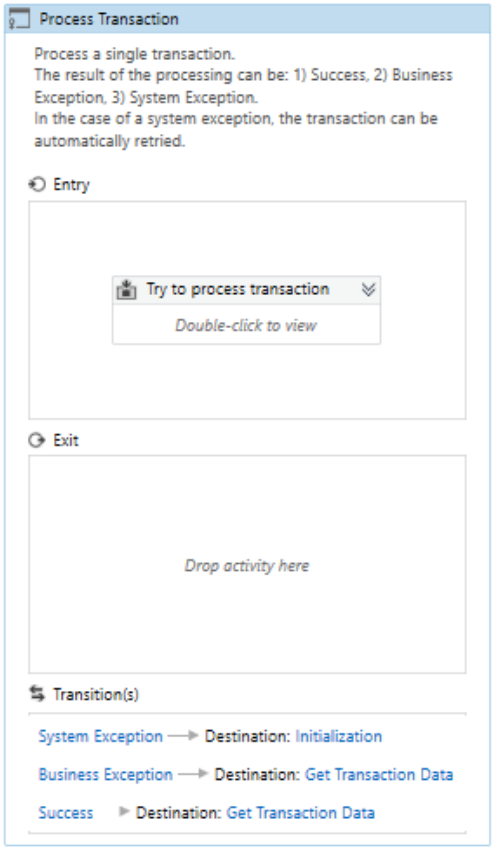
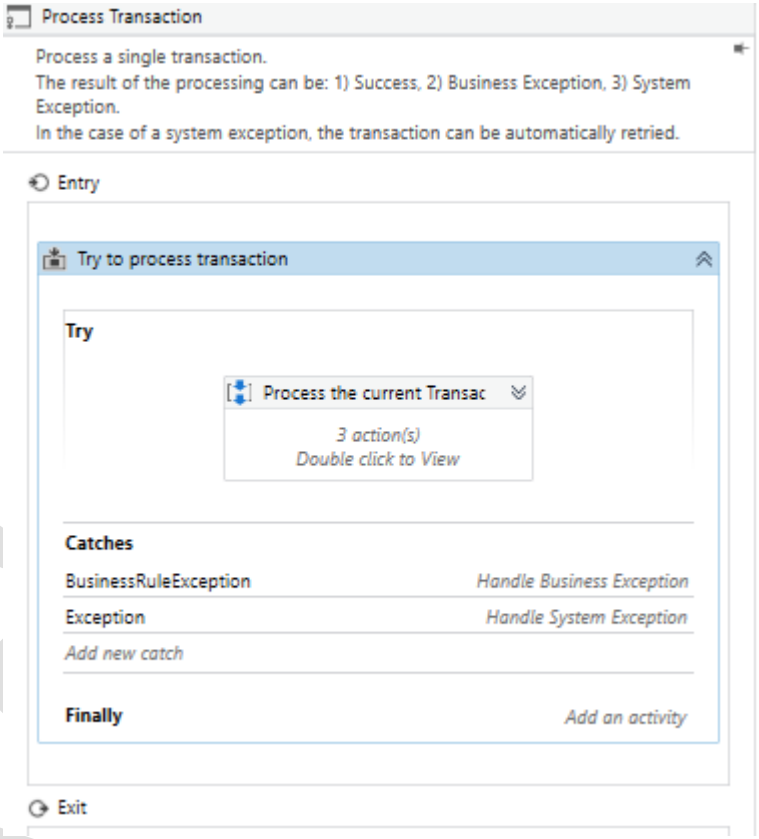
Dive into the Process Transaction and End Process states.

the Set Transaction Status and Retry mechanisms.

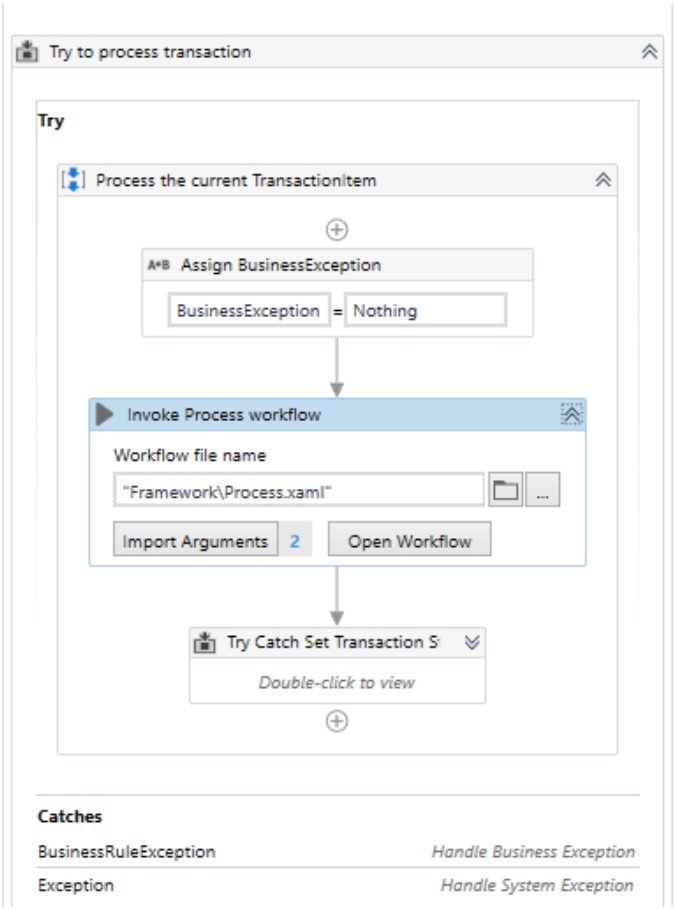
The Process Transaction state starts with a Try Catch activity.
Inside the Try block, we invoke the Process workflow file.

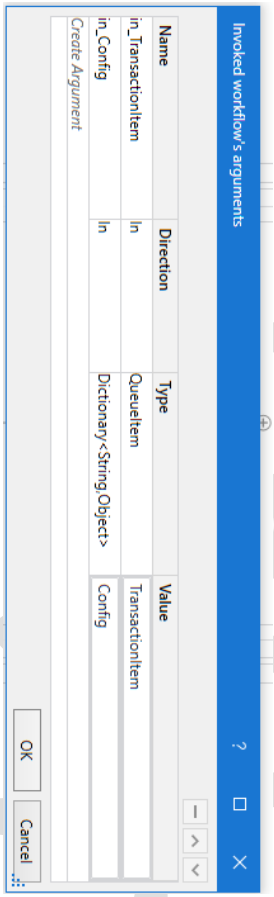




This is where we place the activities which do the actual processing of our data.
In the case of our process, where we type the data from the Transaction Item into the UiDemo application and click Accept.

This workflow takes two input arguments:
In_TransactionItem which holds the transaction data
In_Config which holds the configuration dictionary.



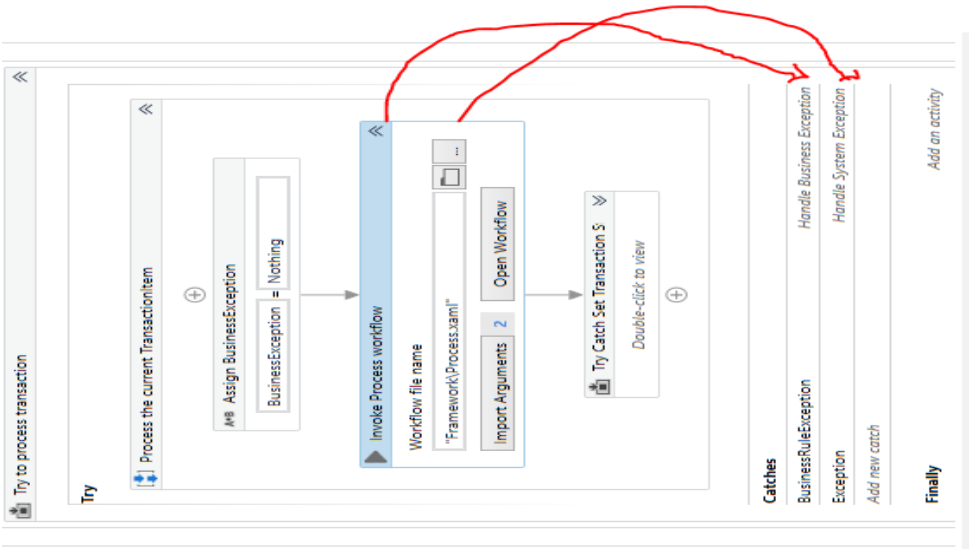


Based on the outcome of the execution of the Process workflow,

we can catch a Business Rule Exception and assign its object to the BusinessException variable.

Note

- Business Rule Exceptions can only be thrown by a Throw activity
- System Exception and assign its object to the SystemException variable.
- It can consist of any unhandled exceptions thrown in the process.



Of course, no catch means the Process workflow has been executed successfully.
The Robot selects the state the process transitions to, based on one of these three outcomes.

A System Exception will direct the process to the Initialization state while both a Business Rule Exception and a No Catch will direct it to the Get Transaction Data state.

A key component of the REFramework is invoked in the Finally block of this Try Catch activity:

The Set Transaction Status workflow.
This workflow sets the statuses for the queue items, taking into account the result of the Process workflow and the number of available retries.

The Set Transaction Status workflow uses quite a few arguments:

- the Config, the SystemError, and the BusinessRuleException arguments, which are used to determine the next state, as well as TransactionItem and TransactionID, as input arguments.
- We can find two In/Out arguments here: RetryNumber and TransactionNumber.
- They are needed as input, but they are also altered by the Set Transaction Status XAML file.
- Whether or not the robot retries a transaction, the Transaction Number or the Retry Number can change.

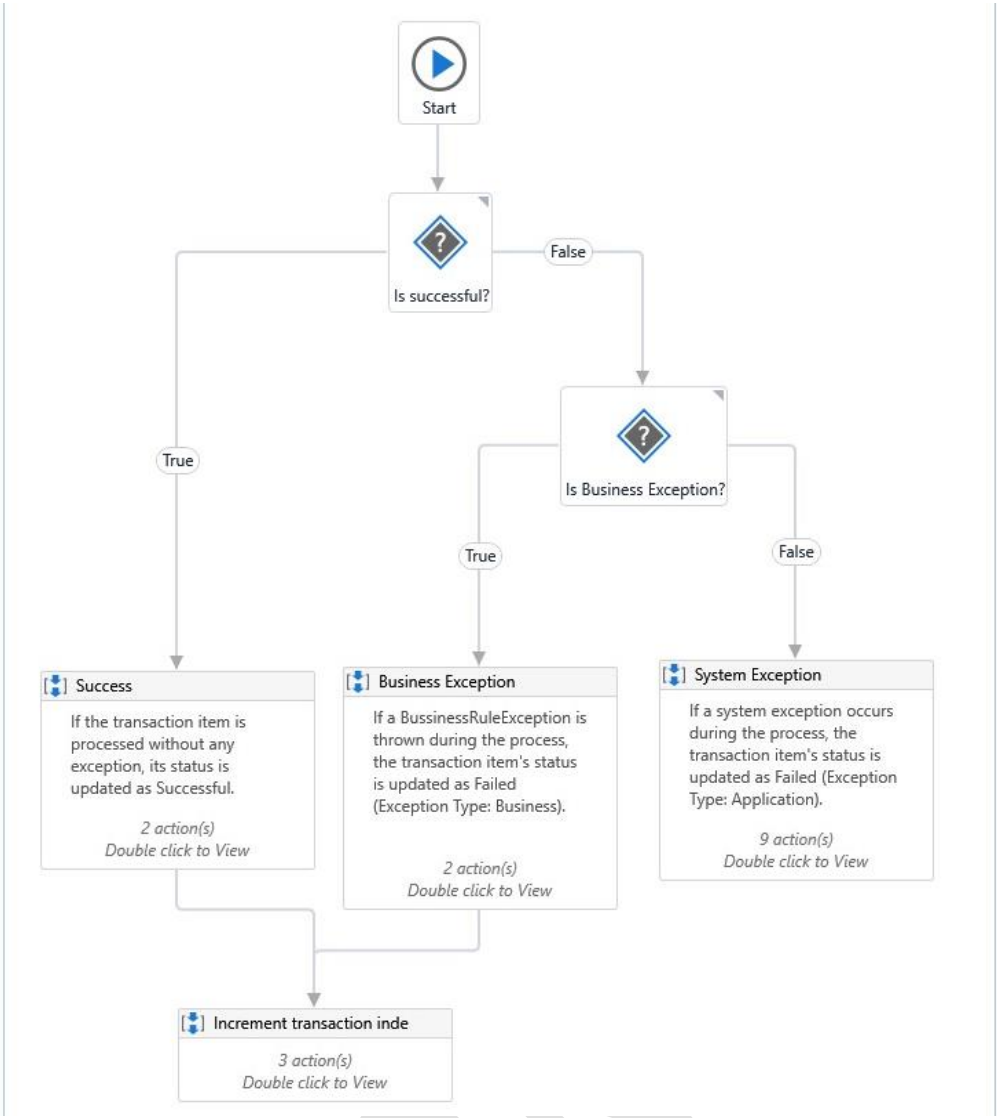
Invoked workflow's arguments

Name	Direction	Type	Value
in_BusinessException	In	BusinessRuleException	Nothing
in_TransactionField1	In	String	TransactionField1
in_TransactionField2	In	String	TransactionField2
in_TransactionID	In	String	TransactionID
in_SystemException	In	Exception	Nothing
in_Config	In	Dictionary<String,Object>	Config
in_TransactionItem	In	QueueItem	TransactionItem
io_RetryNumber	In/Out	Int32	RetryNumber
io_TransactionNumber	In/Out	Int32	TransactionNumber
io_ConsecutiveSystemException	In/Out	Int32	ConsecutiveSystemExceptions

Create Argument

INSIDE SET TRANSACTION WORFLOW:

It's a simple flowchart which executes one of the three sequences. If there is no exception, the Success sequence is executed.


Start
Is successful?
False
True
Is Business Exception?
True
False
Success
If the transaction item is processed without any exception, its status is updated as Successful.
2 action(s)
Double click to View
Business Exception
If a BussinessRuleException is thrown during the process, the transaction item's status is updated as Failed (Exception Type: Business).
2 action(s)
Double click to View
System Exception
If a system exception occurs during the process, the transaction item's status is updated as Failed (Exception Type: Application).
9 action(s)
Double click to View
Increment transaction inde
3 action(s)
Double click to View

Here, we use a Set Transaction Status activity to change the status of the Queue Item to "Successful".

When we use a different Transaction Item type, we need to make some changes to this workflow, as out of the box, the framework is built to work with queue items.

Next, we add a few log fields, the most important of which being the transaction status, number, and ID, and … log the success message.

The additional log fields are only used for this log message, ... hence, we remove them after we log the message by using a Remove Log Fields activity.

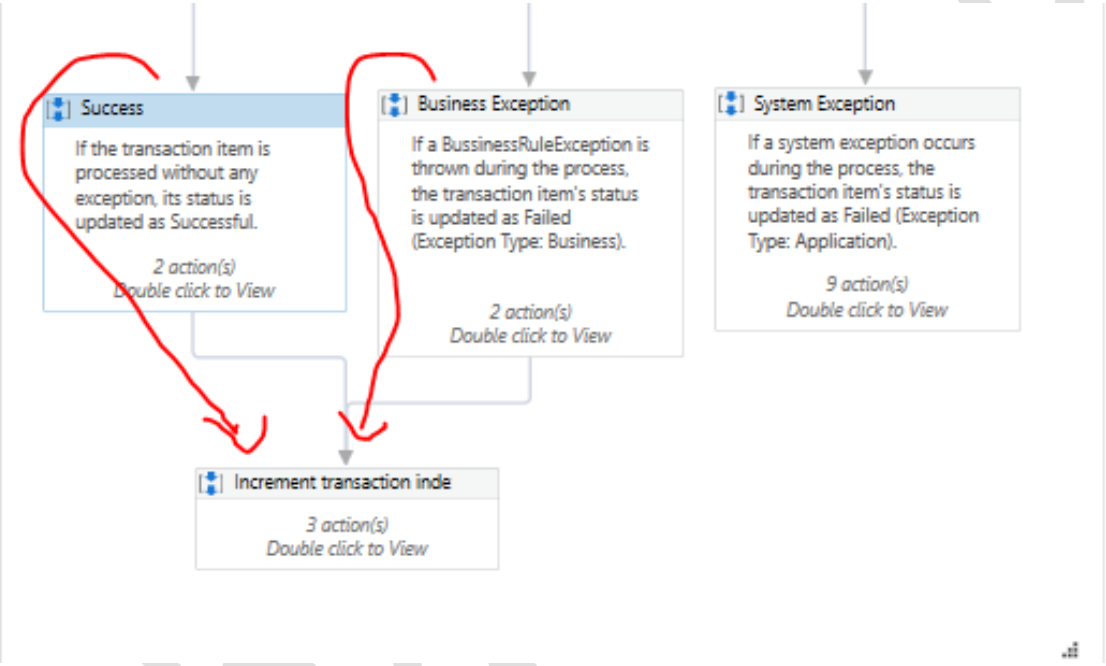
We do this to add more information to the unique log at the end of a transaction.

OK, this covers the activities used to set the Successful status.

1. Going back, if the Robot encounters a Business Exception while processing the transaction,
2. It sets the status to “Failed”, with the error type “Business”, and the reason set to the message of the caught Business Exception.
3. The error message is stored in the Message property of the in_BusinessException argument.
4. Of course, as in the case of the Successful status, we add the log fields, log the message, then remove the log fields.
5. OK, we’re done with the Business Exception status as well.

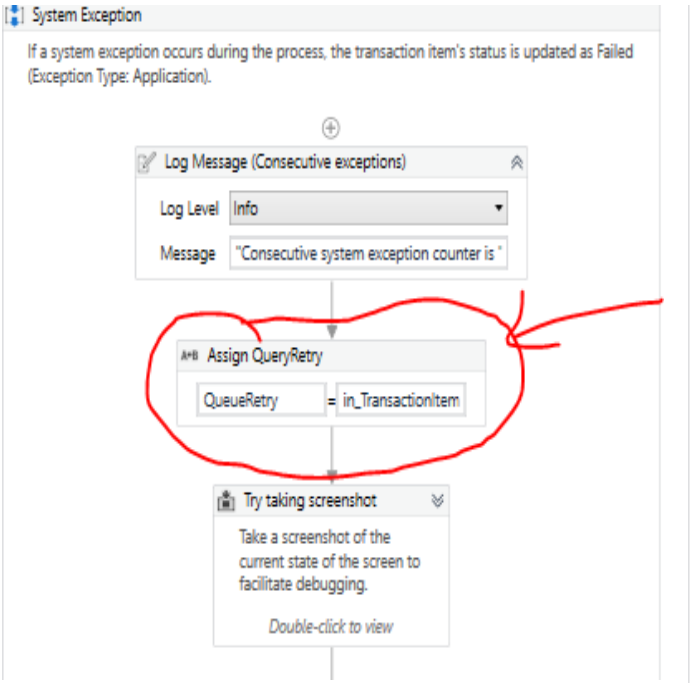
It’s important to note that for both the items which were successfully processed and the ones that failed with a Business Exception, the following step is to move to the next transaction.

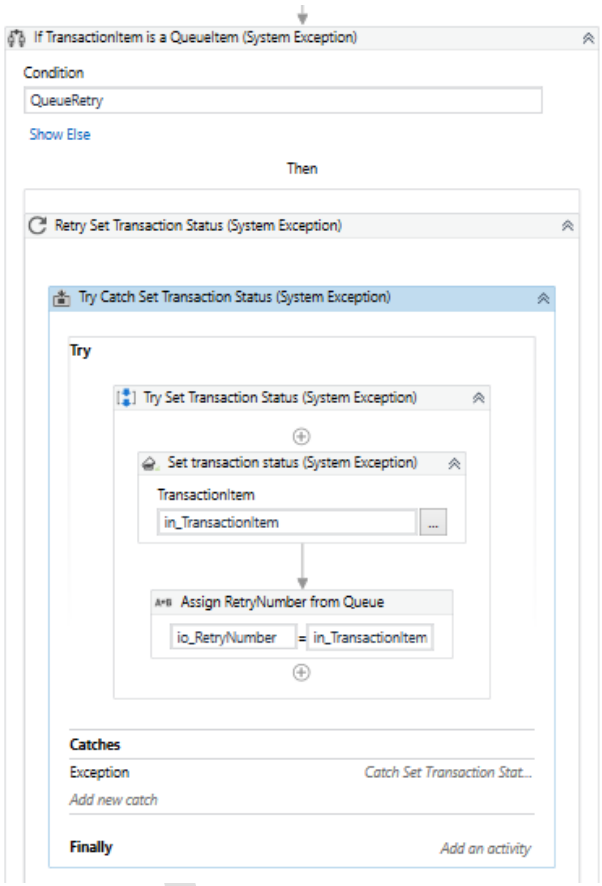
Before we do that, we need to increment the transaction number stored in the in/out Transaction Number argument and reset the retry counter stored in the in/out RetryNumber argument to zero.



what happens when a transaction fails with a System Error. This case requires that the Robot takes more actions as it may need to retry the transaction.

- The first thing we do in this sequence is to assign the value True to the QueueRetry flag if in_TransactionItem is not nothing and the type of the Transaction Item is QueueItem.
- The Robot uses the QueueRetry flag to determine whether it should execute the Orchestrator retry mechanism or the one based on the Config file.
- Next, we set the QueueItem status to Failed … with the error type Application, and the message stored in the property of the in_SystemException argument as the reason.
- Next, we add the additional log fields, … after which, we invoke the RetryCurrentTransaction workflow file.





Retry Mechanism:

- the framework uses the retry mechanism in the Orchestrator queue, while the MaxRetryNumber in the Config file is zero.
- If the retry option is not enabled, after the first flow decision, Retry Transaction, the No branch is executed.
- The next activities are a Log message with the level Error and an Assign which increments the Transaction Number.
- This means the Robot does NOT retry this transaction, but instead just moves on to the next one.

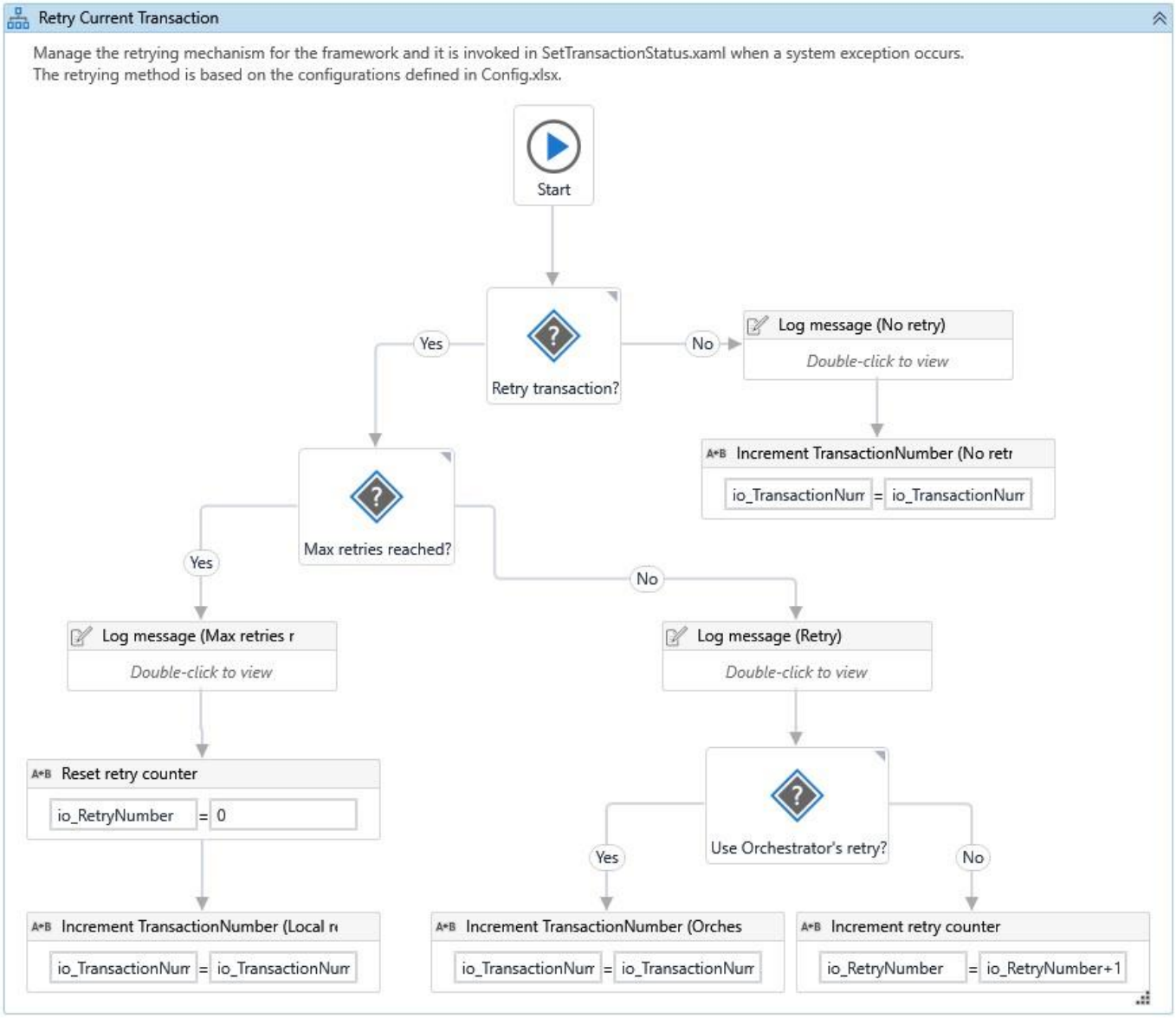
Now, if in our project, the Transaction Item is of a different type than Queue Item (meaning we're not using an Orchestrator queue),

we can enable the retry mechanism simply by setting the **MaxRetryNumber** in the Config file to a value **greater than zero.**

- In that case, the Yes branch of the Retry decision is executed, and next, the Robot checks if the MaxRetryNumber has been reached.
- We don't want to retry a failed transaction forever,
- if the current retry number reaches the maximum value,

- the Robot logs an error message once again, … resets the Retry Number to zero, and increments the Transaction Number.
- But, if the current transaction has been retried fewer times than the maximum retry number indicates, the Robot simply logs a warning.

In the case of a type of transaction item other than a queue item,
the Robot increments the current retry number and that's it: it doesn't increment the transaction number value.
If we use Queue Items, the robot increments the Transaction number.



- Now that we've explored the Retry mechanism, let's go back to the System Exception sequence.
- Next, we remove the added log fields and take a screenshot which is useful for debugging.

- Finally in this sequence, we close all applications before executing the Initialization state and starting them again.
- If an exception occurs while trying to close the applications, we force-close the applications by invoking the Kill all processes workflow in the Catches block.

After restarting all applications, for non-queue item transactions, the same transaction is retrieved in the Get Transaction Data state because the Transaction Number has the same value if retries are still available. If a Queue Item variable is processed, Orchestrator deals with retries, so it increments the Transaction Number.

And with this, we are done with the Set Transaction Status workflow and the Process Transaction state.

Let's navigate back to main.
After we process all transactions, we get no more data, and the process goes into the End Process State.
Here we can find a Try Catch like the one used for System Exceptions. In the Try Block, it will attempt to close all applications.
Should an exception be thrown, the process invokes the KillAllProcesses workflow.

Here are a few reflection points after this video.

1. **Do you think you can explain the key functions of the Process Transaction state?**
2. **What workflows are invoked in this state?**
3. **What are the three paths the Set Transaction Status workflow can take?**
4. **How does the Retry mechanism work with Queue Items?**
5. **How does it work with a different Transaction Item type?**
6. **What workflows are invoked in the End Process state?**