Hurmet Noka

UiPath Certified Professional

LEARNING PLAN CURRICULUM

1. Introduction to RPA and Automation
2. Get Started with RPA Development
3. A Day in the Life of an RPA Developer
4. Variables, Arguments and Control Flow in Studio
5. UI Automation with Studio
6. DataTables and Excel Automation with Studio
7. Data manipulation in Studio
8. Selectors in Studio
9. Project Organization in Studio
10. Debugging in Studio
11. Error and Exception Handling in Studio
12. Introduction to Logging in Studio
13. Orchestrator for RPA Developers
14. Email Automation with Studio
15. PDF Automation in Studio
16. Version Control Systems Integration in Studio
17. RPA Testing with Studio Pro
18. Hand-On Experience for UiPath RPA Associate v1.0

Hurmet Noka

1. Introduction to RPA and Automation

Objectives:

1. **Define RPA and Automation and explain their impact on digital transformation.**

   RPA (Robotic Process Automation) is the technology that enables software 'robots' to carry out repetitive, rule-based digital tasks. Humans typically perform these tasks through the user interface, using the mouse and keyboard. RPA robots are capable of mimicking the human actions, and they are typically more accurate, faster, and more consistent at it.

   Combined with **Artificial Intelligence** (AI), RPA can target more sophisticated work. This opens up endless possibilities on the path towards the fully automated enterprise™.

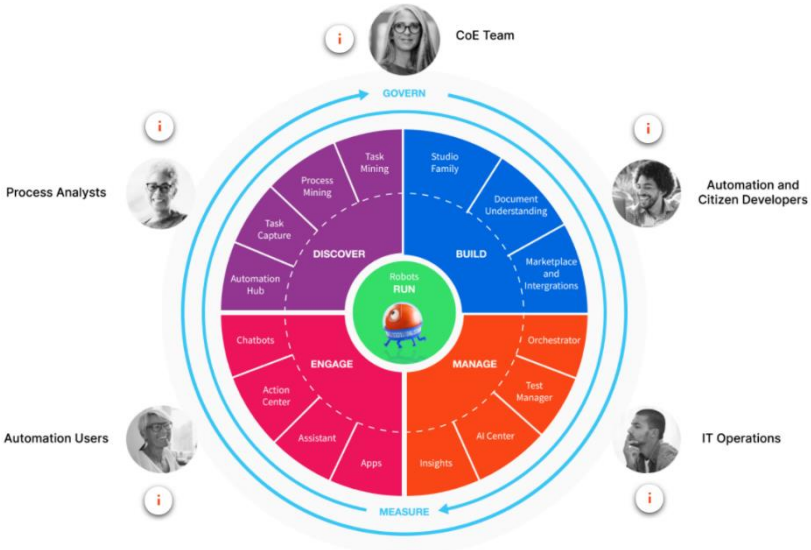
   Automation is a term that describes more accurately these possibilities that exceed the sphere of basic RPA. Sometimes we use these two terms interchangeably, given that RPA is still at the core of automation.

2. **Map and assess some of the business processes that are fit for automation.**

   RPA primarily targets processes that are highly manual, repetitive, rule-based, and have a low exception rate and a standard electronic readable input. Some of the examples are: Payroll processing, Client Information Updating, Contract Renewal Process, Financial Statement Reconciliation, Compliance Reporting, Costumer complaint processing.

3. **Describe the main stages of the typical enterprise automation journey and the role played by the Center of Excellence.**

4. **Explain the components of UiPath Platform and the benefits it brings to the fully automated enterprise.**



2. Get Started with RPA Development

Objectives:

1. **Describe how the UiPath core RPA components (Studio, Orchestrator and Robot with Assistant) work together**.
   As an RPA Developer you will spend most of your time working with Studio, Orchestrator and the Robot. We call them the core components because RPA cannot work without developing automations, running them, and managing the entire ecosystem.
   UiPath Studio is an integrated development environment for RPA developers to design, develop, and debug automation projects. Studio connects with Orchestrator to publish automated processes as NuGet packages to feeds. From there, they are distributed to robots to be executed.
   A software robot is an execution agent that runs automations built with the Studio family and then published as packages either locally, or to Orchestrator.

   There are two types of UiPath robots and they differ both in the way they work and in the way they are licensed:
   Attended robots
   They are digital helpers for human users. They work on the same machines as humans, during the same hours. They are triggered directly by humans (usually through UiPath Assistant) or by an event related to what the human user does.

   UiPath Assistant is the component that provides a friendly interface to interact with attended robots. It is the tool that we use to easily access, manage, and run automations.

   Unattended robots
   These are meant to work non-stop, with as little input from human users as possible. They are deployed on separate machines and their jobs are triggered exclusively from Orchestrator.
   Their interactions with human users are typically handled with as little disruption as possible, by creating and sending requests for human input or validation as tasks. While these await to be processed, unattended robots can continue their work by picking up other jobs. When human input is finally provided, unattended robots can resume their work on the process.

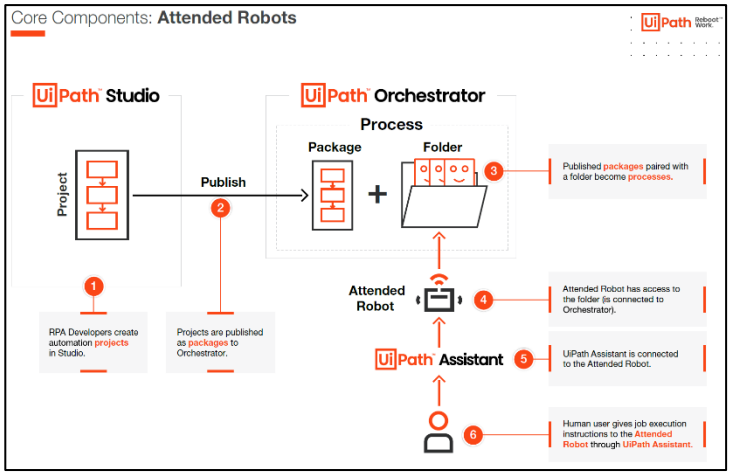
   Orchestrator, the heart of automation management, is a web application that allows managing, controlling, and monitoring the robots and the automations. With Orchestrator we can deploy, trigger, measure, provision, track, and ensure the security of every robot in the organization. Orchestrator also functions as a repository for libraries, reusable components, assets, and processes used by robots or by developers.

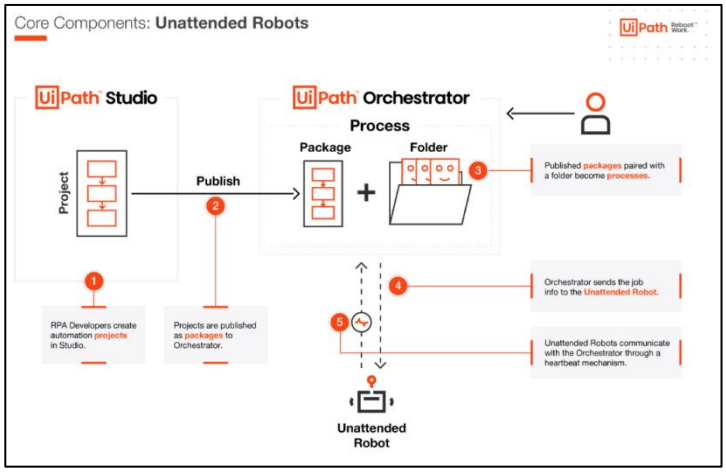
   The main capabilities of Orchestrator:

- Provisioning
- Control and license distribution
- Automation storage and distribution
- Running automation jobs in unattended mode
- Monitoring

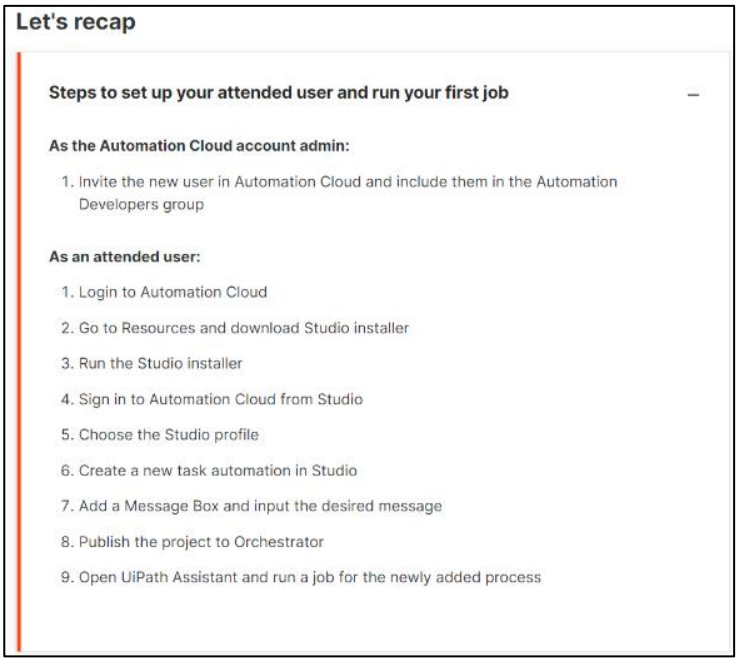
Hurmet Noka

2. **Differentiate between the two types of UiPath robots - attended and unattended.**



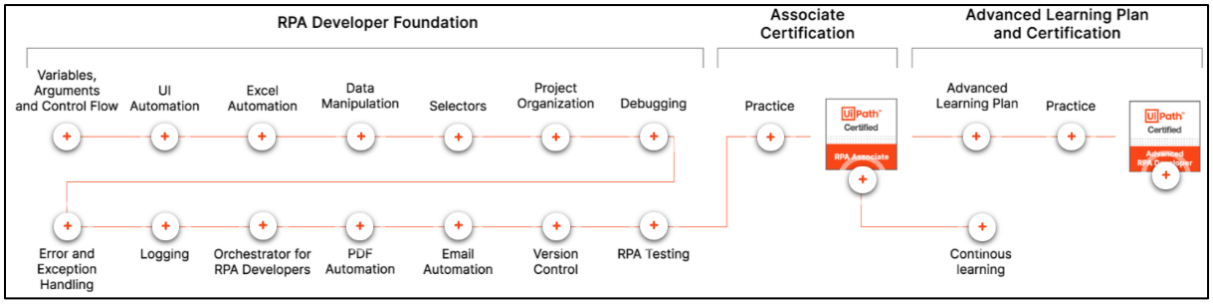


3. **Set up an attended environment with Studio and Assistant, connected to Orchestrator.**



Hurmet Noka

4. **List the key skills required to become an Associate level developer.**



5. **Build an automation project following instructions.**

6. **Access the UiPath Community ecosystem.**

When starting out with UiPath, there's more than training to help you thrive in the world of automation. We believe that collaboration is the best way to learn, so we have designed an entire ecosystem to support your journey. Our community ecosystem helps you connect with an ever expanding group of RPA developers and companies to share knowledge, opportunities and components.

Uipath Community Forum
UiPath Documentation
UiPath certification
UiPath Marketplace

Glossary of Terms and Abbreviations

Studio –

The component of the UiPath RPA Platform for developing Robotic Process Automation projects

Robot –

The Robot is UiPath's execution agent installed on a machine and enabling you to run automation jobs based on processes developed in Studio.

Attended robot –

A type of robot working on the same machines as the humans, during the same hours. They are triggered directly by humans (usually through UiPath Assistant) or by an event related to what the human user does.

Unattended robot –

A robot typically deployed on a separate machine than any human user. Their jobs are triggered exclusively from Orchestrator.

Since robots do not exist as separate entities in Orchestrator, a more technical definition would position unattended robots as execution slots - a user logged in on a machine, executing an automation job, as instructed by Orchestrator.

Hurmet Noka

3. A Day in the Life of an RPA Developer

Objectives:

1. **Explain what the Automation First mindset is**.
   In the Automation First Era, you need to constantly look at your work and the processes in your company through the lens of automation potential. This is what we call the Automation First mindset.
   The automation first three-pronged approach:
   - **A Robot for every person**
   - **Open and Free collaboration**
   - **Robots Learn Skills**

2. **Explain what a process is and differentiate it from a procedure.**
   Definition: A process is a set of interrelated or interacting activities that transforms inputs into outputs.

   Components of a process:
   Inputs - the data that goes in the process.
   Process Flows - the sequences of sub-processes or activities undertaken in the process.
   Source Applications - the applications or systems used to perform the sub-processes or activities of the process.
   Outputs - the result generated by the process.

   Things to remember: The outputs of a process can serve as inputs for other processes.

   A procedure explains:
   - Who is responsible for each part of the process?
   - When each part of the process needs to occur
   - How to handle exceptions
   - The specifications applicable to each part of the process.

   In what concerns the way they are documented, processes and procedures are different. A process is typically documented via a diagram, be it a flowchart or workflow, which aims to highlight the logical sequence of the process steps. A procedure, on the other hand, is often a complex, written document, focused on providing guidelines.

3. **Evaluate if a process is fit for automation.**
   There are two sets of criteria you can use to determine the automation potential: process fitness and automation complexity.

   Process Fitness:
   - Rule-Based
   - Automatable and/or repetitive process
   - Standard Input
   - Stable

   Automation Complexity:
   - Number of Screens
   - Types of applications
   - Business logic scenarios
   - Types and number of inputs

We can sort processes on 4 categories:

- **No RPA** –> Processes where change is frequent, the system environment is volatile and multiple manual even non digital actions are required.
- **Semi-Automation** –> Processes that can be broken into steps that are clearly automated, and steps that need to stay manual (such as validations or usage of physical security tokens)
- **High Cost RPA** –> Processes that are rather digital and can be automated, but use some technologies that are complex (such as OCR) or require advanced programming skills.
- **Zero touch Automation** –> Processes that are digital and involve a highly static system and process environment, so that they can be easily broken into instructions and simple triggers can be defined.

4. **Describe the stages of the automation lifecycle.**

How do RPA project implementations work?

Before getting into the specifics of developing workflows, it's helpful to understand the context you will be working in, the stages of RPA implementations and who you will be working with.

Most often, we will find six stages in RPA implementations:

1. **Prepare RPA** - the processes are defined, assessed, prioritized and the implementation is planned.
2. **Solution Design** - Each process to be automated is documented ("as is" and "to be"), the architecture is created and reviewed, the test scenarios and environments are prepared and the solution design is created and documented for each process.
3. **Build RPA** - The processes are automated, the workflow is tested and validated and the UAT prepared.
4. **Test RPA** - The UAT is performed, the workflow is debugged and the process is signed off.
5. **Stabilize RPA** - The Go-Live is prepared, the process is moved to production, monitored, measured and the lessons learned are documented.
6. **Constant Improvement** - The process automation performance is assessed, the benefits tracked and the changes managed.

Hurmet Noka

4. Variables, Arguments, and Control Flow in Studio

Objectives:

1. **Explain what variables are and what they are used for.**
   Variables are containers that can hold multiple data entries (values) of the same data type. Variables are configured through their properties. You can set them in the Variables panel. The main properties in UiPath are:
   - **Name**
   - **Type**
   - **Default Value**
   - **Scope**

   It should be as descriptive as possible to make your automation easy to read by other developers and to save time. While not the only option, we recommend using **PascalCase** for variable names.

   Creating Variables, there are three main ways to create variables in UiPath:
   - **From the variable Panel**
   - **From Expressions**
   - **From the properties Panel**

2. **Create and configure variables in an automation project.**
3. **Explain what arguments are and what they are used for.**

   What is a workflow?
   A workflow represents a small piece of automation that you can take and re-use in your projects. It is basically your canvas, the place where you design and work with all the UiPath Studio activities and it defines the flow of your automation. Hence the name, workflow.

   UiPath Studio provides you with predefined workflow layouts to suit all the needs of a fast and reliable automation process.

   The workflow layouts are:
   - Sequences
   - Flowcharts
   - State Machines
   - Global Exception Handler

4. **Differentiate between variables and arguments.**
   Arguments are very similar to variables:
   - They store data dynamically
   - They have the same data types
   - They support the same methods and properties

   The difference is that they pass data between workflows, and they have an additional property for this – the direction. Arguments have specific directions: In, Out, and In/Out. These tell the Robot where the information stored in them is supposed to go.
   Arguments are a key component when it comes to building more complex automations, where you need to store and use data between multiple workflows.

Argument names should be in **PascalCase** with a prefix stating the argument direction, such as *in_DefaultTimeout*, *in_FileName*, *out_TextResult*, *io_RetryNumber*.

Creating Arguments, there are three main ways to create arguments in UiPath:

- **From the variable Panel**
- **From Expressions**
- **From the properties Panel**

5. **Use the Invoke Workflow File activity to chain workflow execution and pass data through arguments.**

6. **Differentiate between the most common control flow statements used in UiPath (If statement, Loops and Switch) and configure them according to the specifications.**

There are 4 predefined workflow layouts – **Sequence, Flowchart, State Machine** and **Global Exception Handler**.

We will cover them in depth in the "Project Organization in Studio" course. For now, let's focus on the difference between sequences and flowcharts, as we will use both extensively in our examples throughout the entire course.

- In sequences, the process steps flow in a clear succession. Decision trees are rarely used. Activities in sequences are easier to read and maintain, thus they are highly recommended for simple, linear workflows.
- In flowcharts, the individual activities are a bit more difficult to read and edit, but the flows between them are much clearer.

Use flowcharts when decision points and branching are needed in order to accommodate complex scenarios and decision mechanisms.

The control flow statements

The activities and methods used to define the decisions to be made during the execution of a workflow.

The most common control flow statements are **If, While, Do While, For Each, Switch,** and **Parallel**.

Switch:
It is a type of control flow statement that executes a set of activities out of multiple, based on the value of a specific expression. In other words, we use it instead of an If statement when we need at least 3 potential courses of action.

Parallel:
The Parallel activity enables you to execute two or more child activity branches at the same time. In UiPath Studio, the Parallel activity can be found in the Activities panel, under **Workflow > Control > Parallel**. The Parallel activity finishes only after all child activities are complete or when its CompletionCondition property evaluates to true.

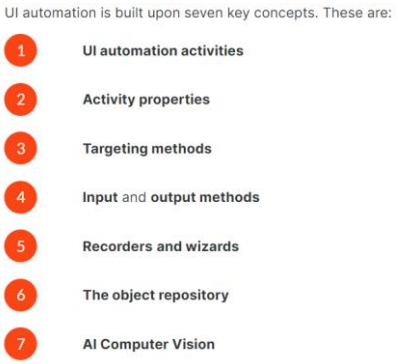
Hurmet Noka

5. Introduction to User Interface Automation

Objectives:

1. **Explain what UI automation is.**
    - UI automation is the process of automating application user interfaces by simulating human input and output actions through specific UI activities.
    - The first step in automating UIs is understanding the logical sequence of steps that a human user would take. The second step consists in translating these steps into UiPath Studio activities and configuring them.
    - The complexity of the UI or of the application is irrelevant. All desktop applications can be automated by using universal or application tailored activities.

2. **Explain the key elements of UI automation with UiPath Studio.**



UI Automation activities let developers quickly write instructions for the Robot to interact with the interfaces of various applications. Activities can be split into:

- **Containers**
- **Input Activities**
- **Output Activities**
- **Synchronization Activities**

Activity properties - Just like for other types of activities, properties determine how the robot performs an action. They can be found in the Properties panel. UI automation activities have specific sets of properties depending on the type of activity.

Targeting methods - Targeting methods are a subset of Properties. They provide several ways to identify the UI element the Robot will be interacting with. The most commonly used targeting method is the selector. The key targeting methods are:

- Selectors
- Fuzzy Selectors
- Image
- Anchors

Input and output methods - As we've discussed already, every time we insert data into an application, or we send a command to a system to produce a change (or to continue), we perform an input action. Output actions are used in UiPath either to extract data (in general, as text) from a UI element. Output methods are what enables output actions to extract data from UI elements.

Input Methods: **Hardware Events, Send Window Message, Simulate**
Output Methods: **Full Text, Native, OCR**

Recorders - is an important part of UiPath Studio that can help you save a lot of time when automating your business processes. This functionality enables you to easily capture a user's actions on the screen and translates them into sequences. These projects can be modified and parameterized so that you can easily replay and reuse them in as many other processes as you need.

The object repository - The object repository ensures the management, reusability, and reliability of UI elements by capturing them as objects in a repository, sharable across projects. It allows for creating and reusing UI taxonomies inside and across automation projects. This is a more advanced feature we will learn more about later.

AI Computer Vision - activity package contains refactored fundamental UI automation activities such as Click, Type Into, or Get Text. The main difference between the CV activities and their counterparts is their usage of the Computer Vision neural network developed in-house by our Machine Learning team. The neural network is able to identify UI elements such as buttons, text input fields, or check boxes without the use of selectors. This is an advanced feature we will learn more about later.

3. **Explain the difference between the modern and the classic design experiences in Studio.**

The key differences between classic design experience and the modern design experience are based on the following Key concepts:

- **UI automation activities**

A separate set of UI activities is available by default in each experience. Regardless of the selected experience of an automation project, modern or classic, the other set of activities can be enabled.
If the default experience is classic, to view the modern activities simply select the "Show Modern" from the Activities panel, or vice versa.

- **Activity properties**

Classic and modern activities have slightly different properties because of different targeting methods and activity behaviours. We will learn more about this in the following lessons.

- **Targeting methods**

When using classic activities, the selector is the most common targeting method for UI elements (99% of the time). In scenarios where it is unreliable, you have the possibility to manually change update the activity to use a fuzzy selectors or anchors.
Modern activities will cycle through the stacked options Selector, Fuzzy Selector, Anchor, and Image to determine the most reliable option in the order we have just presented them. In case the previous method is not reliable, it will automatically choose the next one. This targeting technology is called Unified target.

- **Recorders and wizards**

Modern and classic offer two different sets of recorders:
Modern experience - The App/Web and Computer Vision recorders are available.
Classic experience - The Basic, Desktop, Web, Image, Native Citrix, and Computer Vision recorders are available.
When it comes to Scraping Wizards:
Modern experience - The Table Extraction wizard is available for data scraping.
Classic experience - The Data Scraping and Screen Scraping wizards are available.

- **The object repository**

The Object Repository ensures the management, reusability, and reliability of UI elements by capturing them as objects in a repository, sharable across projects. The Object Repository is only available in the Modern Design Experience.
Don't worry about remembering all of this. It's useful to know what the key areas of difference are and you can reference this course later if you need to.

4. **Enable the modern design experience in Studio.**
5. **Configure a default design experience at Studio level for all new automation projects.**

Hurmet Noka

6. DataTables and Excel Automation with Studio

Objectives:

1. Create, customize and populate DataTable variables.

What are DataTables?

DataTable is the type of variable that can store data as a simple spreadsheet with rows and columns. You can identify each piece of data based on its unique column and row coordinates. Think of it as the memory representation of an Excel worksheet.

In DataTables, the regular convention of identifying the columns and the rows is applied - columns are identified through capital letters, and rows through numbers.

2. Use the most common methods for DataTable manipulation.

Some of the common methods for DataTable Manipulation are:

- Add Data Column
- Add Data Row
- Build DataTable
- Clear DataTable
- Filter Data Table
- Join Data Tables
- Remove Data Row
- Remove Duplicate Rows
- Sort Data Table
- Get Row Item

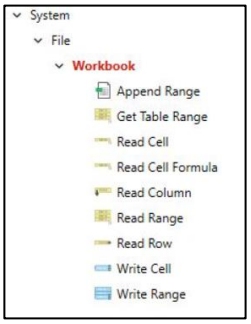
3. Differentiate between the categories of activities used when working with Excel files: Workbook activities and Excel App Integration activities.

Workbook - File Access Level

All workbook activities will be executed in the background.

(+) Doesn't require Microsoft Excel to be installed, can be faster and more reliable for some operations just by not opening the file in the Excel application;

(!) Works only for .xls and .xlsx files.
(!) Doesn't work with .xlsm files.
(!) The file should not be open in Excel at runtime.

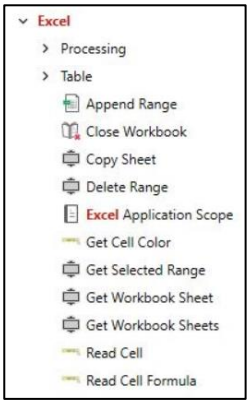


Excel - Excel App Integration

UiPath will open Excel just like a human would.

(+) Works with .xls, .xlsx and .xlsm, and it has some specific activities for working with .csv. All activities can be set to either be visible to the user or run in the background. The file can be open in Excel at runtime.

(!) Microsoft Excel must be installed even when the 'Visible' box is unchecked. If the file isn't open, it will be opened, saved, and closed for each activity.



4. Use specific activities to work with Excel files (read data, write data, save files, and so on).

The Excel Application Scope

The integration with Excel is enabled by using an Excel Application Scope activity. It is a container and all the other Excel activities used to work with the specified Excel file have to be placed inside the container. When the execution ends, the specified workbook and the Excel application are closed.

The Excel Application Scope can be configured to write the output of the activities in the container in a different file.

5. Use the Select method to filter DataTables.

```
Apartments.Select("[Pet friendly] = 'Yes' AND [Price]>400 AND [Price]<800 AND NOT([Number of rooms] = 1)").CopyToDataTable
```

Hurmet Noka

7. Data Manipulation in Studio

What is data manipulation?

Data manipulation is the process of modifying, structuring, formatting, or sorting data in order to facilitate its usage and increase its management capabilities.

Objectives:

1. **Use the most common .NET methods to manipulate String variables.**

   What are they?

   Strings are the data type corresponding to text. It is difficult to imagine an automation scenario that doesn't involve the use of strings.

   Anytime a text needs to be captured, processed, sent between applications, or displayed, strings come in handy (unless the data is structured - like a table).

   String methods

   String manipulation is done by using String Methods borrowed from VB.Net. Below are some of the most common methods used in RPA.

   - **String.Concat**
     Concatenates the string representations of two specified objects.
     Expression: *String.Concat (VarName1, VarName2)*
   - **Contains**
     Checks whether a specified substring occurs within a string. Returns true or false.
     Expression: *VarName.Contains ("text")*
   - **String.Format**
     Converts the value of objects to strings (and inserts them into another text).
     Expression: *String.Format("{0} is {1}", VarName1, VarName2)*
   - **IndexOf**
     Returns the zero-based index of the first occurrence of a specified Unicode character or string within this instance.
     Expression: *VarName1.IndexOf("a")*
   - **LastIndexOf**
     Reports the zero-based index position of the last occurrence of a specified Unicode character or string within this instance.
     Expression: *VarName1.LastIndexOf("author")*
   - **String.Join**
     Concatenates the elements in a collection and displays them as String.
     Expression: *String.Join("|", CollVarName1)*
   - **Replace**
     Replaces all the occurrences of a substring in a string.
     Expression: *VarName.Replace ("original", "replaced")*
   - **Split**
     Splits a string into substrings using a given separator.
     Expression: *VarName.Split("|"c)(index)*
   - **Substring**
     Extracts a substring from a string using the starting index and the length.
     Expression: VarName1.Substring(startIndex, length)

2. Use specific methods to initialize and manipulate List variables.

Lists (or List<T>, as you will encounter them) are data structures consisting of objects of the same data type (for example string or integer). Each object has a fixed position in the list; thus, it can be accessed by index. While arrays are fixed-size structures for storing multiple objects, lists allow us to add, insert and remove items.
Lists can store large numbers of elements - names, numbers, time coordinates and many others. Lists provide specific methods of manipulation, such as:

- Adding and removing items.
- Searching for an element.
- Looping through the items (and performing certain actions on each).
- Sorting the objects.
- Extracting items and converting them to other data types.

Add to Collection - Adds an item to a specified collection. It is equivalent to List.Add(). It can be used, for example, to add a new name to a list of company names.

Remove From Collection - Removes an item from a specified collection and can output a Boolean variable that confirms the success of the removal operation. This activity can be used, for example, to remove an invoice number from a list of invoices to be processed.

Exists in Collection - Indicates whether a given item is present in a given collection by outputting a Boolean as the result. We can use this activity to check whether a list of clients contains a specific name.

Clear Collection - Clears a specified collection of all items. One possible use is to empty a collection before starting a new phase of a process that will populate it again.

3. Use specific methods to initialize and manipulate Dictionary variables.

Dictionaries (or Dictionary<TKey, TValue>, as you will encounter them) are collections of (key, value) pairs, in which the keys are unique. Think of the Address Book in your mobile phone, where each name has corresponding data (phone number(s), email).
The data types for both keys and values have to be chosen when the variable is declared. Data types in Dictionaries can be any of the supported variables (including Dictionaries, for example).
The operations that are most often associated with Dictionaries are:

- Adding and deleting (key, value) pairs.
- Retrieving the value associated with a key.
- Re-assigning new values to existing keys.

Initialization - Just like in the example of Lists, Dictionary variables need to be initialized with instantiated objects. In the previous example, the instantiation and initialization were done inside an 'Assign' activity. However, as you may remember from the Lists chapter, they can also be done from the Variables panel.

Adding Key-Value Pairs - VarName.Add(Key, Value) – adds an item to an existing Dictionary. Because Add does not return a value, we use the Invoke Method activity.

Removing Keys - VarName.Remove(Key) – removes an item from the Dictionary. It can be used in an 'Assign' activity.

Retrieving –

- VarName.Item(Key) – returns the Dictionary item by its key
- VarName.Count – returns an Int32 value of the number of Dictionary items
- VarName.ContainsKey(Key) – checks if the item with the given key exists in the Dictionary and returns a Boolean result
- VarName.TryGetValue(Key, Value) – checks if an item with a given key exists in the Dictionary and returns a Boolean result and the value if found

4. Use the RegEx builder in UiPath Studio to perform complex string manipulation.

Regular Expression (REGEX, or regexp) is a specific search pattern that can be used to easily match, locate and manage text. However, creating RegEx expressions may be challenging.
UiPath Studio contains a RegEx builder that simplifies the creation of regular expressions.
Typical uses of RegEx include:

- Input validation.
- String parsing.
- Data scraping.
- String manipulation.

Matches - Searches an input string for all occurrences and returns all the successful matches.
Output datatype: System.Collections.Generic.IEnumerable<System.Text.RegularExpressions.Match>

IsMatch - Indicates whether the specified regular expression finds a match in the specified input string.

Replace - Replaces strings that match a regular expression pattern with a specified replacement string.

5. Use specific methods to perform data manipulation operations using DateTime variables.

As the name implies, the DateTime type (System.DateTime) enables you to store information about dates and times in variables.
This type of variable can be found in the Browse and Select a .Net type window, under the System namespace System.DateTime.
Typical uses of DateTime variables include appending documents or performing time span calculations.

ToString - Converts variables to string formats. Data can be further manipulated to specific types of strings as well.

DateTime.Parse - Converts variables from string to DateTime variables. This method only works with Invariant culture time formatting variations.

Subtract - Performs a subtract operation on a DateTime variable unit value. Units can vary from days to milliseconds.

Math.Abs - Is used to display an absolute value.
For example, our result is "-5" but we want the result to be displayed as "5".
During processing, we cannot utilize dates with negative numbers which can result from different DateTime operations.

Hurmet Noka

8. Selectors in Studio

Objectives:

1. Explain what selectors are and how they work.

A selector in UiPath Studio is a feature that enables the identification of a specific User Interface element through its address and attributes. These are stored as XML fragments.

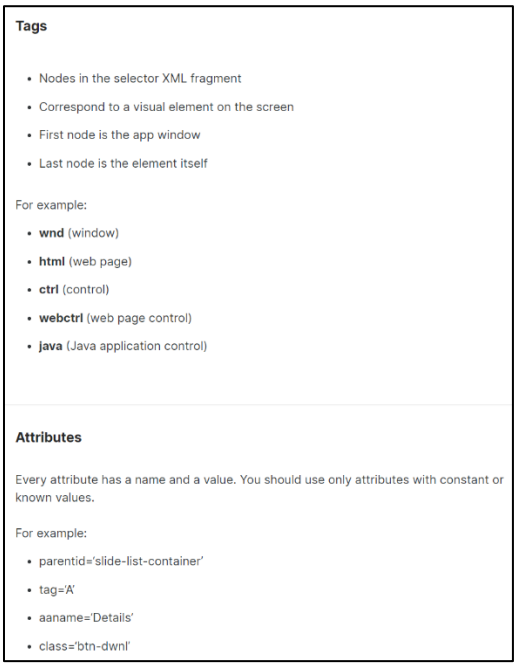
The element identification is done to perform specific activities in an automation project. Selectors are generated automatically every time we use an activity that interacts with graphical user interface elements.

We can think of the element identification process achieved through selectors as a postman that delivers letters to a certain address. In order for the postman to deliver the letters, a specific path is required and must contain structured and hierarchized details such as Country > City > Zip Code > Street Name > Street Number > Apartment Number. Similarly, UiPath Studio requires the detailed path to a specific element within the user interface.

Tags & Attributes of Selectors

As you saw, selectors are made of nodes. And each node is made of tags and attributes. Let's take an example to explain the two. Below is a selector node.

<webctrl parentid='slide-list-container' tag='A' aaname='Details' class='btn-dwnl' />



Tags

- Nodes in the selector XML fragment
- Correspond to a visual element on the screen
- First node is the app window
- Last node is the element itself

For example:

- **wnd** (window)
- **html** (web page)
- **ctrl** (control)
- **webctrl** (web page control)
- **java** (Java application control)

Attributes

Every attribute has a name and a value. You should use only attributes with constant or known values.

For example:

- parentid='slide-list-container'
- tag='A'
- aaname='Details'
- class='btn-dwnl'

2. Choose the correct selectors types and settings when automating.

As previously presented, selectors are automatically generated when UI elements are indicated inside activities or when the recorder is used. Knowing the difference between full and partial selectors is very important when activities generated inside containers are moved outside, or the other way around.

The UI automation classic experience containers are Attach Window, Attach Browser and Open Browser.

Full selectors

Contain all the tags and attributes needed to identify a UI element, including the top-level window;
Generated by the Basic Recorder;
Best suited when the actions performed require switching between multiple windows.

Partial selectors

Don't contain the tags and attributes of the top-level window, thus the activities with partial selectors must be enclosed in containers;
Generated by the Desktop Recorder;
Best suited for performing multiple actions in the same window.

Hurmet Noka

3. Describe how to explore the attributes of User Interface element.

The UI Explorer is the functionality in UiPath Studio that allows you to analyze and edit selectors. It contains a status button showing users the state of the selector, a visual tree panel that displays a navigable UI of each application running at that moment, as well as the selected UI element. The UI Explorer displays all the available tags and attributes and gives you the option to check them in or out.

The UI Explorer can be used whenever the selectors that were automatically generated are not stable or reliable enough. For example, when:

- The selectors change from one execution to another.
- The selectors might change following product updates.
- The selectors use unreliable info, such as index.

4. Fine-tune selectors to improve element identification precision.

Fine-tuning is the process of refining selectors in order to have the workflow correctly executed in situations in which the generated selector is unreliable, too specific or too sensitive with regards to system changes.
It mainly consists of small simple changes that have a larger impact on the overall process, such as adding wildcards, using the repair function or using variables in selectors.

When exactly is fine-tuning of selectors required?

- **Dynamically generated selectors** - As it happens with some websites, the values of the attributes change with each visit.
- **Selectors being too specific** - Some selectors are automatically generated with the name of the file or with a value that changes. Here, placeholders are very useful.
- **System changes** - Some selectors contain the version of the application or another element that changes when the application is updated.
- **Selectors using IDX** - The IDX is the index of the current element in a container with multiple similar elements. This might change when a new element appears in the same container.

How can we fine-tune selectors?

Using wildcards

Wildcards are symbols that enable you to replace zero or multiple characters in a string. These are useful when dealing with dynamically-changing attributes in a selector.
Asterisk (*) – replaces zero or more characters
Question mark (?) – replaces a single character

Using variables

Variables are used as a property for the attribute of your target tag. This allows selectors to easily identify a target element based on the value of the variable or argument, and not an exact string, which might change.
The variable can be changed to interact with a different element, without changing the selector itself.

Using index variables

Index variables are used to access UI elements based on their numerical position in the list or to access specific UI elements in an array or structure. They work by identifying the attributes of the UI elements which are numeric values that are consecutive.
When data is stored in a list structure and you want to refer to them based on their position or numeric value, we use Index Variables. They are stored in defined variable with specific numeric value. For example:
webctrl tableCol='6' tag='TD'tableRow='{{int_Index}}'/>

Managing Difficult Situations

In most of the cases in which the selectors automatically generated are not reliable enough, fine-tuning will solve the issue. However, there are some other situations that we call difficult. Consider the example of a UI element that changes state, position or ID every time the workflow is run.

For these, there are other approaches:

- **Anchor base**
  This is very useful in cases in which the attribute values are not reliable (are generated at each execution, for example), but there is a UI element that is stable and is linked to the target UI element.
  The Anchor Base activity has two parts, one to locate the anchor UI element (like 'Find Element'), and the second to perform the desired activity
- **Relative selector**
  This activity will basically incorporate the information about the anchor's selector in the selector of the target UI element. However, the new selector will probably need additional editing, as some nodes of the first selector will still be in the new one. The solution is to have that part (like a dynamic ID) removed, and the selector will stabilize using the anchor's selector.
- **Visual tree hierarchy**
  The hierarchy in the Visual Tree can improve the reliability of a selector by including the tags and attributes of the element that is above in the hierarchy.
  This is very useful when the target UI element's selector is not reliable, but the selector of the UI element right above in the hierarchy is. However, again, the selector needs further editing and validation, as the dynamic part needs to be removed and, at the same time, you need to make sure that the target element can be identified with a unique attribute.
- **Find children**
  This activity can identify all the children of an element that is more stable. Since its output is the collection of children, you will need to come up with a mechanism to identify only the target UI element (using one of its attributes, that makes is unique between the children, but wouldn't be enough to identify it universally).

Hurmet Noka

9. Project Organization in Studio

Objectives:

1. Choose a suitable project layout for each workflow.

For small processes automated or parts of a larger automation project, there are 3 options of layout – Sequence, Flowchart and State Machine.

Sequence

When to use it?

When there's clear succession of steps, without too many conditions (for example, a UI automation).

Usually, sequences are used to nest workflows and the high-level logic is handled through flowcharts or state machines.

What are the advantages?

Easy to understand and follow, having a top to bottom approach.

Great for simple logic, like searching for an item on the internet.

What are the disadvantages?

Nesting too many conditions in the same sequence makes the process hard to read.

Not suitable for continuous flows.

Flowchart

When to use it?

When you have a complex flow with several conditions, a flowchart is at least visually much easier to understand and follow.

When you need a flow that terminates only in one of several conditions.

What are the advantages?

Easy to understand, as it is similar to logic diagrams in software computing.

The most important aspect of flowcharts is that, unlike sequences, they present multiple branching logical operators that enable you to create complex business processes and connect activities in multiple ways.

What are the disadvantages?

Flowcharts should be used only as the general workflow (with sequences nested inside), not for individual parts of projects (nested inside other workflows).

State Machine

When to use it?

First of all, let's understand what a state machine is. It is an abstract machine consisting of a finite number of pre-defined states and transitions between these states. At any point, based on the external inputs and conditions verified, it can be in only one of the states.

State machines can be used with a finite number of clear and stable states to go through. Some examples from your daily life include vending machines, elevators or traffic lights.

What are the advantages?

Can be used for continuous workflows that are more complex.

Transitions between states can be easily defined and offer flexibility.

Can accommodate processes that are more complex and cannot be captured by simple loops and If statements.

It is easier to cover all the possible cases/transitions with state machines.

What are the disadvantages?

Longer development time due to their complexity: splitting the process into logical "states", figuring out transitions, and so on.

Note: State machines are not to be overused - they should define only the skeleton of the project.

In fact, there are templates built upon state machines especially designed to build large enterprise automations. The most commonly used is the Robotic Enterprise Framework - we have a separate course to cover it in the RPA Developer Advanced learning plan.

2. **Split a complex automation project into functional workflows that can be developed separately.**

   Breaking an automation process down into smaller workflows ensures speed of development and reliability, by allowing independent testing of components while encouraging team collaboration. Both are paramount for the success of such initiatives. Moreover, in complex automations (like most of the enterprise projects are), the question is not if it should be broken down, but how to do it.
   There are multiple ways in which the workflows can be split and at least 3 factors should be considered as breakdown criteria:

   - The application that is being automated
   - The purpose of a certain operation (login, processing, reading a document using OCR, filling in a template, and so on)
   - The length of each workflow
   - Workflow reusability in other projects

   For example, a complex process can be split into workflows for each application, and for each of those applications, split on input, processing or output. If any of these workflows are too long, they can be further split, having in mind also a purpose to do it.

3. **Create and share project templates to speed up development.**
   Libraries

   Extracting workflows and reusing them across an automation project is a good habit for keeping a project organized, readable, and sustainable. At the same time, there are cases in which workflows can be reused in different projects. Consider the sequence of logging into SAP. Every time an automation project deals with SAP, the same sequence of steps will be needed.

   Storing and reusing components in separate projects is done through process libraries. A process library is a package that contains multiple reusable components, which consist of one or more workflows that act as individual activities. Libraries are saved as **.nupkg** files, and then installed in different projects using the Package Manager.

   How do we identify if a part of a project is a reusable? Consider what sequences of activities can be used in several processes. For example: login, logout, starting multiple applications common to several processes, data entry sequences.

4. **Identify reusable components across projects and store them as libraries for future reuse.**
5. **Explain the benefits of using exception handling techniques.**

   It is common for automation projects to encounter events that interrupt or interfere with the projected execution. Some of these are identified in the development and testing phases, and handling mechanisms are implemented.

   Consider an automation where the input data comes from a web form, and the application tries to match the data with the list of existing clients using the last name. But what if the user makes a mistake when spelling the name? Naturally, the name won't be recognized.

   A good project design will include ways of recognizing and identifying the exception, and also patterns of action that are executed only when exceptions are caught. These can be simply stopping the execution, or explicit actions executed automatically within the workflow, or even escalating the issue to a human operator.
   Predicting and treating exceptions can be done in two ways:

   - At activity level, using Try/Catch blocks or Retry Scope.
   - At a global level, using the Global Exception Handler.

Both will be covered in depth in the Error and Exception Handling in Studio course. At this point, it is important to be able to choose the correct type of exception, as the information will be used at a higher level later in the development to make other decisions. The categories of exceptions are:

Application exception

The Application Exception describes an error rooted in a technical issue, such as an application that is not responding. Consider a project extracting phone numbers from an employee database and inserting them into a financial application. If, when the transaction is attempted, the financial application freezes, the Robot cannot find the field where it should insert the phone number, and eventually throws an error. These kinds of issues have a chance of being solved simply by retrying the transaction, as the application can unfreeze.

In managing application exceptions, it is extremely important to have good naming conventions for activities and workflows. This will help with tracking the activity that caused the exception.

Business rule exception

The Business Exception describes an error rooted in the fact that certain data which the automation project depends on is incomplete, missing, outside of set boundaries (like trying to extract more from the ATM than the daily limit) or does not pass other data validation criteria (like an invoice amount containing letters). Business Rule Exceptions do not occur "naturally", they need to be defined using a Throw activity.

Consider a robot that processes invoices, with a business rule set by the process owner that only invoices with the amount below 1 000$ must enter the automated process. For the others, a different flow is applicable, involving a human user to process them.

In this case, retrying the transaction does not yield any chance of solving the issue, instead the business user should be notified about the pending invoice and the case should be treated as a business exception - because is an exception from the usual process flow and the validation is made explicitly by the developer inside the workflow.

As a recommended practice, the text in the exception should contain enough information for a human user (business user or developer) to understand what happened and what actions need to be taken.

6. Explain the benefits of using the versioning capabilities of UiPath to keep the development effort trackable and reliable.

Version Control Systems Integration

Source control systems are particularly useful in larger projects, where multiple teams and individuals contribute in each stage. Source control systems allow users in different teams and locations to access the same resources and work on the same project. They are also used for versioning the code and maintaining the history of all the changes made during development.

Through the Teams page in the backstage view, Studio supports the following source control systems:

Git

With the Git integration you can:

- Clone a remote repository.
- Add a project.
- Commit and push.
- Copy a project to Git.
- Create and manage branches.
- Solve conflicts with File Diff option.

Hurmet Noka

10. Debugging in Studio

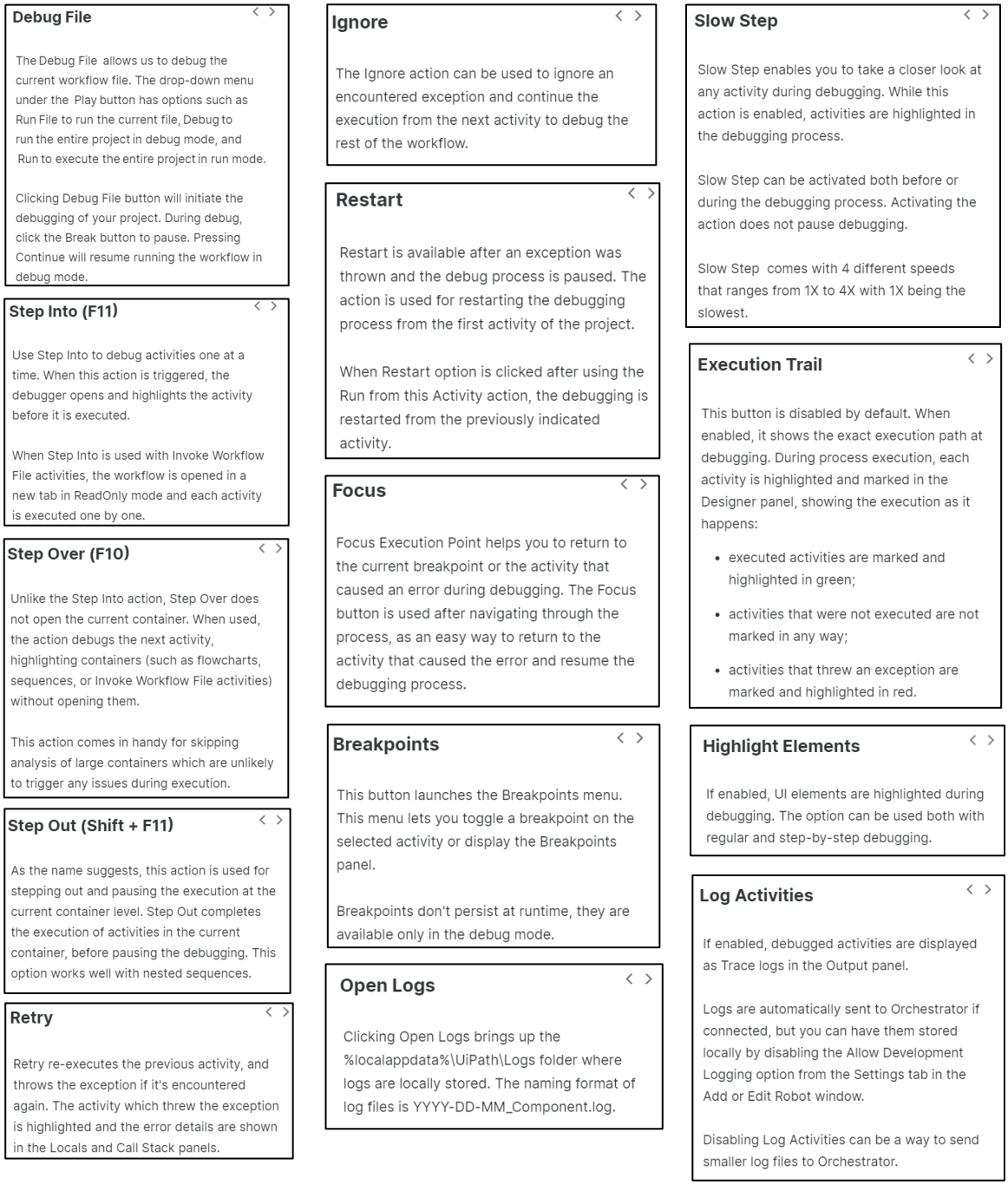
Objectives:

1. Explain what all the debug actions do.

What debugging is?

Debugging is a process of detecting and resolving errors in a project that would cause it to behave unexpectedly or crash. It is usually recommended to perform debugging during the design stage of the automation project, at activity, file, and project level. Let's learn the Debugging actions and panels in the next section.

The debugging actions and panels



Hurmet Noka

2. **Explain what all the debug panels indicate.**

Locals

The Locals panel is only visible while debugging. The panel shows:

- Exceptions - the description and type of the exception.
- Arguments
- Variables
- Properties of previously executed activity - only input and output properties are displayed.
- Properties of current activity

Designer

In debug mode, the Designer panel displays the workflow, the breakpoints on activities and highlights the currently executed activity in yellow and an activity throwing an exception in red.

Highlight current action

In debug mode, the currently running activity is highlighted in the Designer panel. This also applies to actions on which the execution is paused.

Watch

Similar to the Call Stack panel, the Watch panel is only visible during debugging. It can be set to display the values of variables or arguments, and values of user-defined expressions that are in scope. These values are updated after each activity execution while debugging.

Immediate

The Immediate panel is only visible during debugging, and it can be used for evaluating variables, arguments, or statements and inspecting data available at a certain point during debugging. To do so, simply type the variable or argument name in the Immediate window and press Enter.

The Immediate panel keeps the history of previously evaluated statements, and they can be removed using the Clear All context menu option.

Call Stack

The Call Stack panel displays the next activity to be executed and its parent containers when the project is paused in debugging.

The panel is displayed during execution in debug mode and it gets populated after using Step Into, Break, Slow Step, or after the execution was paused due to an error or a breakpoint.

If during debugging, an activity throws an exception, it is marked in the Call Stack panel and the activity is highlighted in red.

Breakpoints

The Breakpoints panel displays all breakpoints in the current project, together with the file in which they are contained. The Activity Name column shows the activity with the toggled breakpoint, while the File Path column displays the file and its location.

The Condition column displays conditions set to breakpoints. The Log Message column shows messages to be logged if the condition is met. Hover over the breakpoint tag on an activity to view its condition and log message.

Output

The Output panel enables you to display the output of the Log Message or Write Line activities, among other things. Exceptions for packages are also displayed in this panel.

From the Output panel, you can show or hide messages that have different log levels (errors, warnings) by clicking the buttons in the panel's header. Double-clicking a message displays further details about it, with the option to copy information.

3. **Use all the UiPath Studio debug features to make your projects run as expected.**

Hurmet Noka

11. Error and Exception Handling in Studio

Errors

Errors are events that a particular program can't normally deal with. There are different types of errors, based on what's causing them - for example:

- **Syntax errors**, where the compiler/interpreter cannot parse the written code into meaningful computer instructions.
- **User errors,** where the software determines that the user's input is not acceptable for some reason.
- **Programming errors,** where the program contains no syntax errors but does not produce the expected results. These types of errors are often called bugs.

Exceptions

Exceptions are events that are recognized (caught) by the program, categorized, and handled. More specifically, there is a routine configured by the developer that is activated when an exception is caught. Sometimes, the handling mechanism can be simply stopping the execution.
Some of the exceptions are linked to the systems used, while others are linked to the logic of the business process.

System exceptions

Below you can find the most common exceptions that you can encounter in projects developed with UiPath. As a general note, almost all exceptions are types derived from **System.Exception**, so using this generic type in a TryCatch, for example, will catch all types of errors.

- **NullReferenceException** - Occurs when using a variable with no set value (not initialized).
- **IndexOutOfRangeException** - Occurs when the index of an object is out of the limits of the collection.
- **ArgumentException** - Is thrown when a method is invoked and at least one of the passed arguments does not meet the parameter specification of the called method.
- **SelectorNotFoundException** - Is thrown when the robot is unable to find the designated selector for an activity in the target app within the TimeOut period.
- **ImageOperationException** - Occurs when an image is not found within the TimeOut period.
- **TextNotFoundException** - Occurs when the indicated text is not found within the TimeOut period.
- **ApplicationException** - Describes an error rooted in a technical issue, such as an application that is not responding.

Business exceptions

Business rule exceptions are separate from all the system exceptions listed above. These describe errors rooted in the fact that certain pieces of data which the automation project depends on are incomplete, missing, outside of set boundaries (like trying to extract more from the ATM than its daily limit), or do not pass other data validation criteria.
Business rule exceptions will not be automatically identified by using the generic System.Exception in a TryCatch activity. They need to be defined by a developer by using the Throw activity and handled inside a TryCatch.

Hurmet Noka

Objectives:

1. Describe the common exception handling techniques and explain when they should be used.

TryCatch, Throw and Rethrow

TryCatch activity catches a specified exception type in a sequence or activity, and either displays an error notification or dismisses it and continues the execution.

As a mechanism, TryCatch runs the activities in the Try block and, if an error takes place, executes the activities in the Catches block. The Finally block is only executed when no exceptions are thrown or when an exception is caught and handled in the Catches block (without being re-thrown).

Try

The activities performed which have a chance of throwing an error.

Catches

The activity or set of activities to be performed when an exception occurs. Please note that multiple exceptions and corresponding activities can be added to this block.

Finally

The activity or set of activities to be performed after the Try and Catches blocks are executed. This section is executed only when no exceptions are thrown or when an error occurs and is caught in the Catches block (without being re-thrown).

Retry Scope

The Retry Scope activity retries the contained activities as long as the condition is not met or an error is thrown.
The Retry Scope activity is used for catching and handling an error, which is why it's similar to TryCatch. The difference is that this activity simply retries the execution instead of providing a more complex handling mechanism.

The activity has 2 main sections:
Action - Holds the activities we want to retry.
Condition - Holds the termination condition. The Retry Scope activity retries the activities in the Action section as long as this termination condition is not met.
It can be used without a termination condition, in which case it will retry the activities until no exception occurs or the provided number of attempts is exceeded.

Additional properties

NumberOfRetries - The number of times that the sequence is to be retried.
RetryInterval - Specifies the amount of time (in seconds) between each retry.

The ContinueOnError Property

The ContinueOnError is a property that specifies if the execution of the activity should continue even when the activity throws an error.
This field only supports Boolean values (True, False). The default value is False, thus, if the field is blank and an error is thrown, the execution of the project stops. If the value is set to True, the execution of the project continues regardless of any error.
Keep in mind that, if the ContinueOnError is set to True on an activity that has a scope (such as Attach Window or Attach Browser), then all the errors that occur in other activities inside that scope are also ignored.

Setting this property to True is not recommended in all situations. But there are some in which it makes sense, such as:

- While using data scraping - So that the activity doesn't throw an error on the last page when the selector of the 'Next' button is no longer found.
- When we are not interested in capturing the error but simply in the execution of the activity.

2. **Use the TryCatch, Throw, Rethrow and Retry Scope activities in your automation projects.**
3. **Use the Global Exception Handler in both attended and unattended scenarios.**

The Global Exception Handler

The Global Exception Handler is a type of workflow designed to determine the process' behavior when encountering an unexpected exception. This is why only one Global Exception Handler can be set per automation project.

In its default configuration, the Global Handler catches the exceptions thrown by any activity in the process at runtime and executes a standard response - ignore, retry, abort or continue, as predefined at design time. For attended scenarios, it can be configured to let the user select the action.

A Global Exception Handler can be created either by adding a new workflow file with this type, or by setting an existing workflow as Global Exception Handler from the Project panel.

How does it work?

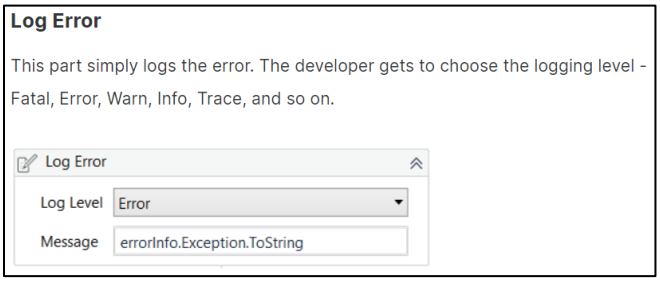
The Global Exception Handler has 2 predefined arguments, that shouldn't be removed:

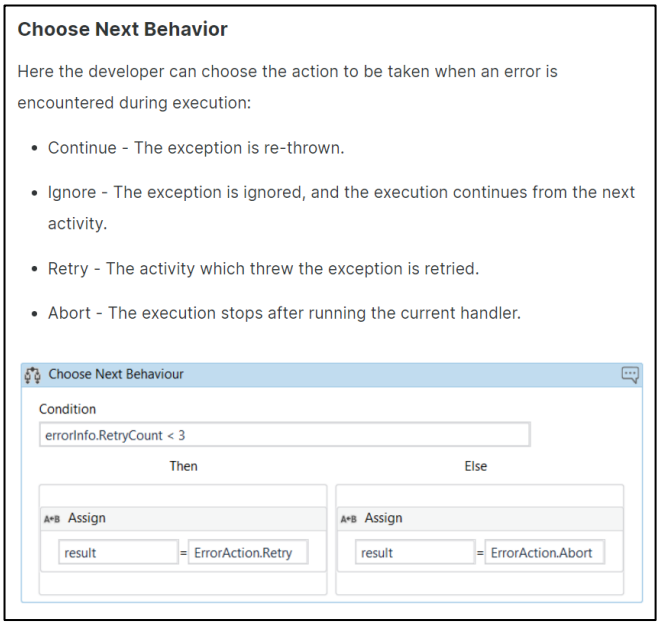
- **errorInfo** with the In direction - contains the information about the error that was thrown and the workflow that failed.
- **result** with the Out direction - used for determining the next behaviour of the process when it encounters the error.
-

The Global Exception Handler contains the predefined activities below. If required, we can also add other activities in our Handler.

Note that the Global Exception Handler is not available for library projects, only processes.

Only uncaught exceptions will reach the exception handler. If an exception occurs inside a TryCatch activity and is successfully caught and handled inside the Catches block (and not re-thrown), it will not reach the Global Exception Handler.





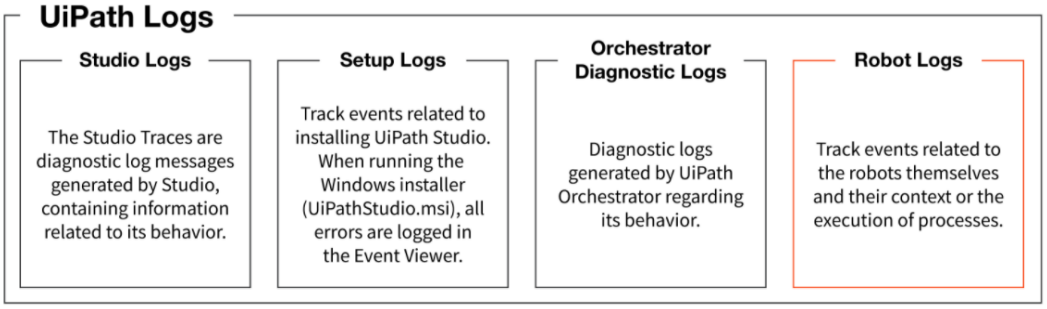
Hurmet Noka

12. Introduction to Logging in Studio

Objectives:

1. Explain what Robot execution logs are.

Types of logs in UiPath
In the beginning, we've mentioned that the focus of this lesson is setting up useful logging when developing project. For a bit of context, let's start by checking out the main types of logs generated by the UiPath platform:



Robot logs
Robot logs are messages generated by the UiPath Robot. There are two types of Robot logs: Robot execution logs and Robot diagnostic logs. Robot execution logs describe the execution of processes while Robot diagnostic logs describe the functioning of the Robot. We will focus on Robot execution logs.

Let's see how default logs and custom logs are created.

Robot execution logs
The types of logs generated while running processes in the studio are a subset of Robot logs, called Robot Execution logs.

Let's see how default logs and custom logs are created.

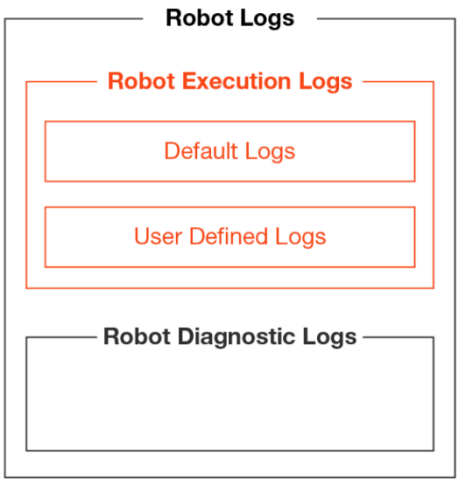
Robot Execution Logs
Robot Execution logs can be used to supervise, diagnose, and debug processes in production, gather process performance data or even track business results like the total value of transactions processed. In this lesson, we are focusing on the **supervising, diagnosing, and debugging** aspects.

2. Explain what default logs, user-defined logs and log levels are.

Robot execution logs can be either **default** logs or **user-defined** logs.

Default logs are generated automatically when certain events take place. The events logged by this category are:

- **Execution start** is generated every time a process is started (Level = Information)
- **Execution end** is generated every time a process is finalized (Level = Information)
- **Transaction start** is generated every time a transaction within a process is started (Level = Information)
- **Transaction end** is generated every time a transaction within a process is finalized (Level = Information)
- **Error log** is generated every time the execution encounters an error and stops (Level = Error)
- **Debugging log** is generated if the Robot Logging Setting is set to Verbose and contains, activity names, types, variable values, arguments, etc. (Level = Trace)



User-defined logs are generated according to the process designed by the user in Studio, when using the Log Message activity or by the Write Line activity.

Logging levels

Logging levels let us sort log messages by severity, or how important a log message is related to the execution of a process. Log levels are used in Orchestrator to filter out logs below a specific level.

Logging Level	Meaning
Fatal	The robot cannot or should not recover from this error. Something has gone critically wrong and the process needs to be stopped. For example, the robot has no means of handling an exception or the website it's interacting with displays a message that it is under maintenance.
Error	An error occurred. The robot will attempt to recover and move on with the next item.
Warn	Any important data that we need to stand out from the rest of the log information.
Info	Information about robot progress. Usually includes when we enter/exit a workflow, when data is read from an external source, etc.
Trace	Information useful while developing/debugging, however not useful and needed in production.
Debugging / Verbose level	Verbose level generates default logs for the execution of each individual activity, allowing a much more in-depth diagnosis by giving more information about the values of variables and arguments We can generate Verbose logs by enabling the '**Log Activities**' option in the Studio.

3. Interpret Robot execution logs.

Accessing and Reading Robot Execution Logs

There are several places where you can access Robot Execution Logs:

- In the **Output Panel** in UiPath Studio for the previous process execution from Studio.
- In the **%localappdata%\UiPath\Logs\<shortdate>_Execution.log** file for all processes ran on the machine from UiPath Studio. Logs will be generated at Trace level and above or Verbose level and above depending on whether the Verbose level is activated or not.
- In the **%localappdata%\UiPath\Logs\<shortdate>_Execution.log** file for all processes ran on the machine from UiPath Assistant. The logs will be generated at the level defined in UiPath Assistant and above.
- In **Orchestrator**, in the Logs section when running processes while connected to Orchestrator. The Logs will be generated at the defined level and above.

As in this course we are learning how to write effective Log messages and interpret log files, we are focusing on accessing logs from the Output panel and **%localappdata%\UiPath\Logs\<shortdate>_Execution.log**.

The anatomy of a log entry

Let's take a moment to understand how logs entries are structured. This can be very helpful when checking execution logs to analyse behaviour or figure out what caused an exception. Logs are in the form of a **JSON**, pretty much just a **key-value pair ("field1:value1," "field2:value2").**

The default log fields are present in all logs:

- **Message**: The Log Message
- **Level**: Defines the log severity
- **Timestamp**: The exact date and time the action was performed
- **FileName**: The name of the .xaml file being "executed"
- **JobId**: The key of the job running the process
- **ProcessName**: The name of the process that triggered the logging
- **ProcessVersion**: The version number of the process
- **WindowsIdentity**: The name of the user that performed the action that was logged
- **RobotName**: The name of the robot (defined in Orchestrator)

Aside from the default fields, logs can also contain type-specific fields and user-defined fields.

- **Type-specific** fields are present depending on the log type, like totalExecutionTimeInSeconds and totalExecutionTime for Execution End.
- **User-defined** fields are defined in Studio (by using the Add Log Fields activity) and appear in all subsequent logs after the activity is generated unless they are (programmatically) removed by the activity Remove Log Fields.

4. Use Log Message activities effectively.

Logging Best Practices

When diagnosing a process in production, you will find that often times, the default exception log messages are helpful, but not sufficient to provide a full picture of why it's failing. User defined logs are the breadcrumbs you can use to track your robot's progress through the process and get a better view of the execution.

Let's take a moment to recap what log levels should indicate in user-defined logs.

Logging Level	Meaning
Fatal	The robot cannot or should not recover from this error. Something has gone critically wrong and the process needs to be stopped. For example, the robot has no means of handling an exception or the website it's interacting with displays a message that it is under maintenance.
Error	An error occurred. The robot will attempt to recover and move on with the next item.
Warn	Any important data that we need to stand out from the rest of the log information.
Info	Information about robot progress. Usually includes when we enter/exit a workflow, when data is read from an external source, etc.
Trace	Information useful while developing/debugging, however not useful and needed in production.
Debugging / Verbose level	Verbose level generates default logs for the execution of each individual activity, allowing a much more in-depth diagnosis by giving more information about the values of variables and arguments. (at trace level). We can generate Verbose logs by enabling the '**Log Activities**' option in the Studio.

On the other hand, **over-logging** can increase the load on Orchestrator, slow your process down and make it hard to diagnose an issue because of the sheer volume of log entries.

The solution is to define the right log messages in strategic points in your project. The Best Practices below cover some healthy uses for identifying events that could lead to issues.

Log message activities should ideally be used:

- At the beginning and the end of every workflow **(Log level = Information).**
- Each time an exception is caught in a Catch block **(Log level = Error).**
- Each time a Business Rule Exception is thrown **(Log Level = Error).**
- When data is read from external sources, for example, log a message at Information level when an Excel file is read **(Log Level = Information).**
- In Parallel or Pick activities, log messages on every branch, in order to trace the branch is **taken (Log Level = Information).**
- In If/Flowchart Decision/Switch/Flow Switch activities (however, since processes might have a lot of these activities, we can decrease the Log Level from Information to trace, so we don't have a lot of these logs in the database).

Hurmet Noka

13. Orchestrator for RPA Developers

Objectives:

1. **Define the purpose and main capabilities of Orchestrator.**

   Orchestrator is the component of the UiPath Platform for managing automations, robots and the related entities. Although having different cloud and on-premises deployment options and a rather complex infrastructure consisting of nodes, servers and high-availability capabilities, the users access it through a simple web interface.

   Orchestrator offers role-based access control and a structure of tenants and folders to replicate organizational structures. Users are able to run the automation workflows developed in Studio and published to Orchestrator using the unattended robot workforce. Furthermore, Orchestrator is used to manage and distribute licenses, as well as for storing automation resources.

   What are Orchestrator's main capabilities?

   - **Provisioning**: Creates and maintains the connection with robots and attended users.
   - **Control and license distribution**: Enables the creation, assignment and maintenance of licenses, roles, permissions, groups, and folder hierarchies.
   - **Running automation jobs in unattended mode**: Enables the creation and distribution of automation jobs in various ways, including through queues and triggers.
   - **Automation storage and distribution**: Allows the controlled storage and distribution of automation projects, assets, and credentials, as well as large files used in automations.
   - **Monitoring**: Allows monitoring of jobs and robots and stores logs for auditing and analytics.
   - **Inter-connectivity**: Acts as the centralized point of communication for third-party solutions or applications.

2. **Describe the Orchestrator entities and what they are meant for, as well as differentiate between the tenant context and the folder context.**

Robot (Orchestrator entity) –

By now you are familiar with UiPath Robot, the component of the UiPath Platform. This is an execution host that runs automation processes published in Orchestrator, as jobs.

In Orchestrator, a robot entity represents an image or the Robot component, controlling its connection and capabilities. The robot entity exists only if it is defined in relation to a user in Orchestrator.

Folder –

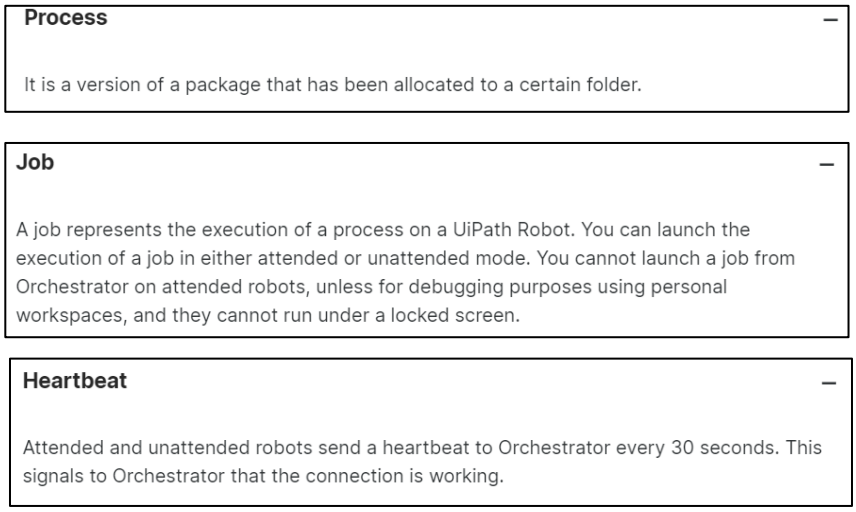
Folders enable the separation and hierarchical organization of automation entities (processes, queues, assets) and the fine-grained configuration of roles and permissions. The hierarchical structure allows up to 6 sub-folders under each first level folder.

Folders help replicate the organizational hierarchies, with the separation of automated processes between teams, segregation of process data, and access control for users. At the same time, when it makes sense, they allow sharing of the resources and assets.

Package –

A project developed in UiPath Studio that is published to Orchestrator as a NuGet package. Multiple versions of the same project can be stored and used.

Packages can also be manually uploaded to Orchestrator.



Tenants and folders

Just like folders, tenants are meant to replicate organizational hierarchies within the same instance of Orchestrator.

From a hierarchy perspective, folders are subdivisions of tenants. And while in the folders case it's more about hierarchization and separation, tenants are clearly isolated from one another.

Consider a typical large company, in which both the data and the business processes are typically separated between divisions like Sales and Finance. But then, the subdivisions would have some of the data or some of the processes separated, at the same time sharing others.

In Orchestrator, some of the entities exist in the tenant context, while others exist in the folder context:

Tenant entities

From the entities defined above, robots are tenant entities. This means that they can be allocated to multiple folders in that tenant. Using roles and permissions, the way robots work with each of the folders can be customized. We'll see that a bit later.

Packages are published to Orchestrator using feeds. The feeds can be configured to be at tenant level, or at folder level. A package published to the tenant feed can be then used in a process in any of the folders. If it is published using a folder feed, it cannot be used for processes in other folders.

There are a couple of other Orchestrator entities which exist at tenant level:

- **User** - Both human users and robots are uniquely identified with users in Orchestrator.
- **Machine** (Orchestrator entity) - These are Orchestrator entities corresponding to the workstations where human users and robots work. Using API keys, they enable the connection between the physical machines and Orchestrator.
- **License** - The right to use Studio and/or Robots, both attended and unattended, is done through licenses. Licenses exist at tenant level, from where they get distributed to users, and consumed when the machines connect to Orchestrator.
- **Webhook** - Webhooks facilitate the communication between Orchestrator and other applications at API level. These are mapped at tenant level, which means they cannot be differentiated between folders and will provide information for the entire tenant.

The tenant menu in Orchestrator offers access to the audit trail and to the notifications and alerts for all the entities on the tenant, as well as to the third-party credential stores, such as CyberArk. It also contains a settings menu for the entire tenant.

Folder entities

From the entities defined at the beginning of the lesson, processes and jobs are folder entities. Packages depend on the feed configuration.

Apart from them, several other entities exist at folder level:

Asset –

An asset is a piece of data stored in Orchestrator for the use of robots. There are four types of assets:

- **Text** - stores only strings (it is not required to add quotation marks).
- **Bool** - supports true or false values.
- **Integer** - stores only whole numbers.
- **Credential** - contains usernames and passwords that the Robot requires to execute particular processes, such as login details.

Assets can have a global value or a value per user. This means that only the designated user will access a certain value stored in that asset.

Storage bucket –

Storage buckets are entities used for storing files which can be used in automation projects.

Queue –

Queues are containers that can hold an unlimited number of items, storing different types of data.

The process of feeding items to a queue is typically different from the process of processing the queue items, and handled by different robots.

Trigger –

Triggers enable the execution of jobs in a structured manner. There are two types of triggers:

- Time triggers: with these, you can schedule the recurrent execution of a process.
- Queue triggers: these enable the execution of a process based on the new items added to a queue.

3. Create, configure and provision unattended robots from Orchestrator.

User creation

In Orchestrator, both human users with attended licenses (Robot or Studio) and unattended robots need to have a corresponding Orchestrator user.

Depending on the deployment type and the organizational setup, users are added and manage in different ways:

- Users can be added **locally** in on-premises Orchestrator.
- Users have to be added in **Automation Cloud**, then in cloud Orchestrator to grant them a license.
- Users can be added from **Active Directory** for both on-premises and cloud Orchestrator if the integration was configured beforehand.

Automatic robot creation

To simplify the attended and unattended robot creation, as well as the license provisioning, the automation robot creation can be enabled at user level, for both attended and unattended robots, and at group level for attended robots.

Machine templates

Robots run on physical or virtual workstations. These are mirrored in Orchestrator by entities called machines. The machines in Orchestrator work as API key generators, authorizing the connection between the robots and Orchestrator.

There are two types of machines in Orchestrator:

- **Machine templates**: this allows the connection to multiple workstations with a single API key.
- **Standard machines**: this allows the connection between Orchestrator and a single machine. This is suitable for scenarios in which robots need to run on specific machines.

License distribution

In Orchestrator, licenses are also called **runtimes**. They are allocated at machine template level, under Machines in the Tenant menu.

The number of runtimes allocated there should be **matched** with the maximum number of users which can run on a single machine connected using that machine template. On a regular Windows machine, only one user can run. But on a Windows server machine, multiple users can run simultaneously.

Licenses are **consumed** as soon as a machine is connected to Orchestrator, no matter the number of users running on it.

4. **Execute jobs using unattended robots in different ways - as individual jobs, with queues, or in large scale deployments.**
5. **Describe how licenses are allocated and consumed in Orchestrator.**

Unattended Automation with Folders

Functionally, the purpose of attended automation is to have the robots ready to take over the undesirable tasks when the human users need it, in their cycle of work and during work hours.

When it comes to unattended automation, the purpose is quite different: the robot needs to be busy as much as possible, with as little human input as possible. In the video below, we'll first see once again how easily unattended robots are provisioned. Then we will create and run jobs, making full use of the license consumption model, process separation based on UI interaction, and job priorities.

The use of folders and job priorities

By accurately reflecting the business hierarchies with the help of folders, roles and permissions, we can control the access to automation processes and make sure the effort is spent where it brings the most return. Job priorities can ensure that business priorities are well reflected in the automation process.

The separation of processes based on UI interaction

Processes in Orchestrator are now differentiated between those that interact with the user interface (known as foreground processes) and those that don't, called background or headless processes.

This has a significant impact on the way automation jobs are executed. An unattended robot can simultaneously run a foreground job and as many headless jobs as available runtimes on a machine.

The license allocation per machine

Allocating licenses (runtimes) per machine ensures that their consumption is optimized. For example, on a Windows Server machine, multiple robots can open sessions and run unattended jobs up to the maximum number of runtimes.

Custom job allocation strategies

Unattended automation was designed in Orchestrator to ensure resource optimization and effectiveness. But there are cases in which the business logic is more important. Orchestrator offers a couple of features and options to allow the customization of the job allocation process, so that certain jobs or resources are available only for some users:

Assets per user

Assets can be created and set so that only given users have access to them.

Job allocation per user or machine

When an unattended job is created, it can be allocated dynamically and any free robot can pick it up. Or it can be configured to be picked up only by a certain user or to be executed on a specific machine.

Trigger allocation per user

Triggers can be set to allocate jobs dynamically, or to allocate the jobs only to a certain user.

6. **Set up large scale unattended robot workforces.**
7. **Use Orchestrators resources directly in Studio.**
8. **Publish, install and update libraries and templates in Orchestrator.**
9. **Store files in storage buckets and use them in automation projects.**
10. **Use triggers and SLAs to automate job execution.**

What are queues?

Queues are containers that can hold an unlimited number of items. Items can store multiple types of data, by default in free form. If a specific data schema is needed, it can be uploaded at queue creation in the form of a JSON file.

Queues in Orchestrator will store items and enable their distribution individually to robots for processing, and monitoring the status of the items based on the process outcomes. As soon as the queue items enter processing, they become transactions. Items are meant to be indivisible units of work: a customer contract, an invoice, a complaint, and so on.

Why do you need queues?

Working with queues is very useful in large-scale automation scenarios underlined by complex logic. Such scenarios pose many challenges—bringing items from multiple sources together, processing them according to a unique logic, efficient use of resources, or reporting capabilities at individual item and queue level, including the use of SLAs.

Consider the customer enrollment process through forms for a retail company: customers may come from different sources (online, partners, own shops, call center), thus having a smooth process of adding them to the processing line is crucial. Further on, working with queues will ensure that these are processed within the SLA time constraints.

Creating queues
Queues are easily created in Orchestrator from the entry in the menu with the same name. They are folder entities, which allows setting up fine-grained permissions.

When creating a queue, you set the maximum number of retries (the number of times you want a queue item to be retried) and the Unique Reference field (select Yes if you want the transaction references to be unique). Once a queue is created, these settings cannot be modified.

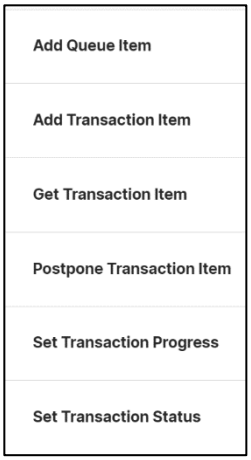
Queues are created empty, but there are specific activities in the UiPath Studio to make the robots populate queues. Bulk upload is also supported directly in Orchestrator, from .csv files.

Populating and consuming queues

To ensure an optimal use of the robots, queues are typically used with the Dispatcher-Performer model of running automations. In this model, the two main stages of a process involving queues is separated:

- The stage in which data is taken and fed into a queue in Orchestrator, from where it can be taken and processed by the robots. This is called Dispatcher.
- The stage in which the data is processed. This is called Performer.

Working with queues and queue items is done using the specific activities located in the UiPath.System.Activities package, under Orchestrator. These are:

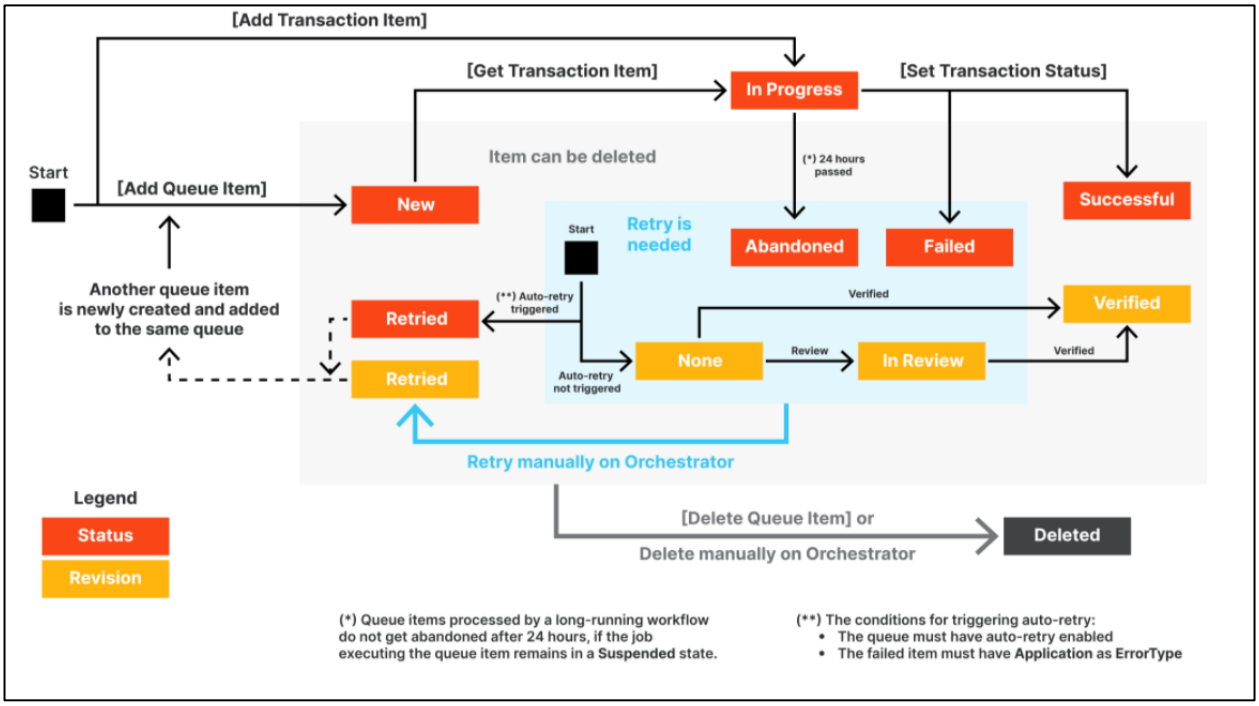


Hurmet Noka

Queue item statuses

A queue item can have one of the statuses below. These will be set automatically following human user and robot actions, and/or using the Set Transaction Status activity. Custom sub-statuses can be set for queue items which are 'In Progress', using the Set Transaction Progres activity.

- **New** +
- **In Progress** +
- **Failed** +
- **Successful** +
- **Abandoned** +
- **Retried** +
- **Deleted** +



Hurmet Noka

Intermission - Transaction Processing Models

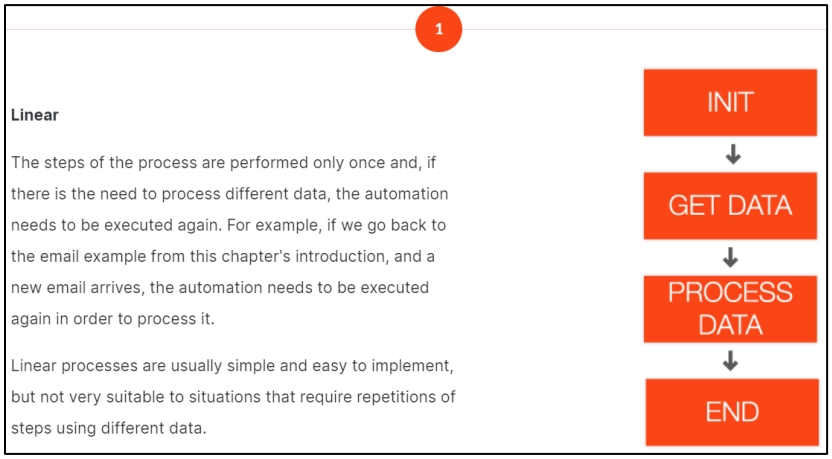
In the Queues lesson we've seen the most complex model of transaction processing. But not the only one. If this is not your first course that covers the capabilities of UiPath Studio, then you've seen the other two models.

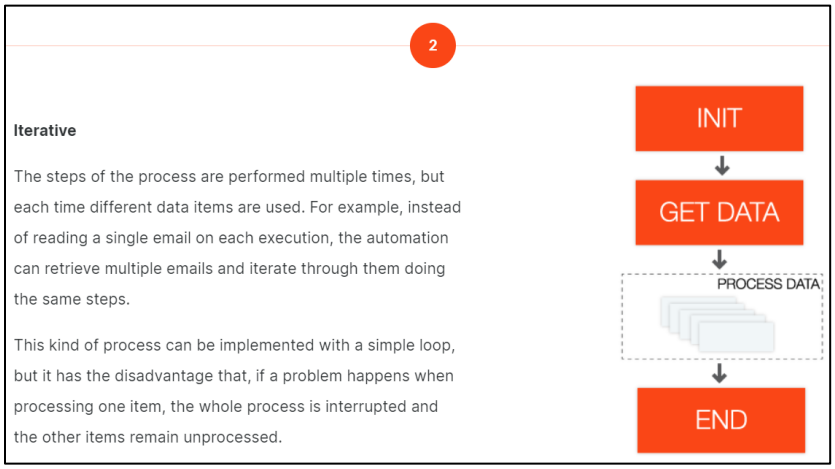
If this is your first course covering UiPath Studio, don't worry! You will easily grasp the other two models of transaction processing.

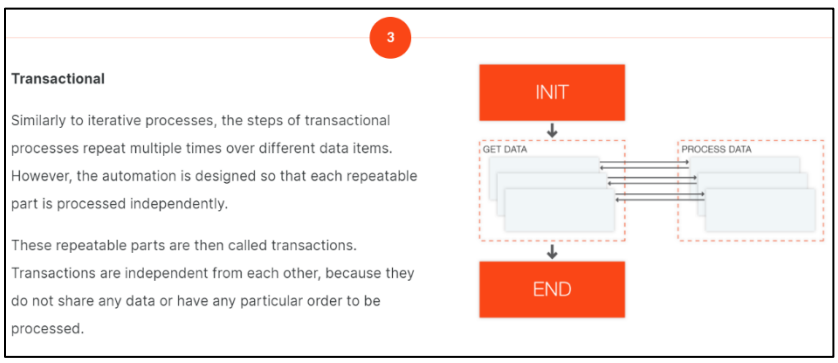
But first, let's make sure we know what a transaction is.

What is a transaction?
A transaction represents the minimum (atomic) amount of data and the necessary steps required to process the data, by fulfilling a section of a business process. A typical example would be a process that reads a single email from a mailbox and extracts data from it. We call the data atomic because once it is processed, the assumption is that we no longer need it as we advance with the business process.

When considering the steps of a business process and how they are repeated, we can divide business processes into three categories:







The three categories of processes can be seen as maturity stages of an automation project, starting with simple linear tasks, which then are repeated multiple times, and finally evolve into a transactional approach.

However, this is not an absolute rule for all cases, and the category should be chosen according to the characteristics of the process (e.g., data being processed and frequency of repetitions) and other relevant requirements (e.g., ease of use and robustness).

What are some business scenarios in which I will use transaction processing?

- You need to read data from several invoices that are in a folder, and input that data into another system. Each invoice can be seen as a transaction, because there is a repetitive process for each of them (i.e. extract the data and input it somewhere else).
- There is a list of people and their email addresses in a spreadsheet, and an email needs to be sent to each of them along with a personalized message. The steps in this process (i.e., get data from spreadsheet, create personalized message, and send email) are the same for each person, so each row in the spreadsheet can be considered a transaction.
- When looking for a new apartment, a robot can be used to make a search according to some criteria. For each result of the search, the robot extracts the information about the property and inserts the data into a spreadsheet. In this case, the details page for each property constitutes a transaction.

Queue triggers and time triggers

What are triggers?
Triggers enable the execution of jobs in a structured manner. There are two types of triggers:

- Time triggers: with these, you can schedule the recurrent execution of a process.
- Queue triggers: these enable the execution of a process based on the new items added to a queue.

Triggers are folder entities, so fine-grained access rights can be defined.

Why do you need triggers?
Time triggers are a great way to automate the execution of a process that needs to have a certain cadence. For example, running a process dealing with the expenses submitted by employees.

Queue triggers are probably the best option to handle queues that are populated with new items at a pace which varies and cannot be predicted. They can also prove to be valuable for processes with strict SLAs and significant consequences of breaching them. Examples may include all the business processes involving customers: on boarding, purchases, and complaints.

Service Level Agreements (SLAs)

What are SLAs in Orchestrator?

An SLA in Orchestrator is just an application of the general concept of SLA. In other words, it is a commitment between a provider and a client with measurable indicators regarding the service provided.

In Orchestrator, this takes the form of a time interval (expressed in days and/or hours) in which a queue item has to be processed. The Risk SLA is a subsequent concept in Orchestrator, meant to identify the point in which corrective action needs to be taken to avoid breaching the SLA.

Why do you need SLAs in Orchestrator?
Automated processes are run based on certain assumptions regarding the volume of the queue items. This dictates the robot capacity allocated to a certain process and queue.

There are cases in which the reality is different from those assumptions. In the absence of an SLA, the breaches may be noticed too late and with costly consequences.

Hurmet Noka

Practice 1 - Connect an Unattended Robot to Orchestrator

You're probably set with your attended environment. Studio is installed and licensed and you have an attended robot connected to Orchestrator via interactive sign-in, and controlled through UiPath Assistant.

It's time to take your environment further by provisioning an unattended robot and connecting in to the same Orchestrator. You need a separate machine to deploy it and an Orchestrator with an unattended license (which could be a community Automation Cloud).

1

In **Automation Cloud** and **Orchestrator**:

1. Add a user in Automation Cloud, in the Automation Users group.
2. Open Orchestrator and go to Users at Tenant level.
3. Add the user in Orchestrator and grant them the 'Allow to be Automation User' role.
4. Toggle the 'Automatically create an unattended robot for this user' from the Unattended Robot tab. Add the domain (machine name if you're using a Windows machine) and the Windows login username (and the Windows password, if there is one).
5. Create a machine template under Machines at Tenant level and assign at least an unattended execution slot.
6. Create a folder. If possible, publish a process to it.
7. Assign both the user and the machine template to the folder.

2

On the **machine** where the unattended robot will be deployed:

1. Make sure the UiPath Agent is installed. Use the Enterprise installer if you have access to it. Otherwise, the community installer should work, but you will also have UiPath Assistant (this is not an issue).
2. If it's not open, open UiPath Agent from the folder where UiPath was installed.
3. Right-click the UiPath Agent and open Orchestrator settings.
4. Input the machine key corresponding to the machine template and the Orchestrator URL (from the browser where Automation Cloud is open, down to the tenant name).
5. Click Connect.

Steps 2-4 above can be achieved through UiPath Assistant if it is installed.

Hurmet Noka

14. Email Automation in Studio

Email Automation Overview

Introduction

Today, there are lots of ways to communicate. But when it comes to critical processes, email is still at the top of the list for most of the companies.
Let’s take for example some servers that generate different alert messages throughout the day. To speed up things, we may want to automate the process of distributing the alerts, so the right people get the right alert messages in time. One way to do this is by filtering emails based on different criteria like subject, date, body content, or sender.
Studio provides email automation capabilities through the UiPath Mail Activities Pack. Let's find out more about it and put it to work.

The UiPath Mail Activities Pack

This activity package is designed to facilitate the automation of any mail-related tasks, covering various protocols, such as IMAP, POP3 or SMTP. UiPath also features activities that are specialized for working with Outlook and Exchange.
To better understand the usability of the mail activities in the package, let's take a high-level look at how we can automate the distribution of email alerts generated by multiple servers.

The "catch and dispatch" process

Consider three servers generating different types of alerts, which are being sent to a single mailbox account.
From there, the Level 1 technicians have to acknowledge and filter the emails.
Afterwards, they follow specific procedures to either escalate or handle the alerts themselves.
Since this is a repetitive task that has a stable process and clear guidelines, it is the perfect candidate for automation.
Let's take a quick look at the main steps of the process.

Objectives:

1. **Install the dedicated email activities in UiPath Studio.**
2. **Retrieve email messages based on the email client and server in use.**

Filtering Emails

In the 2020.10 version of the Mail Activities Package, there are two main ways to filter emails:

- By using control flow activities.
- By using 'Get Outlook Mail Messages' activity.

The filtering options provided with 'Get Outlook Mail Messages' are:

- **Filter -** A string used as a filter for the messages to be retrieved. Accepts JET queries or DASL queries. Let's assume that you use Outlook and need to retrieve only the email messages that have "Critical" in the beginning of the subject line. In this case, you can also use SQL filtering, and your expression would look like this: **"@SQL=""urn:schemas:httpmail:subject"" like 'Critical%'"**
- **FilterByMessageIds -** Returns only mail messages that match the specified message IDs. If set, the Filter option is ignored.
- **OnlyUnreadMessages -** Specifies whether to retrieve only unread messages. By default, this check box is selected.
- **Top** - The number of messages to be retrieved starting from either the newest or the oldest, depending on the OrderByDate parameter.

Aside from the filtering options provided by the 'Get Outlook Mail Messages' activity, you can also use a For Each activity to indicate the actions to be taken based on the specified conditions.

UiPath Studio provides a variety of filtering options. Depending on the email server that you are using, choose the activity and option that best suits your needs.

3. **Automate the interaction with emails by filtering and downloading attachments.**
4. **Use the different 'Send Email' activities available.**

Sending Emails

UiPath Studio has multiple activities integrated that help you send emails based on the server in use.
'System.Net.Mail.MailMessage' represents the main data type when working with emails in UiPath. You will be using this data type whenever you need to get or forward email messages.
When sending an email using one of the email activities in Studio, you can add a subject, custom body, attachments, or even use a template to do all these. You can also use an **Orchestrator Asset** or the **'Read Text File'** activity to access a template.

5. **Use message templates to send emails.**

Practice 2 solution

1. **Open Studio** and start a new process.
2. Add a new '**Sequence**' type of activity to start your workflow and drag a 'Get Outlook Mail Messages' type of activity inside it.
3. Now, in the Properties pane, add "**Inbox**" as the input for MailFolder. Check the '**MarkAsRead**' and '**OnlyUnreadMessages**' boxes. In the '**Filter**' expression field, add **"subject]='IDs Request'"** to search the Inbox only for emails that have "IDs Request" as the subject line.
4. Add '**For Each**' type of activity and drag a new sequence inside it.
5. Add an 'Assign' type of activity and rename it to 'Assign to store the sender email address'. Within it, **press CTRL+K to create a new variable** named '**SendersEmail**' and assign the following expression to it:

   SendersEmail = **email.From.ToString**

6. Next, drag a '**Save Attachments**' activity. Type **"SavedAttachment\"** inside the 'FolderPath' field and '**email**' inside the 'Mail Message' field. In the Output Attachments field, **press CTRL+K to create a new variable** named '**Attachments**' (stores all the saved attachments).
7. Afterwards, create and populate the ID column in the Excel file. For this you must add an '**Excel Application Scope**' activity (rename it accordingly). Type '**Attachments(0).ToString**' as the location of the files.
8. Next, drag a new '**Sequence**' activity and name it 'Process Excel'. Within the new sequence, drag an Excel '**Write Cell**' activity and name it **"Write ID Header Cell"**. Add '**"Sheet1"**' as the 'Sheet Name', **"E1"** as 'Range', and **"ID"** in the expression value field.
9. Now, add a '**Get Cell Color**' activity and rename it as "**Get Email Header Cell Color**". Type '**"Sheet1"**' in the 'Sheet Name' and **"D1"** in the 'Cell' value field.
   To set the ID Header range color, drag a '**Set Range Color**' activity. In the 'Color' value field, **press CTRL+K to create a new variable** and name it '**HeaderColor**'. Type **"E1"** in the 'Range' value field, and '**"Sheet1"**' in the 'SheetName' value field.
10. Next, add an Excel '**Read Column**' activity and rename it as "Read Column To Get the number of Data Rows". Type '**"Sheet1"**' as the 'Sheet Name' and '**"A1"**' in the 'Cell' value field.
11. Add another Excel '**Write Cell**' activity to write the formula throughout the new ID column. Type '**"Sheet1"**'as 'Sheet Name', add **"E2:E"+ DataColumn.Tostring.Length.ToString** in the 'Range' value field, and **"=CONCATENATE(LEFT(A2,3),LEFT(B2,3),RIGHT(C2,3))"** in the 'Input' value expression field.
12. To send the email with the updated Excel file, add a new '**Sequence**' activity **outside the 'Excel Application Scope'** one and rename it as "Send Email with updated file".
13. Afterwards, drag a '**Read Text File**' activity, as you will use it to read the email template.
14. Next, download the email template that you will use when sending emails.

Hurmet Noka

16. Version Control Systems Integration in Studio

Objectives:

1. Explain what version control systems are.

What are version control systems?

Version control systems are tools used by software development teams to manage the collaboration on large projects. A version control system allows developers to track a code change, review the history of the code, and revert to a previous version of the project, if needed.

Here are some benefits of using version control systems:

- **Enhanced collaboration -** Team members can work freely on any file at any time and merge the changes into a common version at the right time.
- **Storing versions -** Only the current version is stored on the disk, all the others are in the system.
- **Restoring previous versions -** Restore older versions of the file at any time.
- **Tracking different project versions -** New versions are usually saved with change descriptions. Versions of the same file can also be compared.

2. Identify the version control systems that are integrated in UiPath Studio.

The version control systems UiPath Studio is integrated with are Git, TFS, and SVN. The connection to a version control system is done at project level. To manage your connections, access Studio, go to the Backstage view, and click the Team tab.

Alternatively, the Add to Source Control button in the status bar offers shortcuts to Git Init, Copy to Git, Add to TFS, and Add to SVN.

Which version control system should you choose?

Regardless of the type of version control system used, project files are stored on a server where you push your files after you have completed your work on your local machine.

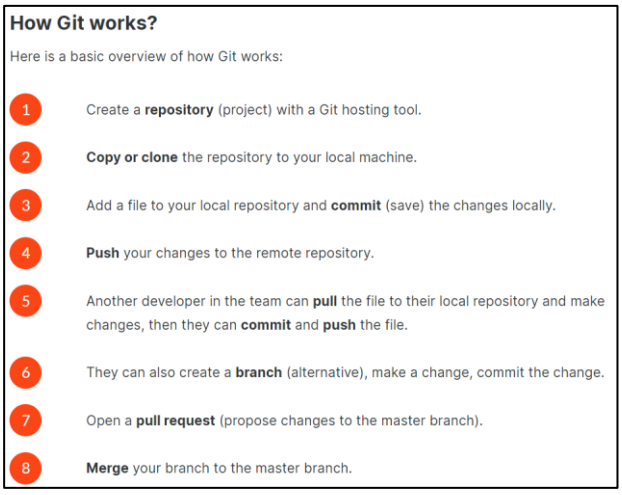
However, deciding whether to use a centralized version control system like SVN or a distributed version control system like Git affects how you commit changes. This will be covered later in this course

3. Perform the most important actions within a version control system integration (we will use Git as an example).

A Closer Look at Git

Git is an open-source version control system. Git is distributed, unlike older centralized version control systems such as SVN and CVS, which allows every developer to have a complete history of their code repository locally. This makes the initial clone of the repository slower, but subsequent operations such as commit, blame, diff, merge, and log dramatically faster.

Git also has good support for branching, merging, and rewriting repository history. The pull request is one popular feature that allows teams to collaborate on Git branches and efficiently review each other's code. Git is the most widely used version control system in the world and is considered the modern standard in software development.



Hurmet Noka

SVN

With a centralized system, the SVN version control system stores all files and historical data on a central server. The developers commit their changes to this central server repository.

Trunk: The trunk is the hub of your current, stable code and product. It only includes tested, unbroken code.
Branches: Here you add the new codes and features. Using a copy of the trunk code, team members conduct research and development in the branch. This allows each team member to work on the enhanced features without disrupting each other's progress.
Tags: Tags are a duplicate of a branch at a given point in time. Tags are not used during development but are used during deployment after the branch's code is completed. Marking code with tags makes it easy to review and, if necessary, revert your code.

GIT

Git uses a central repository and a series of local repositories. Local repositories are exact copies of the central repository containing their complete history of updates. The Git workflow is similar to SVN, but with an extra step. To create a new feature, we need to take an exact copy of the central repository to create our local repository on your local machine (consider this as our "local trunk").

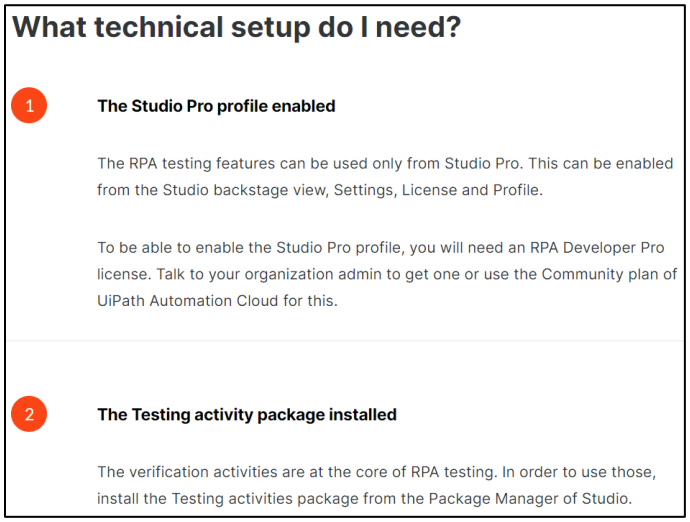
Then we work on our local repository exactly as we would in SVN by creating new branches, tags, etc. When we're done, we merge our branches into our local repository (i.e. local trunk). When we're ready to merge into the central repository, we push our changes from our local repository to the central repository.

Some of the benefits include the following:

- It's faster to commit.
- No more single point of failure.
- It's available offline.

Hurmet Noka

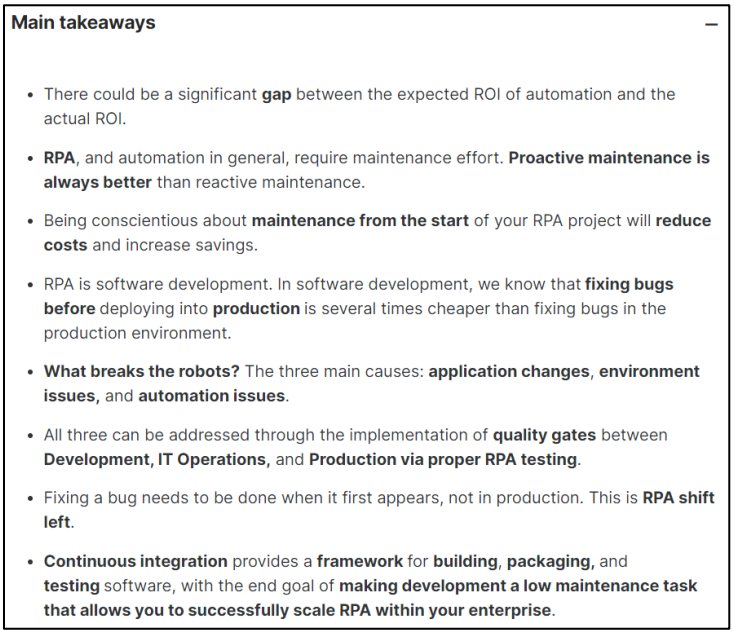
17. RPA Testing in Studio Pro



Objectives:

1. **Explain the causes that affect the robot maintenance and how they can be tackled.**
   RPA is software. It goes through all the software development stages, including testing. RPA testing deals with issues that are typically discovered in production, but aims to do it as early as possible in the process.

   Watch the video below for an introduction to RPA testing, to the types of issues it addresses, as well as how can these be tackled early.



2. **Build the case for RPA testing.**
3. **Describe the levels of RPA testing.**

Levels of RPA testing

Since RPA development goes through the regular software development stages, RPA testing is not at all different from software testing. There are four main levels: unit (component) testing, integration testing, system testing, and acceptance testing.

Unit (component) testing

It's called unit testing because the functionality of an automation project should be broken down into units or components (in the form of workflows) which can be tested individually.

Unit testing has the greatest effect on the quality of the project when it's integrated in the development process, and done by the developers themselves.

In unit testing, non-essential or complex objects should be replaced with mock objects.

Integration testing

In this type of testing, the focus is on the interactions between the components or systems. This is typically a subsequent phase after assuring the quality at component level.

This can be done by professional testers or by developers. It can also be done in a continuous integration/continuous deployment framework, in which a test set is executed every time a new functionality is added to an existing automation.

System testing

The focus here is the entire system. In a large end-to-end automation, this would come after the different components have been tested in different configurations and scenarios.

System testing requires a dedicated pre-production environment, where testers can get the same experience that the user would have after the automation goes live. The PDD should contain many useful information to create test cases for this level.

Acceptance testing

This also deals with the entire system, but is performed by business users. The goal here is to check if the target audience interacts with the software differently than the developers have intended.

4. **Create basic and data-driven test cases for RPA workflows.**

UiPath Test Suite offers a complete solution for RPA testing during automation development, as well as during integration and before moving it into production. Depending on the deployment model, RPA testing can be integrated in a continuous integration environment.

But the first goal of RPA testing—at least chronologically—is to address application issues and automation issues as early as possible, in the RPA development phase. Watch the video below to see the built-in features of UiPath Studio Pro for RPA unit testing.

The structure of a test case

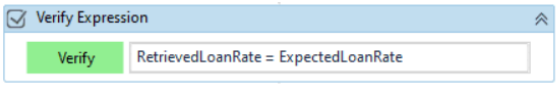
The easiest way to create an RPA test case in Studio Pro is to right-click a workflow in the Project panel and select "Create test case". The test case thus created will have the behavior-driven development structure.

The verification activities

These activities are meant to be used in the Then block, to verify the outcomes of a test case execution.

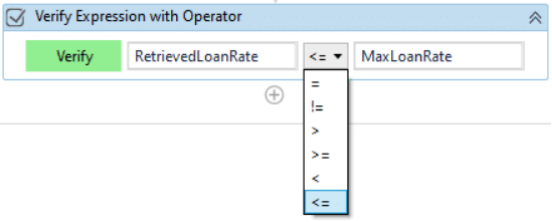
Verify Expression

This activity verifies a single expression (e.g. whether two variables are equal). Its outcome can be True or False.



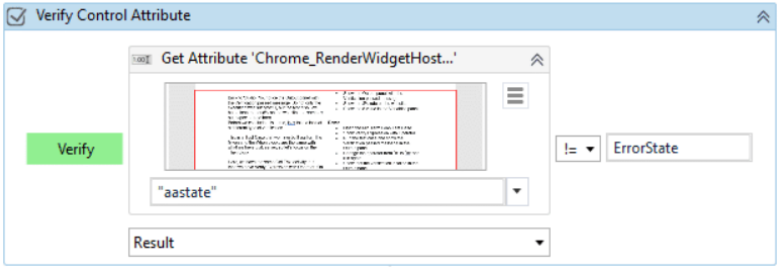
Verify Expression with Operator

This activity compares the outcomes of two expressions, variables or arguments using 6 predefined operators. Its outcome can also be True or False.



Verify Control Attribute

This is the most versatile activity in the first version of the Testing Activities Package. It allows the comparison of a property returned by another Activity with an expression, variable or argument. Its outcome can also be True or False.



Basic vs. data-driven test cases

Basic test cases run with a single set of data, and typically verify the outcomes against static values. To test an RPA workflow in different scenarios, data-driven testing is used. This typically uses an Excel file containing data variations, and the test case runs once for each chosen data set.

A basic test case can be easily converted to a data-driven one by adding the file with the data variations. When doing so, arguments are automatically created with the names of the columns in the data variation file.

5. **Use the activity coverage tools of Studio Pro to assess whether RPA workflows are well covered by the test cases.**
6. **Use mock testing to simulate real objects in RPA testing scenarios.**
   Automation projects often deal with business applications not having separate non-production environments. In such cases, testing the RPA workflows during development can become risky, through the effects in production, or costly, through the special resources needed.

   Mock testing, or mocking, offers a way of replacing dependencies by creating objects that simulate the behavior of real objects. These objects are typically out of scope for the testing effort, at least in the unit testing stages.

7. **Follow the good case practices identified by RPA developers from real automation projects.**

> **Modularity in your automation project**
>
> Good testing starts well before the test cases. An automation project should be broken down in atomic workflows, each of them having a single purpose and the smallest possible number of actions.
>
> This way, it will be easy to understand and unit-test it.

> **Modularity in your test cases**
>
> A test case should have one purpose and ideally contain one verification. On the other hand, every feature should have a unit test. If there are exceptions, create separate test cases for each of these.

> **Clean structure**
>
> Test cases created from workflows have the Given-When-Then structure. You can use other structures or frameworks, but make sure you keep it clean.
>
> If the Given block becomes too cumbersome, it might be a sign that the workflow you are trying to unit-test should be more atomic.

> **Autonomy**
>
> Test cases should be autonomous, meaning that one test case should not depend on another test case's run.

> **Mocking**
>
> Use mocking whenever there are complex steps irrelevant to the purpose of the test case.

Hurmet Noka

Keep test cases up to date

Update the test cases with each change request.

Reusability

If there are test cases relevant for other automation projects, use the Import Test Case feature in Studio Pro. If you need to rename the test cases, do it only in Studio Pro, not outside.

After unit testing

If your development model is configured for Continuous Integration / Continuous Deployment, include the test cases in the CI/CD pipeline.

Prepare an RPA test set that can be run by the IT team in the pre-production environment, whenever there's an environment change. On the other hand, run the test set whenever you commit a change to your automation project.

Hurmet Noka

PRACTICE TEST

1. A developer needs to ensure a string has a valid email format. How can this be verified?

- Using the Format Value activity
- Using the Contains function with '@.com'
- Using a Send SMTP Mail Message activity
- Using Regular Expressions

2. Which data type does the Selector property of an activity accept?

- System.String
- UiPath.Core.Selector
- System.Security.SecureString
- System.Xml.XmlElement

3. A developer needs to create a variable to store the value of an email address. Which variable property or field is optional?

- Default
- Variable type
- Scope
- Name

4. During development, a breakpoint is enabled at a particular activity. Later, the developer published the package to UiPath Orchestrator 2020.10. When the process runs from a UiPath Robot in an unattended mode, what is the expected behavior?

- Process generates an exception error due to the breakpoint
- Breakpoint will have no impact on the process execution
- Breakpoint will cause the process to pause indefinitely
- Process causes the robot to crash due to the breakpoint

5. What describes the file access level needed when using the Workbook Read Range activity?

- Works with .xls and .xlsx files, and Excel must be installed
- Works with .xlsx files and Excel does not need to be installed
- Works only with .xlsm files and Excel must be installed
- Works only with .xls files and Excel does not need to be installed

6. What are the differences between partial selectors and full selectors?

- Partial selectors are recommended to perform multiple actions in the same window.
  Full selectors include information about the top-level window.
- Partial selectors do not include information about the top-level window.
  Full selectors are recommended to perform multiple actions in the same window.
- Partial selectors include information about the top-level window.
  Full selectors are recommended to perform multiple actions in the same window.
- Partial selectors are recommended when switching between multiple windows.
  Full selectors do not include information about the top-level window.

7. Which activity can be used to retrieve the value from a specific column, from a DataRow object?

- Get Row Item
- Get Transaction Item
- Filter Data Table
- Remove Data Row

8. Based on best practice, how many Global Exception Handlers can be set per automation project?
- One for the entire project
- One for each workflow in the project
- One for each activity in the project
- An unlimited number in the project

9. Which function is available with an attended robot?
- Process execution can be manually triggered from an Orchestrator shared folder
- Process execution can be scheduled from Orchestrator
- Process execution can continue under a locked screen
- Processes can be started from UiPath Assistant or the command prompt

10. Which elements must exist for each Dictionary object?
- Item and Collection
- Key and Collection
- Key and Value
- Item and Values

11. When reading a CSV file with the Read CSV activity, which property needs to be enabled in order to access data in the output datatable by the column names in UiPath Studio 2020.10?
- PreserveFormat
- Has headers
- Delimiter
- IgnoreQuotes

12. What is the correct expression for displaying the current date in the Output panel using a Write Line activity in a "dd-MM-yyyy" format?
- Format(Now,"dd-MM-yyyy")
- Now.CompareTo("dd-MM-yyyy")
- Now.ToString("dd-MM-yyyy")
- Now."dd-MM-yyyy"

13. Which UiPath Studio panel can be used to change the scope of a variable?
- Locals
- Outline
- Project
- Variables

14. What is a unique feature of the UI Explorer tool?
- Allows the developer to view the full UI hierarchy tree
- Allows the developer to repair a selector
- Allows the developer to use wildcards in a selector
- Allows the developer to indicate a selector

15. A developer is reviewing the file contents of a UiPath Studio project. In which location are the UiPath dependencies specified?
- .entities
- .settings
- project.json
- .screenshots

16. Which function does the Anchor Base activity provide?

- Searches for a UI element by attaching to a reliable selector field and sends the tab hotkey until it is in the desired field
- Searches for a UI element by using another UI element as an anchor and searches the specified area around the anchored element
- Confirms the selector for the element is "True" based on a nearby anchored element
- Confirms the page is fully loaded and that all elements are in the same place as where the automation was developed

17. What describes the relationship between UiPath Robots and UiPath Orchestrator?

- UiPath Orchestrator consumes the Queue Items uploaded to the UiPath Robot
- Attended robots can be connected to UiPath Orchestrator without Windows Credentials
- Unattended robots can be connected to UiPath Orchestrator without the Windows username
- UiPath Robots can be deleted if they have pending or active jobs in UiPath Orchestrator

18. Which data type is used to store data extracted from a Microsoft Excel file using a Read Range activity?

- DataTable
- Array
- String
- Object

19. A developer needs to design a process in the following way:
The process needs to validate the Payment Status field in an application.
There are three possible values in the Payment Status field named: Pending, Paid, or Awaiting Approval.
For "Pending", the process should check the "Due Date" field. If the Due Date has lapsed, raise an alert. Otherwise, do nothing.
For "Paid", change the Overall Status to Completed.
For "Awaiting Approval", send an alert email to the email address in the Account Owner field.
Based on best practices, which activity is recommended for designing the process flow for Step 2?

- Set Transaction Status
- Switch
- If
- Report Status

20. A developer wants to remove the company name from the "UiPath Certification" string. Which string method will return "Certification"?

- "UiPath Certification".Substring(6)
- "UiPath Certification".Substring(6).Trim
- "UiPath Certification".Substring(6,14)
- "UiPath Certification".Substring(14,6)

21. Which component of the UiPath platform is used to develop workflows?

- UiPath Orchestrator
- UiPath Studio
- UI Explorer
- UiPath Agent

22. A developer wants to build a project which iterates through multiple digital PDF invoice files to retrieve the Total Value. Assuming all the files have the same structure, which action should be performed based on best practices?

- Use the Find Image and Get Text activities in an Anchor Base activity
- Use OCR activities to output the total value from all invoices
- Ensure the selectors are valid for all invoices
- Ensure the total values are copied to the clipboard by using Hotkeys

23. A user needs to download a report for each month of the year and decides to loop over the months using a For Each activity. Which data type is best-suited to hold the values of the names of the months?

- Dictionary<Object,String>
- List<String>
- String
- String[]

24. A user wants to write a data table to an .xlsx file; however, the file does not exist. What will happen if the Excel Write Range activity is used to write the data table?

- Process will continue the execution without creating the file
- Process will throw an execution error
- File will be created and the data will be located in the file
- File will be created but the file will not contain any data

25. How can a full list of attributes of UI elements be displayed?

- Using the UI Explorer tool
- Using the "Indicate on Screen" button
- Using the Output panel
- Using the Outline panel

26. What is a characteristic of an Orchestrator Asset?

- Assets can hold entire DataTable variables
- All values from any Asset type are encrypted
- Asset types can be changed after creation of the asset
- Assets values can be defined for each user

27. Which characters can be used as wildcards while editing a selector?

- ? and *
- _ and *
- % and !
- % and _

28. In UiPath Studio, which argument direction(s) exist?

- In arguments only
- Out arguments only
- In/Out arguments only
- In, Out, and In/Out arguments

29. A customer calls into a Call Center to get an update on their current outstanding payment. Consider the following use case:

    The customer provides the Customer ID to the call center representative.
    The call center representative inputs the Customer ID into a web application.
    The robot scrapes the data from the web application.
    The robot performs calculations.
    The outstanding amount is provided to the customer over the telephone by the representative.

    What is the number of steps that require manual intervention?

- One
- Two
- Three
- Five

30. What describes a function of UiPath version control?

- Stores only one version of a project on the version control system
- Prohibits the comparison between multiple file versions
- Enables an effective way to connect to GIT, TFS, and SVN at the same time
- Provides an effective way to manage larger projects that require collaboration between multiple users

31. A developer wants to retrieve all rows in a Data Table variable with a "Price" value greater than 1000. Which activity can the developer use?

- Output Data Table
- Sort Data Table
- Filter Data Table
- Lookup Data Table

32. When using an Excel Application Scope activity, which activity should be used to sort a table directly in an .xlsx file?

- Sort Data Table
- Get Table Range
- Select Range
- Sort Table

33. What are the differences between variables and arguments?

- Variables must be assigned to an argument value
  Arguments always have a direction
- Variables always have a direction
  Arguments can be assigned to a variable value
- Variables pass data between activities inside the same workflow
  Arguments pass data between workflow files
- Variables pass data between workflow files
  Arguments pass data between activities inside the same workflow

34. A developer wants to retrieve all UI elements from a form based on a certain filter condition. Which activity should the developer use?

- Find Element
- Set Clipping Region
- Mouse Trigger
- Find Children

35. Which type(s) of automation does UiPath support?

- Hybrid and Unattended automation only
- Unattended automation only
- Attended and Hybrid automation only
- Attended, Hybrid, and Unattended automation

36. A developer has added an Invoke Workflow File activity in Main.xaml to invoke the ProcessInvoices.xaml workflow. The developer needs to pass a DataTable called dt_Invoices from Main.xaml to ProcessInvoices.xaml.

    Which action needs to be performed?

- Declare an "In" argument in ProcessInvoices.xaml
- Declare an "Out" argument in Main.xaml
- Declare an "Out" argument in ProcessInvoices.xaml
- Declare an "In" argument in Main.xaml

37. What is the correct sequence of Queue Item statuses if all statuses are involved in the transaction execution of a Queue Item?

    Instructions: From the drop-down list, select the correct sequence number in which each status occurs.

- Failed
- In Progress
- Successful
- Retried

38. A developer has a workflow in which the value of a counter needs to be evaluated before the body of the loop is executed. Which Control Flow activity should be used?

- Do While
- Switch
- If
- While

39. What is a feature of the Native screen scraping method?

- Supports Citrix
- Extracts the text position
- Runs in the background
- Extracts hidden text

40. When using the For Each activity to loop through the output of a Get Outlook Mail Messages activity, how should the TypeArgument property be set?

- System.Web.UI.WebControls.MailMessageEventArgs
- System.Net.Mail.MailMessage
- UiPath.Mail.MailMessageExtensions
- System.Net.Mail.Attachment

41. How can a sequence named InitAllSettings.xaml be triggered from within Main.xaml?

- Using the Open Application activity
- Using the Invoke Workflow File activity
- Using the Invoke Method activity
- Using the Invoke Code activity

42. In UiPath Studio, which activities can be used as an anchor?

- Wait Element Vanish or Wait Image Vanish
- Get Text or Get Visible Text
- Find Image or Find Element
- Element Exists or Image Exists

43. What is the main feature of a flowchart?

- Determines the project's behavior when encountering an execution error
- Provides a condition to facilitate the transition from one state to another
- Enables the creation of complex business processes and connects activities in multiple ways by branching multiple logical operators
- Provides the ability to seamlessly go from one activity to another and act as a single Block activity for linear processes