Get Transaction Data state

we will learn about the workflows, arguments, and variables used in this state and about something called the Stop mechanism.

Get Transaction Data.

This state retrieves one by one the transaction items containing the data to be processed.

For EX, this is where the Robot retrieves the Queue Items containing the Cash In, On Us Check and Not On Us Check values.

---

what is a transaction?

- We define a transaction as the repetitive part of our process. It 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.

TransactionNumber
In the REFramework, the transactions are numbered.
The value is stored in the TransactionNumber variable and incremented with each iteration.
The TransactionNumber variable is initialized with the value - one.

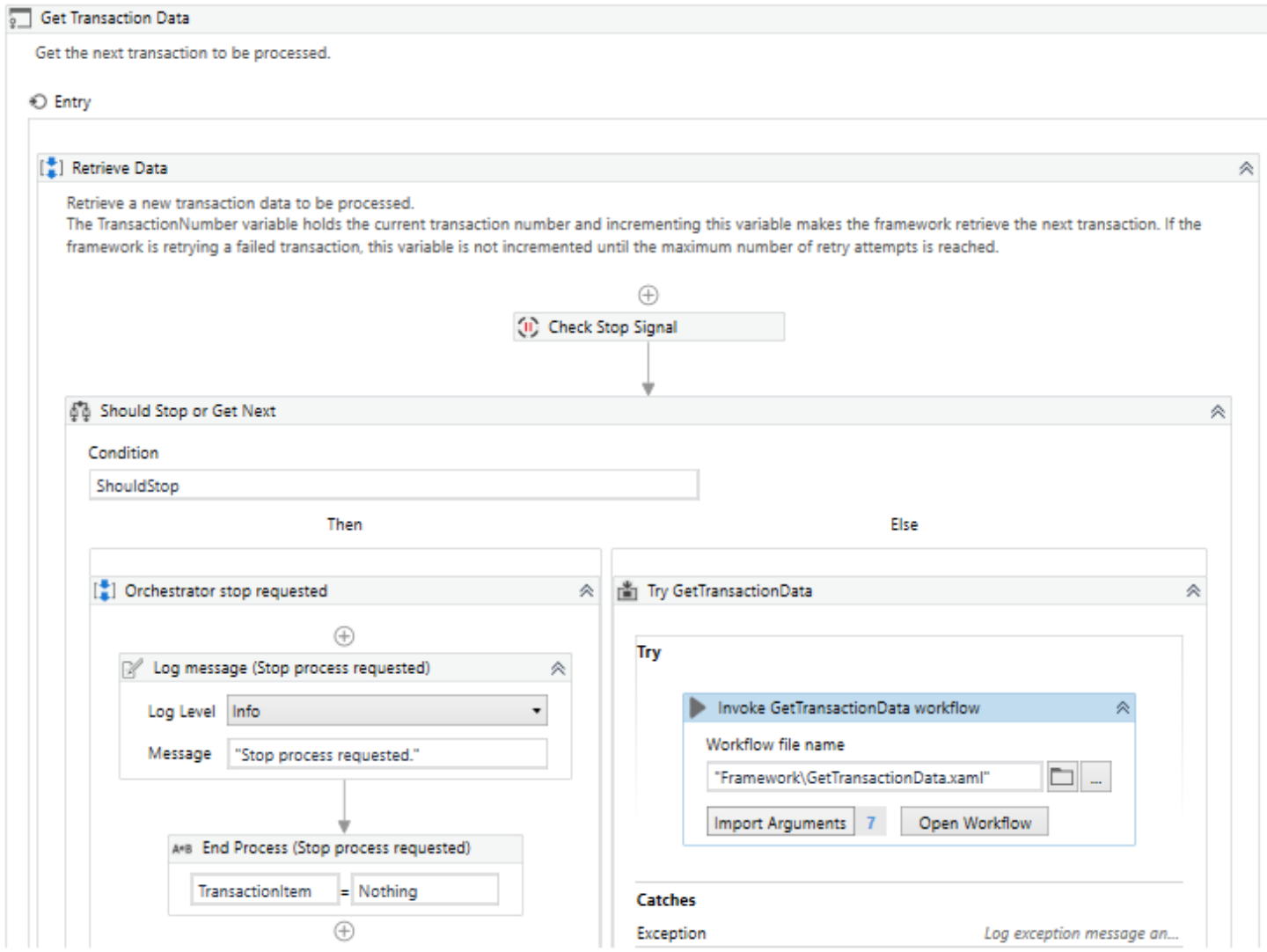
TransactionItem.
The next important variable is TransactionItem.
By default, it's of the QueueItem type, but the type can be changed if a different one is needed (for example, DataRow or MailMessage).

In our case, we need it as a queue item, as we uploaded the transactions from the Excel file to a Queue in Orchestrator.

When **Orchestrator queues are used**, the transaction data retrieval is handled by the **Get Transaction Item activity** included by default.
In this case, it is not necessary to make any modifications to the GetTransactionData.xaml workflow.

When the Robot gets a Queue Item from Orchestrator,

- the status of the Item changes to In Process and it does not get updated until we set it to Success or Failed by using a Set Transaction Status activity, after the item is processed.
- This is why, if we want to elegantly stop a process, we should do it after finishing processing the current item or just before getting a new queue item.
- So, the first thing we do in the Get Transaction Data state is check if a Stop Signal was sent from Orchestrator.
- The Should Stop activity gives us the option to soft stop our process execution, before starting a new transaction.
- If the stop command is received, the automation transitions to the End Process state.
- If no stop command is detected, we invoke the GetTransactionData workflow file.



Alright, let's have a look at the arguments used in this workflow.

- We have the following two IN arguments: TransactionNumber and the Config dictionary.
- We also have a few OUT arguments, the most important being TransactionItem.
- The others: TransactionID, TransactionField1, and TransactionField2 are used for logging purposes, as we will see in one of our next videos.

Invoked workflow's arguments

Name	Direction	Type	Value
in_TransactionNumber	In	Int32	TransactionNumber
in_Config	In	Dictionary<String,Object>	Config
out_TransactionItem	Out	QueueItem	TransactionItem
out_TransactionField1	Out	String	TransactionField1
out_TransactionField2	Out	String	TransactionField2
out_TransactionID	Out	String	TransactionID
io_dt_TransactionData	In/Out	DataTable	dt_TransactionData
Create Argument			

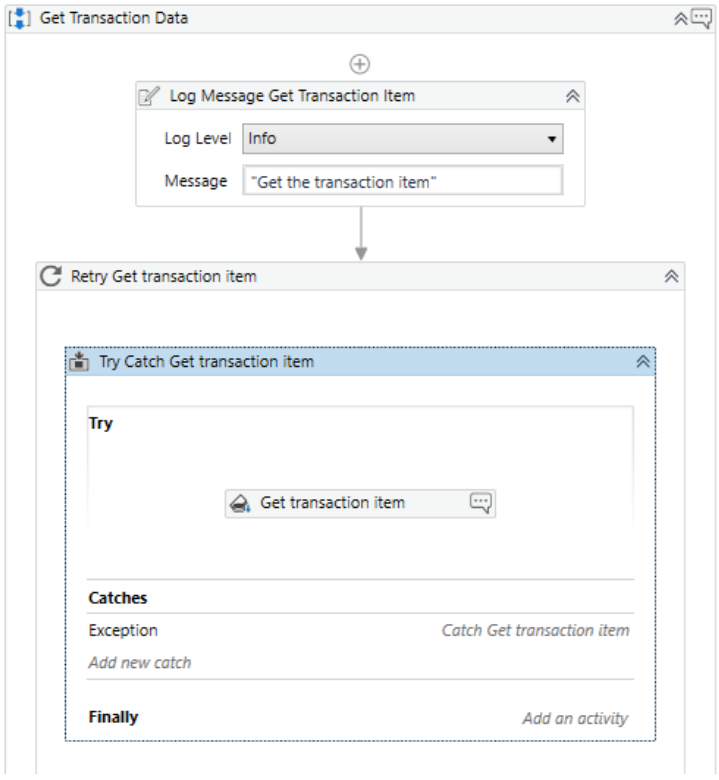
- TransactionData can be used as an alternative to store the collection of transaction items when Orchestrator Queues are not used.
- As seen here, by default the type is set to DataTable. Note that we are not using this argument in the current process.

KAHANI FROM ACTIVITY

- Get a transaction item from a specified source (e.g., Orchestrator queues, spreadsheets, databases, mailboxes or web APIs).
- If there are no transaction items remaining, out_TransactionItem is set to Nothing, which leads to the End Process state.
- For cases in which there is only a single transaction (i.e., a linear process), use an If activity to check whether the argument in_TransactionNumber has the value 1 (meaning it is the first and only transaction) and assign the transaction item to out_TransactionItem. For any other value of in_TransactionNumber, out_TransactionItem should be set to Nothing.

  If there are multiple transactions, use the argument in_TransactionNumber as an index to retrieve the correct transaction to be processed. If there are no more transactions left, it is necessary to set out_TransactionItem to Nothing, thus ending the process.

  Get a transaction item from the specified Orchestrator queue.
  If queues are not used in this process, replace this activity with the appropriated logic to retrieve transaction items.
  For example, if transactions are rows from a DataTable, the row corresponding to the current transaction is retrieved at this point.



Now, which activity would you expect to be executed first?

- Indeed, the first activity is Get Transaction Item, which outputs the queue item from the queue specified in the Config file.
- Remember earlier, how we mentioned three OUT arguments that are used for logging purposes?
- Here is where we assign values to them. Let's find out more about them.
- The first one, TransactionID should be unique for each transaction. By default, it is set to the current system timestamp.
- The second and third log fields will be used to log information about the processed transaction: Transaction Field 1 and Transaction Field 2.
- In our case, we set the first one to the concatenation of three values retrieved from the queue item: Cash In, On Us Check and Not On Us Check.
- For another example, if transaction items are invoices, then out_TransactionID can be the invoice number, out_TransactionField1 can be the invoice date,
- and out_TransactionField2 can be the invoice amount.

Now, at some point during process execution, all transactions will have been retrieved and processed, and the process needs to stop as no more transactions are available.
This occurs when TransactionItem becomes Nothing or Null.

When using an Orchestrator queue, that happens by default when there are no more new items to be sent. If we change the type of the TransactionItem argument, we need to Assign it to Nothing when there are no more transactions to be processed.

From this state, as seen here in the Main workflow, we have two possibilities.

One is that all transactions are processed, and no more data is available. In this case, the process goes to the End Process state and stops.
The other is that a new transaction item is retrieved so the robot executes the Process Transaction state next.

With this, we conclude our overview of the Get Transaction Data state. Here are a few things to check after having gone through this demo:

Can you explain what a transaction is?
What role does the Get Transaction Data State play in the framework?
Why and how do we use the Stop mechanism?
What workflows do we invoke in this state?
What arguments and variables do we use?