Practice 3 - Queues

You are tasked with automating the process of inserting transactions in the teller application "UiDemo". The transaction-related input data is stored in an Excel file. Due to the very large number of transactions, multiple robots are required to work at the same time. According to the Solution Architect (SA) and the Business Analyst (BA), the process will need to be allocated to as many as 5 robots.

The BA has identified one recurring Business Exception:

- Invalid input data - Some values are not numbers.

Your task:

- bullet

  Download UiDemo and the input Excel file below.

- bullet

  Build a Dispatcher (Producer) process that adds each row in the input file as a queue item to an Orchestrator queue.

- bullet

  Build a Performer (Consumer) process that gets each queue item and enters the data in UiDemo.

- bullet

  Include an exception handling mechanism to catch non-integer values.

- bullet

  Make sure the correct statuses are set on the transactions in Orchestrator.

- bullet
  Include a mechanism to terminate the process safely.

Prerequisites for downloads
Get the UiDemo application and the Transactions Excel file to get started.

UIDemo.zip
17.7 KB

UiDemo Login credentials:

- Username: admin
- Password: password

Transactions.xlsx
10.2 KB

Practice 8 - Solution

1. Create a queue in Orchestrator

1. Under the *Queues* section in an Orchestrator folder, click the **Add** button.
2. Click the **Name** field and enter a name, such as “Queue1”.
3. Make sure the **Auto Retry** option is checked and in the **Max # of retries** section, type 2.

2. Create the Dispatcher

1. Create a new sequence and give it a name - i.e. **AddItems**.
2. Read the excel file as a **DataTable** using a **Read Range**.
3. Add a **For Each Row** activity.
4. Drag and drop an **Add Queue Item** activity and place it in a **For Each Row** activity.
5. Select the QueueName field in the Properties panel and type the name of the queue created during the previous steps. Make sure you place the text between quotation marks.
6. The **ItemInformation** field is where the values of the transaction item can be added. This process works similarly to passing arguments to an invoked workflow. Press the **...** button. An **ItemInformation** window enabling you to create an argument is displayed.
7. Click **Create Argument**.
8. Name the first argument “CashIn”.
9. Set the **ArgumentType** to **String** and the **Direction** to **In.**
10. Fill in the **Value** field by entering the information on the first column of the Data Table - row("CashIn").ToString.
11. Do the same steps (7-10) for the second field and third field (Name them: OnUsCheck and NotOnUsCheck).
12. Run the workflow.
13. Check the values in Orchestrator

1. Locate the queue you created earlier.
2. Press the **More Actions** button and select **View Transactions**. The previously created transaction items are displayed.
3. Press the **View Details** button to see the values you entered in Studio.

3. Create the Performer

1. Create a new sequence in Studio. Name it “ProcessTransactions”.
2. Drag and drop a **Get Transaction Item** activity to the *Designer* panel.
3. Double-click the **Get Transaction Item** activity to see the corresponding *Properties* panel.
4. Fill in the **QueueName** field by entering the previously created queue.
5. Click the **TransactionItem** field and press **Ctrl + K** to create a new variable of type Transaction Item.
6. Type a name, such as "TransactionItem", and press **Enter**.
7. **Run** the project. Return to Orchestrator and notice what happened. The status of the first transaction item was changed from “New” to “In progress”. That happened because Orchestrator has not been instructed on the status of that item.
8. Go back to your open project in Studio. Add an **If** activity and use the following expression as the **condition**: TransactionItem IsNot Nothing. Place a **Terminate Workflow** activity in the Else block. This will ensure that the workflow ends when there are no items left.
9. Add a Try Catch activity in the Then block of the If activity. In the Try section, add 3 Assign activities.
10. Create 3 variables corresponding to the fields that will be filled in in the UIDemo app (CashIn, OnUsCheck, NotOnUsCheck). Place each variable on the left of each of the 3 Assign activities.
11. On the right side of the **Assign** activities, add the expressions that extract the corresponding data from the transaction items and cast it to Integer. For example, for the CashIn field, it will look like this: cint(TransactionItem.SpecificContent(“CashIn”)).
12. Now use these values to input them inside the **UiDemo.exe**. Open the application and Log In. (The credentials are provided in the beginning of the exercise). Keep the application open for the purpose of the exercise.
13. Use a **Type into** activity for each of these variables inside the Try section. After this three **Type Into**, use a **Click** activity for the **Accept** button inside the UiDemo.
14. To make sure we inform Orchestrator about this Queue Item, drag a **Set Transaction Status** activity to the Designer panel.
15. Double-click the **Set Transaction Status** to see its corresponding *Properties* panel.
16. Click the **TransactionItem** field to enter the **TransactionItem** variable, which was created during the previous steps.

17. Ensure the Status field is set to “Successful”.
18. In the Catches section, add the invalid cast exception as 'System.InvalidCastException'. Then add a Set Transaction Status, fill in the transaction item variable and set the status to 'Failed' with an explanatory message - e.g. "Conversion from String to type 'Integer' is not valid."

4. Terminating and stopping the process

Each running process in Orchestrator has a Running state associated with it. The user has the option to stop or kill the process. Kill will halt the process immediately, which may cause unwanted errors.

Stop will do it in a safe way, by closing applications and logging out. But this requires the use of the Should Stop activity in our workflow. It informs whether the **Stop** button has been pressed in Orchestrator by the time the control flow reached the **Should Stop** activity. This is similar to a Save Game function in a computer game. Let's do that.

1. Go to the *Project* panel of the Performer and open the *Main* workflow. Drag and drop here a **Flowchart** activity.
2. Create a new variable of type Int32 called ‘counter’ with the default value set on ‘1’.
3. Add a **Log Message** with the message “Processing Transaction number "+counter.ToString
4. From the Project panel drag and drop inside the **Flowchart** the *ProcessTransactions.xaml* workflow and connect it with **Start** point of the Flowchart.
5. Use an **Assign** activity for increasing the value of the counter with 1 each time the process is run. Connect this activity with the previous one.
6. Drag and drop a **Should Stop** activity and connect it with the previous **Assign** activity used. In the *Properties* panel and inside the **Result** press **Ctrl+K** to create a new variable in order to store the output of this activity. Call it ‘shouldStop’. Check the *Variables* panel and you will see that ‘shouldStop’ was created with the type Boolean.
7. Further, add a **Flow Decision** activity and connect it. In the *Properties* panel in the Condition property set the variable ‘shouldStop’ as we will need to check if the process should stop or not.
8. Use a **Log Message** activity for the **True** case. Write inside of it a representative message like “Process was stopped. “
9. For the **False** case, connect the Flow Decision with the first Log message activity from the top. Your workflow should look like this:

5. Publishing the Performer

1. From the top menu of the Studio press **Publish**. A pop-up will show up and you should see the information about your package. Click **Publish**.
2. Go to Orchestrator -> Tenant -> Packages. Search for your project here and you should see it.
3. Go to *Processes* in a folder in which you have at least an unattended robot connected and create a new **Process**. In the **Package Name** select the package that you just published.
4. Navigate to *Jobs* and create a new **Job** for your process. Select the Process then the robot on your machine and click **Start**.

Download

Download the workflow below to see the complete solution.

Practice 3 - Queues.zip
96.2 KB