checklist before completion

Additionally, the following should be done:

1. Remove all unused screen shots.
2. Remove all unused variables and arguments
3. Remove unused – commented out – code.