UNIT -IV
RPA BOT Models -Exception Handling
RPA BOT Models: Attended Vs Unattended Bots- Monitor Events Triggers for Attended Automation. Exception Handling: Debugging and Exception Handling - Debugging Tools & best practices.
Deploying and Maintaining the BOT: Publishing the Automation solution using publish utility - creating a provision robot from the server - connecting a robot to server - deploy the robot to server.

RPA BOT Models

In **Robotic Process Automation (RPA)**, the term "bot models" typically refers to the different types or approaches used in developing and deploying RPA bots. These models describe how bots interact with systems, handle tasks, and integrate with other components of business processes. Below are some common **RPA Bot Models**:

1. Attended Bots

- **Definition**: Attended bots are robots that require human initiation or intervention to perform tasks. They typically work on the user's machine and assist them by automating repetitive tasks. These bots are designed to be activated by the user and are ideal for processes that require human interaction.
- **Use Cases**:
  - Data entry based on user inputs.
  - Automating actions while the user is working (e.g., populating fields in a form, or retrieving data).
- **Example**: A bot that helps a customer service agent by looking up customer records in an application after the agent requests it.

Key Features:

- Triggered by users (button press, shortcut, etc.).
- Operates on the user's machine or environment.
- Immediate support in the background or on-demand.

2. Unattended Bots

- **Definition**: Unattended bots work autonomously without human intervention. They run in the background and handle entire processes end-to-end without needing a user to start them. These bots are typically deployed on virtual machines or servers and run continuously to automate business processes without human oversight.
- **Use Cases**:
  - Batch processing tasks (e.g., end-of-day processing, data extraction from various systems).

    - Automatic report generation and distribution.
    - Integration with back-office systems (e.g., ERP systems like SAP).
- **Example**: An RPA bot that runs at night to reconcile data between two different systems and generates a report.

Key Features:

- Runs without human initiation.
- Deployed on virtual machines or servers.
- Scheduled or triggered by specific events.
- Can work on multiple processes without human intervention.

3. Hybrid Bots

- **Definition**: Hybrid bots combine the features of both attended and unattended bots. These bots are typically designed to function autonomously but may require some human intervention in certain scenarios, especially for exception handling or approval-based processes.
- **Use Cases**:
    - A bot that autonomously handles the initial part of a process, but asks for human approval or input when it encounters an exception or an unclear decision.
- **Example**: An invoice processing bot that handles invoice extraction automatically but requires manual approval if the invoice amount exceeds a certain threshold.

Key Features:

- Can work autonomously or with human interaction.
- Ideal for processes with decision points or exceptions.
- Flexibility to switch between attended and unattended modes.

4. Citrix Bots (Virtual Desktop Bots)

- **Definition**: Citrix bots are specialized RPA bots designed to interact with applications running in a virtual environment, such as those hosted on **Citrix** or other Remote Desktop services. These bots can operate in environments where traditional direct application automation is not possible, like when using virtual desktops or virtualized applications.
- **Use Cases**:
    - Automating legacy applications that run in virtual environments.
    - Interacting with systems that are hosted in virtualized environments (e.g., Citrix, VMware).
- **Example**: A bot that automates data entry in an old application running on a Citrix server.

Key Features:

- Capable of interacting with virtual environments.
- Simulates user actions through image recognition and screen scraping.
- Can work with legacy systems and applications that lack direct API support.

5. Process-Oriented Bots

- **Definition**: Process-oriented bots are designed to automate specific, repeatable business processes. These bots are built with a clear sequence of activities to accomplish the goal, typically for processes with defined workflows.
- **Use Cases**:
  - Accounts payable or receivable automation.
  - Order processing.
  - Data extraction and manipulation from reports or forms.
- **Example**: A bot that automates the process of approving purchase orders after verifying the details against a central system.

Key Features:

- Focused on automating a specific business process.
- Tasks are sequential and typically well-defined.
- Involves multiple systems or applications for end-to-end automation.

6. Task-Oriented Bots

- **Definition**: Task-oriented bots are smaller, more focused bots that automate individual tasks or steps within a larger process. These bots are typically not designed to handle the entire workflow but instead focus on specific, repeatable actions.
- **Use Cases**:
  - Data entry in a single system.
  - Web scraping tasks for extracting specific information.
  - Checking inventory levels in a database.
- **Example**: A bot that logs into an email system and downloads all attachments, then processes them into a specified format.

Key Features:

- Focused on a single, repetitive task.
- Often used as components within larger workflows.
- May be called by other bots or workflows.

7. AI-Powered Bots (Cognitive Bots)

- **Definition**: AI-powered bots use artificial intelligence (AI) and machine learning (ML) algorithms to handle tasks that require cognitive abilities, such as recognizing unstructured data, understanding natural language, or making decisions based on complex inputs.

- **Use Cases**:
    - Invoice processing using OCR (Optical Character Recognition) for extracting unstructured data.
    - Customer service bots that understand and respond to natural language inquiries (e.g., chatbots).
- **Example**: A bot that reads invoices in various formats (PDF, scanned images) and extracts key data like amounts, dates, and vendor details using AI-powered OCR.

Key Features:

- Incorporates AI/ML for processing unstructured data.
- Capable of learning from data and improving its performance over time.
- Can handle tasks like text analysis, sentiment analysis, and predictive analytics.

8. Self-Healing Bots

- **Definition**: Self-healing bots are designed to autonomously detect and correct issues during execution. These bots can detect when something goes wrong (e.g., a system update changes a UI element or a workflow fails) and automatically adjust their operations to continue functioning properly.
- **Use Cases**:
    - Handling system changes or UI modifications that break automation scripts.
    - Automatically adjusting to new workflows when minor process changes occur.
- **Example**: A bot that continues to run even if the layout of an application changes, automatically adjusting the selectors or actions it uses to interact with the application.

Key Features:

- Automatic detection and correction of issues.
- Helps reduce the need for manual intervention in bot management.
- Makes bots more resilient to system changes.

Summary of Bot Models:

- **Attended Bots**: Require human interaction to start or assist.
- **Unattended Bots**: Work autonomously without human intervention.
- **Hybrid Bots**: Can function both autonomously and with human involvement when needed.
- **Citrix Bots**: Automate virtualized desktop applications.
- **Process-Oriented Bots**: Focus on automating a complete business process.
- **Task-Oriented Bots**: Automate specific tasks within a process.
- **AI-Powered Bots**: Use cognitive capabilities like AI/ML for handling complex tasks.
- **Self-Healing Bots**: Automatically adjust to changes in systems to maintain functionality.

1. Attended Vs Unattended Bots

Attended Bots vs Unattended Bots in Robotic Process Automation (RPA)

In **UiPath** and other RPA platforms, bots are typically classified into two categories based on their mode of operation: **Attended Bots** and **Unattended Bots**. These categories describe how the bots interact with users, initiate processes, and operate within the business workflow.

Here's a detailed comparison between **Attended** and **Unattended Bots**:

Attended Bots:

Definition:

Attended bots are designed to assist users with specific tasks by interacting with them in real time. These bots require human intervention to initiate and typically work alongside users during business processes.

Unattended Bots:

Definition:

Unattended bots are designed to operate autonomously without any human intervention. These bots can be triggered by events, schedules, or systems, and they usually run on virtual machines or servers in the background, managing entire processes end-to-end.

Comparison: Attended vs Unattended Bots

Aspect	Attended Bots	Unattended Bots
Trigger	Human-initiated, triggered by the user	Event-driven, scheduled, or triggered by other systems
User Interaction	Requires user involvement during execution	No user interaction required during execution
Execution Location	Runs locally on user's machine (desktop)	Runs on virtual machines, servers, or cloud infrastructure
Operation Mode	Assists users with tasks in real time	Operates autonomously in the background
Complexity	Typically simpler processes, often task-specific	Can handle entire processes, including complex workflows
Use Case Examples	Customer service assistance, data entry, form filling	Batch processing, ERP automation, back-office tasks
Scalability	Limited by user involvement, not easily scalable	Highly scalable with the ability to run multiple bots simultaneously
Feedback	Provides real-time feedback to users	No immediate feedback unless errors occur, handled through logs or alerts

Aspect	Attended Bots	Unattended Bots
Cost of Infrastructure	Low, since it operates on individual desktops	Higher, requires servers or cloud infrastructure

---

When to Use Attended Bots:

- **Customer Support**: When a bot needs to assist an agent in real-time while they handle customer inquiries.
- **Manual Process Automation**: When automation requires user input or decision-making during the process.
- **On-Demand Automation**: When users need to trigger automation for specific tasks, such as extracting and entering data.

When to Use Unattended Bots:

- **End-to-End Process Automation**: For processes that don't require human intervention, such as back-office workflows or batch data processing.
- **Night-time or Off-Hours Automation**: When automation needs to run outside of regular working hours without user involvement.
- **High-Volume, Scalable Tasks**: For automating tasks that need to be performed at a large scale, such as handling thousands of invoices or transactions.

2. Monitor Events Triggers for Attended Automation

In **UiPath**, **Monitor Events** and **Triggers** are key components for creating **Attended Automation**. These allow your bot to respond to specific events or conditions and automate tasks based on them. For **Attended Bots**, events typically involve the user's actions, such as clicking a button or changing the state of an application. By monitoring these events, the bot can trigger specific actions in real time.

Here's an overview of how to work with **Monitor Events** and **Triggers** for **Attended Automation** in UiPath:

1. Monitor Events in UiPath

Monitor Events in UiPath allow you to track specific user or system interactions that trigger automated actions. These events can be mouse clicks, keyboard presses, window state changes, or application events. Attended automation often requires real-time monitoring of these events to provide support to users.

Some common examples of events you may want to monitor include:

- **Clicking a button** in an application.
- **Text input** by the user.
- **Changes in window state** (e.g., when an application window becomes active or inactive).
- **UI element visibility** or status change.

How to Monitor Events:

UiPath provides several activities to monitor events and respond to them:

1. **Monitor Events Activity**:
    - The **Monitor Events** activity allows you to monitor events like **UI Automation**, **Hotkeys**, and **Triggers**.
    - It listens for certain conditions or user actions and executes actions when those conditions are met.
    - **Workflow Example**:
        - The **Monitor Events** activity will continuously check for specific events and triggers. If an event occurs, the bot will execute the relevant action.

    Steps:

    - Add the **Monitor Events** activity to your workflow.
    - Inside the **Monitor Events** container, you can place other activities like **Click**, **Type Into**, or custom logic based on what event is triggered.
    - Define the events you want to monitor, such as mouse clicks or keyboard inputs.

    Example:

```
Monitor Events
  On Button Clicked:
    // Handle the action when the button is clicked
  On Hotkey Pressed:
    // Handle the action when a hotkey is pressed
```

2. **Hotkey Triggers**:
    - You can monitor hotkey triggers using **Hotkey** events, where the bot listens for specific key combinations, such as **Ctrl + Shift + N**.
    - This allows the bot to respond immediately when the user presses a particular hotkey.

2. Using Triggers in Attended Automation

Triggers in UiPath can be used to initiate or activate a workflow based on certain predefined conditions, such as changes in an application or specific system events. Attended bots often rely on triggers to respond to user input or actions, creating an interactive experience.

Types of Triggers:

1. **Element Trigger**:

- This trigger monitors whether a specific UI element appears or changes in a specific way (e.g., a button becomes visible or enabled).
- Commonly used to wait for an element to be available or clickable before interacting with it.

2. **Hotkey Trigger**:
   - A hotkey trigger listens for specific key combinations that the user presses.
   - This is useful for creating shortcut commands for the bot's execution, allowing the user to trigger automation at will.
3. **State Trigger**:
   - This trigger listens for changes in the state of an application, window, or system (e.g., when a window becomes active or when a file is downloaded).
4. **UI Automation Trigger**:
   - This trigger responds to changes in the UI, such as detecting when a particular button or field becomes enabled, visible, or clickable.
   - Used to trigger actions based on user interaction with the UI of an application.

How to Create Triggers:

1. **Element Exists**:
   - The **Element Exists** activity is often used with a trigger-like behavior. It waits for a UI element to appear or change status, like detecting if a button is enabled or a text field becomes editable.
2. **On Element Appear**:
   - This is an event trigger that monitors whether a UI element appears or becomes visible on the screen.
   - You can use this to trigger specific actions when a user interacts with an application and causes an element to appear.

   Example:

```
On Element Appear (button "Submit" is visible):
    // Trigger the bot to click the "Submit" button
```

3. **On Hotkey Pressed**:
   - Define a hotkey that can trigger the bot's action. For instance, a user can press **Ctrl + Shift + A** to start a process that the bot will handle.

3. Example Workflow Using Monitor Events and Triggers in Attended Bots

Let's consider an example where an **Attended Bot** helps a user by listening for a button click and responding with a predefined action, like filling in a form.

Scenario:

The bot is set to monitor a button click, and once the button is clicked, the bot performs some actions like typing data into a form or interacting with another application.

Workflow:

1. **Monitor Events Activity** is used to continuously listen for user interactions.
2. Once the **On Button Click** event occurs, the bot triggers a sequence to type information into a form.
3. The bot could also listen for hotkey combinations (e.g., **Ctrl + Alt + F**) to perform a specific task when the user needs assistance.

Steps to implement:

- Use the **Monitor Events** activity.
- Inside the **Monitor Events** container, use an **On Element Clicked** event to trigger the bot's actions.
- If the event is a button click, then the bot performs actions such as **Click**, **Type Into**, or other necessary actions.

```
Monitor Events
    On Button Clicked (SubmitButton):
        // Action triggered when button is clicked
        Type Into (Field1): "Hello"
        Type Into (Field2): "World"
```

You could also add **Hotkey** triggers like this:

```
Monitor Events
    On Hotkey Pressed (Ctrl+Shift+N):
        // Action triggered when hotkey is pressed
        Open Application (e.g., Notepad)
```

4. Best Practices for Using Monitor Events and Triggers in Attended Automation:

1. **Non-Blocking Operations**:
    - Use **Monitor Events** to listen to events without blocking other processes. This allows the bot to run in the background and respond when needed, while still performing other tasks when idle.
2. **Handle Multiple Events**:
    - You can monitor multiple events at the same time using multiple event handlers within the **Monitor Events** activity. For example, the bot can listen for both a button click and a hotkey press simultaneously.
3. **Error Handling**:
    - Add appropriate exception handling for scenarios where the event does not occur as expected. For instance, you may want to handle cases where the monitored UI element does not appear after a timeout.
4. **User Feedback**:
    - Provide feedback to the user when the bot is waiting for input or monitoring events, such as showing a loading indicator or prompt, to improve the user experience.
5. **Efficient Monitoring**:

- Avoid using overly broad monitoring triggers that might lead to unnecessary resource consumption. Focus on specific events that are relevant to the workflow.

Exception Handling:

Exception Handling in UiPath

Exception handling is an essential part of creating reliable and resilient RPA processes. In UiPath, **exception handling** allows you to manage errors or unexpected situations that may arise during the execution of your automation workflow. Without proper exception handling, your automation might fail unexpectedly, causing disruptions and requiring manual intervention.

UiPath provides several activities and techniques to handle exceptions effectively in automation workflows.

Types of Exceptions in UiPath

1. **System Exceptions**: These occur when there's a problem with the system or environment where the automation is running (e.g., network issues, application crashes).
2. **Business Exceptions**: These exceptions arise due to logic or business-related issues (e.g., missing data, incorrect values, or conditions not met in the process).
3. **User-Defined Exceptions**: These are exceptions explicitly thrown by the developer to handle specific situations in the workflow.

1. Try-Catch Activity

The **Try-Catch** activity is the primary way to handle exceptions in UiPath. It allows you to define a block of code (the **Try** section) that can throw an exception, and then define how to handle that exception (in the **Catch** section). You can also define actions in the **Finally** section that will always run, whether or not an exception occurs.

Syntax:

- **Try**: The block of code where exceptions might occur.
- **Catch**: The block where you handle specific exceptions.
- **Finally**: The block that is always executed (optional, but useful for cleanup actions).

Example:

```
Try
    // Your code that might throw exceptions
    Open Application
    Read Data from File
Catch (ExceptionType1 e1)
    // Handle the specific exception type (e.g., handle missing file)
    Log Message: "File not found!"
Catch (ExceptionType2 e2)
    // Handle another type of exception
```

```
    Log Message: "Network error occurred!"
Finally
    // Clean-up code (e.g., closing applications, releasing resources)
    Close Application
```

2. Common Exception Types

Here are some common exceptions you may encounter in UiPath workflows:

- **Business Rule Exception**: A custom exception for business logic failures.
- **File Not Found Exception**: Raised when a file that is being accessed does not exist.
- **Null Reference Exception**: Raised when trying to access an object that is null.
- **Index out of RangeException**: Raised when an index is outside the bounds of an array or list.
- **Timeout Exception**: Raised when a time limit for a specific operation (e.g., waiting for a UI element) exceeds.
- **System Exception**: The base class for all exceptions related to system failures (e.g., network errors).

3. Handling Multiple Exceptions

You can use multiple **Catch** blocks in a single **Try-Catch** activity to handle different types of exceptions differently. For example, you can have one block to handle **FileNotFoundException** and another for **Timeout Exception**.

Example:

```
Try
    // Code that could throw multiple exceptions
    Open Excel File
Catch (FileNotFoundException e)
    // Handle file not found exception
    Log Message: "The specified file is missing!"
Catch (TimeoutException e)
    // Handle timeout exception
    Log Message: "The operation timed out!"
Catch (Exception e)
    // General exception handler for other unforeseen errors
    Log Message: "An unexpected error occurred: " + e.Message
Finally
    // Clean-up actions
    Close Excel Application
```

4. Throw Activity (Raising Exceptions)

Sometimes you may want to throw an exception intentionally when a certain condition is met (e.g., invalid input or unexpected data). The **Throw** activity allows you to raise exceptions manually.

Syntax:

```
Throw New BusinessRuleException("Custom message")
```

Example:

```
If some_condition IsNot Met
    Throw New BusinessRuleException("The process cannot continue due to a
missing value.")
```

The **Throw** activity can be used to generate **business exceptions** when a certain condition is not met, allowing you to control the flow of your automation by providing meaningful error messages.

5. Retry Scope Activity

The **Retry Scope** activity allows you to retry a specific action or sequence of activities if an error occurs, up to a certain number of retries. This is particularly useful in situations where a failure might be temporary (e.g., a network delay or temporary unavailability of an element).

Usage:

- You define the **Action** to be retried and the **Condition** that determines whether to continue retrying.
- You can specify the number of retries and the time between each retry.

Example:

```
Retry Scope
    Action
        // Try to click a button
    Condition
        // Check if the button is visible
        Element Exists (button)
    RetryCount = 3
    DelayBetweenRetries = 5000ms
```

In this example, the bot will try to click the button up to 3 times, waiting 5 seconds between each attempt, until the button appears.

6. Global Exception Handling with the Global Handler

UiPath provides a **Global Exception Handler** that can be configured to manage exceptions that might occur across an entire workflow. This is useful for setting up a uniform exception handling mechanism that can catch all exceptions at a higher level and apply consistent error logging or recovery actions.

How to Set Up:

1. In the **Main.xaml** file, go to the **Project** tab.
2. Select **Settings** and enable the **Global Exception Handler**.
3. Create a separate workflow for handling exceptions globally, where you can define:
    - Logging the error message.
    - Sending notifications (e.g., email).

- Retry mechanisms or recovery actions.

7. Best Practices for Exception Handling

1. **Log Exceptions**: Always log exception details, such as the type of error, stack trace, and specific messages, to help with troubleshooting.
   - Use the **Log Message** activity to log detailed error information.
2. **Handle Expected Exceptions**: Anticipate common failure points (e.g., missing files, network issues) and handle them gracefully with custom messages or recovery steps.
3. **Use Finally for Cleanup**: Always use the **Finally** block in your exception handling to ensure that clean-up actions (like closing applications or releasing resources) are executed regardless of whether an exception occurs.
4. **Avoid Catching Everything**: Avoid catching every exception type using a generic **Catch (Exception e)** block unless you have a specific reason. It's better to catch specific exceptions to handle them appropriately.
5. **Notify Stakeholders**: For critical failures, consider sending notifications (emails, messages) to the relevant stakeholders for immediate attention.
6. **Retry Logic**: Implement retry mechanisms where errors may be temporary (e.g., network-related issues) using the **Retry Scope** or **Invoke Method** activities.

Example of a Complete Exception Handling Workflow:

```
Try
    // Try to open an Excel file
    Open Excel Application
    Read Data from Excel
Catch (FileNotFoundException e)
    // If file not found, log and notify user
    Log Message: "File not found!"
    Send Email Notification: "The specified file is missing."
Catch (TimeoutException e)
    // If operation times out, retry or handle
    Log Message: "The operation timed out. Retrying..."
    Retry Logic (Retry Scope)
Catch (Exception e)
    // Handle all other unforeseen exceptions
    Log Message: "An unexpected error occurred: " + e.Message
    Send Email Notification: "An unexpected error occurred in the
automation."
Finally
    // Ensure Excel is closed even if an error occurs
    Close Excel Application
```

3. Debugging and Exception Handling

Debugging and Exception Handling in UiPath

In UiPath, **debugging** and **exception handling** are two critical practices that help ensure your automation works efficiently and can recover gracefully from errors during execution. Debugging helps developers find issues in the workflow during development, while exception

handling ensures that the automation can handle errors in production or runtime without failing abruptly.

Let's break down both practices and their best practices:

1. Debugging in UiPath

Debugging allows you to examine how the automation behaves, track variable values, and pinpoint issues in the process.

Key Debugging Tools and Techniques in UiPath:

1. **Breakpoints**:
   - A **breakpoint** allows you to pause the execution of your workflow at a specific activity or point in the process.
   - **How to set a breakpoint**: Right-click on an activity and select **Toggle Breakpoint** or press **F9**. You'll see a gray dot next to the activity, indicating where execution will pause.

   Example: If you're trying to debug an Excel read operation, you can set a breakpoint before reading the data to check if the file path and other parameters are correct.

2. **Step Into / Step Over**:
   - **Step Into (F11)**: Moves inside the invoked workflow or method and steps through each activity.
   - **Step Over (F10)**: Executes the current activity but skips over invoked workflows or methods.

   Example: Use **Step Into** when you want to see detailed activity execution, such as when invoking a custom workflow or method, and **Step Over** when you don't need to step into invoked methods.

3. **Watch Panel**:
   - The **Watch Panel** allows you to monitor the values of variables during execution.
   - You can add variables to the Watch Panel to track their values in real-time while stepping through the automation.

   Example: During a `For Each` loop, you can monitor the loop variable to see how it changes during each iteration.

4. **Output Panel**:
   - The **Output Panel** shows log messages, errors, and debugging information.
   - You can use **Log Message** activities to write to the Output Panel, providing insights into variable values or custom messages during execution.

   Example: Adding a **Log Message** activity can help you log the value of a specific variable (e.g., `FilePath`) to verify if the correct file is being processed.

5. **Immediate Panel**:
    - The **Immediate Panel** is useful for evaluating expressions and testing code snippets during debugging.
    - You can type expressions or variable names here to quickly check their values.

    Example: If you're uncertain about the value of a variable, you can type `myVariable.ToString()` in the Immediate Panel to check its current value during the debug session.

6. **Highlight Activities**:
    - Highlighting activities lets you visually see which activity is being executed. This is helpful for larger workflows where you want to observe what's happening at any given time.

2. Exception Handling in UiPath

Exception handling ensures that your automation handles errors smoothly and can either recover from them or log them for future analysis. It involves managing exceptions to avoid automation failures and improve robustness.

Key Exception Handling Activities in UiPath:

1. **Try-Catch Activity**:
    - The **Try-Catch** activity is the fundamental way to handle exceptions. It consists of three sections:
        - **Try**: The block of activities that might throw an exception.
        - **Catch**: Handles the specific exception that was thrown.
        - **Finally**: Executes regardless of whether an exception occurred or not (used for cleanup).

    Example:

```
Try
    // Code that might throw an exception
    Open Excel Application
    Read Data from Excel
Catch (FileNotFoundException e)
    // Handle file not found exception
    Log Message: "The specified Excel file was not found."
Catch (TimeoutException e)
    // Handle timeout exception
    Log Message: "The operation timed out."
Catch (Exception e)
    // Handle all other exceptions
    Log Message: "An unexpected error occurred: " + e.Message
Finally
    // Close Excel Application
    Close Application
```

In this example, if the Excel file is not found, the **Catch** block will handle the error by logging it and not causing the automation to fail abruptly. The **Finally** block ensures that Excel is closed even if an error occurs.

2. **Throw Activity**:
    - The **Throw** activity allows you to manually raise an exception. You can use it to throw an exception under specific conditions, such as when certain conditions aren't met in your automation.

Example:

```
If someCondition Is False
    Throw New BusinessRuleException("Invalid data. Cannot proceed.")
```

This is useful when you want to enforce business rules, and if the data doesn't meet the condition, an exception is raised with a custom error message.

3. **Retry Scope Activity**:
    - The **Retry Scope** activity allows you to attempt an action multiple times before it fails. It's particularly useful for temporary errors such as network issues or element not being available immediately.
    - You can set the **RetryCount** and **DelayBetweenRetries** to control how many times the action should be retried and how long to wait between retries.

Example:

```
Retry Scope
    Action: Try to click a button
    Condition: Check if the button exists
    RetryCount: 3
    DelayBetweenRetries: 5000ms
```

In this example, if the button is not available initially, the automation will retry clicking the button up to 3 times, with a 5-second delay between each attempt.

4. **Global Exception Handler**:
    - The **Global Exception Handler** provides a way to handle exceptions across all workflows in a project.
    - You can define the exception handler globally so that all exceptions are handled uniformly across different parts of the project (e.g., logging, sending notifications, retrying workflows).

How to Enable:

- Go to the **Project** tab, select **Settings**, and enable the **Global Exception Handler**.
- This will allow you to create a separate workflow (e.g., **GlobalExceptionHandler.xaml**) where you can handle exceptions uniformly for the whole project.

3. Best Practices for Debugging and Exception Handling

Debugging Best Practices:

1. **Set Breakpoints Thoughtfully**: Set breakpoints at key locations where you suspect issues may occur (e.g., before file read/write, before interacting with an element).
2. **Inspect Variables**: Use the **Watch Panel** to monitor variable values during execution to ensure the correct values are being passed through the automation.
3. **Log Information**: Add **Log Message** activities to track the progress and variables. This is helpful when debugging complex workflows.
4. **Step Through Workflow**: Use **Step Into** and **Step Over** to examine each activity in detail or skip over methods that don't require debugging.

Exception Handling Best Practices:

1. **Catch Specific Exceptions**: Always try to catch specific exceptions (e.g., `FileNotFoundException`, `TimeoutException`) rather than a generic `Catch (Exception e)` block. This way, you can handle each type of error more appropriately.
2. **Log Errors**: Always log exception details, including the exception message, stack trace, and any relevant context (e.g., input data).
3. **Graceful Recovery**: Use the **Finally** block to perform necessary cleanup actions like closing applications or releasing resources.
4. **Use Retry Logic**: For temporary issues (e.g., network timeouts, UI element loading delays), implement **Retry Scope** to retry failed actions.
5. **Alert Stakeholders**: For critical issues, send email notifications or raise alerts to notify the responsible parties of the error (especially for business-critical automation).
6. **Use Custom Exceptions**: Create custom exceptions (e.g., `BusinessRuleException`) to handle business-specific issues and provide more meaningful error messages.

4. Example of Debugging and Exception Handling in Action

Here's a full example workflow that includes both debugging and exception handling:

```
Try
    // Debugging: Set a breakpoint here
    Open Excel Application
    Read Data from Excel (Sheet1)

    // Debugging: Use Watch Panel to monitor variable value
    For Each row In DataTable
        Log Message: "Processing row " + row.ToString()
        // Some data processing

    // Example where an exception might occur (e.g., file not found)
Catch (FileNotFoundException e)
    Log Message: "The specified file was not found. Please check the file
path."
    // Send email notification about the missing file
Catch (TimeoutException e)
    Log Message: "The operation timed out. Retrying..."
```

```
    // Retry logic if needed (Retry Scope)
Catch (Exception e)
    Log Message: "An unexpected error occurred: " + e.Message
Finally
    // Ensure that Excel is always closed
    Close Excel Application
    Log Message: "Process completed."
```

4. Debugging Tools & best practices

Debugging Tools and Best Practices in UiPath

Debugging is an essential part of RPA development in UiPath. It helps you identify and resolve issues in your workflows, ensuring that the automation runs smoothly and performs as expected. Below are the **debugging tools** available in UiPath and **best practices** for effectively using them.

1. Debugging Tools in UiPath

a. Breakpoints

- **What It Is**: Breakpoints allow you to pause the execution of your workflow at a specific point. This helps you inspect variables, states, and the flow of execution up until that point.
- **How to Use**:
  - Right-click on the activity where you want to set a breakpoint and select **Toggle Breakpoint**.
  - Alternatively, press **F9** to set a breakpoint.
- **When to Use**:
  - When you want to check intermediate values or the state of variables during execution.
  - To pause before complex operations (e.g., reading/writing files, interacting with UI).

b. Step Into / Step Over

- **Step Into (F11)**: Moves inside the invoked workflow or method and steps through each activity.
- **Step Over (F10)**: Executes the current activity but skips over invoked workflows or methods.
- **How to Use**:
  - **Step Into**: To go into an invoked workflow and see what's happening inside.
  - **Step Over**: To execute an activity and skip over any invoked workflows or methods you don't need to debug.
- **When to Use**:
  - Use **Step Into** when debugging custom workflows or invoked methods.
  - Use **Step Over** for high-level operations or workflows that don't need deep inspection.

c. Watch Panel

- **What It Is**: The **Watch Panel** allows you to monitor the value of specific variables or expressions during execution.
- **How to Use**:

    - Right-click on the variable in the Variables panel or directly in the workflow and select **Add to Watch**.
    - The variable will appear in the Watch panel where you can track its value in real time.
- **When to Use**:
    - To keep track of specific variables (e.g., loop counters, file paths) and see their values at different stages of the workflow.
    - Especially useful when debugging loops or conditions.

d. Output Panel

- **What It Is**: The **Output Panel** displays log messages, exceptions, and debug information.
- **How to Use**:
    - Use the **Log Message** activity to log custom messages during execution.
    - Check the Output Panel during debugging to view these messages.
- **When to Use**:
    - Log key variable values or status messages at critical points in the workflow to understand the execution flow.

e. Immediate Panel

- **What It Is**: The **Immediate Panel** allows you to evaluate expressions or variables during debugging and get immediate feedback.
- **How to Use**:
    - You can type expressions (e.g., `myVariable.ToString()`) or evaluate variables directly in the panel.
- **When to Use**:
    - To quickly check the value of variables or evaluate expressions without stopping the workflow.

f. Highlight Activities

- **What It Is**: The **Highlight Activities** feature highlights the currently executing activity to help you visually track the execution.
- **How to Use**:
    - Enable it in the debug settings or click the "Highlight Activities" button to turn it on.
- **When to Use**:
    - To see exactly which activity is being executed, especially in long or complex workflows.

2. Best Practices for Debugging in UiPath

a. Set Breakpoints Strategically

- **Where to Place Breakpoints**:
    - Set breakpoints before key operations, like reading files, interacting with UI elements, or invoking custom workflows.
    - Place breakpoints after decision-making activities (e.g., `If` conditions or `Switch` activities) to verify correct flow.

- **Avoid Overusing Breakpoints**: Set breakpoints only at critical points to avoid slowing down the debugging process. Too many breakpoints can make the debugging process less efficient.

b. Use the Watch Panel for Variables

- **What to Track**:
  - Use the Watch Panel to monitor important variables such as counters, flags, or data input values. This helps you verify that the variables are behaving as expected.
  - Track variables that influence the flow of your automation, such as elements being clicked, file paths, or data being processed.
- **When to Add Variables to Watch**:
  - Add variables that might change unexpectedly, such as those inside loops or conditional branches.

c. Use Log Messages to Track Progress

- **Purpose**: Use **Log Message** activities to print relevant information to the Output Panel at various points in the automation.
- **What to Log**:
  - Log the values of variables, messages about key process steps, and error messages.
  - Log entries before and after significant operations, like reading files, writing to databases, or opening applications.
- **When to Use**:
  - When testing your workflow in non-debug mode to track if the process is functioning as expected.
  - To track progress and identify where the workflow fails in production.

d. Simplify Complex Workflows

- **How**: If you have a very large and complex workflow, break it into smaller sub-workflows. Debugging small, modular workflows is easier than debugging large monolithic ones.
- **Why**: Debugging smaller workflows allows you to isolate the issue to a specific part of the process, making it easier to identify and fix errors.

e. Step Through the Workflow

- **Step Through Using "Step Into"**: Use the **Step Into** feature to step through workflows and activities in fine detail. This allows you to see exactly how data flows through your automation and spot any potential issues.
- **Avoid Stepping Through Complex Loops**: If your loop is large or involves repetitive actions, try using **Step Over** to skip over activities that don't require deep inspection.

f. Test with Different Scenarios

- **Test Various Inputs**: Always test your workflows with multiple input data sets to ensure they behave correctly in various scenarios.

    - For example, test with empty data, unexpected file formats, incorrect paths, and other edge cases.
- **Simulate Errors**: Try to simulate errors like network timeouts or UI elements not being found to see how your workflow behaves under these conditions.

g. Take Advantage of the Immediate Panel

- **Use for Quick Testing**: The **Immediate Panel** is an excellent tool for testing out expressions or evaluating variables without interrupting the workflow.
    - Use it to check the value of a variable mid-debugging or to test small code snippets.
- **Quick Validation**: When you're unsure if a specific piece of code or expression is correct, use the Immediate Panel to quickly validate your assumptions.

h. Use "Highlight Activities" During Debugging

- **Why**: The **Highlight Activities** feature is helpful for tracking where the process is in large workflows. It visually highlights the currently executing activity, making it easier to follow the flow of execution.
- **When to Use**: Especially useful when debugging workflows with many activities or when the issue is related to how the activities are executing in real-time.

i. Debug in Isolation

- **Isolate Problematic Sections**: If you're not able to identify the issue in a complex workflow, isolate the problematic section by commenting out other parts of the workflow and running only the suspect section.
- **Why**: This reduces complexity and allows you to focus only on the part of the workflow causing the issue, making it easier to identify bugs or errors.

3. Advanced Debugging Techniques

a. Debugging in Different Environments

- **Develop in Studio**: Debug workflows in the **Studio** environment with full access to debugging tools, such as breakpoints, the Watch Panel, and the Output Panel.
- **Test in Attended Mode**: For **Attended Automation**, you can use **UiPath Assistant** to debug workflows interactively by seeing how it interacts with the user interface in real-time.
- **Test in Unattended Mode**: For **Unattended Automation**, use **Orchestrator** or execute robots in the background to see how your workflow behaves when running without direct human interaction.

b. Use the Debugging Logs

- **Why**: UiPath creates detailed logs during execution. You can examine the logs to understand why a process failed, especially in production or unattended automation.
- **Where to Find**: Logs can be found in **Orchestrator** logs or on the local machine where the robot is running.

- **What to look for**: Pay attention to error messages, warning logs, and trace logs to understand the source of issues.

Deploying and Maintaining the BOT:

Deploying and Maintaining the BOT in UiPath

Once automation (bot) is developed and thoroughly tested in UiPath, the next important phases are **deployment** and **maintenance**. These steps ensure that your bot runs in a production environment efficiently, and any issues or updates are handled appropriately over time. Let's break down the process of deploying and maintaining a bot in UiPath.

1. Deploying the BOT in UiPath

Deployment refers to the process of transferring your robot from the development environment to the production environment. This includes moving the workflow to the right platform, scheduling, and ensuring it works as expected.

a. Deploying to Orchestrator

UiPath **Orchestrator** is the central platform for managing, scheduling, and monitoring robots. It plays a vital role in deployment, as it provides a seamless interface to deploy and manage robots.

Steps for Deploying a Bot to Orchestrator:

1. **Publish the Process from UiPath Studio**:
   - After your workflow is ready and tested in **UiPath Studio**, you can **publish** it to **Orchestrator**.
   - Go to **UiPath Studio** → Click on **Publish** (or **Publish to Orchestrator**) in the **Design** tab.
   - When publishing, make sure you select the correct Orchestrator instance where the robot should be deployed.
   - Choose an appropriate package name, version, and description for the process.
2. **Create a Robot in Orchestrator**:
   - Once the process is published, navigate to **Orchestrator** and create a robot under the **Robots** section.
   - Choose the appropriate type of robot (e.g., **Attended**, **Unattended**, or **Non-production**).
   - Assign the robot to an environment and ensure it is linked to an available machine.
3. **Create an Environment**:
   - In **Orchestrator**, create an **Environment** (if not already created) to group robots together based on deployment (e.g., Development, Test, and Production).
   - Assign the robot to this environment to streamline scheduling and management.
4. **Create a Process in Orchestrator**:
   - In **Orchestrator**, navigate to the **Processes** section and click **Create Process**.
   - Select the correct package (the one you published from Studio) and assign it to the environment where the robot will be running.
   - Now, the bot is ready to be executed or scheduled in this environment.

5. **Schedule the Process**:
    - In **Orchestrator**, go to the **Schedules** section to schedule when the process should run.
    - Configure the time, recurrence, and robot allocation.
6. **Assign the Process to Robots**:
    - Ensure that the **robot** is assigned to the process and that the correct **environment** is used.
    - This allows the robot to execute the process at the scheduled time.

b. Deploying the BOT Manually (for Small Scale)

If you're not using Orchestrator (e.g., in a small-scale or personal project), you can manually deploy the bot by running it from UiPath Studio or using **UiPath Assistant**.

1. **Publish the Process Locally**:
    - If you're not using Orchestrator, you can still publish the process locally in **.nupkg** format.
    - Store the package in a local folder.
2. **Run the Process from UiPath Assistant**:
    - You can run the automation directly from **UiPath Assistant**, a desktop application where you can manage and trigger bots.
    - Simply install UiPath Assistant on the machine, add the process, and trigger it manually or on-demand.

2. Maintaining the BOT in UiPath

Once the bot is deployed, the next step is to **maintain** it to ensure its continued efficiency and stability. This phase includes monitoring performance, handling issues, troubleshooting errors, and updating workflows.

a. Monitoring Bots

Monitoring bots ensures that your automation is running as expected and helps you catch any errors that might occur during execution.

Tools for Monitoring Bots:

1. **UiPath Orchestrator**:
    - **Dashboard**: Provides real-time visibility of all running bots, their statuses, and logs.
    - **Jobs**: You can monitor individual jobs, check whether they are running, completed, or failed.
    - **Logs**: View the logs of bot executions in **Orchestrator** to understand what happened during the job execution and diagnose issues.
    - **Alerts**: Set up alerts in Orchestrator to notify you when a bot encounters issues (e.g., job failure or performance degradation).
2. **UiPath Assistant**:
    - **Status Monitoring**: For Attended Bots, UiPath Assistant allows you to monitor the bot's current status, trigger manual execution, and review logs.

- **Queue Monitoring**: Monitor queue items and ensure they are being processed as expected.

3. **Log Files**:
   - **Local Logs**: Each UiPath robot generates log files that provide detailed information about process execution. You can check the local log files in the **Logs** folder.
   - Logs help in identifying issues with execution (e.g., missing files, connection errors, UI element not found).

b. Error Handling and Troubleshooting

Bots might encounter errors due to various reasons, such as system failures, network issues, or unexpected changes in the environment.

Error Handling Strategies:

1. **Exception Handling in Workflow**:
   - Use **Try-Catch** blocks in workflows to manage errors gracefully.
   - Use **Retry Scope** to handle temporary errors, such as network timeouts or unavailable UI elements.
   - Configure logging in the **Catch** block to capture error messages for further investigation.
2. **Use Global Exception Handlers**:
   - Set up **Global Exception Handlers** in UiPath to handle exceptions that occur at the workflow level across the entire project.
   - This ensures that even unexpected errors are caught and logged properly.
3. **Testing and Debugging**:
   - Run periodic tests to ensure that the bot is still functioning correctly after updates or system changes.
   - Use the debugging tools, such as **Breakpoints**, **Watch Panel**, and **Output Panel**, to investigate issues and determine root causes.
4. **Error Logging and Notifications**:
   - Set up **Email** or **SMS alerts** for critical errors or failures to ensure that the responsible personnel are immediately notified.
   - Use **Orchestrator Alerts** for automatic notification upon bot failure or other issues.

c. Updating the BOT

As business processes evolve or environmental changes occur, your bots may need to be updated to reflect new requirements or handle changes (e.g., UI changes, new APIs, new software versions).

Steps to Update the Bot:

1. **Update the Workflow in UiPath Studio**:
   - Make necessary changes to the automation in **UiPath Studio** based on new requirements or fixes for issues.
   - Ensure that the changes are tested thoroughly in the **Development** environment.
2. **Publish the New Version to Orchestrator**:

    - Once changes are made, republish the updated process to **Orchestrator** with a new version number.
    - Ensure that the **robot** running the updated process is linked to the latest version in Orchestrator.
3. **Test the Changes**:
    - Test the updated workflow in a **Test** or **Staging** environment to ensure that the changes do not introduce new errors or issues.
4. **Rollout the Update**:
    - Once testing is successful, deploy the updated workflow to production. If using **Orchestrator**, this can be done by updating the process version linked to your production environment.
5. **Monitor After Updates**:
    - After deployment, monitor the robot closely for any issues and check if the updates work as expected. Review logs, jobs, and alerts for errors.

d. Performance Tuning

To maintain an optimal performance of bots, consider the following performance management strategies:

1. **Optimize Workflow Design**:
    - Reduce the complexity of workflows by breaking them into smaller reusable components.
    - Use **parallel processing** (if needed) to speed up processing time.
    - Avoid excessive logging and unnecessary delays in your workflows.
2. **Queue Management**:
    - Use **Queues** in Orchestrator to manage large volumes of tasks efficiently. Make sure that robots are able to process queue items without performance issues.
    - Set up **Retry Mechanism** for failed queue items to ensure tasks are reattempted if an error occurs.
3. **Resource Management**:
    - Ensure that robots have sufficient system resources (e.g., CPU, memory) to execute workflows without performance degradation.
    - Use **Orchestrator's performance metrics** to monitor resource usage and make adjustments accordingly.

3. Best Practices for Maintaining a BOT

1. **Routine Monitoring**: Regularly check Orchestrator dashboards, logs, and job statuses to ensure everything is running smoothly.
2. **Proactive Issue Resolution**: Set up alerts for critical errors to ensure that issues are addressed quickly. Resolve issues before they impact business operations.
3. **Backup and Versioning**: Always maintain a backup of your bot workflows and any configuration settings in case you need to roll back to an earlier version.
4. **Periodic Updates and Testing**: Periodically update the workflows as business requirements evolve and test new updates in a non-production environment before rolling them out.

5. **Optimize for Scalability**: As the volume of automation grows, consider optimizing for scalability by using Orchestrator's features such as **scaling robots**, **load balancing**, and **queue management**.
6. **User Training**: Ensure that end-users and stakeholders know how to handle simple issues (e.g., manually triggering bots or identifying errors) to reduce the dependency on developers.

5. Publishing the Automation solution using publish utility

Publishing the Automation Solution Using UiPath Publish Utility

Publishing an automation solution in UiPath is the process of making your automation available for deployment, sharing, and execution in environments like **UiPath Orchestrator** or locally. The **Publish Utility** is an essential tool in UiPath that allows you to easily package and publish your automation solution to an orchestrator or a local machine.

Why Use Publish Utility?

- **Centralized Management**: Helps manage, deploy, and monitor your automation from **UiPath Orchestrator**.
- **Version Control**: Ensures that your automation is always published with a version number, enabling rollbacks to previous versions if needed.
- **Automation Distribution**: Facilitates the distribution of automation packages across different environments (e.g., test, production).

Steps to Publish an Automation Solution Using UiPath Publish Utility

1. Preparing the Automation Solution for Publishing

Before you publish, ensure that your automation solution is ready and tested in **UiPath Studio**.

1. **Complete the Development**:
   - Ensure that all workflows (XAML files) are completed and debugged.
   - Validate that the automation runs successfully in **UiPath Studio**.
2. **Organize the Project**:
   - Your automation solution should be organized into relevant folders in **UiPath Studio**.
   - Ensure that all necessary libraries, dependencies, and packages are included.
   - Check the **project.json** file to ensure that dependencies and settings are correctly defined.

2. Publishing the Automation Using Publish Button in UiPath Studio

Here are the steps to publish the automation directly from **UiPath Studio**:

1. **Open UiPath Studio**:
   - Launch **UiPath Studio** and open the project you wish to publish.

2. **Select the Publish Option**:
    - Once your project is open, go to the **Design** tab in the top menu.
    - Click on the **Publish** button located in the toolbar.
3. **Choose the Publishing Destination**: After clicking on **Publish**, you will be presented with several publishing options:
    - **Publish to Orchestrator**:
        - Select **Orchestrator** as the destination if you want to publish the automation to **UiPath Orchestrator**. You will need to have an active connection with **Orchestrator** configured in **UiPath Studio**.
        - The **Orchestrator URL**, **Machine Key**, and **Robot Connection** should be configured correctly.
        - Select the correct **Package Name** and **Version**. The version helps in versioning your automation.
    - **Publish to a Local Folder**:
        - If you wish to save the package locally, select a folder where the `.nupkg` (NuGet package) file will be saved.
        - This package can later be manually deployed or shared.
4. **Confirm and Publish**:
    - Once the destination is selected, confirm your choices (including the version of the automation) and click **Publish**.
    - If publishing to Orchestrator, make sure the **Orchestrator** is connected and accessible from your Studio.
    - The package will be published, and the status will appear in the **Output** panel.

3. Publish to UiPath Orchestrator

Publishing to **UiPath Orchestrator** allows you to deploy the automation solution across multiple environments and robots.

1. **Set Up Orchestrator Connection**:
    - Ensure that your **UiPath Studio** is connected to **UiPath Orchestrator**.
    - Go to **Settings** > **Orchestrator Settings** in **UiPath Studio** and enter the correct **Orchestrator URL** and **Machine Key** (from Orchestrator).
    - Check if your machine is connected to Orchestrator (this can be verified under **Robots** in Orchestrator).
2. **Publish the Package**:
    - When you click **Publish**, if connected to Orchestrator, the package will automatically upload to **Orchestrator**.
    - You can see the published version in **Orchestrator** under **Packages**.
3. **Deploy in Orchestrator**:
    - After publishing, you can create a new **Process** in Orchestrator and link it to the published package.
    - You can assign it to a robot and schedule it for execution.

4. Publish to Local Machine

If you prefer not to use **Orchestrator**, or for small-scale deployments, you can publish the automation locally.

1. **Choose Local Folder**:
    - After selecting the **Publish** option in **UiPath Studio**, choose **Local Folder** as the destination.
    - This will save the automation package in a `.nupkg` format (NuGet package), which you can later distribute.
2. **Distribute the Package**:
    - You can move or copy this package to another machine and manually run the automation through **UiPath Assistant**.
    - Alternatively, you can manually upload the `.nupkg` package to **Orchestrator** later if needed.

5. Versioning the Automation Package

Each time you publish a new version of your automation solution, you need to define a new version number for your automation package. Versioning is critical for:

- **Managing multiple versions**: You can revert to a previous version if there's an issue with the latest one.
- **Keeping track of changes**: Helps in tracking updates and changes made over time.

Versioning Tips:

- Increment the version number each time you publish a new package. A common practice is to use **Semantic Versioning** (e.g., **1.0.0**, **1.1.0**, **2.0.0**).
    - **Major** (1.x.x): Significant changes, such as breaking changes or major updates.
    - **Minor** (x.1.x): New features or enhancements that are backward compatible.
    - **Patch** (x.x.1): Bug fixes or minor improvements.

6. Retrieving Published Packages from Orchestrator

After publishing your automation solution to **Orchestrator**, you can view and manage the package versions:

1. **Go to Orchestrator**:
    - Navigate to **Orchestrator** and log in.
    - Go to the **Packages** tab to see all the published automation packages.
2. **Check Package Version**:
    - In the **Packages** section, you can view different versions of your automation, their statuses, and other details.
    - From here, you can choose which version of a package you want to deploy to a robot.
3. **Deploy and Monitor**:

- Create a new **Process** or update an existing one to deploy the automation package to a robot.
- Monitor the job execution and logs from **Orchestrator** to ensure everything runs as expected.

7. Best Practices for Publishing Automation Solutions

- **Always Version Your Packages**: Ensure each new version is properly versioned. This helps in rollback and troubleshooting if needed.
- **Test Before Publishing**: Always test the automation in the development or staging environment before publishing to production.
- **Check Dependencies**: Ensure that all required dependencies (like libraries, packages, and custom activities) are included in the project before publishing.
- **Use Orchestrator for Centralized Management**: If your organization uses Orchestrator, publish to Orchestrator for better control and monitoring of robots.
- **Backup Packages**: Keep a backup of automation packages, especially if they are deployed to production.
- **Document Versions**: Maintain good documentation on what changes each version includes for transparency and traceability.

6. Creating a provision robot from the server

Creating and Provisioning a Robot from the Server in UiPath

Provisioning a robot from a server in UiPath involves setting up a **UiPath Robot** in **UiPath Orchestrator**, so that it can execute automation workflows. The robot can be either **Attended** (which requires user interaction) or **Unattended** (runs without user interaction).

Steps for Creating and Provisioning a Robot from the Server

1. Prepare the Orchestrator Environment

To provision a robot, ensure that your **UiPath Orchestrator** is already set up and configured. You need to have access to your **Orchestrator** account and the necessary permissions to create and manage robots.

1. **Access Orchestrator**:
   - Open **UiPath Orchestrator** in a web browser.
   - Log in with your credentials.
2. **Create an Environment (Optional)**:
   - An environment is a logical grouping of robots. You may want to create an environment before provisioning the robot.
   - Navigate to **Tenant** → **Environments** → **Add** to create a new environment.

2. Install the UiPath Robot on the Server (Machine)

If you're provisioning a robot on a server, you need to ensure that the **UiPath Robot** is installed on that server.

1. **Download and Install UiPath Robot**:
    - Go to the **UiPath website** and download the **UiPath Studio** (which includes the robot) installer.
    - Run the installer on the server you plan to provision the robot for.
    - Follow the installation steps to complete the setup. Ensure that you install **UiPath Robot** as part of the installation.
2. **Connect the Robot to Orchestrator**:
    - After installing UiPath Robot on the server, you will need to connect it to **UiPath Orchestrator**. This can be done by setting up the **Robot** in the **UiPath Assistant** (installed along with UiPath Robot).

3. Set Up the Robot in UiPath Orchestrator

Now, to create the robot in **Orchestrator**, follow these steps:

1. **Navigate to Robots in Orchestrator**:
    - Log in to **UiPath Orchestrator**.
    - In the **Orchestrator** dashboard, navigate to **Robots** in the **Management** section.
2. **Add a New Robot**:
    - Click on **Add** to create a new robot.
3. **Configure Robot Details**: Fill in the following details for the new robot:
    - **Robot Name**: A unique name for the robot.
    - **Robot Type**: Choose the type of robot:
        - **Attended**: Requires a user to be logged in and interacts with the robot.
        - **Unattended**: Can run without user intervention, typically used for server-side automation.
        - **Non-Production**: Used for testing purposes.
    - **Machine Name**: Select the machine (server) on which this robot will run. If the machine is not listed, ensure the machine has been created and connected to Orchestrator (explained later).
    - **Domain and Username**: Enter the **Windows domain** and **username** for the robot if using an **Unattended** robot (this is necessary for login credentials on the server).
    - **Provisioning Type**: Choose **Automatic** for unattended robots if you want the robot to provision automatically without manual login.
    - **Orchestrator URL**: Provide the **Orchestrator URL** if not pre-configured.
4. **Assign the Robot to an Environment**:
    - After setting up the robot, assign it to an **environment**. The environment helps manage which robots are part of which process or automation.
5. **Save the Robot**:
    - Click **Save** to create the robot in Orchestrator.

4. Configure the Machine (Server) in Orchestrator

Before the robot can function, you need to ensure that the machine (server) is registered in **Orchestrator**. This is done using **Machine Keys**.

1. **Add a New Machine**:
    - Go to **Orchestrator → Machines → Add Machine**.
    - Provide the **Machine Name** and **Type** (standalone or machine template).
    - After saving, you'll be provided with a **Machine Key**.
2. **Connect the Robot to the Machine**:
    - Go to **UiPath Assistant** on the server (machine where the robot will run).
    - Open **UiPath Assistant**.
    - Click on **Settings** (gear icon) → **Orchestrator Settings**.
    - Paste the **Orchestrator URL** and **Machine Key** from Orchestrator into the respective fields.
    - The robot will now be connected to the machine and ready to execute automation tasks.

5. Provision the Robot on the Server

After the robot is created in Orchestrator and the machine is connected, you can now **provision the robot**.

1. **Provisioning via Orchestrator**:
    - Ensure the **robot** is assigned to a machine (created earlier) in **Orchestrator**.
    - The **UiPath Assistant** on the server will recognize this and will now be able to communicate with Orchestrator to retrieve and run automation tasks.
2. **Start the Robot**:
    - In **Orchestrator**, select the robot you've created.
    - You can now start, stop, or schedule the robot to run tasks.
    - You can also monitor the robot's activity and check logs to ensure the robot is functioning correctly.

6. Testing and Verification

1. **Run a Job**:
    - Go to **Orchestrator → Jobs → Start Job**.
    - Select the robot and the automation process to run.
    - If everything is configured properly, the job should start and the robot should execute the automation.
2. **Monitor Robot Status**:
    - After starting a job, you can monitor the robot's status under **Jobs**.
    - **Running**: The robot is executing the job.
    - **Successful/Failed**: View the status of the job to see whether it completed successfully or encountered errors.
    - **Logs**: Check the logs to troubleshoot any issues with the robot's execution.

7. Additional Configuration (For Unattended Robots)

If the robot is **Unattended**, you may need to ensure the following configurations for smooth operation:

1. **Windows Credentials**:
   - For **Unattended** robots, ensure that the robot has proper login credentials (Windows username and password) configured in **Orchestrator**.
   - This allows the robot to execute processes without requiring user intervention.
2. **Machine Provisioning**:
   - If you're using **Unattended** robots in virtual machines or server environments, ensure that the machine has been provisioned correctly in **Orchestrator** with all necessary settings.

7. Connecting a robot to server

Connecting a Robot to a Server in UiPath

Connecting a **UiPath Robot** to a server involves registering the machine in **UiPath Orchestrator** and linking the **Robot** installed on the server to Orchestrator. Once connected, the robot can start executing automation tasks remotely, either as **Attended** (with user interaction) or **Unattended** (running autonomously).

Steps to Connect a Robot to a Server

1. Install UiPath Robot on the Server

Before connecting the robot to the server, you need to install **UiPath Robot** on the server machine. If you have already installed **UiPath Studio** on the server, the robot component will also be installed automatically.

Steps to Install the Robot:

1. **Download UiPath Studio & Robot**:
   - Visit the UiPath website to download the **UiPath Studio** installer (the Robot is included in this installation).
2. **Install UiPath Studio and Robot**:
   - Run the downloaded installer on the server.
   - During installation, ensure you install **UiPath Robot**. If you only need the Robot, you can choose to install the Robot only (without the Studio).
3. **Launch UiPath Assistant**:
   - After installation, open the **UiPath Assistant** on the server machine. The Assistant allows you to configure and manage the robot's connection with **Orchestrator**.

2. Register the Machine in UiPath Orchestrator

You need to register the server machine in **UiPath Orchestrator** to enable the robot to connect to it.

Steps to Register a Machine in Orchestrator:

1. **Log into Orchestrator**:
   - Open **UiPath Orchestrator** in your browser and log in using your credentials.
2. **Navigate to Machines**:
   - Go to the **Tenant** section → **Machines** tab.
3. **Add a New Machine**:
   - Click **Add Machine** to create a new machine record.
4. **Configure Machine Details**:
   - **Machine Name**: Provide a unique name for the server machine (e.g., `ServerMachine`).
   - **Machine Type**: Choose the type of machine:
     - **Standard Machine** (for individual robots).
     - **Template Machine** (for multiple robots running on the same template machine).
   - **Machine Key**: After creating the machine, a **Machine Key** will be generated. You will need this key to connect the robot to Orchestrator.
5. **Save the Machine**:
   - Click **Save**. The **Machine Key** is displayed, and you need to copy this key to connect the robot on the server.

3. Connect the Robot to Orchestrator

Now that the machine is registered in Orchestrator, you can connect the **Robot** on the server to Orchestrator using the **Machine Key**.

Steps to Connect the Robot to Orchestrator:

1. **Open UiPath Assistant on the Server**:
   - Open **UiPath Assistant** on the server where the robot is installed.
2. **Go to Robot Settings**:
   - Click on the **Settings** icon (gear icon) at the top-right of **UiPath Assistant**.
3. **Enter Orchestrator Details**:
   - Under **Orchestrator Settings**, enter the following details:
     - **Orchestrator URL**: Provide the URL of your **UiPath Orchestrator** (e.g., `https://platform.uipath.com`).
     - **Machine Key**: Paste the **Machine Key** that you copied from Orchestrator when you created the machine.
4. **Click Connect**:
   - After entering the details, click **Connect**. The robot will now connect to Orchestrator, and its status will change to **Connected** in UiPath Assistant.

4. Create and Provision the Robot in UiPath Orchestrator

Once the robot is connected to Orchestrator, you need to create a robot record and assign it to an environment.

Steps to Create and Provision the Robot:

1. **Navigate to Robots in Orchestrator**:
   - In **UiPath Orchestrator**, go to the **Robots** tab under **Tenant**.
2. **Click Add to Create a Robot**:
   - Click **Add** to create a new robot.
3. **Configure Robot Details**:
   - **Robot Name**: Give the robot a name (e.g., `ServerRobot`).
   - **Robot Type**: Choose between:
     - **Attended Robot**: Requires user interaction and is typically used on desktops.
     - **Unattended Robot**: Runs autonomously and is typically used on servers.
   - **Machine Name**: Select the machine (the server) that you previously created and registered in Orchestrator.
   - **Domain and Username**: For **Unattended Robots**, specify the **Windows domain** and **username** the robot will use to log into the server.
4. **Assign to an Environment**:
   - Assign the robot to an existing **Environment** or create a new environment in Orchestrator. Environments group robots that will work on specific automation tasks.
5. **Save the Robot**:
   - Click **Save** to create the robot in Orchestrator.

5. Deploy the Robot to the Server and Start Jobs

Now that the robot is created and provisioned, you can deploy processes to the robot and start automation jobs.

Steps to Deploy the Robot and Start Jobs:

1. **Create or Choose a Process**:
   - In **Orchestrator**, go to the **Processes** tab and either create a new process or select an existing process to deploy.
2. **Assign Process to Robot**:
   - In the **Jobs** section of Orchestrator, click **Start Job**.
   - Select the robot (the one you just created and connected) and the process you want it to execute.
   - You can choose to run it in **Attended** or **Unattended** mode, depending on your robot type.
3. **Monitor the Job**:
   - Once the job is started, you can monitor its status in the **Jobs** section of Orchestrator. It will show if the job is **Running**, **Completed**, or **Failed**.
   - Check the **Logs** for details on execution, and make any necessary troubleshooting if an error occurs.

6. Verifying the Connection and Robot Operation

Once everything is set up, you should verify that the robot is connected and running correctly:

1. **Robot Status in UiPath Assistant**:
   - Check that the robot's status in **UiPath Assistant** on the server is **Connected**.
2. **Check Job Execution**:
   - In **Orchestrator**, monitor job execution logs for any errors or issues. If the job starts successfully and completes, the robot is functioning correctly.
3. **Robot Health**:
   - In **Orchestrator**, you can also view the **Health** tab for the robot to check the overall status and troubleshoot any connectivity issues.

8. Deploy the robot to server

Deploying a UiPath Robot to a Server

Deploying a **UiPath Robot** to a server involves setting up the robot on the server, registering it in **UiPath Orchestrator**, and ensuring that the robot can execute automation tasks. This process is commonly used for **Unattended Robots**, which can run autonomously without human intervention, but can also be applied to **Attended Robots** that require user interaction.

Here is a step-by-step guide to deploy a robot to a server:

1. Install UiPath Robot on the Server

Before deploying the robot, you need to install the **UiPath Robot** on the server where the automation will run.

Steps for Installing UiPath Robot:

1. **Download UiPath Studio**:
   - Download the **UiPath Studio** installer from the UiPath website.
   - This installer includes both **UiPath Studio** and **UiPath Robot**.
2. **Run the Installer**:
   - Execute the installer on the server machine.
   - Choose **Robot** during the installation (you can skip Studio installation if you don't need it).
3. **Verify Installation**:
   - After installation, launch the **UiPath Assistant** on the server machine.
   - The **Assistant** allows you to configure and manage the robot's connection to **Orchestrator**.

2. Register the Machine in UiPath Orchestrator

The next step is to register the **server machine** in **UiPath Orchestrator** so the robot can communicate with Orchestrator.

Steps to Register the Machine:

1. **Log in to UiPath Orchestrator**:
   - Open **UiPath Orchestrator** in your web browser and log in using your credentials.
2. **Navigate to Machines**:
   - In the Orchestrator dashboard, go to **Tenant** → **Machines** section.
3. **Create a New Machine**:
   - Click on **Add** to create a new machine.
   - Enter the machine details:
     - **Machine Name**: A unique name for the machine (e.g., `ServerMachine`).
     - **Machine Type**: Choose between **Standard Machine** or **Template Machine**.
     - **Machine Key**: After saving the machine, a **Machine Key** will be generated. This key is needed to link the robot on the server with Orchestrator.
4. **Save the Machine**:
   - Click **Save** to create the machine and note down the **Machine Key**.

3. Connect the Robot to Orchestrator

Now, you'll connect the **UiPath Robot** on the server to **UiPath Orchestrator** using the **Machine Key**.

Steps to Connect the Robot:

1. **Open UiPath Assistant**:
   - Open the **UiPath Assistant** on the server.
2. **Access Settings**:
   - In **UiPath Assistant**, click on the **Settings** icon (gear icon) in the top-right corner.
3. **Enter Orchestrator URL**:
   - Under **Orchestrator Settings**, input the **Orchestrator URL** (e.g., `https://platform.uipath.com` or your custom Orchestrator URL).
4. **Enter Machine Key**:
   - Paste the **Machine Key** that you copied earlier from **UiPath Orchestrator** into the **Machine Key** field.
5. **Click Connect**:
   - After entering the details, click **Connect**. The robot will now be connected to **Orchestrator**.
6. **Verify Connection**:
   - The robot's status in **UiPath Assistant** will show as **Connected** if the connection is successful.

4. Create and Provision the Robot in Orchestrator

Once the machine is connected to Orchestrator, you need to create the robot record in **Orchestrator** and assign it to an environment.

Steps to Create and Provision the Robot:

1. **Navigate to Robots in Orchestrator**:
   - In **UiPath Orchestrator**, go to **Tenant** → **Robots** section.
2. **Add a New Robot**:
   - Click **Add** to create a new robot.
3. **Configure Robot Details**:
   - **Robot Name**: Enter a unique name for the robot (e.g., `ServerRobot`).
   - **Robot Type**: Select **Attended** or **Unattended** based on the robot's use:
     - **Attended Robot**: Requires user interaction.
     - **Unattended Robot**: Can run without human intervention, typically used for automation on servers.
   - **Machine Name**: Select the machine (server) you registered in Orchestrator earlier.
   - **Domain and Username**: For **Unattended Robots**, specify the **Windows domain** and **username** for the robot to use when logging into the server.
4. **Assign Robot to Environment**:
   - Select an **Environment** or create a new one to group robots that will run specific automation tasks.
5. **Save the Robot**:
   - Click **Save** to create the robot in **Orchestrator**.

5. Deploy the Robot to the Server

Now that the robot is registered and connected, you can deploy automation processes to it.

Steps to Deploy the Robot:

1. **Create or Select an Automation Process**:
   - In **UiPath Orchestrator**, navigate to the **Processes** section.
   - Either upload a new **.nupkg** file (process package) or select an existing process that you want to deploy to the robot.
2. **Start a Job**:
   - Go to **Jobs** in Orchestrator and click on **Start Job**.
   - Choose the robot you created earlier (e.g., `ServerRobot`).
   - Select the **process** to be executed by the robot.
3. **Monitor the Job**:
   - After the job starts, you can monitor its progress and status under the **Jobs** tab in Orchestrator.
   - The status will show if the job is **Running**, **Successful**, or **Failed**.
   - Check the **Logs** for detailed information about the job execution and any errors encountered.

6. Verify Robot Operation

Once the robot is deployed, you should verify that it is functioning correctly.

Steps to Verify:

1. **Check Robot Status**:
    - In **UiPath Assistant**, check that the robot's status is **Connected** and there are no errors.
2. **Review Logs**:
    - In **Orchestrator**, go to the **Jobs** section and check the **Logs** to verify if the robot has executed the automation process successfully.
3. **Troubleshooting**:
    - If there are issues, review the **Orchestrator Logs**, **Robot Logs**, and **UiPath Assistant** for any error messages that can guide you in troubleshooting.