An overhead view of five people sitting around a dark conference table in a modern office setting. They are engaged in a meeting, with some looking at laptops and others at documents. The scene is dimly lit with blue ambient lighting.

Автоматизация UiPath

Пошаговая инструкция

UiDemo - пошаговая инструкция

Стратегия: Поскольку нам нужны несколько роботов для параллельной обработки, нам необходим надежный способ разделения рабочей нагрузки. Лучшим подходом является использование Очередей Orchestrator (Orchestrator Queues). Создадим два проекта автоматизации. Назовем их Диспетчер (Dispatcher) и Исполнитель (Performer) Первый считывает файл Excel и загружает транзакции в очередь, а второй обрабатывает загруженные элементы очереди.

Рабочий процесс	Вход	Выход
Диспетчер	Файл Excel	Элементы очереди
Исполнитель	Элементы очереди	Обработанные транзакции

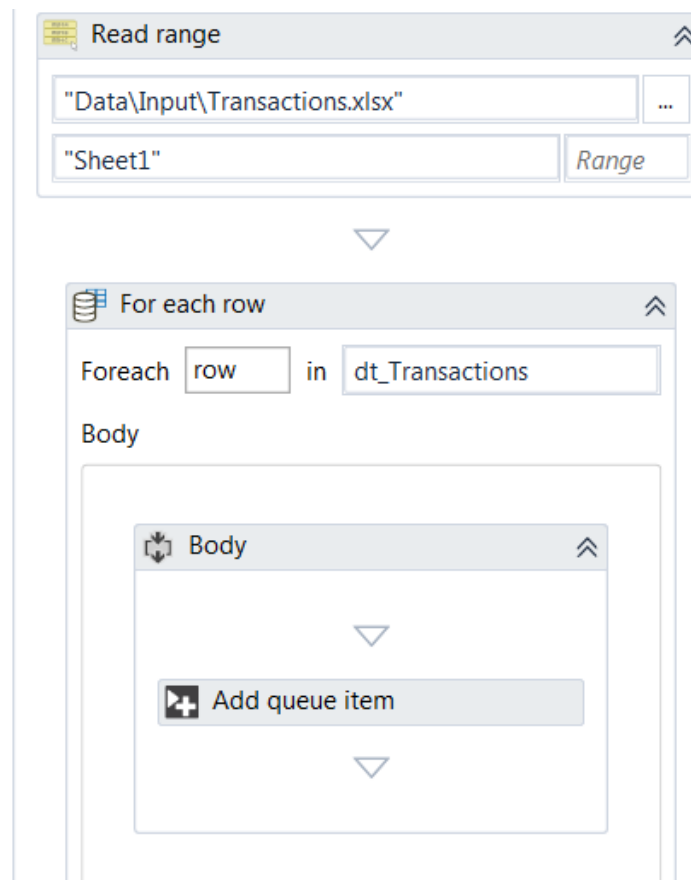
- Начнем с шаблона REFramework, так как он очень хорошо подходит для цели Исполнителя. Первое, что нам нужно сделать, это сначала переименовать папку "ReFramework_UiDemo", а затем отредактировать файл Project.json в корневой папке. Давайте изменим имя на "UiPath_REFrameWork_UiDemo", и Описание на "Demonstrating the REFramework with UiDemo". Это должно выглядеть так:

```
"name": "UiPath_REFrameWork_UiDemo",
"description": "Demonstrating the REFramework with UiDemo",
```

- Функция Диспетчера в данном случае проста – робот должен прочитать файл Excel и добавить элемент очереди для каждой строки. Это, как правило, отдельный процесс, который планируется запустить перед исполнителем, но для простоты и наглядности тестирования мы будем включать его в Исполнителя.
 - Создадим новую последовательность и назовем ее **Dispatcher – UploadQueue**.
 - Для извлечения информации из файла Excel и сохранения выходных данных в переменной DataTable требуется операция **Read Range** (Чтение диапазона).
 - Каждая строка является элементом транзакции, поэтому далее нам понадобится операция **For Each Row** (Для каждой строки). Используйте действие **Add Queue Item** (Добавить элемент очереди) в разделе **Body**.
 - Измените свойство "ItemInformation", добавив три аргумента: CashIn, OnUsCheck и NotOnUsCheck. Это должно выглядеть так:

Name	Direction	Type	Value
CashIn	In	String	row("CashIn").ToString
OnUsCheck	In	String	row("OnUsCheck").ToString
NotOnUsCheck	In	String	row("NotOnUsCheck").ToString

- Заполните поле **QueueName**, а затем создайте очередь с тем же именем в Orchestrator. Чтобы повторить неудачные транзакции, при создании очереди просто задайте значение 2 для свойства **RetryNumber**.
- Последовательность Диспетчера должна выглядеть так:



- Давайте запустим эту последовательность и проверим очередь на новые элементы.

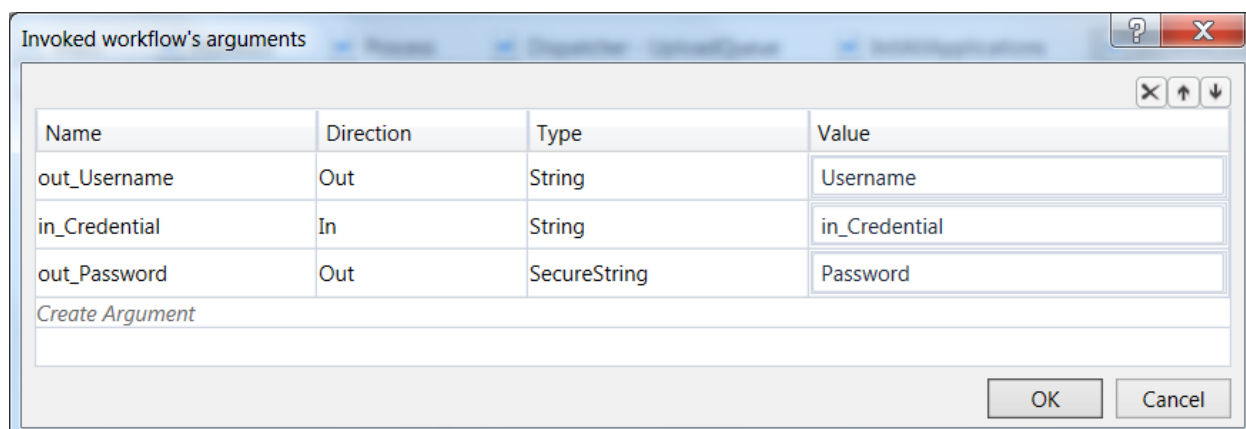
Порядок! Процесс Диспетчера был построен. Давайте теперь перейдем к **Исполнителю**!

- Входные данные для Исполнителя представляют собой очередь в Orchestrator, что означает, что во Framework необходимо внести совсем небольшие изменения. Мы начинаем с конфигурационного файла и изменим **"QueueName"** на то, что мы использовали ранее. Нам нужен еще один параметр – учетное имя для UiDemo. Значение **MaxRetryNumber** в списке

констант должно быть равно 0, так как мы используем функцию повтора очереди в Orchestrator. Таблица "Параметры" должна выглядеть так:


Name	Value	Description
QueueName	REFramework_UiDemo	Orchestrator Queue Name. Be sure to match the two names.
UiDemoCredential	UiDemoCredential_REFramework	Credential used to log in to UiDemo

- Сохраните и закройте конфигурационный файл.
- Следующим шагом инициализируем процесс.
 - Во-первых, мы должны авторизоваться в UiDemo, прежде чем можно будет выполнять повторяющиеся транзакции.
 - Мы всегда должны размышлять в категориях многократно используемых компонентов - процесс авторизации определенно попадает под эту категорию. Рекомендуется сгруппировать все файлы xaml в папках в соответствии с именами приложений.
 - Создадим **подпапку UiDemo** в корневом каталоге.
 - Затем создадим новую последовательность в Studio и назовем ее UiDemo_Login.xaml.
 - Переместим файл в папку UiDemo.
- Файл UiDemo_Login.xaml должен начинаться с описания. Мы также можем использовать аннотацию в самом первом действии, будь то последовательность или блок-схема, для указания функции каждого компонента и использования аргументов. Как правило, мы добавляем Precondition (Предусловие), например, в случае когда конкретный экран должен быть активным заранее, . Также включается действие POST. Как правило, последовательность входа должна также включать действия, которые инициализируют приложение, но мы сделаем это в следующих уроках. В этом случае предположим, что приложение уже было открыто при запуске процесса авторизации. "The application started" ("Приложение запущено") является предусловием нашего компонента.
- Следующим шагом является определение аргументов. Нам нужны только входной аргумент с именем Credential и два выходных для хранения имени пользователя и пароля. Обратите внимание, что аргумент Credential действует как идентификатор, используемый для извлечения имен пользователей и паролей из различных хранилищ учетных данных. Учетные данные и аргумент имен пользователей принадлежат к типу данных String (Строка), а пароль — к SecureString - особому классу .NET, используемому для защиты конфиденциальной информации. При именовании аргументов наилучшей стратегией является использование префикса, через который задается тип аргумента. Это также помогает нам отличать аргументы от переменных. Создайте входной аргумент "in_Credential" с типом String.
- Есть много вариантов хранения учетных данных. Например, мы можем использовать файл **GetAppCredentials.xaml**, о котором мы говорили в предыдущем видео.
 - Первая попытка в последовательности — извлечение Credential Asset (Ресурса учетных данных) из Orchestrator. Однако также можно использовать диспетчер учетных данных Windows. Выбор между двумя вариантами за вами, но мы рекомендуем создать Orchestrator Credential Asset, чтобы обеспечить к нему глобальный доступ .
 - Итак, используйте действие **Invoke Workflow** ("Вызов рабочего процесса") и укажите файл GetAppCredentials.xaml.
 - Импортируйте аргументы.
 - Аргумент in_Credential должен иметь значение in_Credential.
 - Создайте две переменные для хранения имени пользователя и пароля.
 - Панель аргументов должна выглядеть так:



Примечание: В продуктиве может понадобиться избежать использования файла GetAppCredentials.xaml для более управляемого поведения. Обычно, точное расположение набора учетных данных известно, что позволяет легко их получить. Если учетные данные хранятся в Orchestrator в качестве ресурсов (Assets), просто используйте действие **Get Credential** ("Получить учетные данные") в категории **Orchestrator** в разделе **Assets** ("Активы").



- Далее, нам нужно ввести имя пользователя и пароль, а затем применить **действие Click** на кнопку авторизации.
 - Убедитесь, что свойство **SimulateClick** включено.
 - Чтобы ввести пароль, необходимо применить действие **Type Secure Text** ("Напечатать защищенный текст") с паролем в свойстве **SecureText**.
 - Задействуйте свойство **SimulateType**.
 - Окончательный рабочий процесс должен выглядеть так:


UiDemo_Login

Logs in UiDemo using the Credential name input argument

Precondition: UiDemo application started
Post action: UiDemo main window opened



▼


Invoke GetAppCredentials workflow


"Framework\GetAppCredentials.xml" ...

Edit Arguments Import Arguments



▼


Type into Username


Username



Username +

▼


Type secure text Password


Password

▼


Click Log In


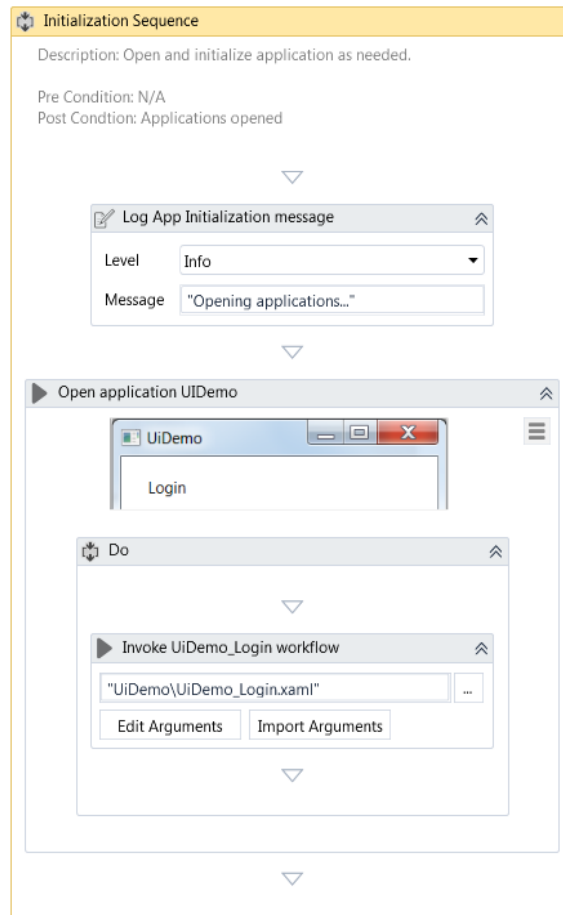
▼

После завершения любого компонента мы должны тщательно его протестировать, чтобы сократить время последующей отладки. Значения аргументов по умолчанию могут быть полезны, но иногда в качестве аргументов необходимо передать более сложные объекты. В этом случае можно создавать тестовые последовательности (Test Sequences) для создания объектов и передачи их в качестве аргументов.

- Далее идет файл `InitAllApplications.xaml`. Мы используем только одно приложение: `UiDemo`.
 - Нам необходимо применить действие **Open Application** ("Открыть приложение")
 - Укажите окно авторизации (Log In Window) в `UiDemo`. В свойстве **FileName** есть локальный путь приложения, но мы должны избегать таких жестко привязанных значений в наших рабочих процессах. Более того, если мы планируем использовать несколько роботов, мы должны помнить, что разные роботы могут иметь разные пути для одного и того же приложения.
 - К счастью, мы можем использовать текстовый актив ("Text Asset") Orchestrator для хранения значений для каждого робота (**Per Robot**). Мы не должны жестко привязывать имена активов, так как они могут меняться от одной среды к другой, так что давайте снова используем конфигурационный файл и запишем имя ресурса (Asset name) в таблице активов (**Assets sheet**). Это должно выглядеть так:

Name	Asset
UiDemoPath	UiDemoPath

- Словарь файла конфигурации уже используется в качестве входного аргумента в рабочем процессе `InitAllApplications.xaml`, поэтому давайте изменим путь к файлу на `in_Config("UiDemoPath").ToString`. Далее необходимо вызвать рабочий процесс `UiDemo_Login`, импортировать аргументы и передать учетные данные из словаря файла конфигурации следующим образом: `in_Config("UiDemoCredential").ToString`.
- Окончательный рабочий процесс должен выглядеть так:



- Чтобы протестировать InitAllApplications, нам нужно запустить файл _Test.xaml, который сначала считывает конфигурационный файл, а затем вызывает наш файл. Порядок! Всё работает.
- Затем необходимо сконфигурировать действия CloseAllApplications и KillAllProcesses. Оба достаточно просты и должны выглядеть так:

Normal App Closing Sequence

Description: Here all working applications will be soft closed

Pre Condition: N/A
Post Condition: Applications closed

▼

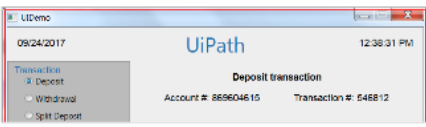
Log message

Level: Info

Message: "Closing applications..."

▼

Close application UiDemo



▼

Kill Processes

Description: Here all working processes will be killed

Pre Condition: N/A
Post Condition: N/A

▼

Log message

Level: Info

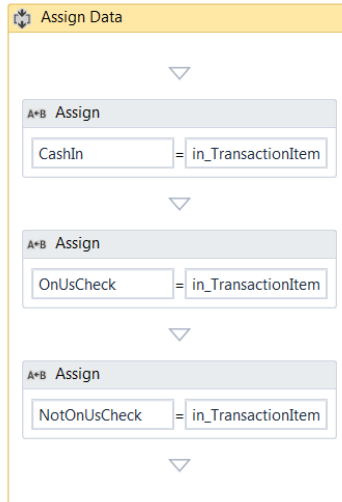
Message: "Killing processes..."

▼

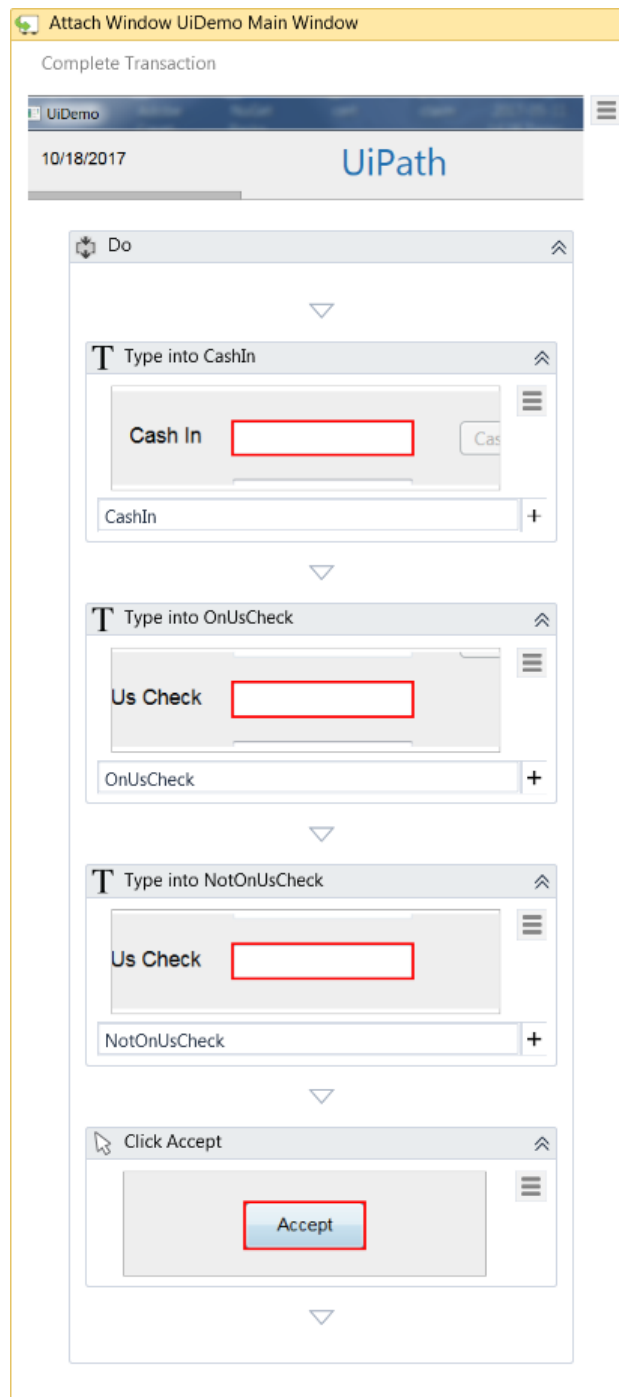
Kill process UiDemo

▼

- Теперь нам нужно заполнить файл Process.xml. Поскольку есть два решения, и второе происходит из первого, то лучшим раскладом для нашего процесса является блок-схема (Flowchart). После создания одного из них скопируйте аннотацию и аргументы из существующей последовательности процессов. Как правило, необходимо использовать динамические данные из элемента очереди (Queue) несколько раз, поэтому рекомендуется создать для него набор локальных переменных.
- Создайте последовательность и используйте действие **Assign** ("Назначить") для задания значений переменных следующим образом:



- Нам нужно проверить допустимость входных данных и является ли каждое значение числом.
 - С этой целью мы можем использовать метод **Double.TryParse**, а затем сохранить результаты в переменных с типом данных Double.
 - Используйте действие Flow Decision и установите условие **Double.TryParse(CashIn, dbl_CashIn) AND Double.TryParse(OnUsCheck, dbl_OnUsCheck) AND Double.TryParse(NotOnUsCheck,dbl_NotOnUsCheck)**.
 - Если это недопустимо, используйте действие **Throw** с новым объектом **BusinessRuleException** в качестве свойства **Exception** ("Исключение"), как показано ниже: **new BusinessException("Входные данные недопустимы")**.
- Если входные данные допустимы, добавьте другое Flow Decision, чтобы проверить, будет ли значение **dbl_CashIn** больше 1000.
 - Если это так, мы используем другое действие **Throw** с другим Сообщением об исключении (Exception message).
 - Если значение меньше, чем 1000, используем действие **Attach Window** ("Прикрепить окно") на главном экране UiDemo , три действия **Type Into**, и действие Click для кнопки **Accept** ("Принять").
 - Контейнер **Attach Window** должен выглядеть так:



- Блок-схема нашего процесса должна выглядеть следующим образом:

