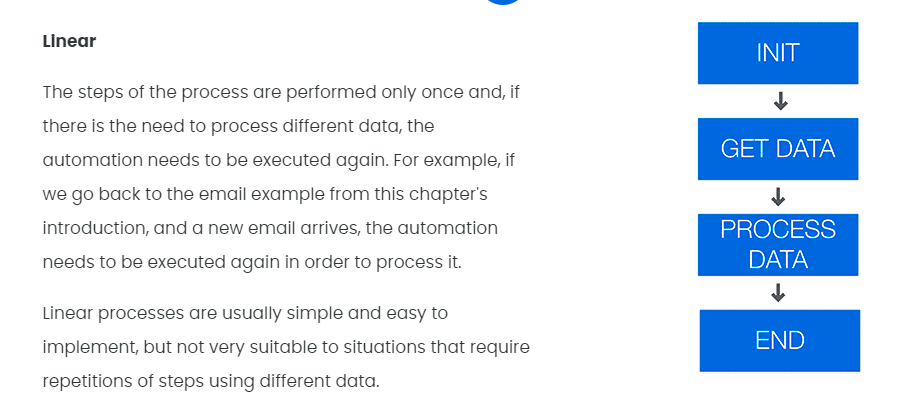
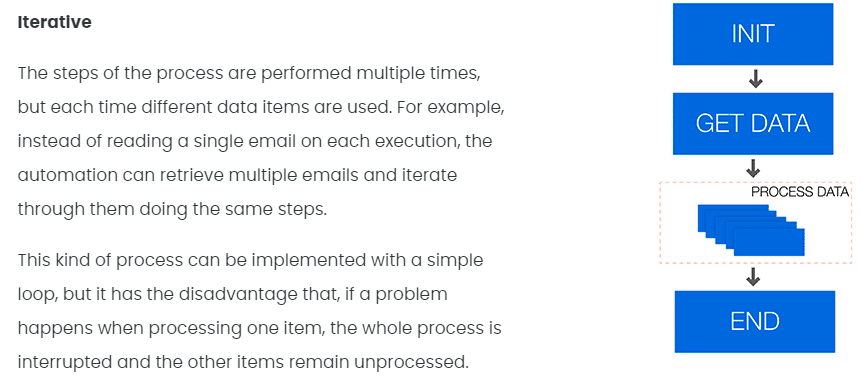
Introduction of Robotic Enterprise Framework

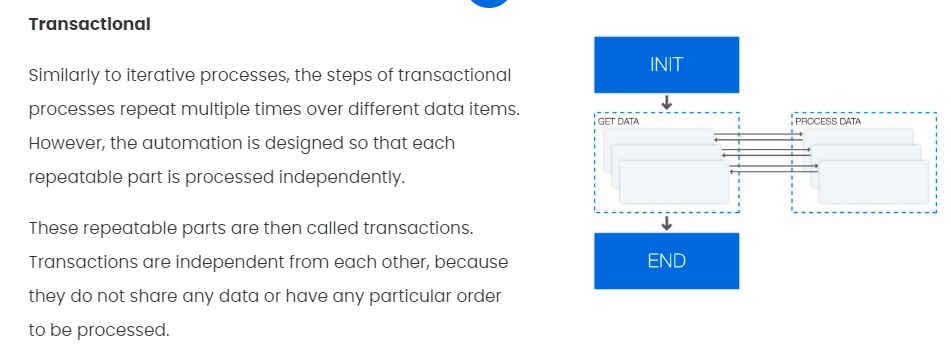
**What is a transaction?**

A transaction represents the minimum (atomic) amount of data and the necessary steps required to process the data, as to fulfill a section of a business process. A typical example would be a process that reads a single email from a mailbox and extracts data from it.

We call the data atomic because once it is processed, the assumption is that we no longer need it going forward with the business process.







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

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

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

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

**RE Framework**

**What is it?**

Generally speaking, a framework is a template that helps you design (automation) processes. At a barebones minimum, a framework should offer a way to store, read, and easily modify project configuration data, a robust exception handling scheme, event logging for all exceptions and relevant transaction information.

The REFramework is implemented as a state machine, which is a type of workflow that has two very useful features:

* States that define actions to be taken according to the specified input
* Transitions that move the execution between states depending on the outcomes of the states themselves.

You may remember state machines from the Project Organization lesson. One of the examples presented was the typical air conditioner:

* It has the OFF State, from where it moves to an IDLE State by pressing the ON/OFF button;
* From the IDLE State, it moves to either HEAT or COLD States when the temperature inputted by the user is lower and higher respectively than the current one. Once the desired temperature is achieved, it moves back to the IDLE State;
* From the IDLE State, it can move to the OFF State when the ON/OFF button is pressed;
* All the conditions that trigger the movements between the states are Transitions.

Based on a similar idea, REFramework has 4 main states that are usually common to business processes:

* 1

**Initial State**

This is where the process starts. It's an operation where the process initializes the settings and performs application checks in order to make sure all the prerequisites for the starting the process are in place.2

**Get Transaction Data State**

Gets the next transaction item. This can be a queue item or any item of a collection.

By default, transaction items are queue items, but this can be easily changed to suit your needs. This is also the state in which the Developer should set up the condition to exit this state when there are no items to process.

**Process Transaction State**

Performs actions/applies logic in various applications for the transaction item obtained at the previous step. Once a transaction item is processed, the process continues with the next available transaction item.

* 4

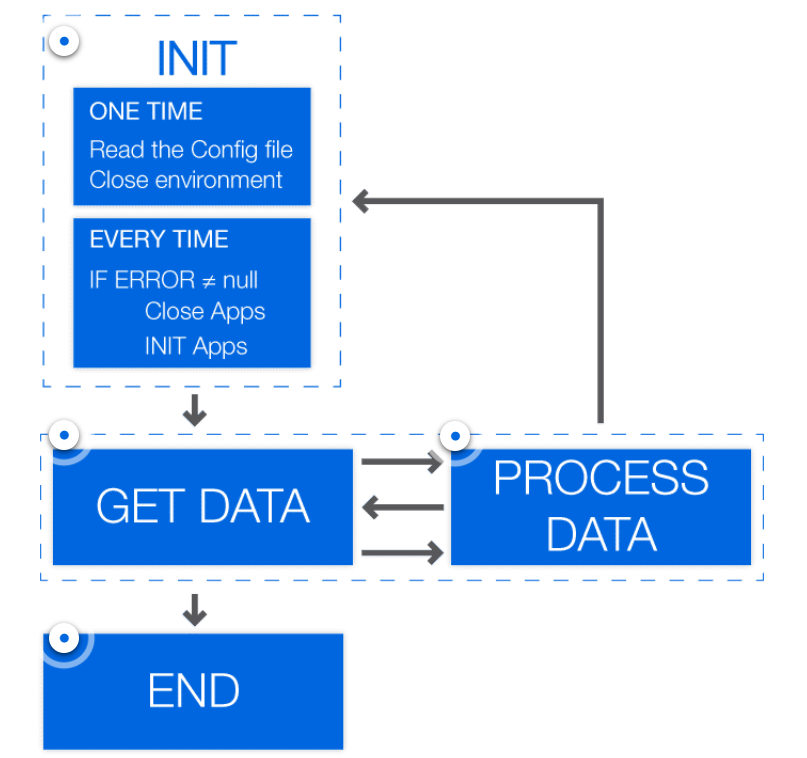
**End Process State**

The process ends (and the applications opened during the automation should be gracefully closed).

To have a better understanding of how we can use REFramework, let's go through the following scenario:

There is a list of people and their email addresses in a spreadsheet. An email needs to be sent to each of them with a personalized message based on a template.

Select each marker below to see how the process can be implemented using the 4 states:



REFramework Features

**Settings**

In many processes, it is common to have certain settings and configuration values that are read during the initialization phase. Examples of settings include URLs to access web applications, Orchestrator queue names and default logging messages.

The REFramework keeps track of such data by reading them from a configuration file (Config.xlsx) and storing them in a Dictionary object (Config) that is shared among the different states. This offers an easy way to maintain projects by changing values in the configuration file, instead of modifying workflows directly.

**Logging**

Another powerful feature of the REFramework is the built-in logging mechanism. Most of the workflows that compose the framework use Log Message activities that output details of what is happening in each step of the execution.

This can be used not only to find problems and help in the debugging process, but also to create visualizations and reports about the execution of the process (for example, how many invoices are processed each day, how many failures happen and what are the main causes of failures) and about the process itself (for example, what is the aggregated value of all reports processed in a month).

**Business Exception & Application Exception**

During the execution of most processes, there can be situations that are not part of the normal execution flow and they need to be addressed to achieve a more robust automation.

As an example, consider a process that uses a few web applications, but at a certain point the web browser freezes. If an activity (for example, Click) tries to interact with a frozen application, it will probably fail and return an exception.The REFramework was designed to enable recovery from exceptions by either attempting to process the transaction again (i.e., retrying) or by skipping that transaction.

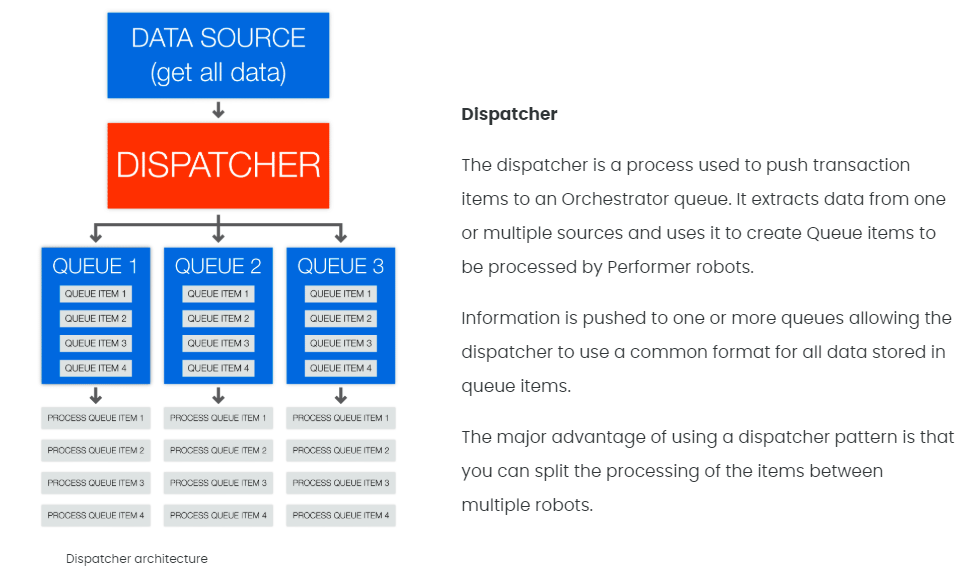
If the cause of the problem can be fixed by restarting applications, then the framework automatically does that and tries again to process the same transaction. These exceptions are called Application Exceptions.

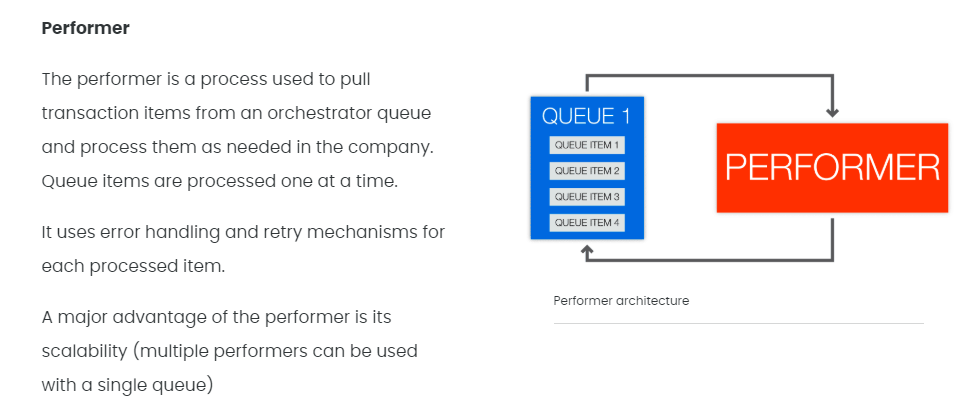
If the problem is related to the data itself or an underlying business requirement (for example, don't process invoices with value more than a certain amount), then that transaction is skipped and the framework proceeds to the next transaction. These exceptions are called Business Rule Exceptions.

Dispatcher and Performer

Although the REFramework can be used with different types of data sources, it provides a particularly strong integration with Orchestrator queues. When queues are used, it is possible to define priorities and deadlines for transaction items and to track retrying attempts of failed transactions.

The use of queues also enables an execution pattern called Dispatcher & Performer, which divides the process in two main phases: dispatching items to be processed and adding them to a queue, followed by retrieving an item from a queue and performing the process using that item. The second part of the process is generally built using the REFramework.





Dispatcher & performer model advantages:

* 1
* Better separation of processes (between dispatcher and peformer)
* 2
* Better separation & distinction between architecture and process layers
* 3
* Better error handling and retry mechanism
* 4
* Possibility to run processes across several machines (availability)
* 5
* Better re-usability within your project's created components
* 6
* Improved built-in configuration & Orchestrator integration
* 7
* Previous workflows created without REFramework can be easily adapted and deployed in order to use REFramework and the dispatcher/performer model

**Let's recap the Activities and Methods used**

**1. Dispatcher** - get entries from an Excel file and add them as queue items in **Orchestrator**

We started the project as a **sequence** and added a '**Try Catch**' activity. In the '**Try**' block:

* We used a '**Read Range**' to read the first 10 lines (A1:C11) from an Excel file and store them in a newly created **DataTable variable**;
* We used a '**For Each Row**' to loop through the rows of the **DataTable variable** and add each of them in the queue using the '**Add Queue Item**' activity.
* We used a '**Log Message**' activity to log the fact that the dispatching process was successful.

**2. Performer**- get items (transactions) from the queue and process them

a. In the **Init state**:

* **InitAllSettings** workflow invoked to initialize all settings and data for the applications and processes used. In this part, the settings, constants and assets in the **Config.xlsx** file from the Data folder are read;
* **InitAllApps** workflow invoked to open apps and check their status, credentials and so on;
* **KillAllProcesses** workflow invoked to leave a single instance of the applications used.

Any **system error** sends the process to the End Process state.

b. In the **Get Transaction Data state**:

* In a **Sequence**, use a '**Should Stop**' activity to verify if the **Orchestrator** sends a stop signal.
* If not, invoke the **'GetTransactionData**' workflow that uses a '**Get Queue Item**' activity to get the transaction information from the Orchestrator.

When there's no data to be read, it moves to the End Process state.

c. In the **Process Transaction state**:

* The process is a **Sequence** with a 'Try Catch' activity set to catch both system exceptions and business exceptions.
* Inside the '**Try**' block, the **ProcessTransaction workflow**is invoked that has a simple process to:
  + **Attach** to UiDemo application
  + 3 '**Type into**' activities to fill in the Cash in, On Us Check, Not On Us Check fields;
  + '**Click**' "Accept" button.

d. In the **End Process state**, the applications closed are used.

Doc - <https://docs.uipath.com/studio/docs/project-organization>

GitHub link – for RE-Framework - <https://github.com/UiPath/ReFrameWork>