

CenturyLink IT

RPA Logging Standards

Release V0.7

Prepared By:

Paul Gullette – Lead Software Developer

Updated By:

Murugavel Dhanavel – Software Developer

Document Information

This section identifies information specific to this document.

Document History

This is an evolving document for RPA Logging Standards.

|  |  |  |
| --- | --- | --- |
| Ver | Date | Description of Changes |
| 0.1 | 6/15/2018 | Created |
|  |  |  |
| 0.2 | 6/18/2018 | Added formatting/versioning/author/toc |
| 0.3 | 6/19/2018 | Finished defining required states. Added additional logging requirements. |
| 0.4 | 6/20/2018 | Added default metrics, exception logging details. |
| 0.5 | 6/22/2018 | Updated screenshot labels, added url to GitLab repository containing template. |
| 0.6 | 6/27/2018 | Updated screenshot of template. Updated text based on provided feedback. |
| 0.7 | 06/03/2018 | Updated to include logging standard to control concurrent runs and to use common logon ids across use cases, added condition to check for concurrency error before entering InitAllApplications |

This document was created within CenturyLink IT. CenturyLink IT has made every effort to ensure this document is accurate at the time of printing.

Copyright and Trademarks

© 2019 CenturyLink, Inc. All rights reserved. The CenturyLink mark, pathways logo and certain CenturyLink product names are the property of CenturyLink, Inc. All other marks are the property of their respective owners.

Contents

Purpose 4

Logging 4

Logging Definitions 4

Splunk Event Screenshot 5

Default Fields 6

Custom Fields 7

UIPath Robotic Enterprise Framework 7

REF States 8

CTL\_UIPath\_Transactional\_Template Screenshot 9

Obtaining The Template 9

Usage of The Template 9

Initialize Process State 10

State Requirements In Order 10

Transitions 10

Fetch Next Transaction Items State 11

State Requirements In Order 11

Transitions 11

Load Single Transaction Item State 12

State Requirements In Order 13

Transitions 14

Process Single Transaction Item State 14

State Requirements In Order 15

Transitions 15

logF\_TransactionStatus Assignment 15

End Process 16

State Requirements In Order 16

Transition Logging 16

Transition Requirements In Order 16

Exception Logging 17

Exception Logging Requirements In Order 17

Other Required Logging 17

External .XAML File Logging Requirements In Order 17

Default Metrics 18

Default Metrics Table 18

# Purpose

To ensure consistency across automations, this document will serve as the guidelines that define the type of log events, the content of the log events and the locations where log events need to be present. In addition, a consistent approach for starting development of a new automation within UIPath will also be outlined.

# Logging

A fundamental component of any application is its logging. Logging historically has been used to indicate errors or exceptions, and sometimes during development, logging may be set to a verbose level to provide much more detail about what an application is doing. For RPA, logging will be the basis for consistently providing details of each automation both in terms of what is happening as well as any errors or exceptions. UIPath has a robust logging implementation that allows for consistently logged items coupled with an ability to extend the kind of contextual information needed. For practical reasons and for the success of the RPA platform, logging will be used to tell the story of each automation empowering a rich set of metrics, alarming, indicators of performance issues, and many other benefits.

## Logging Definitions

A logging event will consist of a mixture of default fields as well as custom fields. Custom field provide a consistent starting point for all report writing and dashboards created using logging information. Here is a screenshot of an event from Splunk showing the result of logging an event with custom fields added.

### Splunk Event Screenshot



Below is a table with some of the default fields that will be present on every logged event:

### Default Fields

|  |  |
| --- | --- |
| Field Name | Field Description |
| fileName | The name of the .XAML file which logged the event |
| fingerprint | A unique identifier for every log event |
| jobId | A unique identifier that exists for the duration of a given automation run. This value can be used to group all logged events for a given job execution. |
| level | The log level of the message. Can be Information, Error, Warn, Fatal, Debug. |
| logType | Indicator of how the log event was created. Default log events are generated by UIPath, and they will have a value of “Default”. A user-initiated log event will have a value of “User”. |
| machineName | The computer host name where the log event occurred |
| message | The content of the message argument provided by the developer when implementing the Log Message activity |
| processName | The name of the process as defined in the project.json file |
| processVersion | The version of the process as defined in the project.json file |
| robotName | The name of the robot that is executing the automation |
| timeStamp | A fully granular date/time string including subsecond precision as well as timezone offset |
| windowsIdentity | The windows user id that was responsible for initiating the bot execution |

There are additional default fields that will be present under certain circumstances. Examples include log events that occur during a transaction involving an Orchestrator queue, log events that indicate the start of a robot’s execution as well as the end.

In addition to the default logging and default logging fields provided by UIPath, the RPA standards also define the need for custom logging fields. Each of these custom fields are prefixed with a “logF\_” to avoid name clashes. This ensures none of the custom fields overlap or replace default fields.

## Custom Fields

|  |  |
| --- | --- |
| Field Name | Field Description |
| Template Version -1 | |
| logF\_IntakeID | The intake number determined at the time the intake was created. This value will be used to map any additional items related to this intake such as business process name, business unit name, etc. |
| logF\_StateName | The name of the current state within the state machine |
| logF\_TransitionName | The name of the current transition within the state machine |
| logF\_TransitionSource | The source of the transition |
| logF\_TransitionDestination | The destination of the transition |
| logF\_TransactionID | A unique identifier within a job representing the current item being processed |
| logF\_TransactionDetails | Any additional information related to the current transactional item. For instance, if the logF\_TransactionID is an order id, this field would contain any related order information that might be pertinent to the automation such as order type, customer, location, etc. This field will be a dictionary including all related fields and their values rather than defining an unknown number of fields with the logF\_ prefix. |
| logF\_TransactionStatus | An indicator set after Process Single Transaction Item State has completed. Possible values are Success, Business Exception and Technical Exception. |
| logF\_TemplateVersion | An incrementing version integer that will be preset within the template |
| logF\_ExceptionDetails | The body of the exception |
| logF\_ExceptionType | The type of exception which can be either Business or Technical |
| Template Version -2 | |
| logF\_ProcessinstanceID | Indicates the number of parallel instances of the same process running in orchestrator under same tenant |
| logF\_bConcurrentError | Indicates whether the maximum number of allowed concurrent runs is exceeded or not |
| Template Version -3 | |
| logF\_CommonApp | The Acronym of the application which uses the common logon id if any. Acronym for any application should be from CMS Viewer. Eg: “SFC” for ‘Salesoforce.com Consumer’  - If there are more than one app using common id, separate the acronyms by comma. Eg: “SFC,TAG”  - When there are no common application ids used, have the value as “N/A” |
| logF\_CommonID | The common user id used across multiple use cases to login to a specific application  If there are more than one app using common id, separate the acronyms by comma in the same order as application acronyms. Eg: “RPASFC,RPACOMMON”  When there are no common application ids used, have the value as “N/A” |
| logF\_CommonAction | The action performed by the use case in that application using the common id. Possible values are Retrieve/Create/Update/Delete  If there are more than one app using common id, separate the actions by comma in the same order as application acronyms. Eg: “Retrieve/Update, Create”  When there are no common application ids used, have the value as “N/A” |
| logF\_CommonActionDescription | Details of action being done in the application using common id.  If there are more than one app using common id, separate the actions by comma in the same order as application acronyms. Eg: “Retrieve order data/Update provisioning details, Create new telephone numbers”  When there are no common application ids used, have the value as “N/A” |

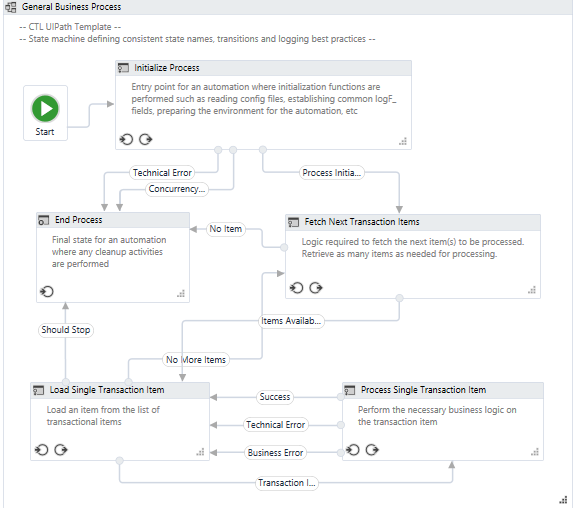
# UIPath Robotic Enterprise Framework

Each workflow automation will be developed using the UIPath Robotic Enterprise Framework (REF). The Robotic Enterprise Framework utilizes the concept of a state machine for defining a process’s execution. A state machine defines a distinct set of states that the application can exist in at any given time, and further there are transitions defined between the states. These states allow for a separation of concerns within the development process allowing for better consistency as well as providing more opportunity for reuse of common components. Five distinct states are defined within the state machine. They are listed below:

## REF States

|  |  |
| --- | --- |
| State Name | State Description |
| Initialize Process | Entry point for an automation where initialization functions are performed such as reading config files, establishing common logF\_ fields, preparing the environment for the automation, etc |
| Fetch Next Transaction Items | Logic required to fetch the next item(s) to be processed. Retrieve as many items as needed for processing. |
| Load Single Transaction Item | Load an item from the list of transactional items |
| Process Single Transaction Item | Perform the necessary business logic on the transaction item |
| End Process | Final state for an automation where any cleanup activities are performed |

## CTL\_UIPath\_Transactional\_Template Screenshot



## Obtaining The Template

The CTL\_UIPath\_Transactional\_Template is hosted in a GitLab repository. It can be cloned using this url:

https://ne1itcprhas62.ne1.savvis.net/RPA\_DEV/CTL\_UIPath\_Transactional\_Template

## Usage of The Template

Each RPA will be unique based on the business unit’s needs. As such, the specific business logic required will also be unique. The template defines the requirements for the state machine, the transitions and consistent logging practices. Within each state, it is up to the developer(s) to determine how to structure the logic as it makes sense. If it is advantageous to place logic in an external .XAML file, and then to invoke that file within a state or multiple states, that is perfectly acceptable. The only requirements for using an external .XAML file are for logging which are outlined in this document.

# Initialize Process State

Activities within this state will be run once at the beginning of a job. Any kind of environment prep, logins to applications, etc will be performed during this state. One required operation that will happen in this state is the parsing of the config file. The config file will define necessary variables for the operation of the automation such as assets, application urls, retry counts, etc. Each config file will also contain the unique value for the custom logF\_IntakeID field. The functions performed within this state are to be wrapped in a Try/Catch activity. Any caught exceptions should set the variable, exTechnicalError, to the exception object. This state allows to control the concurrency of the process runs based on different factors. Please refer to the documentation in the Cipher Knowledge base link on the usage of CheckConcurrentRuns.xaml - <https://cipher.ctlhub.com/?route=kb/manage&id=212&categoryid=10&rpa_platform_id=1>

## State Requirements In Order

|  |  |  |
| --- | --- | --- |
| Step | Action | Detail |
| 1 | Add Log Fields Activity | logF\_StateName = “Initialize Process”  logF\_TemplateVersion = “3” |
| 2 | Log Message Activity | Message = “Starting”  Level = “Info” |
| 3 | Parse Config File |  |
| 4 | Add Log Fields Activity | logF\_IntakeID = value from config  logF\_CommonApp = value from config  logF\_CommonID = value from config  logF\_CommonAction =value from config  logF\_CommonActionDescription= value from config |
| 5 | Log Message Activity | Message = “Config parsed”  Level = “Info” |
| 6 | Add Log Fields Activity | logF\_ProcessInstanceID = value from CheckConcurrentRuns workflow  logF\_bConcurrentError = value from CheckConcurrentRuns.xaml |
| 7 | Perform any additional initialization |  |
| 8 | Log Message Activity | Message = “Ending”  Level = “Info” |
| 9 | Remove Log Fields Activity | “logF\_StateName” |

## Transitions

|  |  |  |
| --- | --- | --- |
| Name | Condition | Destination |
| Process Initialized | exTechnicalError Is Nothing | Fetch Next Transaction Items |
| Technical Error | exTechnicalError IsNot Nothing | End Process |
| Concurrency Error | bConcurrencyError | End Process |

# Fetch Next Transaction Items State

Fetch the transaction items that need to be processed from external source which could be a database, web application, excel file, etc. Persist these items to a list<Dictionary<String,Object>> variable named lstTransactionItems. This state will be called as many times as needed if a workflow requires multiple fetches to obtain transaction items. Once this state is no longer able to retrieve any further transaction items, it will transition to End Process state.

The functions performed within this state are to be wrapped in a Try/Catch activity. Two kinds of exceptions can be generated to indicate the kind of exception encountered.

1. BusinessRuleException should be caught if thrown.
2. Generic Exception should be caught if thrown.

## State Requirements In Order

|  |  |  |
| --- | --- | --- |
| Step | Action | Detail |
| 1 | Add Log Fields Activity | logF\_StateName = “Fetch Next Transaction Items” |
| 2 | Log Message Activity | Message = “Starting”  Level = “Info” |
| 3 | Reset variables | Set any variables used for fetching transaction items such as lstTransactionItems to Nothing |
| 3 | Fetch next transaction item(s) |  |
| 4 | Log Message Activity | Message = “Ending”  Level = “Info” |
| 5 | Remove Log Fields Activity | “logF\_StateName” |

## Transitions

|  |  |  |
| --- | --- | --- |
| Name | Condition | Destination |
| Items Available | lstTransactionItems.Count > 0 | Load Single Transaction Item |
| No Item | lstTransactionItems Is Nothing OrElse lstTransactionItems.Count = 0 | End Process |

# Load Single Transaction Item State

Within this state, a single transaction item will be selected from the lstTransactionItems list. As soon as this item is loaded, two key logging fields are to be populated.

1. logF\_TransactionID (string)

should be set to the unique value of the item that needs to be processed. This value should be unique for the current job execution. For example, if the lstTransactionItems list contains multiple orders, then a likely unique candidate for logF\_TransactionID would be the order number as long as each order number within the current job is unique.

1. logF\_TransactionDetails (Dictionary<String,Object>)

should be set to any additional items pulled from the lstTransactionItems list. If processing orders for example, logF\_TransactionDetails may contain the CUID of the user on whose behalf the order is being worked. Any other contextually relevant information related to the current transaction item should be present in the logF\_TransactionDetails field.

Once both logF\_TransactionID and logF\_TransactionDetails have been set, use the “Add Log Fields” activity to add both fields. These fields will continue to be in scope until the current transaction item has been completely processed either successfully or with an exception. This will ensure that any log activity that occurs as part of processing a transaction item will also include the information about the transaction item. If the transactional item is retrieved from the Orchestrator queue, the entire queue item can be set to the logF\_TransactionDetails field which will include any information that is part of the queue item.

The retrieved transaction item should be stored in the dictCurrentTransactionItem dictionary. This variable will make it consistent to refer to the current item during processing. It also ensures that each iteration of getting the next transaction item can consistently reset variables.

The functions performed within this state are to be wrapped in a Try/Catch activity. Two kinds of exceptions can be generated to indicate the kind of exception encountered.

1. BusinessRuleException should be caught if thrown.
2. Generic Exception should be caught if thrown.

## State Requirements In Order

|  |  |  |
| --- | --- | --- |
| Step | Action | Detail |
| 1 | Add Log Fields Activity | logF\_StateName = “Load Single Transaction Item” |
| 2 | Remove Log Fields Activity | Remove:  logF\_TransactionID  logF\_TransactionDetails  logF\_TransactionStatus |
| 3 | Reset variables | Any variables used throughout the processing of a transaction item should be reset. For example, the exTechnicalError, exBusinessRuleError, dictCurrentTransactionItem should be set to nothing. |
| 4 | Log Message Activity | Message = “Starting”  Level = “Info” |
| 5 | Fetch single transaction item from lstTransactionItems | Use an integer counter, iTransactionItemsProcessedCount, to fetch the next item from the list |
| 6 | Set key transaction item logging fields | Assign the unique logF\_TransactionID field based on something unique for the current job execution, and set any additional related transactional data to the logF\_TransactionDetails field |
| 7 | Set retrieved transaction item to consistent variable | Assign the retrieved transaction item to the dictCurrentTransactionItem dictionary |
| 8 | Log Message Activity | Message = “Loaded transaction item”  Level = “Info” |
| 9 | Log Message Activity | Message = “Ending”  Level = “Info” |
| 10 | Remove Log Fields Activity | “logF\_StateName” |

## Transitions

|  |  |  |
| --- | --- | --- |
| Name | Condition | Destination |
| Transaction Item Loaded | ﻿dictCurrentTransactionItem IsNot Nothing | Process Single Transaction Item |
| No More Items | ﻿iTransactionItemsProcessedCount >= lstTransactionItems.Count | Fetch Next Transaction Items |
| Should Stop | ShouldStop | End Process |

# Process Single Transaction Item State

Perform the business-specific logic required to process the transaction item. The majority of business logic is likely to exist within this state. After all functions have completed, increment the iTransactionItemsProcessedCount integer to indicate the transaction item has been processed. As this state likely represents the majority of the business logic required to complete the automation of an item, it is encouraged to add additional informational log messages within the various steps. If a transaction item takes 5 minutes to process, but there is only two log events at the beginning and end of this state, it will be difficult to do any further analysis into how those 5 minutes were spent. It is also likely that this state will include logic that overlaps with other automations. Reuse of common functionality is encouraged.

The functions performed within this state are to be wrapped in a Try/Catch activity. Two kinds of exceptions can be generated to indicate the kind of exception encountered.

1. BusinessRuleException should be caught if thrown. and set the variable, exBusinessRuleError, to the exception object.
2. Generic Exception should be caught if thrown and set the variable, exTechnicalError, to the exception object.

## State Requirements In Order

|  |  |  |
| --- | --- | --- |
| Step | Action | Detail |
| 1 | Add Log Fields Activity | logF\_StateName = “Process Single Transaction Item” |
| 2 | Log Message Activity | Message = “Starting”  Level = “Info” |
| 3 | Run business logic to process transaction | Use appropriate functions to fully process current transaction item |
| 4 | Log Message Activity | Message = “Ending”  Level = “Info” |
| 5 | Remove Log Fields Activity | “logF\_StateName” |

## Transitions

|  |  |  |
| --- | --- | --- |
| Name | Condition | Destination |
| Success | ﻿exTechnicalError Is Nothing And exBusinessRuleException Is Nothing | Load Single Transaction Item |
| Business Error | ﻿exTechnicalError Is Nothing And exBusinessRuleException IsNot Nothing | Load Single Transaction Item |
| Technical Error | ﻿exTechnicalError IsNot Nothing And exBusinessRuleException Is Nothing | Load Single Transaction Item |

Within each Transition out of the Process Single Transaction Item State, it is required to set the status of the transaction. The logF\_TransactionStatus field will indicate the result of processing the transaction item as a Success, Business Exception or Technical Exception. The table below defines the values based on the transition.

## logF\_TransactionStatus Assignment

|  |  |
| --- | --- |
| Transition | logF\_TransactionStatus Value |
| Success | ﻿Success |
| Business Error | Business Exception |
| Technical Error | Technical Exception |

# End Process

This is the final state of any workflow. Once this state has been reached, no further states are possible. Within this state, any kind of cleanup that hasn’t already been done can be performed.

## State Requirements In Order

|  |  |  |
| --- | --- | --- |
| Step | Action | Detail |
| 1 | Add Log Fields Activity | logF\_StateName = “End Process” |
| 2 | Log Message Activity | Message = “Starting”  Level = “Info” |
| 3 | Perform any final cleanup |  |
| 4 | Log Message Activity | Message = “Ending”  Level = “Info” |
| 5 | Remove Log Fields Activity | “logF\_StateName” |

# Transition Logging

Within each transition between states, additional log events need to be present. Since additional logic can be performed within a transition, knowing the timings as well as any related events that occur within a transition is crucial.

## Transition Requirements In Order

|  |  |  |
| --- | --- | --- |
| Step | Action | Detail |
| 1 | Add Log Fields Activity | logF\_TransitionName = “<Exact name of transition as defined in respective Transitions tables above>”  logF\_TransitionSource = “<Exact name of the state the transition is originating from>”  logF\_TransitionDestination = “<Exact name of the state the transition is transitioning into>” |
| 2 | Log Message Activity | Message = “Transitioning”  Level = “Info” |
| 3 | Perform any additional transition logic |  |
| 4 | Remove Log Fields Activity | “logF\_TransitionName”  “logF\_TransitionSource”  “logF\_TransitionDestination” |

# Exception Logging

Exceptions are a common part of any workflow, and within this standard, they are used both to indicate expected and unexpected conditions. As the contents of an exception and its reasons can vary, a standard definition of logging these exceptions is provided. Within the catch block of a Try/Catch activity, follow these guidelines.

## Exception Logging Requirements In Order

|  |  |  |
| --- | --- | --- |
| Step | Action | Detail |
| 1 | Add Log Fields Activity | logF\_ExceptionDetails = “<Exception body>”  logF\_ExceptionType = “<Business|Technical>” |
| 2 | Log Message Activity | Message = “Exception occurred”  Level = “Error” |
| 3 | Remove Log Fields Activity | “logF\_ExceptionDetails”  “logF\_ExceptionType” |

# Other Required Logging

Within the business logic that is unique to an RPA, it is likely and encouraged to look for opportunities to reuse existing workflows. Logins to common applications, taking screenshots and sending an email are examples of logic that can likely be reused between automations. One of the techniques to reuse existing logic is to invoke an external workflow. This activity allows for including the functionality provided by an already written .XAML file. There is a folder called CTL\_Common as part of the template which will be populated over time to include generic, reusable components.

Any time an external .XAML file is invoked from within a workflow, some consistent logging messages need to be added. One of the default fields provided by UIPath when using the “Log Message” activity is the fileName. Any Log Message activities that occur within an external .XAML file will include the name of that file by default.

## External .XAML File Logging Requirements In Order

|  |  |  |
| --- | --- | --- |
| Step | Action | Detail |
| 1 | Log Message Activity | Message = “Starting”  Level = “Info” |
| 2 | Perform business logic | Run any activities as required. Any additional Log Message activities will automatically include the fileName as a default field. |
| 3 | Log Message Activity | Message = “Ending”  Level = “Info” |

# Default Metrics

When the standards outlined within this document are followed, a default set of metrics will be available for any workflow. The table below outlines these metrics.

## Default Metrics Table

|  |  |  |
| --- | --- | --- |
| # | Metric Name | Detail |
| 1 | Number of Bots | Distinct number of bots utilized |
| 2 | Number of Processes | Distinct number of processes |
| 3 | Number of Executions | Total number of executions |
| 4 | A breakdown of (1, 2, 3) by Business Unit |  |
| 5 | Time-based stats | View of execution metrics over time including   1. Completed without error 2. Completed with error 3. Not completed |
| 6 | Top 5 processes with errors | View of most common processes with any kind of error |
| 7 | Top 5 handled errors | View of most common processes with caught Business or Technical exceptions |
| 8 | Top 5 unhandled errors | View of most common processes with uncaught exceptions |
| 9 | Bot utilization | Number of hours used by the bots |
| 10 | Average execution time | Average time taken broken down by   1. Process 2. Business Unit |
| 11 | SLA breaches (if applicable) | View of the number of executions that exceeded SLA where SLA is measured from the time a transaction item is loaded until the time that item is successfully processed |
| 12 | Historical rollups of stats | Views of previous time period stats to show trending |
| 13 | ROI Saved | A dollar amount calculated based on externally provided business multipliers coupled with the time taken from loading a transaction item until the time that item is successfully processed |
| 14 | FTE Saved | A numeric indicator of savings based on externally provided business multipliers coupled with the time taken from loading a transaction item until the time that item is successfully processed |