CSX Enterprise Framework

May 2020
CSX internal training

Rosen Monov
rosen.monov@servicecentrix.com
RPA Architect

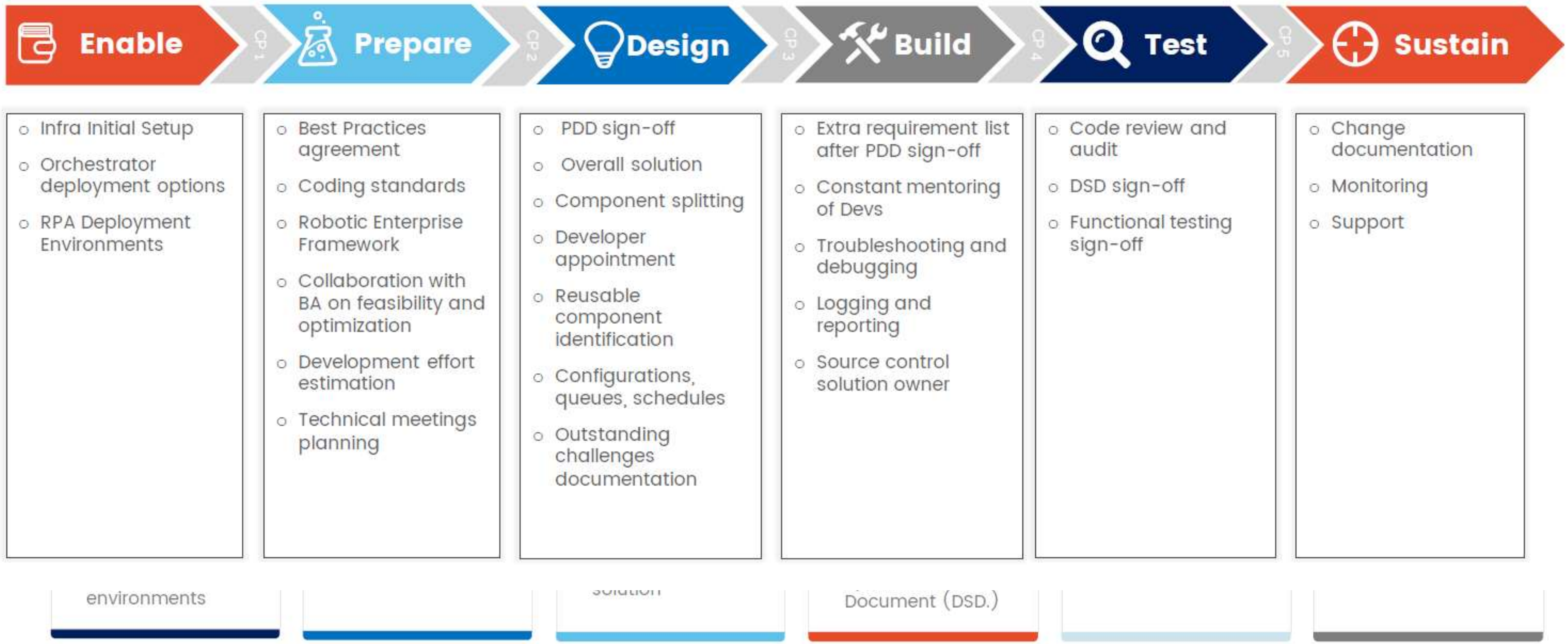
ServiceCentrix

Content

- Solution Architect role
- Transactional processing
- UiPath Framework
- Framework template SCX 0.3f

Solution Architect role

Responsibilities and ownership of the SA



Enable
- Infra Initial Setup
- Orchestrator deployment options
- RPA Deployment Environments

Prepare
- Best Practices agreement
- Coding standards
- Robotic Enterprise Framework
- Collaboration with BA on feasibility and optimization
- Development effort estimation
- Technical meetings planning

Design
- PDD sign-off
- Overall solution
- Component splitting
- Developer appointment
- Reusable component identification
- Configurations, queues, schedules
- Outstanding challenges documentation

Build
- Extra requirement list after PDD sign-off
- Constant mentoring of Devs
- Troubleshooting and debugging
- Logging and reporting
- Source control solution owner

Test
- Code review and audit
- DSD sign-off
- Functional testing sign-off

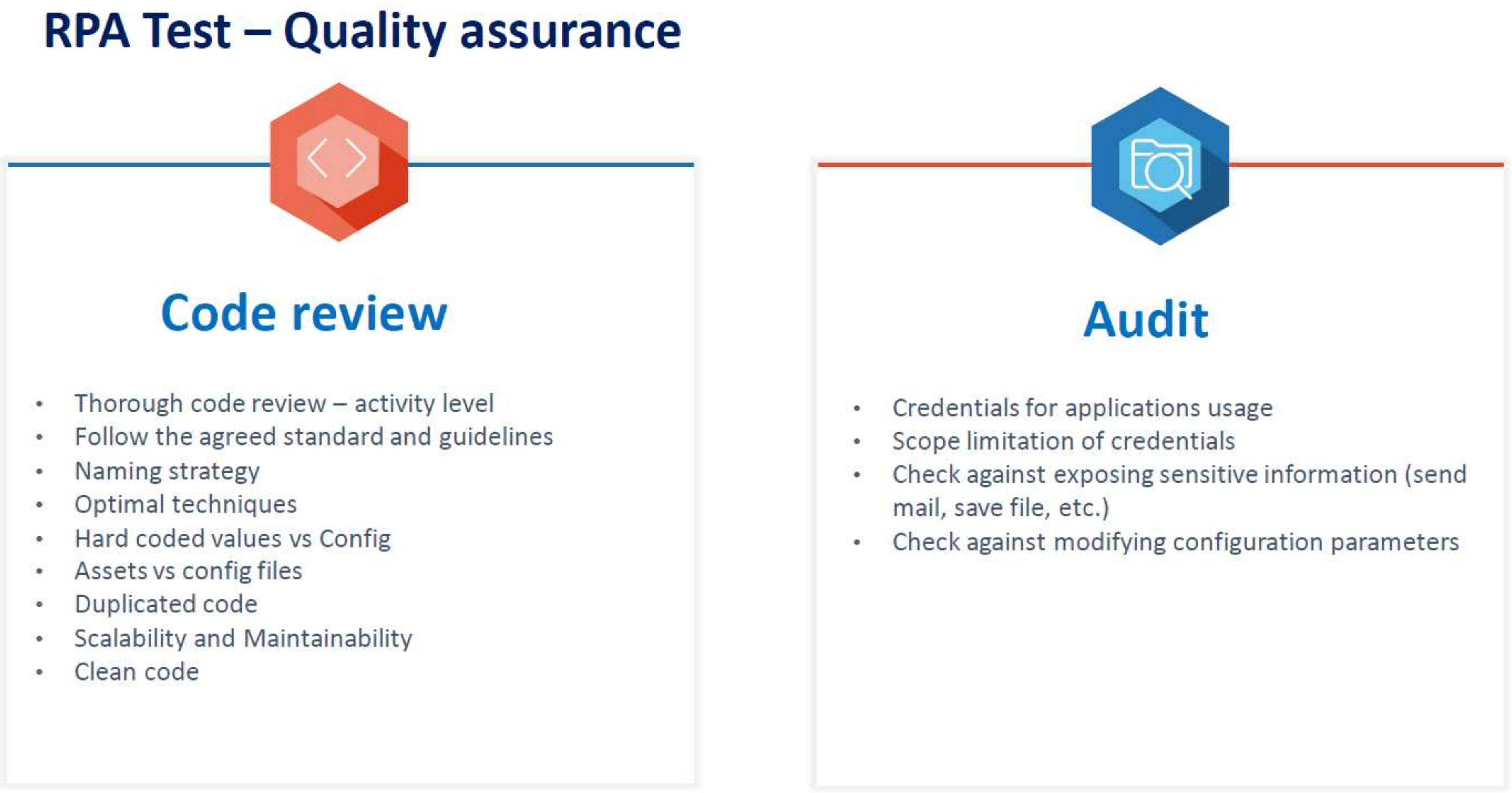
Sustain
- Change documentation
- Monitoring
- Support

environments

solution

Document (DSD.)

Solution Architect role QA



RPA Test – Quality assurance

Code review

- Thorough code review – activity level
- Follow the agreed standard and guidelines
- Naming strategy
- Optimal techniques
- Hard coded values vs Config
- Assets vs config files
- Duplicated code
- Scalability and Maintainability
- Clean code

Audit

- Credentials for applications usage
- Scope limitation of credentials
- Check against exposing sensitive information (send mail, save file, etc.)
- Check against modifying configuration parameters

Solution Architect role
Competitive factors

- Using one FW template
- Folder structure
- WF naming convention
- Variable naming convention
- Best practice
- Lessons learned
- Snippets and reusable components
- Quality assurance
- Facilitated support
- Knowledge base and Team capacity

Transaction

Source: training **Robotic Enterprise Framework Overview**

- **Transaction Processing**

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

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.

Transaction example 1

Examples

- Withdrawal from an ATM
  - Authenticate - Check the account, plastic, validity, expiration, pin
  - Authorize - Check the amount available to withdraw
  - Execute – count and release to tray
  - Taken – Make sure the customer has taken the money

Breaking obstacles: If there is no enough money available in the machine, the transaction will be cancelled.

The transaction is consider successful only if all steps, E2E, have been executed successfully

Transaction example 2

Real live scenario: We provide a service to thousands of end users and it is time to apply monthly service tax

Charging credit cards with monthly service tax

Each CC will be processed, but some of them may fail for various reasons:

- NW issue with the payment portal (System error)
- One card issuer (Visa) does not respond (System error)
- NW connection with the DB is broken (System error)
- The CC has expired (Business error)
- No enough amount (Business error)
- The CC is locked (Business error)

Transaction example 2

- Now what would happen if the whole charging process stops?
- Can we restart it?
- How to make sure CC will not be re-charged again? Identify exactly that charge is not reliable, we do charge same end user for other services too, or upon plan changes.
- Do we need to retry all failures? May be only those system errors should be retried?
- How can we get visibility how the process is progressing, and the success rate?

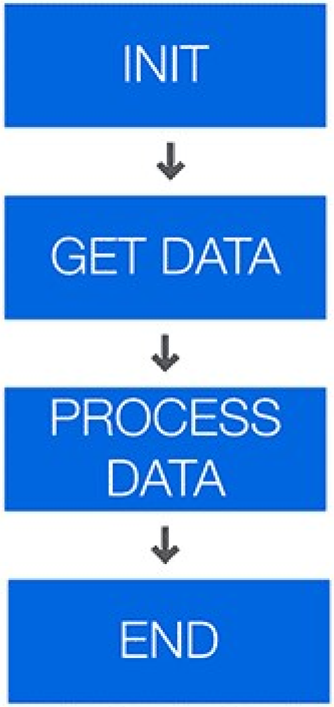
Solution is a TASK-TRANSACTION log.

Run the whole process - Linear

- **Linear**

The steps of the process are performed only once and, if there is the need to process different data, the automation needs to be executed again. For example, if we go back to the CC example, and we had a NW issue for one CC, the automation needs to be executed again in order to process it.

Linear processes are usually simple and easy to implement, but **not very suitable** to situations that require repetitions of steps using different data.

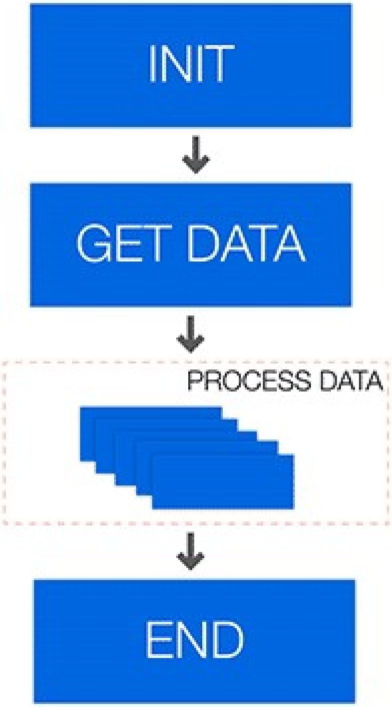


Run the whole process - Iterative

- **Iterative**

The steps of the process are performed multiple times, but each time different data items are used. For example, instead of reading a single CC on each execution, the automation can retrieve multiple CC and iterate through them doing the same steps.

This kind of process can be implemented with a simple loop, but it has the disadvantage that, if a problem happens when processing one item, **the whole process is interrupted** and the other items remain unprocessed

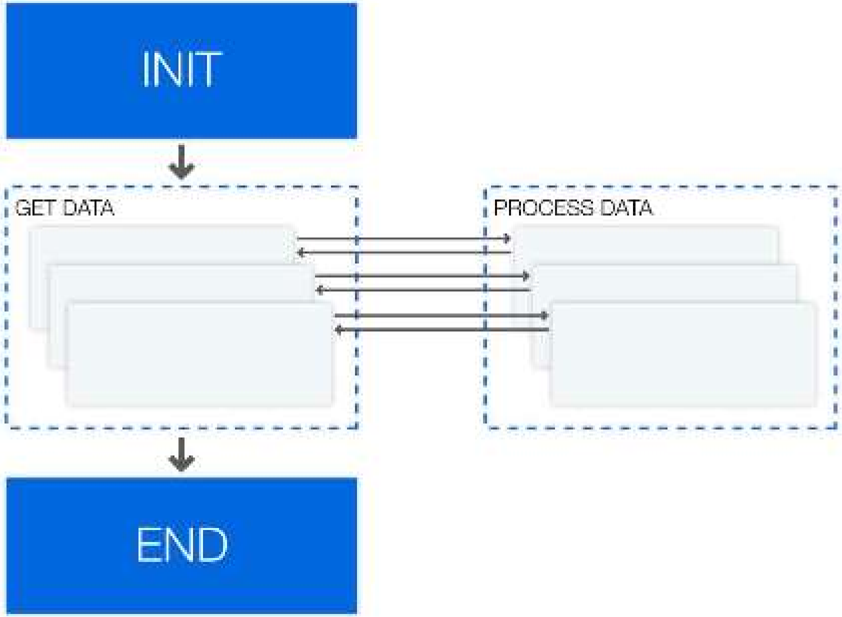


Run the whole process - Transactional

- **Transactional**

Similarly to iterative processes, the steps of transactional processes repeat multiple times over different data items. However, the automation is designed so that **each repeatable part is processed independently**.

These repeatable parts are then called transactions. Transactions are independent from each other, because they do not share any data or have any particular **order to be processed**.



Run the whole process 1

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.

Run the whole process 2

However, this is not an 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).

For example, a process for sending daily report can be Linear.

Nevertheless, if we already have a queue, adding such task is cheap, but would enable:

- Deliver evidence that report has been generated and sent out to the right recipients
- If the process fails, it can be retried
- Can initiate email notification and eventual investigation if the report fails continuously

Business scenarios

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.

What is a Framework?

- 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.

How Framework catches exceptions

What is the mechanism to keep the process running

- CatchException_prj

UiPath REFramework

- 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.

- 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.

UiPath REFramework States

- **Init 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.

- **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.

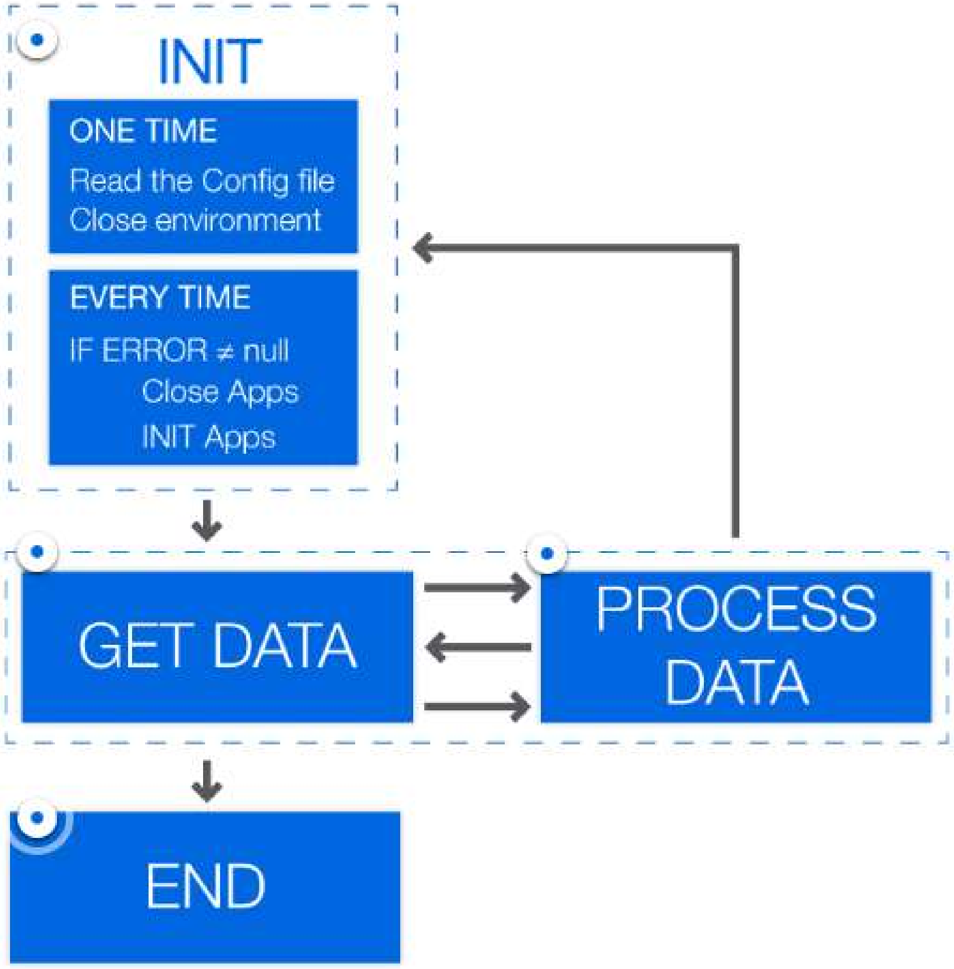
- **Process Transaction State**

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.

- **End Process State**

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

UiPath REFramework States



INIT

ONE TIME

Read the Config file

Close environment

EVERY TIME

IF ERROR ≠ null

Close Apps

INIT Apps

GET DATA

PROCESS DATA

END

REFramework Features

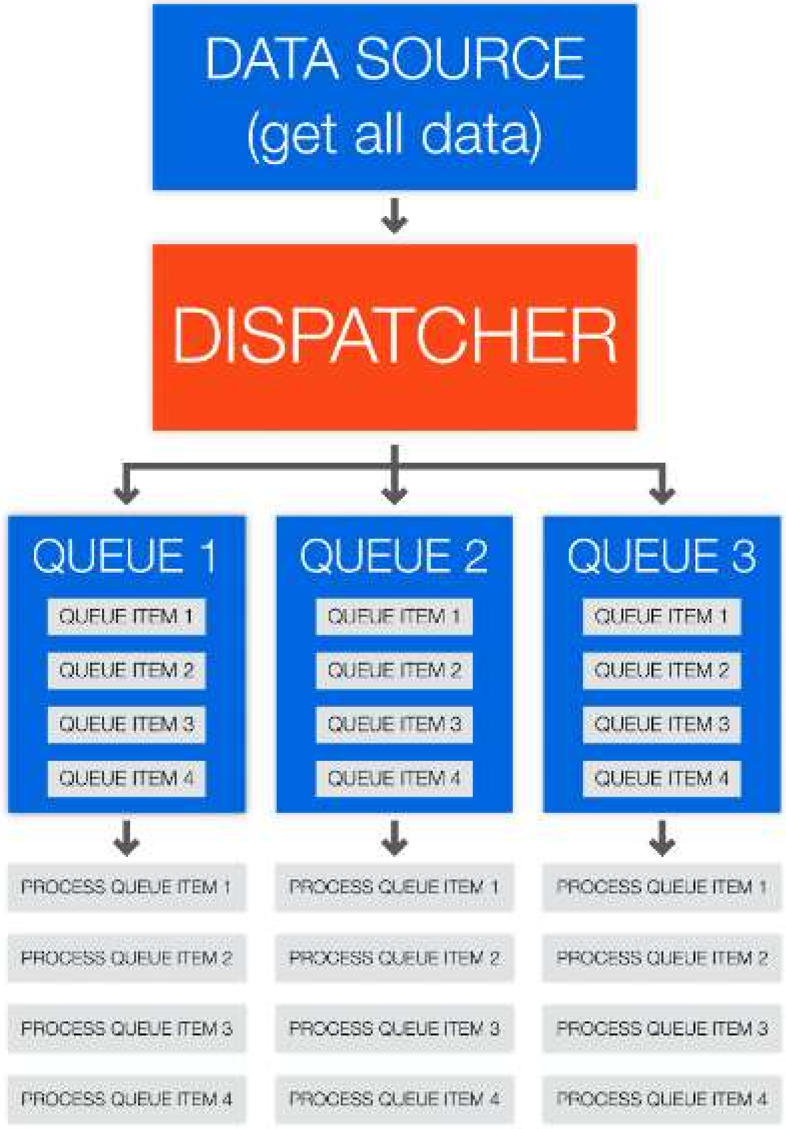
Dispatcher & Performer 1

- **Dispatcher**

The dispatcher is a process used to push transaction items to a queue. It extracts data from one or multiple sources and uses it to create Queue items to be processed by Performer robots.

Information is pushed to one or more queues allowing the dispatcher to use a common format for all data stored in queue items.

The major advantage of using a dispatcher pattern is that you can **split** the processing of the items between multiple robots

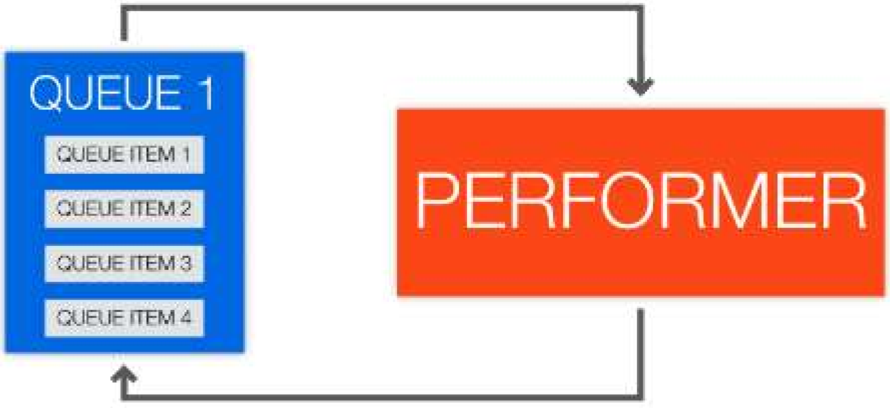


ServiceCentrix

Dispatcher & Performer 2

Performer

- The performer is a process used to pull transaction items from a queue and process them as needed in the company. Queue items are processed one at a time.
- It uses error handling and retry mechanisms for each processed item.
- A major advantage of the performer is its **scalability** (multiple performers can be used with a single queue)



Performer architecture

ServiceCentrix

Dispatcher & Performer 3

- Role of state **Init**
  - To validate all applications (and connections) are alive and the robot can login
  - If possible, leave some apps open for the Dispatcher
- What if we do not validate applications and connections?
  - Risk to get every transaction fail all possible times
  - No point of trying any transaction further
- What if the email does not work?
  - Depends how critical is the role of the email
- What if Init fails?
  - Retry Init, then send email to IT support

Dispatcher & Performer 3

- Role of module **Dispatcher.xaml**
  - Adds transactions to the queue
  - Where Dispatcher should be placed? Init, at the bottom of "First run"
  - To make quick and simple task list
  - To be reliable, collecting minimum information. If it fails, the whole Init would fail as well
  - To avoid multiple processing of same items
- What if Dispatcher fails?
  - No point to continue, unless there are pending transactions, not dependent on the failed app
- How many Dispatchers we may have?
  - As many as Actions we create

Dispatcher & Performer 4

- Role of state **Get Transaction Data**
  - To retrieve a transaction and pass it to Performer. No selection/order criteria in case of Orchestrator.
  - Retrieving the transaction changes its status to “In progress”
- Role of module **Process.xaml**
  - To initiate any coming transaction. For example “Date-time Process started”
  - There could be logic, but usually there is no business code inside, it simply invokes corresponding application-driven-Process.xaml
  - To pass Result upon success

Dispatcher & Performer

Dispatcher & performer model advantages:

1. Better separation of processes (between dispatcher and performer)
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 and productivity)
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

How the REFramework works 2

- Failure in a sub-Process for a single transaction
  - Execution stops immediately
  - You cannot expect any output arguments from any module
  - All local variables are destroyed
  - The execution goes to SetTransactionStatus
  - Application is abandoned in unknown state, next would be closing and reopening all applications. You are not able to perform a rollback.
- What you have left?
  - You have error.Message and error.Source
  - You still have the transaction row, but it is too late for update – data are gone
  - Config is also available
- What should you do?
  - Leave a line in field History to explain what happened

How the REFramework works 3

- Keep TR up to date inside the workflow
  - Update data fields as soon as you receive them
  - Update History field with a row after each important stage, to be seen what point you have reached
  - In order to update TR on stages, before anything bad happens, you may need to pass the TR as argument to each business process workflow. This may not be necessary for screen layer workflows
  - If you need to perform a rollback upon **Business exception** – easy, just create a clone for “recovery actions” or “rollback”, eventually send email, and finally do throw Business exception.
  - If you need to perform a rollback upon **System error:** Surround the critical workflows with try catch, remember raised Exception into variable, and after “recovery actions” throw same or similar Exception again to break the execution and let the framework to handle it.

Manage exceptions

- You may not have exhaustive list of all exceptions upfront
- Build list of System and Business exceptions during the development as you face or set them.
- Ask the customer of the corresponding actions for each exception. Assume and propose simply stopping the process and leaving a note to the T log.
- Any deviations from *out of the box design* (simply breaking the execution), like taking “recovery actions” and notifications, can be very expensive in term of effort and project complexity.
- You may end up with a table of exceptions, containing module name, technical reason, business explanation, recovery actions, and notification with list of recipients.

Corporate template SCX 0.3f

ServiceCentrix

SCX Framework 0.3f 1

Specifics and advantages

- Much cheaper. Designed to use Excel file as a transaction queue, no orchestrator required, providing similar productivity*1 and transactional processing
- More reliable: No single point of failure: instead of NW connection to Orchestrator it uses local resources for transactional processing.
- The queue can be used as a pre-built reporting tool. Filtering might be required.

SCX Framework 0.3f 2

Specifics and advantages

- The config file allows switching between testing and production environment
- The config file allows mode switch regarding step-by-step execution (Dialog mode) and Debug mode (for troubleshooting with detail output)
- The data folder placement is flexible
- Multipurpose queue: allows reusing the same queue for various processes
- Can be easily re-integrated with Orchestrator

SCX Framework 0.3f 3

- Can avoid duplicate transactions

Problem: The dispatcher would take all items (for example Requests) from the application after filtering, then will put them into the queue. What if after processing the **same items** appear each time we open the application? If this is an Orchestrator queue, there would be duplication, i.e. requests would be added, then processed, each time the dispatcher runs.

Solution: SCX framework checks for item duplication, using various criteria, for example when the item was last updated. This check is located in module QAddTransaction, using “Select” statement.

SCX Framework 0.3f 4

- Can get next transaction based on criteria or order. For example Failed first, Pending first, etc.
- Can check previous transactions for existence of same ticket/request. For example, can find how many documents were uploaded recently for this request, and expect same amount in current download. Allows cross-transaction data usage.
- Can filter existing transactions on specific status and trigger actions. For example, if there are requests in pending signature, can send emails to corresponding agents. Cross-transaction data usage again.
- Can make several attempts to run Init state
- Facilitated support (modular, folder, naming)

SCX Framework 0.3f 5

Disadvantages

- The excel file may get corrupted
  - SCX framework does not engage Excel. Nevertheless, be careful how you use Excel
  - Make file backup once a day. Delete old backups
- The excel file may become too slow, if more than 10 thousand rows
  - Backup the file periodically, then use a new queue from template

Solution: SCX Framework 4 using SQL server

Exotic frameworks 1

- Case pre-building task list is impossible

Scenario: At Post Bank there is a button, which shows just one Credit Card that need to be processed.

Solution: Place the Dispatcher in Get Transaction Data (see next slide)

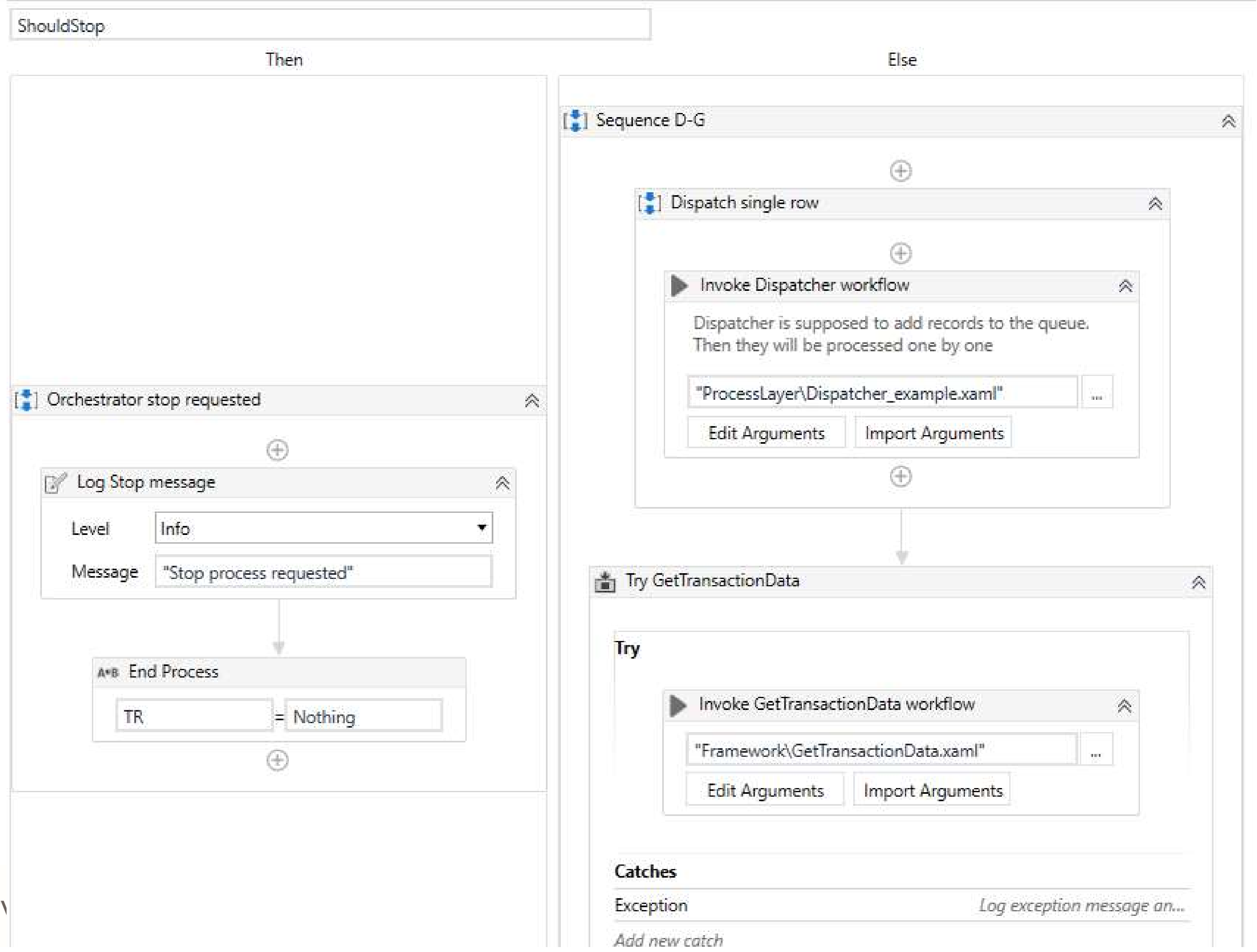
Common\Framework\Framework SCX 0.3c example SingleTID.zip

- Case a transaction need to be put in sleeping state, then checked each time the process runs

Scenario: Robot 1 has uploaded documents for end user signature, and checks periodically if they have been signed or not yet

Solution: Implementing status “Pending” plus field “MainRunTime” to avoid cycling on same Get Transaction.

Framework SingleTID



ShouldStop

Then

Orchestrator stop requested

Log Stop message

Level: Info

Message: "Stop process requested"

End Process

TR = Nothing

Else

Sequence D-G

Dispatch single row

Invoke Dispatcher workflow

Dispatcher is supposed to add records to the queue.
Then they will be processed one by one

"ProcessLayer\Dispatcher_example.xaml"

Try GetTransactionData

Try

Invoke GetTransactionData workflow

"Framework\GetTransactionData.xaml"

Catches

Exception — Log exception message an...

Exotic frameworks MultiAction

- Case the process is split into several sequent and relatively independent Actions

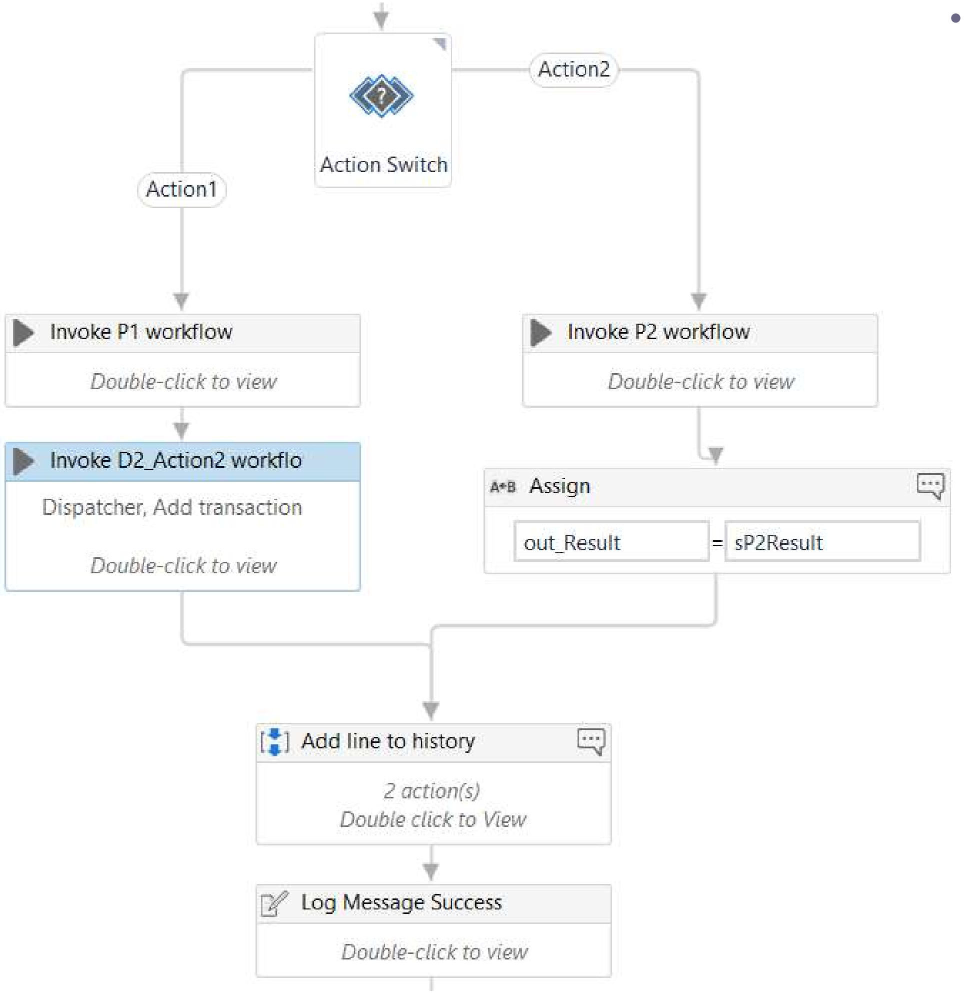
Scenario 1: Upon transaction successful completion a new transaction should be generated for a follow up process.

Scenario 2: The process is too heavy/complex to be executed as a single transaction. It is much better to be split into several steps, where each one can fail and then retried.

Solution:

Use MultiAction queue. It has field “Action” that indicates which Action-Process.xaml should be invoked. Each such process ends with adding the next action-transaction

Framework MultiAction 2



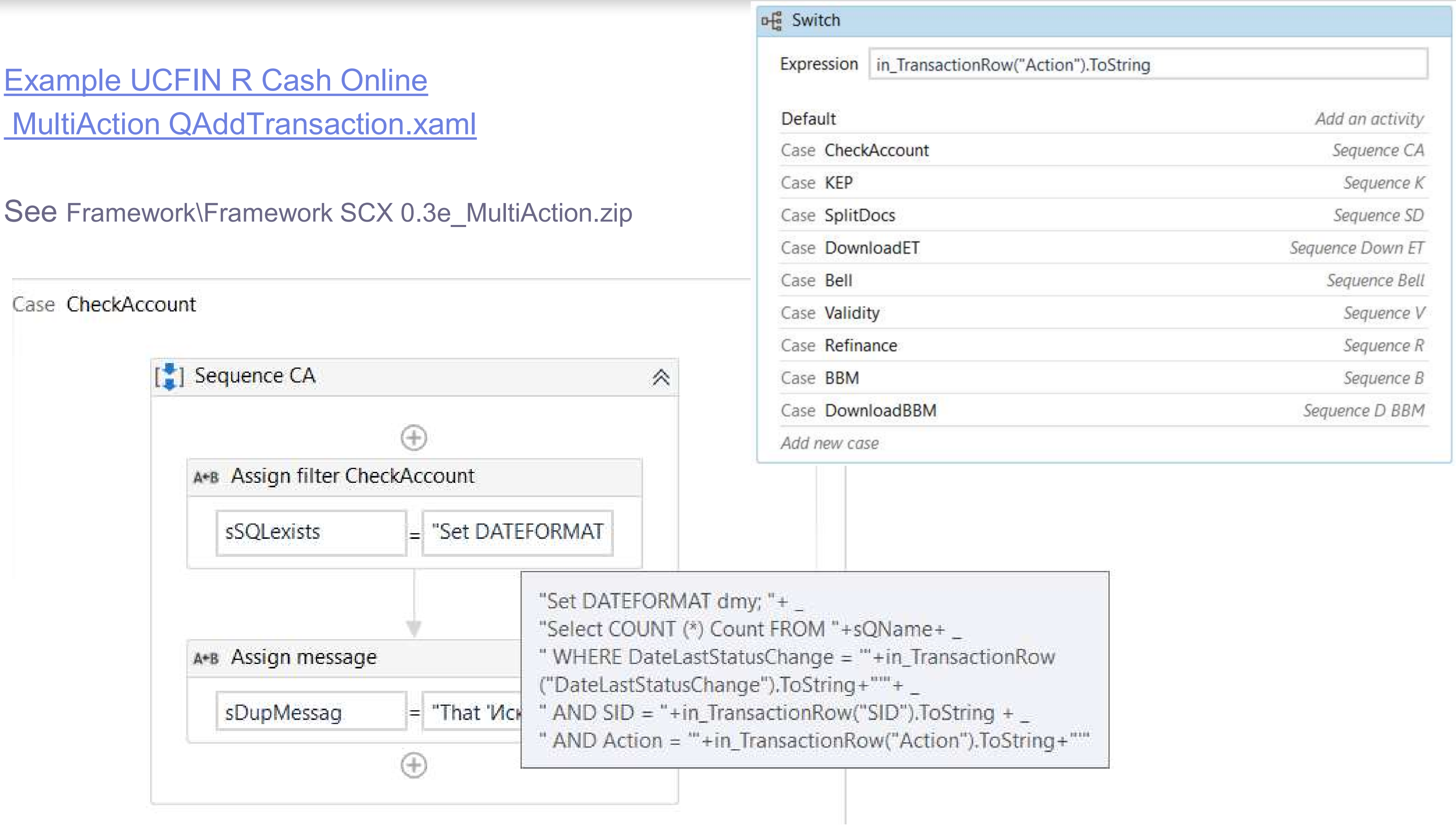
- example MultiAction QAddTransaction.xaml

ServiceCentrix

Framework MultiAction   3

Example UCFIN R Cash Online
 MultiAction QAddTransaction.xaml

See Framework\Framework SCX 0.3e_MultiAction.zip



Switch

Expression: in_TransactionRow("Action").ToString

Default	*Add an activity*
Case CheckAccount	*Sequence CA*
Case KEP	*Sequence K*
Case SplitDocs	*Sequence SD*
Case DownloadET	*Sequence Down ET*
Case Bell	*Sequence Bell*
Case Validity	*Sequence V*
Case Refinance	*Sequence R*
Case BBM	*Sequence B*
Case DownloadBBM	*Sequence D BBM*
Add new case	

Case CheckAccount

Sequence CA

Assign filter CheckAccount

sSQLexists = "Set DATEFORMAT

Assign message

sDupMessag = "That 'Иск

```
"Set DATEFORMAT dmy; "+ _
"Select COUNT (*) Count FROM "+sQName+ _
" WHERE DateLastStatusChange = '"+in_TransactionRow
("DateLastStatusChange").ToString+"'"+ _
" AND SID = "+in_TransactionRow("SID").ToString + _
" AND Action = '"+in_TransactionRow("Action").ToString+"'"
```

ServiceCentrix

Exotic frameworks   Pending

- Case process need to wait, Pending status

Implemented in SCX 0.3f

Challenge: Avoid taking the same transaction in Pending status indefinitely

  - In Main run time is remembered
  - Argument in_MainRunTime is passed to QGetTransaction.xaml, which does not take those in pending withing same Runtime

"[Status] = 'New'

OR ([Status] = 'Pending' AND [**MainRunTime**] < '"+in_MainRunTime.ToString("yyyy-MM-dd HH:mm:ss")+"')"

  - Whenever status is put back to Pending, MainRunTime is updated by *SetTransactionStatus.xaml*

Exotic frameworks   2 frameworks

- Case Dispatcher is complex and may fail at specific item, or need to process exceptions

Examples

- **Exceptions**: Processing each email, downloading attachments, validate their content, then finally adding transaction rows or eventually replying back to the sender
- **Separate schedule**: Dispatcher should run many times throughout the day, then the process should run just once

Solution

- Put the Dispatcher in a separate framework
- The Data folder can be shared (config file)

Framework challenges 1

- Sometimes a failure may require **reverse actions** to be performed, before marking the transaction as failed.
- If it is about a **business issue**, where throwing the error is more or less under control, such code can be placed in the workflow itself, right before throwing the BE.
- In case of **system error**, the execution will return back to SetTrasnactionStatus.xaml, but it would be hard to determine what has happened and where, since output arguments from the failed workflow cannot be passed. We may know only its name, and the message. Both are not reliable, moreover, maybe even we have lost control over the application. In such case, the solution is to **surround the critical pace with Try-catch** and take reverse actions immediately after catching the error. Hereby we can put the right comment into the transaction row, as well as to send a warning email notification, along with exception's details.

SCX Framework 0.3f
Config file

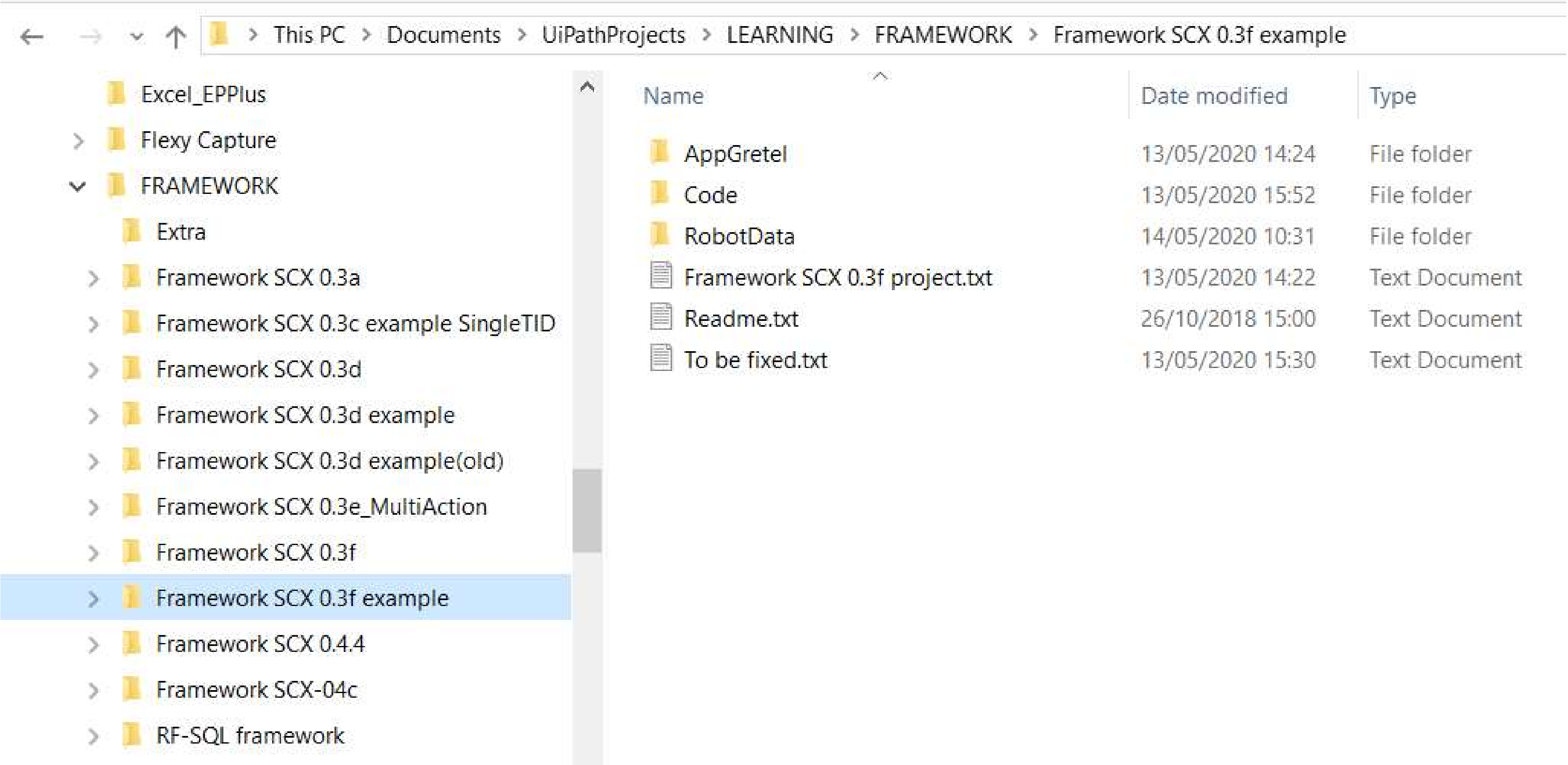
- Two environments. Switching is easy
- Dialog mode, Debug mode
- Queue definition with predefined fields

Using SCX Framework 0.3f
Folders and config file location

- Finding Config file is hardcoded in InitAllSetting.xaml
- Row “DataPath” exists in config file solely to indicate it will be filled automatically
- Why DataPath is not retrieved from Config file?
- Config.xlsx is always in the data folder
- Last action in InitAllSetting.xaml sets default DateTime order dd/MM/yyyy, instead of default US order month-day
- Folder *Code* can be renamed
- Folder *RobotData* can be renamed, if DataPath.ini is used
- Keep both Code and Data folder on non-system drive
- Folder Documentation is good for BR, release information, Ops manual, etc.

Using SCX Framework 0.3f
Data folder and config file location O1

- Option 1: One folder for both Code and Data (Unattended)



This PC > Documents > UiPathProjects > LEARNING > FRAMEWORK > Framework SCX 0.3f example

- Excel_EPPlus
- Flexy Capture
- FRAMEWORK
  - Extra
  - Framework SCX 0.3a
  - Framework SCX 0.3c example SingleTID
  - Framework SCX 0.3d
  - Framework SCX 0.3d example
  - Framework SCX 0.3d example(old)
  - Framework SCX 0.3e_MultiAction
  - Framework SCX 0.3f
  - Framework SCX 0.3f example
  - Framework SCX 0.4.4
  - Framework SCX-04c
  - RF-SQL framework

Name	Date modified	Type
AppGretel	13/05/2020 14:24	File folder
Code	13/05/2020 15:52	File folder
RobotData	14/05/2020 10:31	File folder
Framework SCX 0.3f project.txt	13/05/2020 14:22	Text Document
Readme.txt	26/10/2018 15:00	Text Document
To be fixed.txt	13/05/2020 15:30	Text Document

ServiceCentrix

Using SCX Framework 0.3f
Data folder and config file location O1

- Advantages
  - Keep the folder structure simple, it works from any root folder
  - Can have several independent robots/frameworks on same machine
  - Compact. Moving robot to another machine is easy
- Disadvantages
  - Will not work if project packed then run from robot tray.
  - Why ?

PackageLocation.PNG

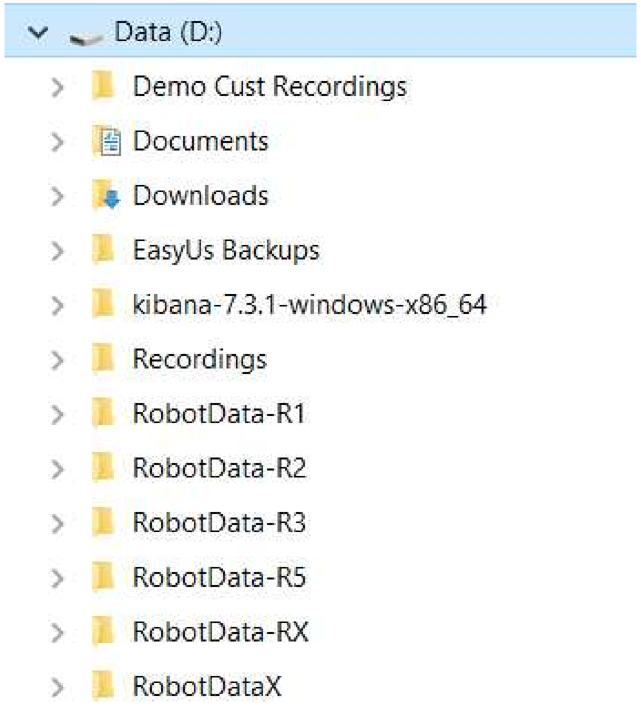
- How to set?

This is by default

(file name is Code\DataPath.ini.notused)

Using SCX Framework 0.3f
Data folder and config file location O2

- Option 2: Any folder for Code and specific folder Data (Attended)



Data (D:)
- Demo Cust Recordings
- Documents
- Downloads
- EasyUs Backups
- kibana-7.3.1-windows-x86_64
- Recordings
- RobotData-R1
- RobotData-R2
- RobotData-R3
- RobotData-R5
- RobotData-RX
- RobotDataX

- Advantages
  - Can have several independent robots/frameworks on same machine
  - Works from schedule, starting Main.xaml
  - Works as a robot package from system tray
- Disadvantages
  - Moving to another machine is more complex
  - Robots may use wrong data folder

  Solution: there is value *RobotName* in config

- How to set?

Rename file *Code*\DataPath.ini.notused to *DataPath.ini* and enter the full path to the data folder

ServiceCentrix

Using SCX Framework 0.3f

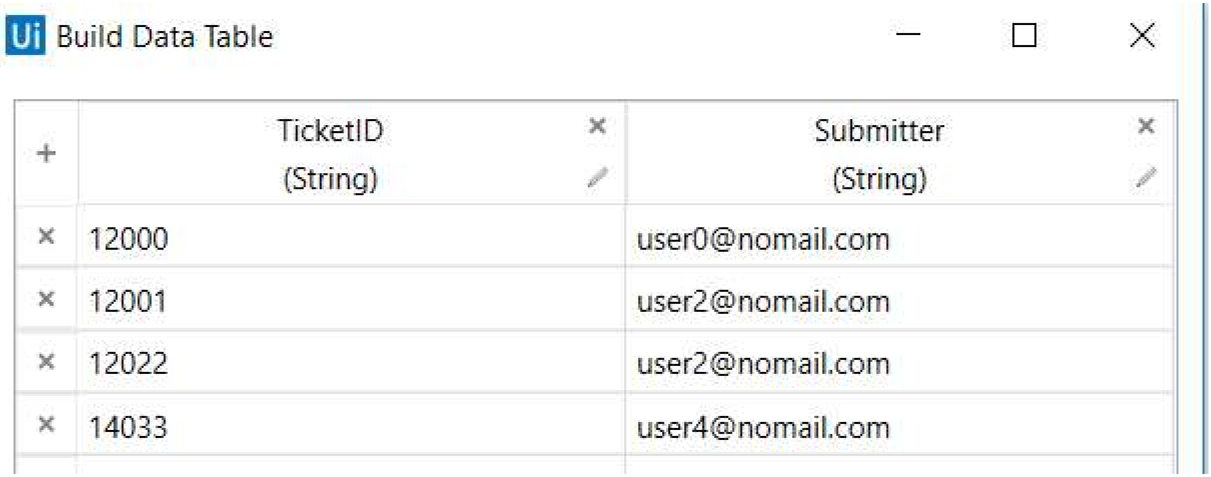
- Design the queue
  - Determine what is going to be the scope of a single transaction.
  - If we need to use multiple records per transaction, like many items per one purchase order, we can store these items as an excel file, named with the transaction id and located in folder Temp *1.
  - Fill tab QueueDefinition
  - Make fields for critical values. Short list can be stored in a single string field
  - Always store Date-time values as a Datetime filed, to be able to compare them and avoid formatting issues.
  - Fields like Item id, for example RequestID, TicketID, etc. is more practical to be stored as strings
  - The queue file will be created upon first call of QAddTransaction

Using SCX Framework 0.3f Dispatcher

- Make Dispatcher

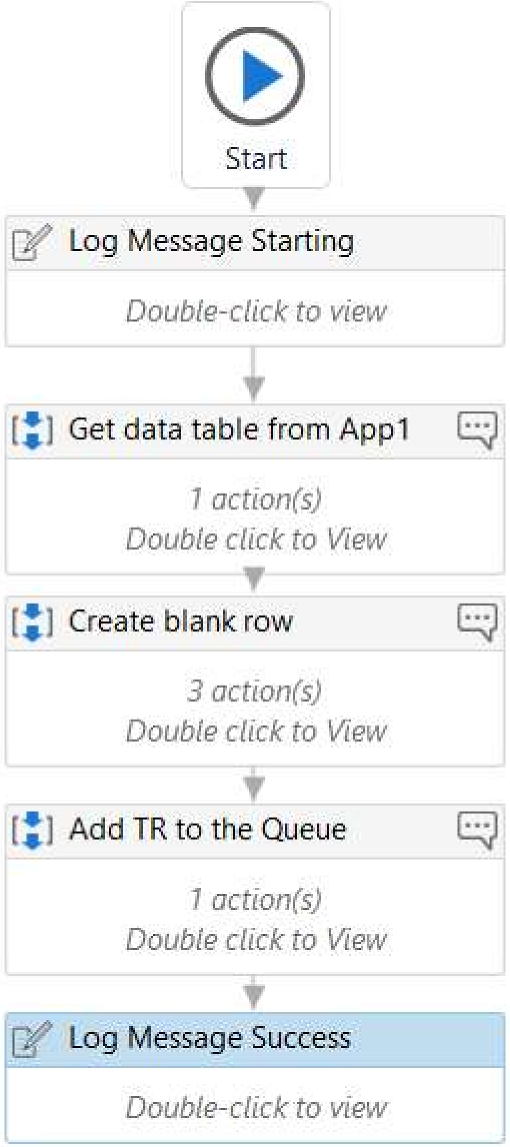
Look at file “Framework SCX 0.3 example.zip” *Dispatcher.xaml*

- Get data table from App1 is a dummy sequence. In reality you should set a filter and get records that need to be processed into a table **t1.** For demo purposes the job is done by this activity:



Build Data Table

TicketID (String)	Submitter (String)
12000	user0@nomail.com
12001	user2@nomail.com
12022	user2@nomail.com
14033	user4@nomail.com



Start

Log Message Starting

Double-click to view

Get data table from App1

1 action(s)
Double click to View

Create blank row

3 action(s)
Double click to View

Add TR to the Queue

1 action(s)
Double click to View

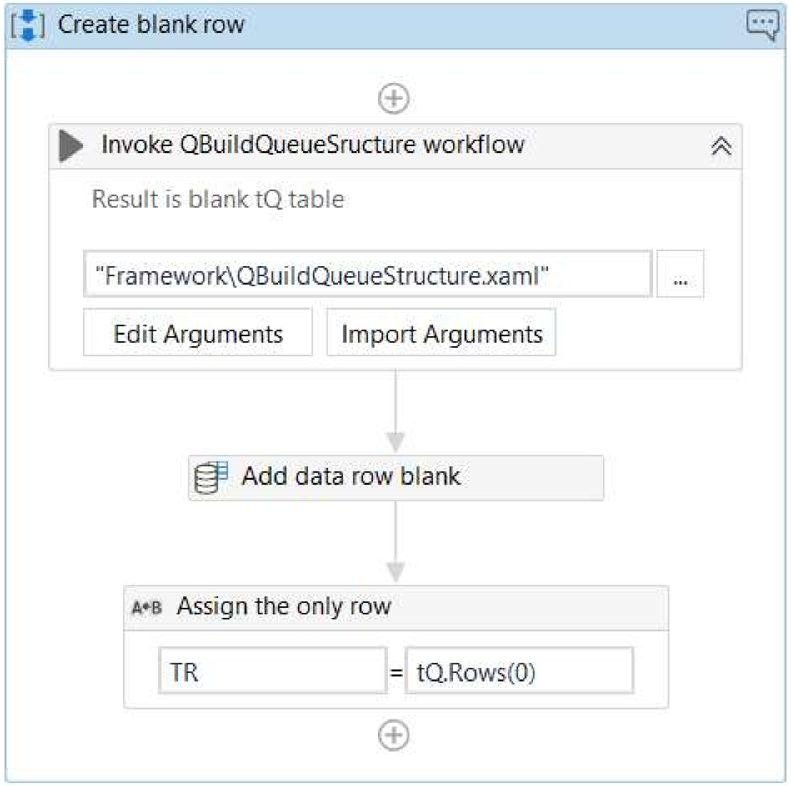
Log Message Success

Double-click to view

ServiceCentrix

Using SCX Framework 0.3f Dispatcher

- A blank transaction row is created. Same row will be reused many times
- QBuildQueueStructure will provide an empty table **tQ**, which is built with all fields and data types according to the config file.
- Each time you need to add records to the queue (in each Dispatcher) you have to use that sequense



Create blank row

Invoke QBuildQueueSructure workflow

Result is blank tQ table

"Framework\QBuildQueueStructure.xaml"

Add data row blank

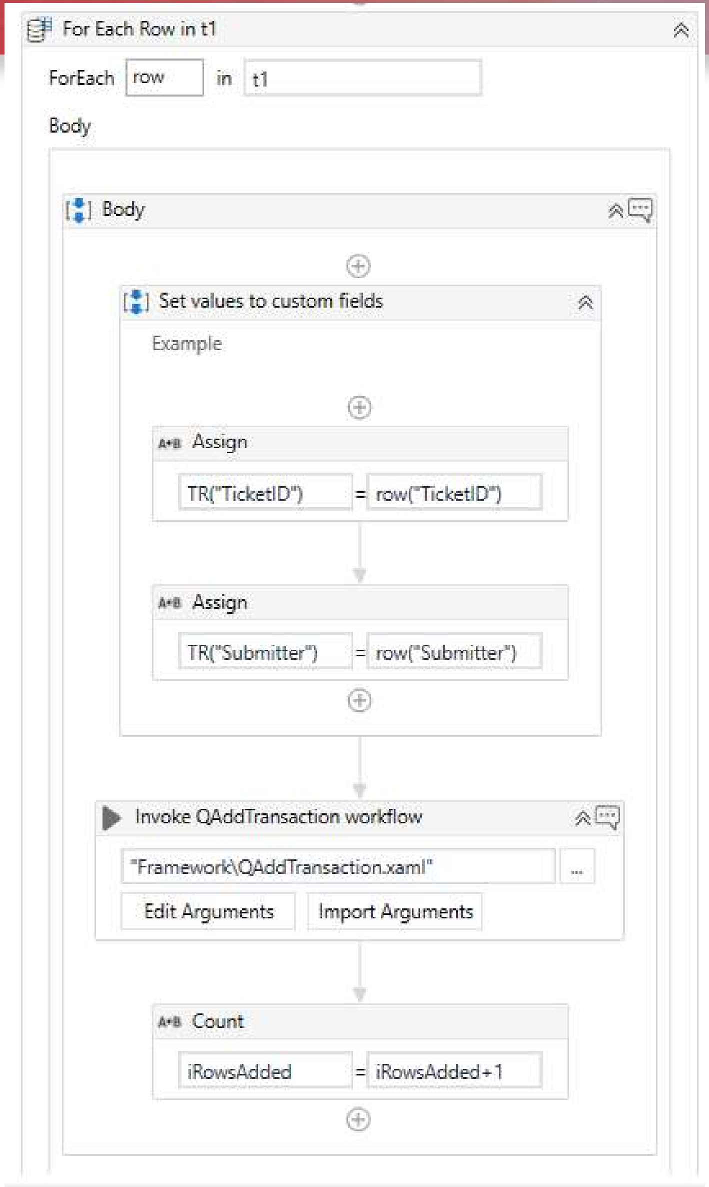
Assign the only row

TR = tQ.Rows(0)

Using SCX Framework 0.3f Dispatcher

- Each record, retrieved from our application, will be set as value to our Transaction Row (TR), then added as a new transaction to the queue

- You do not need to take care about “mandatory fields”, they will be populated automatically, according to the rules of table tQ.



For Each Row in t1

ForEach row in t1

Body

Body

Set values to custom fields

Example

Assign

TR("TicketID") = row("TicketID")

Assign

TR("Submitter") = row("Submitter")

Invoke QAddTransaction workflow

"Framework\QAddTransaction.xaml"

Count

iRowsAdded = iRowsAdded+1

ServiceCentrix

Using SCX Framework 0.3f Dispatcher

- Why simply reading from the queue file should not be used?
- Reading a range from Excel file would provide field names without data type, i.e. all fields in the resulting table would be data type “Object”.
- Using QBuildQueueStructure will enforce DataType for each field

Demo ExcelReadRange.xaml

Using SCX Framework 0.3f Dispatcher

- Make your own dispatcher
- Test your dispatcher, running

Process\TestInvoke\TestInvoke_Dispatcher.xaml

- Modify query
  - *Delete all rows: Do not delete content, mark the waste rows and delete them*
  - *Delete all rows or the last ones, but not in the middle. This would break the queue: ThransactionID must confirm to same row number+2*
  - *Add/remove/modify fields*

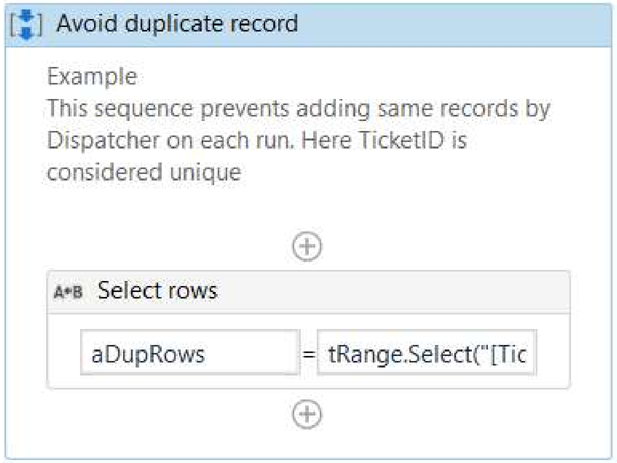
  You can either modify config file, then regenerate the queue, or edit config file and update the queue file accordingly

Using SCX Framework 0.3f Dispatcher

- Customize QAddTransaction

This is necessary if you want to avoid adding same records each time you run the dispatcher. Customize the select statement. Currently it checks for same TicketID:

tRange.Select("[TicketID] = '"+in_TR("TicketID").ToString +"' ")



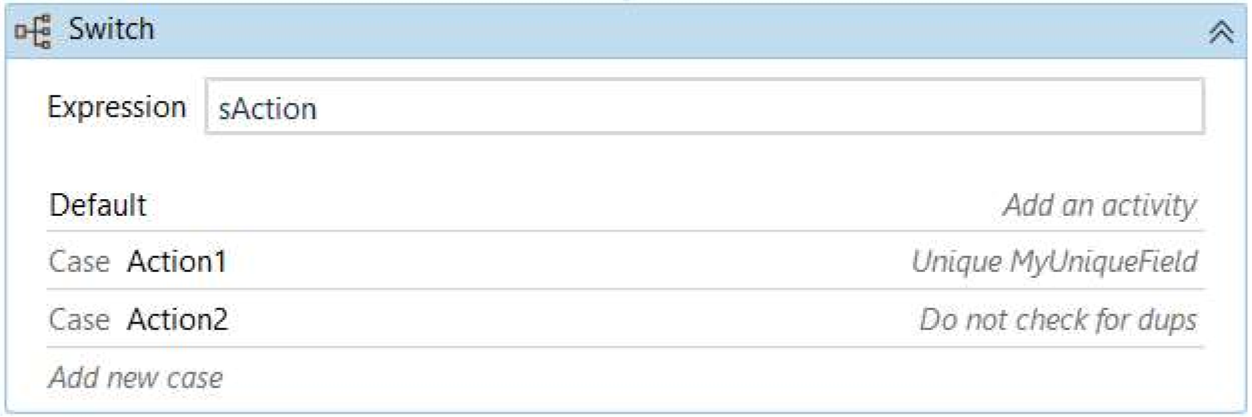
Avoid duplicate record

Example
This sequence prevents adding same records by Dispatcher on each run. Here TicketID is considered unique

Select rows

aDupRows = tRange.Select("[Tic

- In case of MultiAction queue, you can use a switch



Switch

Expression: sAction

Default	*Add an activity*
Case Action1	*Unique MyUniqueField*
Case Action2	*Do not check for dups*
Add new case	

ServiceCentrix

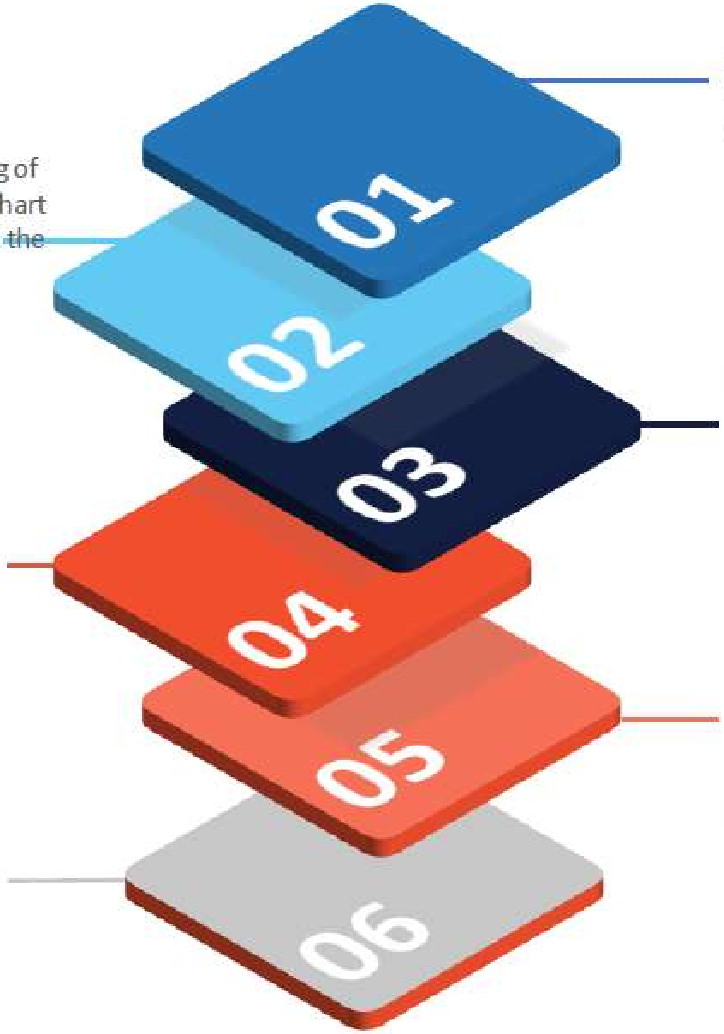
Using SCX Framework 0.3f Performer

- Customize Process
  - Keep it simple. Build it up invoking workflows
  - Assign argument out_Result, if you want to replace default Result value “Record processed” with something else, for example “Request approved”
  - Be aware if BE or SE happens anywhere, **no arguments** will be returned
  - Pass TR to invoked workflows, if you want them to be able updating the transaction with their own history, as well as other fields. Recommended for **Business layer** workflows. Otherwise get out argument from them and update TR in Process.

Using SCX Framework 0.3f
Modular structure

- Why we need modules?
  - Stable during modification
  - Version control
  - Avoid code duplication
- Application and business modules
  - Always use framework
  - For new modules start form template. Mandatory argument is *in_config*, and optional is *in_TR*
  - Place all application-related workflows in one folder, for example App1SAP
  - Non application-related workflows should be in folder Common

Workflow abstraction



1. Framework layer: High level State Machine using the REFrameWork with overall exception handling.

2. Business Process layer: Modeling of the business process rules - a flowchart integrating other components from the other levels.

3. Services layer: Additional service components with internal exception handling. State machine template.

4. Application Process layer: Components working with applications and implementing process rules. The argument names can be related to the business process. This component will navigate, write and extract.

5. Application Screen layer: Generic components where arguments are not related to the business process rules. Separate workflows can be created to navigate, input, and extract data.

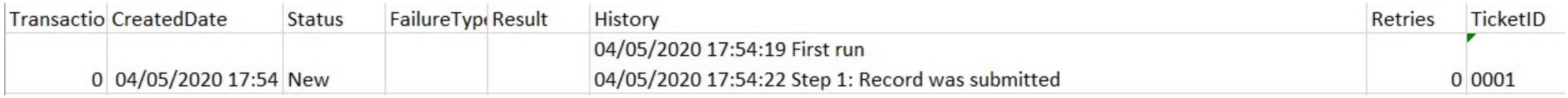
6. Data layer: Interact only with easily accessible (local) data. Orchestrator activities allowed. Not for application interaction.

SCX Framework 0.3f
Unit testing

- Make unit testing of each module
  - For each module create calling workflow for unit testing, in folder TestInvoke. Use snippets template and name it as module’s name.
  - The test module may pass a TR, and save the result into TR.xlsx

See template Code\Snippets\TestInvoke_TR_X.xaml

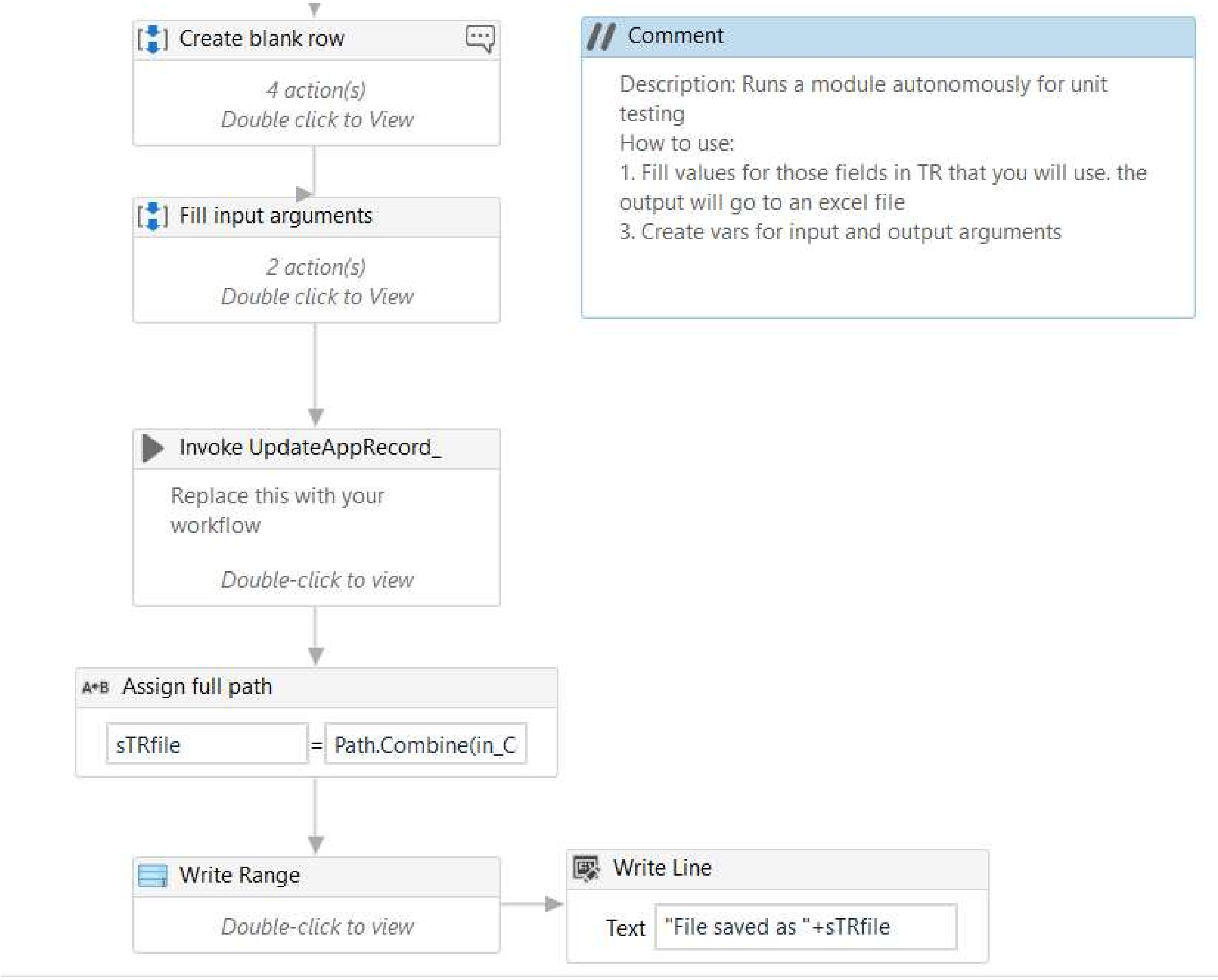
Here is the output produced:



Transactio	CreatedDate	Status	FailureTyp	Result	History	Retries	TicketID
0	04/05/2020 17:54	New			04/05/2020 17:54:19 First run 04/05/2020 17:54:22 Step 1: Record was submitted	0	0001

SCX Framework 0.3f
Unit testing

TestInvoke_TR_X.xaml



Create blank row

4 action(s)
Double click to View

Fill input arguments

2 action(s)
Double click to View

Invoke UpdateAppRecord_

Replace this with your workflow

Double-click to view

Assign full path

sTRfile = Path.Combine(in_C

Write Range

Double-click to view

Write Line

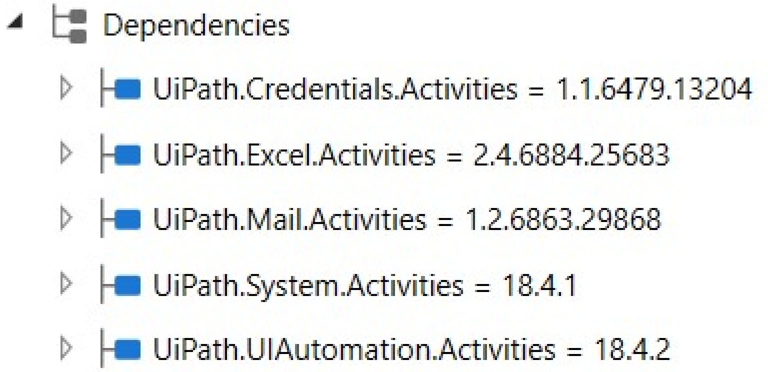
Text "File saved as "+sTRfile

Comment

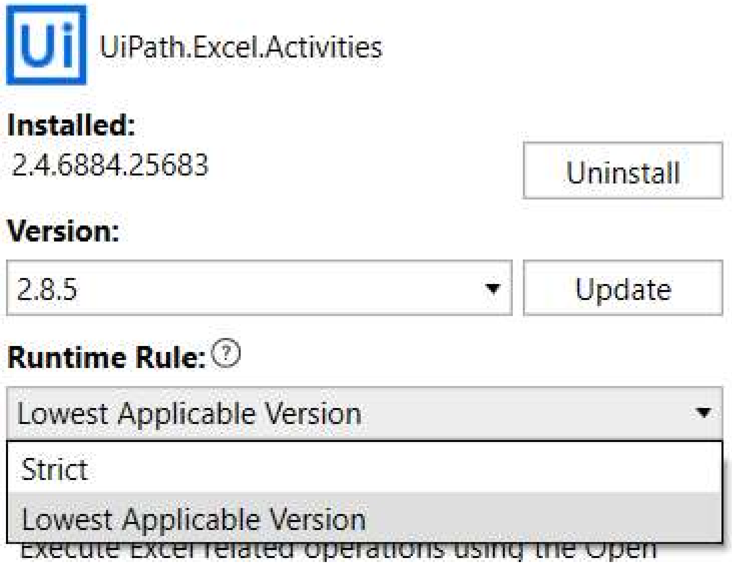
Description: Runs a module autonomously for unit testing
How to use:
1. Fill values for those fields in TR that you will use. the output will go to an excel file
3. Create vars for input and output arguments

SCX Framework 0.3f
UiPath package version

- Update Activity packages. In SCX framework they are 18.x, but can be updated to 19.x or 20.x
- Set Runtime rule, for a robot should be “Strict”
- Remove not used dependencies
- Upon first framework load you can even delete project.json, if you want to use your already installed packages, or if your Studio version is too old



Dependencies
- UiPath.Credentials.Activities = 1.1.6479.13204
- UiPath.Excel.Activities = 2.4.6884.25683
- UiPath.Mail.Activities = 1.2.6863.29868
- UiPath.System.Activities = 18.4.1
- UiPath.UIAutomation.Activities = 18.4.2



UiPath.Excel.Activities

Installed:
2.4.6884.25683

Version:
2.8.5

Runtime Rule:
Lowest Applicable Version

Execute Excel related operations using the Open

SCX Framework 0.3f
Recap

- Create folder structure, Datapath.ini
- Design queue (part of solution design)
- Config: Set queue name and fields
- Config: Set app URL, credentials, etc.
- Make screen layer workflows (Login_Appx, FindRecord, GoToDetials, etc.
- Make Business layer workflows (ReadData, Calculate…) they may need TR passed
- Build Dispatcher, set AvoidDuplications (QAddTransaction.xaml), Unit test, integrate in Framework
- Build Process, Unit test, integrate in Framework

Thank you !

www.servicecentrix.com
info@servicecentrix.com

ServiceCentrix