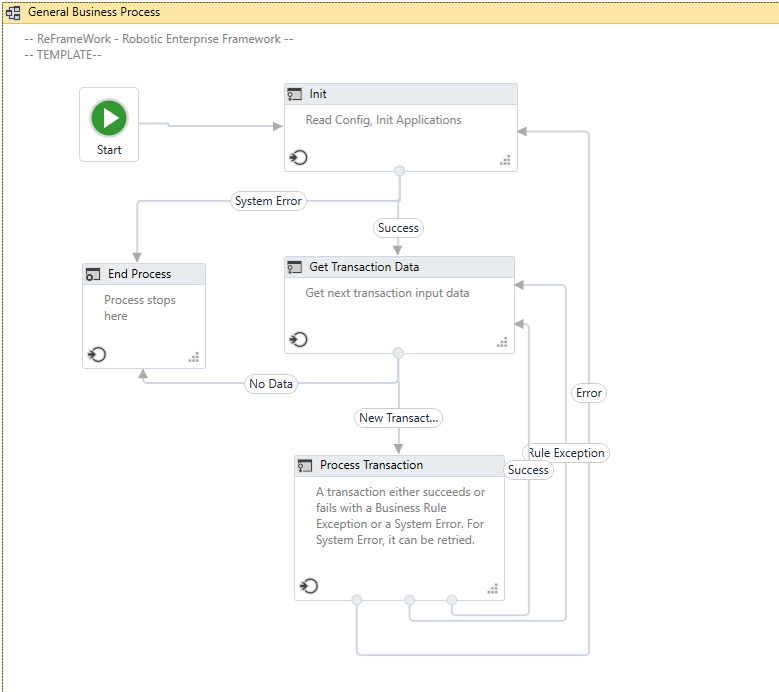
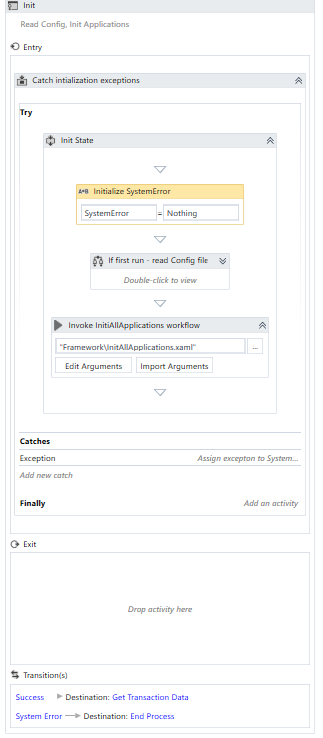
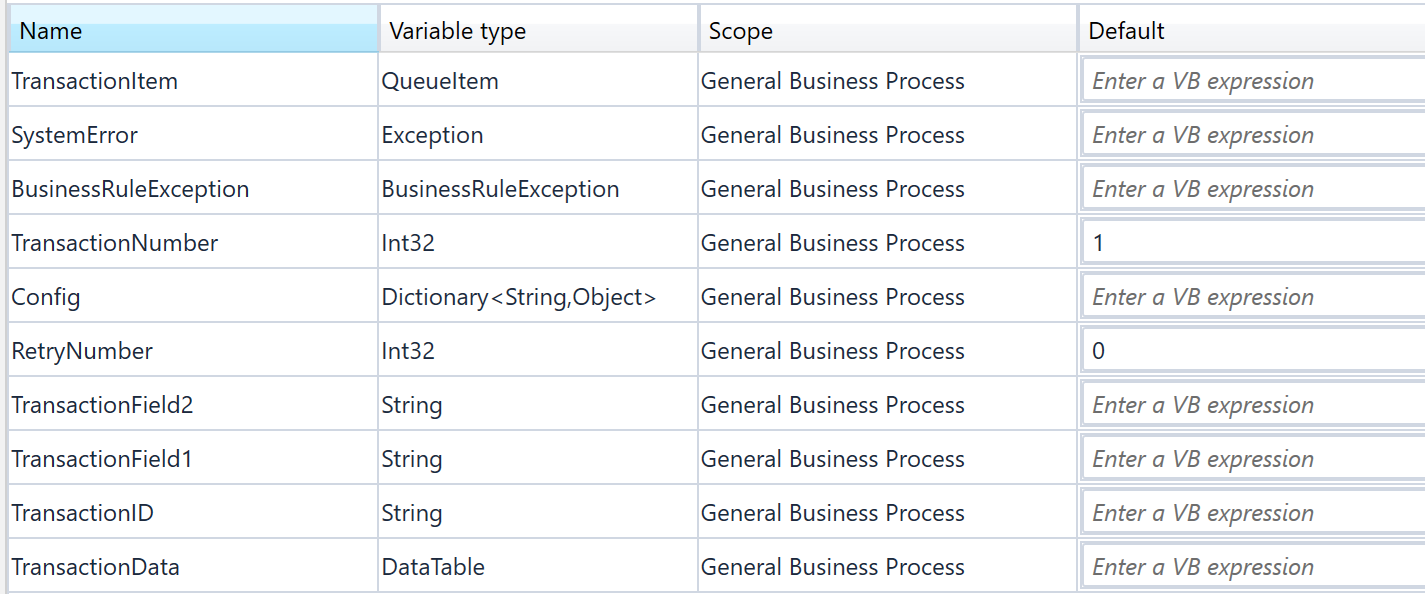


It has 4 States : Init, Get Transaction Data, Process Transaction, End Process.

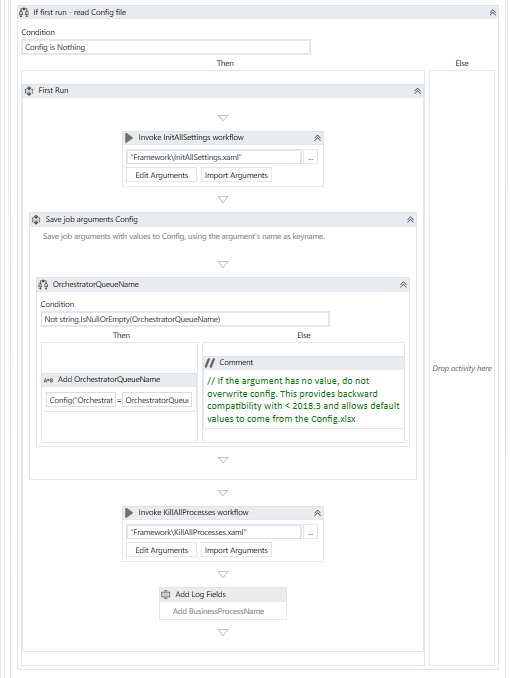


**Init :** Robot read the configuration and initialize the application.



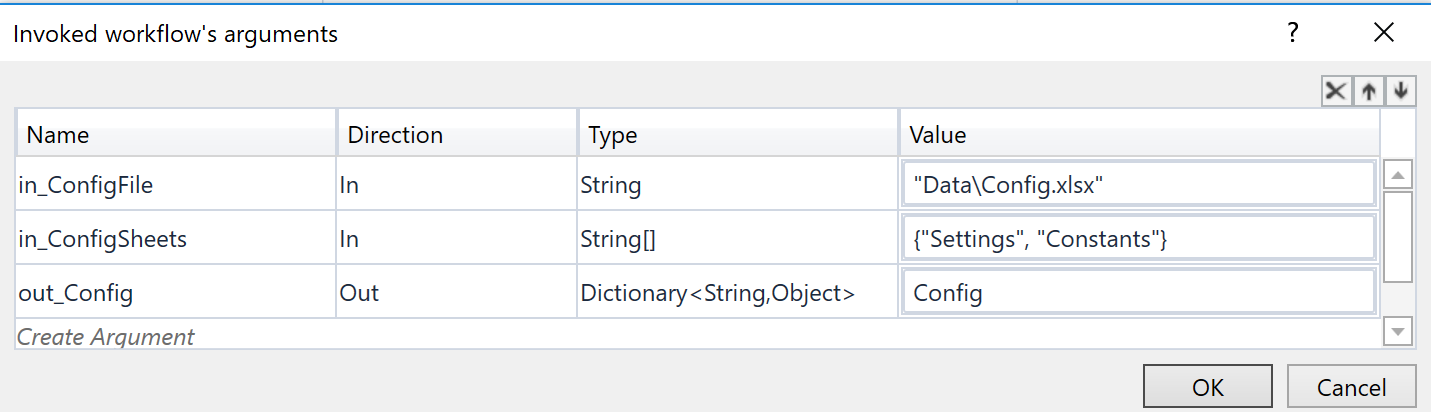


1. Create a sequence.
2. Take an Assign activity: System Error, Type Exception.
3. If First – Run Config File.
   * Take an If block and check for config is nothing
   * We run config file for the First run only
   * It is a dictionary of <String, Object>.
   * In the true part of If config is nothing, create a sequence.



In the Sequence to read the config file in the first run:

Invoke the workflow: InitAllSettings.xaml, this has the below three arguments:



The input arguments are the path of the config file located by default in the Data folder and config sheets, an array of Strings which indicates the sheets that contain configuration data.

The Output argument is our config dictionary to be used throughout the process.

The Default Data Sheet has 3 Tabs: Settings, Constants, Assets.

* Settings – Any configuration related to business process, URLs, filepath, credentials name.

Name – Always contains string, which is key in the dictionary.

Value – Holds the dictionary Value.

Description – Detailed Description.

* Constants- Store technical settings useful for the developers.
  + Contains information – No of Retry, TimeOut delay , Image accuracy and static log messages.
  + Useful in switching environments from Dev to test to Prod.
  + MaxRetryNumber- Used to retry the transaction that have failed with application exception or system error.
* Assets-

KillAllProcessesWorkFlow – To make sure robot starts in a clean and controlled environment.

**Invoke InitiAllApplications workflow** –

Starts all the applications.Worked as we pass arguments.

The Xaml file should be parameterized depending on what applications are used in the process.

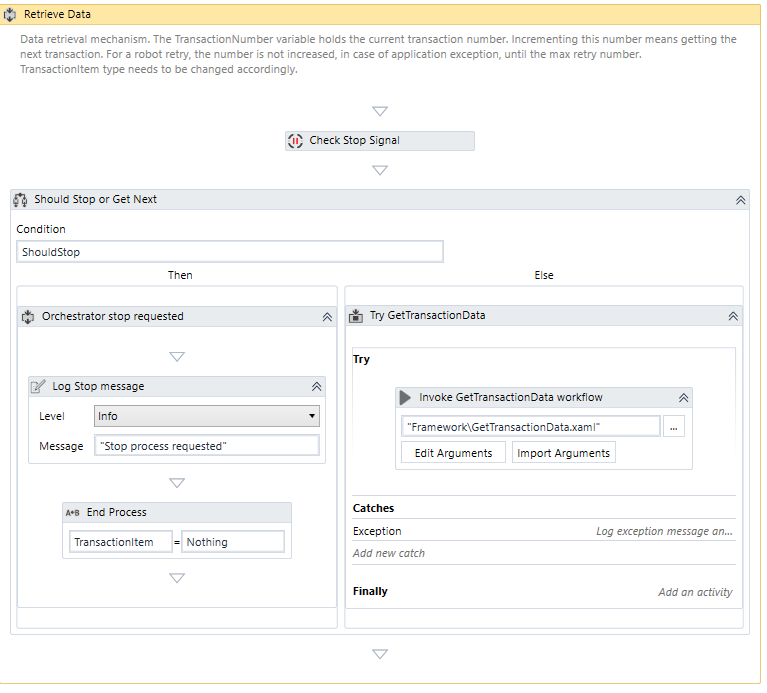
**Get Transaction Data state**

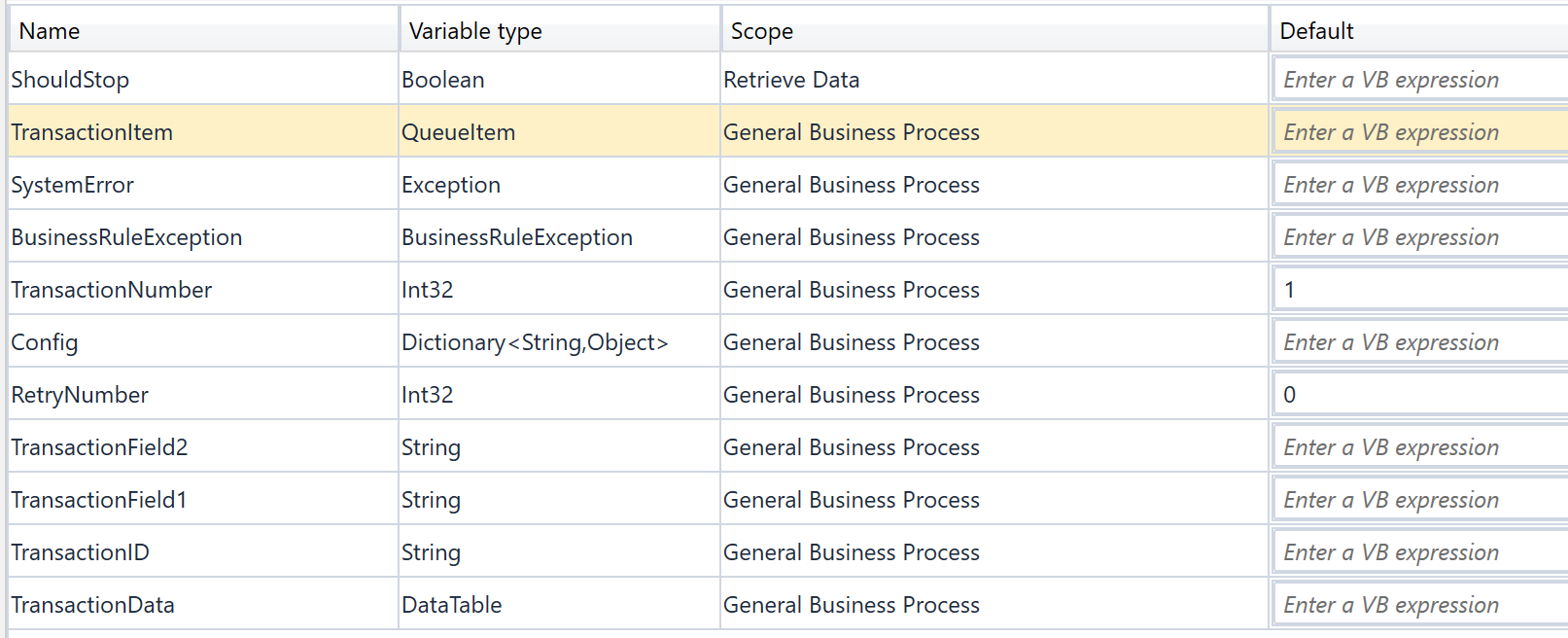
Data retrieval mechanism. The TransactionNumber variable holds the current transaction number. Incrementing this number means getting the next transaction. For a robot retry, the number is not increased, in case of application exception, until the max retry number.

TransactionItem type needs to be changed accordingly.

* We define a transaction as a repetitive part of the process.
* With each execution, the transactions are numbered.
* Transactionnumber: The attained value is stored in the transaction number default value =1
* TransactionItem: Queue Item

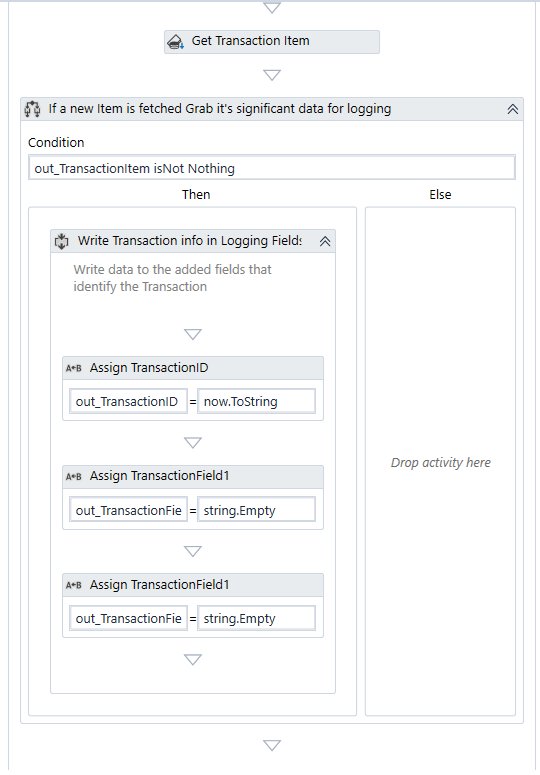
1. We check if a Stop signal was sent from orchestrator, If so we soft stop the robot by going to the end process state.



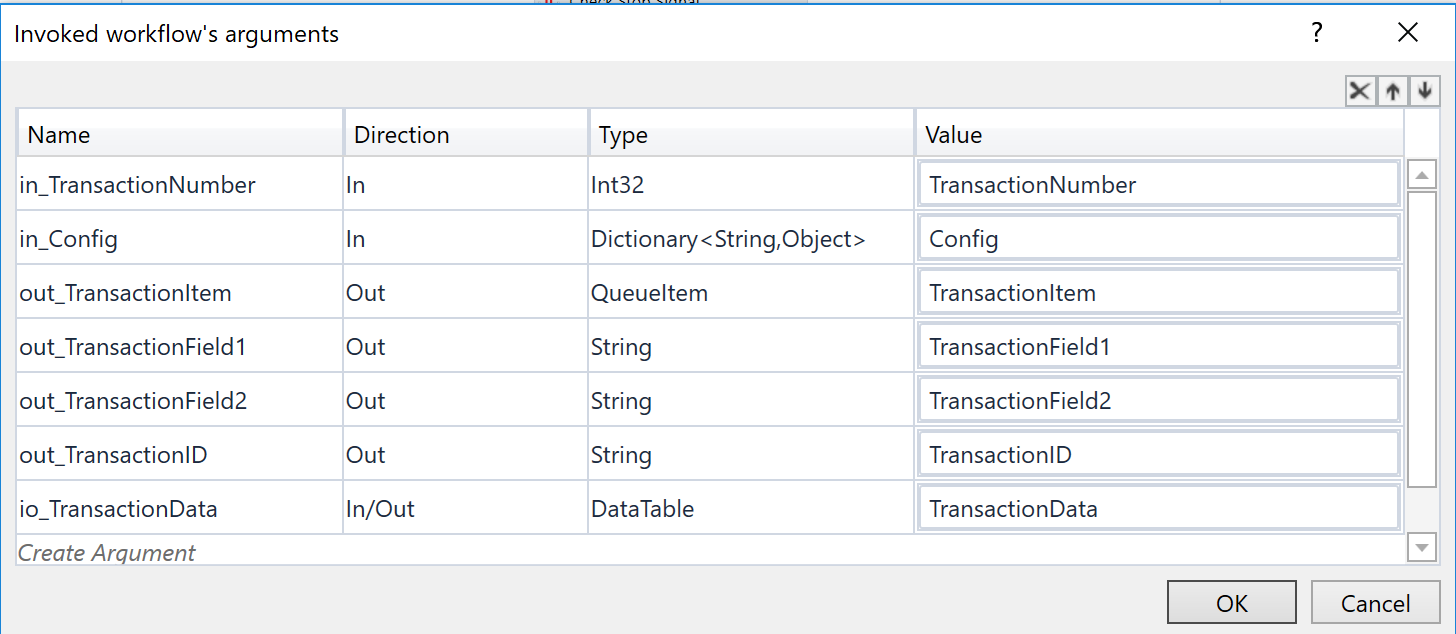


1. If there is NO Stop signal from Orchestrator, we invoke the getTransactionData WorkFlow.

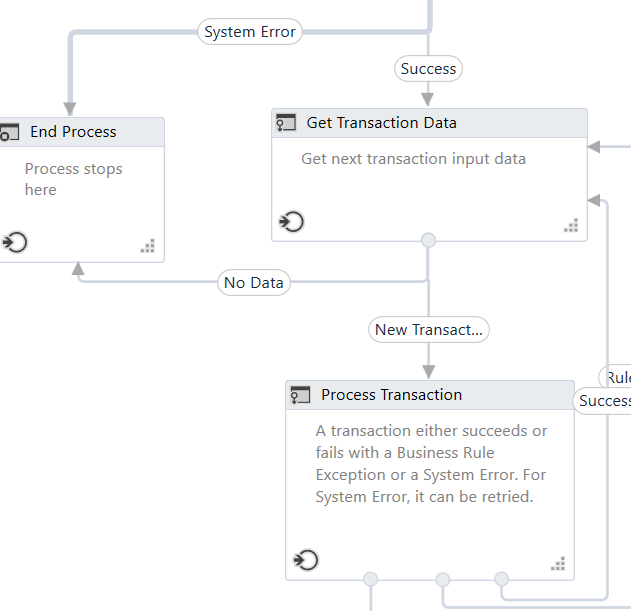
Below is what getTransactionData WorkFlow. Looks like



Arguments of getTransactionData WorkFlow.

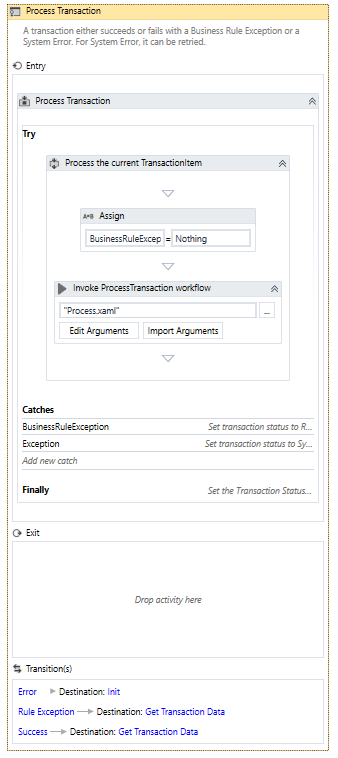


* We pass 2 input arguments : transactionnumber and config dictionary and get transactionitem as output.
* Opening the workflow we can see first activity is Get Transaction Item which outputs a queue item.
* The returned transaction Id is set to current Time stamp only in case when we do nat have a unique transaction id else use the ticket number or receipt number.



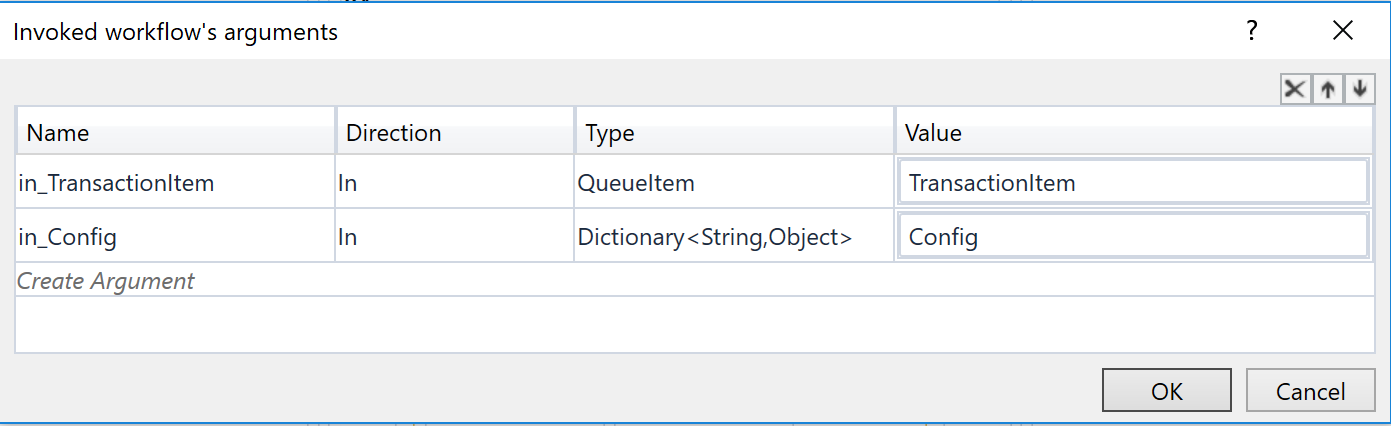
If TransactionItem is Nothing, that means we have **No data** to process. End Process will be called and closeallApplications.xaml is invoked.

Else , we Move to Process Transaction

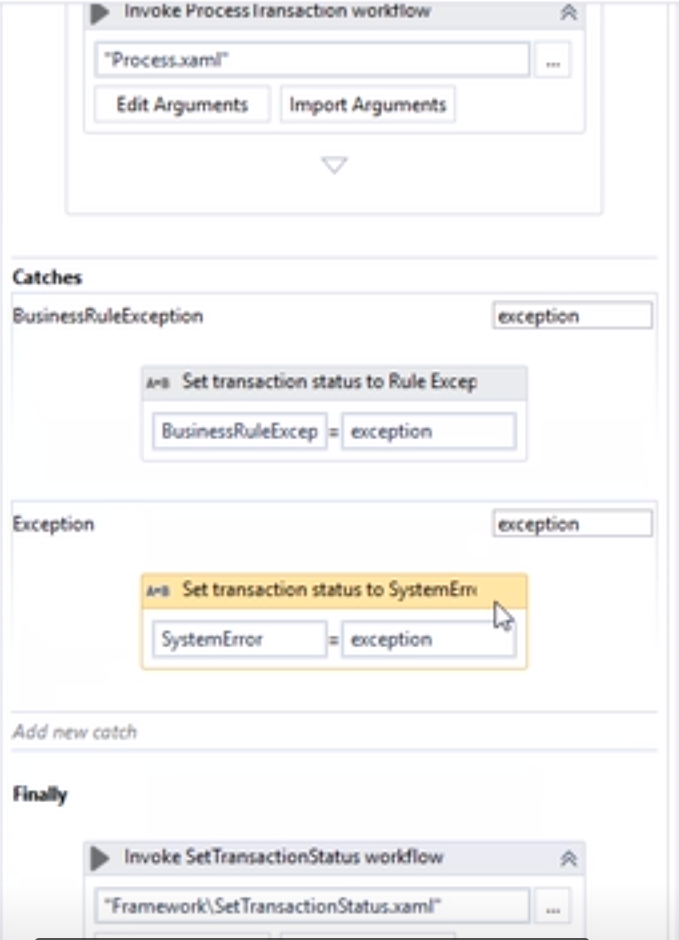


We invoke Process.Xaml File

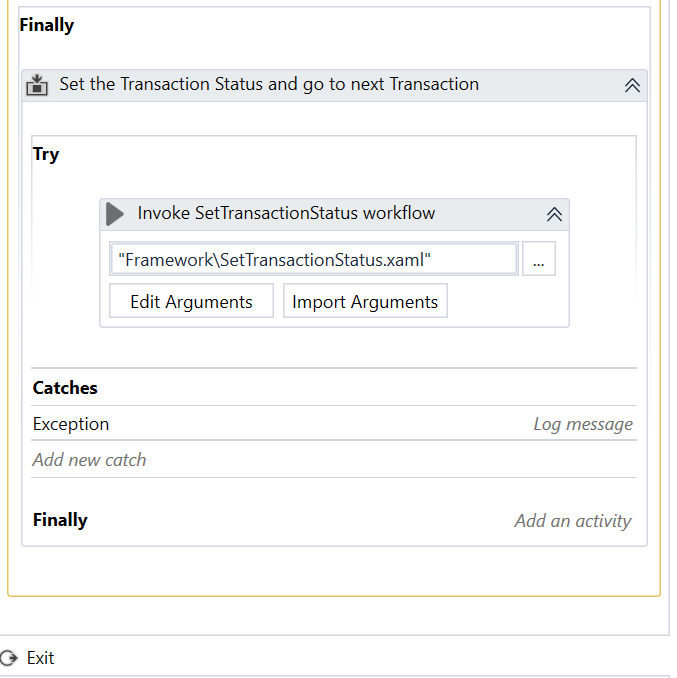
It has 2 input arguments:

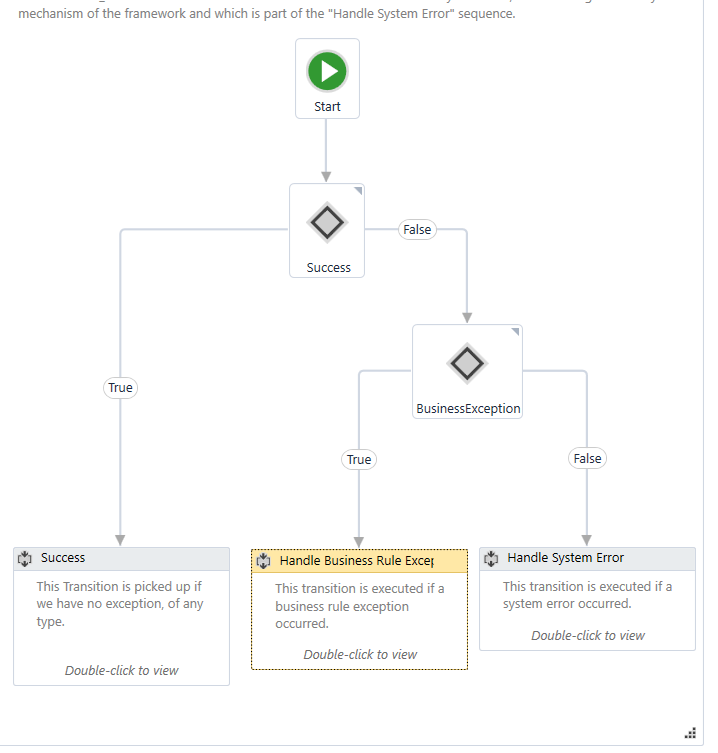


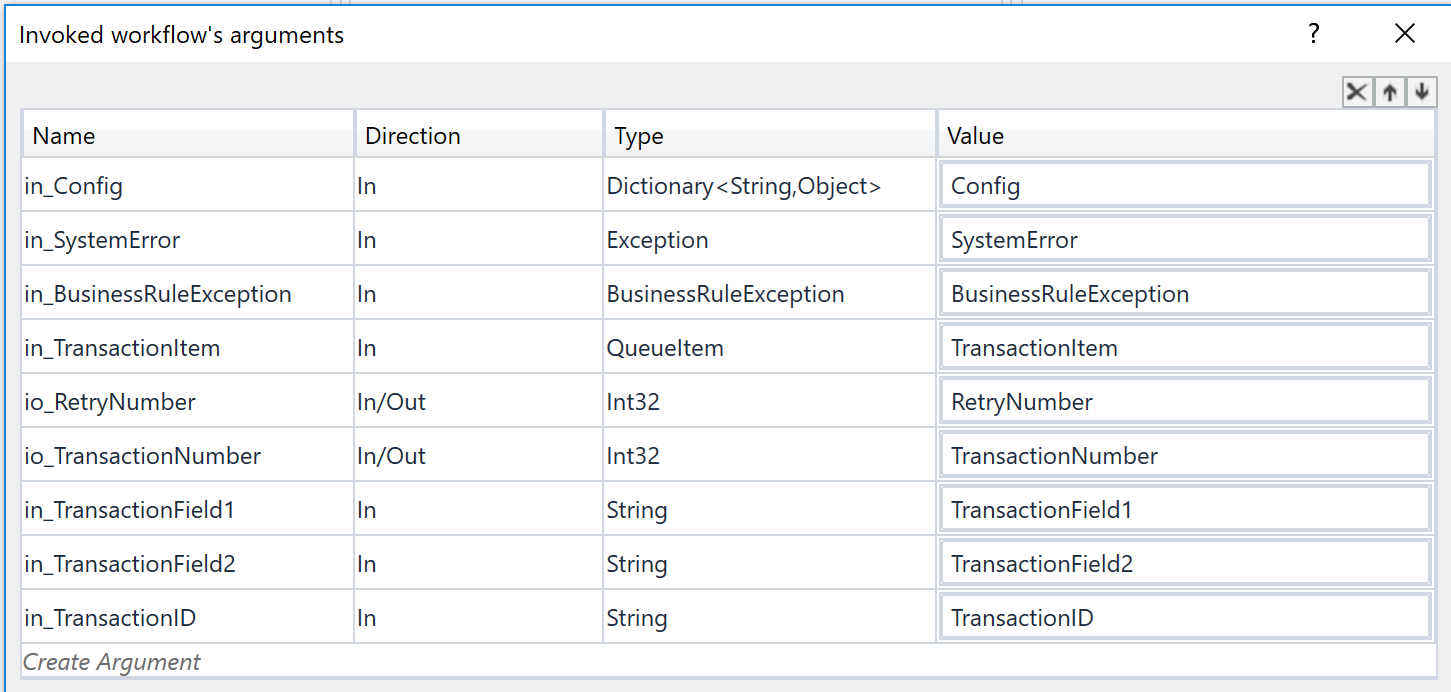
Based on the outcome of the execution of the process, we can catch a BusinessRuleException, which is always thrown by a throw activity, a system exception which can consists of any unhandled exception thrown in the process or No Catch which means the process has ended successfully. Based on one of these outcomes a new state is selected.



In the end we have a finally block which has Set TransactionStatus.Xaml

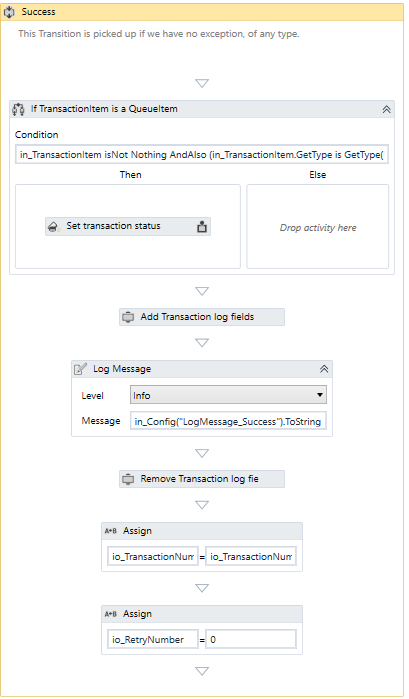






This is a simple workflow executing one of the 3 sequences

1. If there is NO exception Success sequence is executed.



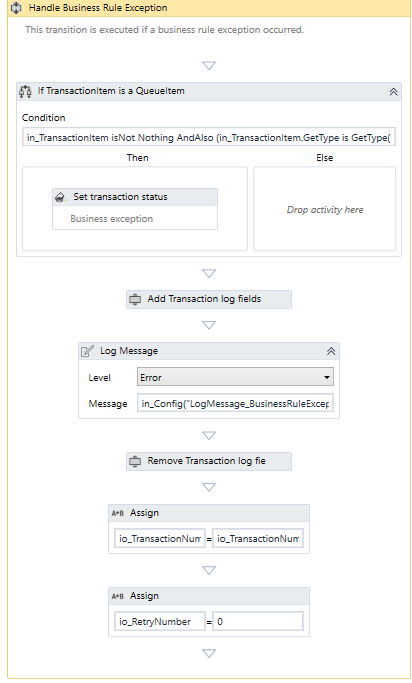
Set the status of the transaction to successful.

log some messages

Increment the transaction number by 1.

Set current Retry number to 0.

1. If we get a business exception, we have to handle it.

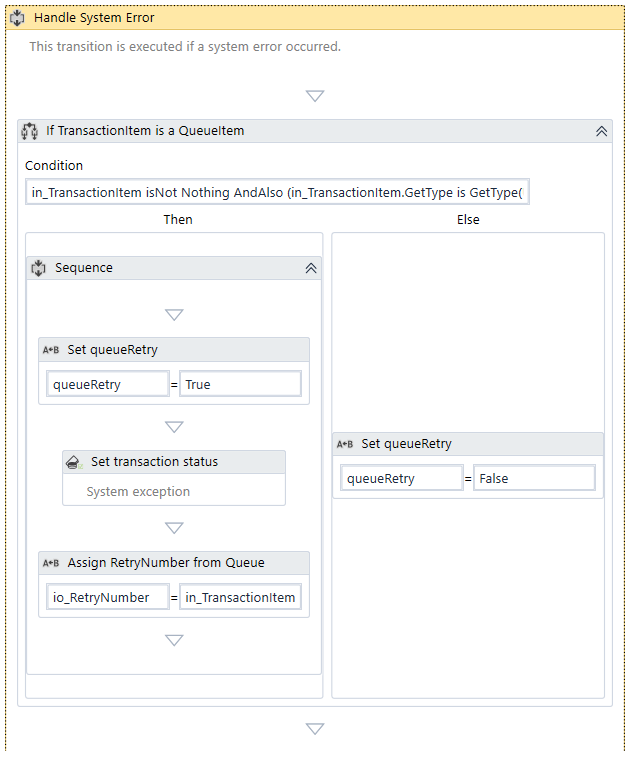


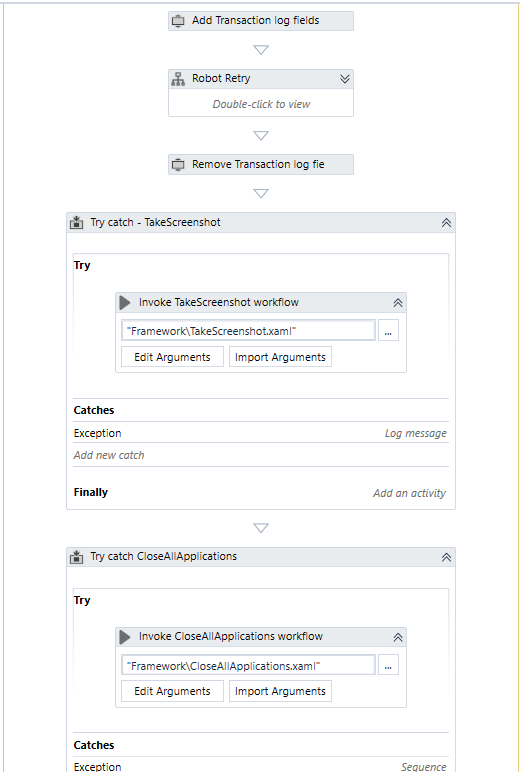
Set the status to fail

Error Type – Business

Reason – Exception message.

3.Handle System Error:

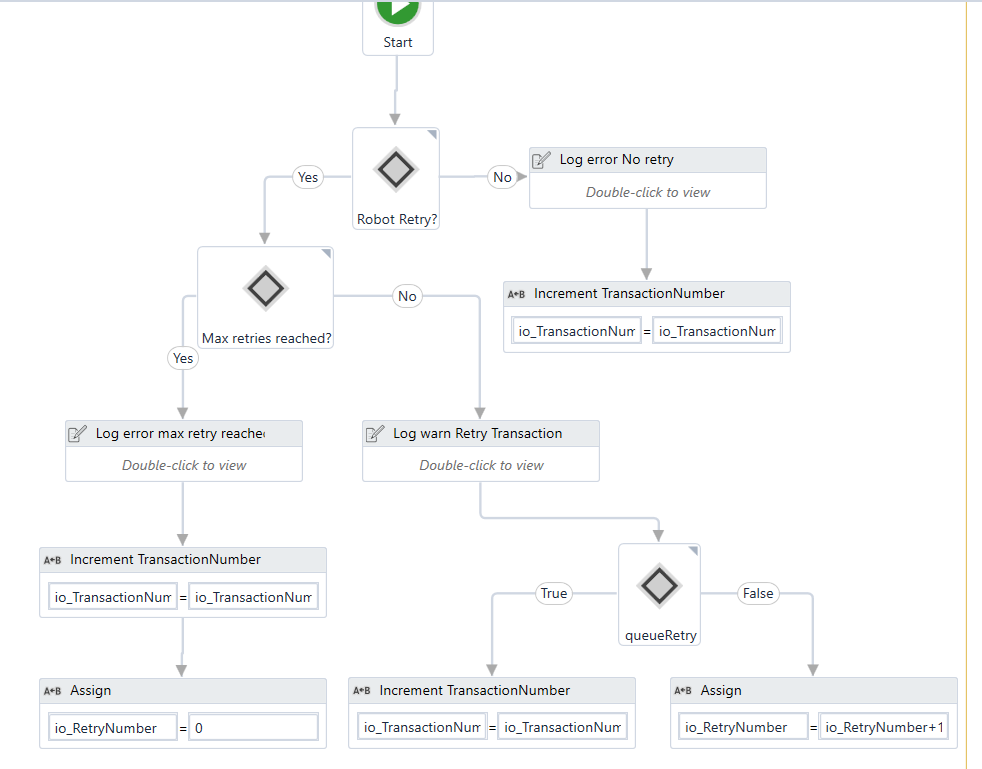




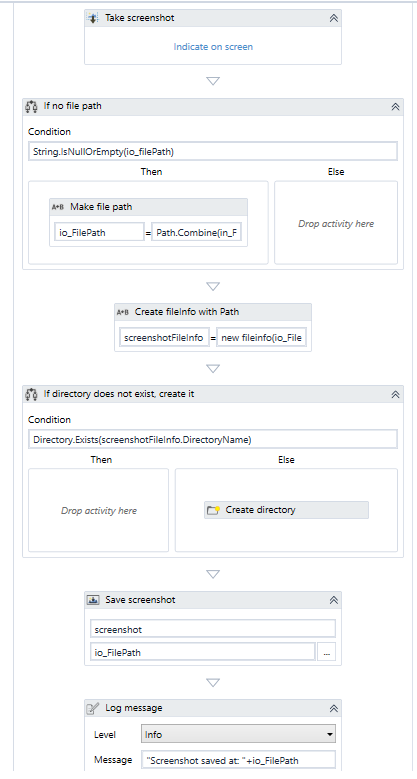
We have extra variable here: QueueRetryFlag – used to log a warning message if the queue item is being retried and an error, otherwise.

If we use queue item, the flag is set to true, but if we want to change the type of the transaction item to something else. We need to remove the entire sequence.

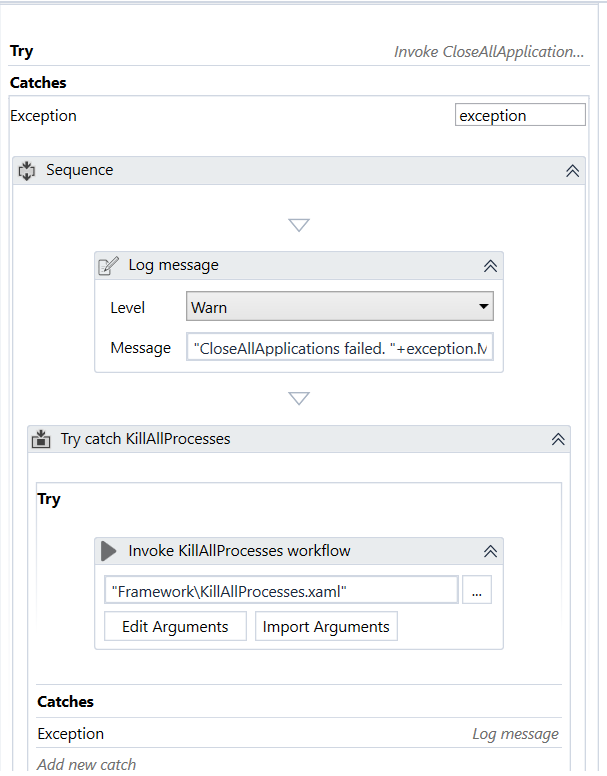
Then we have something called Robot Retry



In case of system exception, we invoke another reusable file Take screenshot.xaml



The last step in handling the system error is to try close all applications, if this Fails kill all processes to make sure that everything is closed.



**GetAllCredential.Xaml File**

Used to retrieve the credentials used by the robot to access different applications.

We use- **Get Orchestrator Credentials**

It gives us the secure username and the secure password.

Secure string is a dot net class to handle text that should be kept confidential.

If the retrieval of the credentials from orchestrator fails,we catch the exception and use GetSecure Credentail activity which attempts to get the credentials from the windows.

If this also fails, we dialog box appears for the user to enter credentials.