Manual del Framework de Practia en UiPath

Contenido

Acerca del Framework

Este Framework es una plantilla (template) que permite a los desarrolladores RPA de Practia diseñar procesos que mínimamente obtienen datos, los procesan y emiten los resultados producidos; a grandes rasgos, su propósito es promover un entendimiento común de todos los procesos desarrollados en el área y, por supuesto, facilitar el mantenimiento de estos.

Al ser un marco de referencia para los desarrolladores, el Framework no tiene el propósito de brindar soluciones específicas y de profundidad para los distintos procesos que pueden llegarse a implementar; por el contrario, y como fue indicado al inicio, se debe entender como un marco de trabajo que permite a los desarrolladores hablar en un mismo "lenguaje".

Volver al Contenido

Glosario

- **Transacción:** Representa un conjunto de datos obtenido en el estado *Data Mapping* y procesado en el estado *Data Processing;* una vez procesado, puede generar un nuevo conjunto de datos de salida que es exportado en el estado *Data Output*.

  Dependiendo del proceso y especialmente de los datos de entrada (Input), una transacción puede ser de cualquier tipo como, por ejemplo, un *DataTable* proveniente de un archivo de Excel, un *QueueItem*, una colección de *MailMessages*, un *String* proveniente de un archivo de texto, entre otros.

- **Directorio raíz:** El directorio raíz es el espacio de trabajo en el que se alojarán todos los elementos necesarios para el funcionamiento del proceso; por defecto, el directorio raíz cuenta con los siguientes subdirectorios: *Input, Output, Logs* y *Settings*.

Volver al Contenido

Componentes del Framework

La siguiente tabla detalla los workflows invocados por los estados que componen el Framework y el orden de invocación de estos.

Ubicación del componente	Componente	Estado que lo invoca
Raíz del Template	Main.xaml	
Settings Load	SettingsLoad.xaml	Settings load
Settings Load	TryToCloseSettingsLoad.xaml	Settings Load
Inicialization	ValidateInternetAvailability.xaml	Inicialization
Inicialization	ValidateDictionaryConstants.xaml	Initialization
Inicialization	ValidateFoldersExistance.xaml	Initialization
Initialization	CreateFolder.xaml	Initialization

Inicialization	KillAllProcesses.xaml	Inicialization
Inicialization	StartAllProcesses.xaml	Inicialization
Data Mapping	GetTransactionItem.xaml	Data Mapping
Data Processing	ProcessTransactionItem.xaml	Data Processing
Data Processing	SetTransactionItemStatus.xaml	Data Processing
Exception Handler	GetUiPathExecutionLog.xaml	Exception Handler
Exception Handler	TakeScreenshot.xaml	Exception Handler

Volver al Contenido

Dependencias

A continuación, se listan las dependencias necesarias para el funcionamiento del Framework:

- Microsoft Activities 1.0.1
- Microsoft.Activities.Extensions 2.0.6.9
- UiPath.Credentials.Activities 1.1.6479.13204
- UiPath.Excel.Activities 2.7.2
- UiPath.Mail.Activities 1.7.2
- UiPath.System.Activities 19.10.1
- UiPath.UIAutomation.Activities 19.11.1

Volver al Contenido

Variables globales

Las variables globales son aquellas cuyo alcance se encuentra el flujo principal de trabajo (main.xaml). Estas variables se utilizan para almacenar información que estará disponible en tiempo de ejecución y podrán ser accedidas desde cualquier estado.

En la columna *Escrita en* se indica en qué estado se inicializa y/o modifica la variable y en la columna *Leída en* se indica en que estados y/o workflows es leída.

Nombre de la variable	Tipo de variable	Escrita en	Leída en
bool_IsTransactionItemConsumer	Boolean	Settings load	Data Mapping Data Processing
bool_processConsumesAssets	Boolean	Settings load	SettingsLoad.xaml Initialization
bool_returnToDataMapping	Boolean	Data Processing Data Output Exception Handler	Data Output Exception Handler

bool_StateStatus	Boolean	Main.xaml	Main.xaml
dicConfig	Dictionary <String, String>	SettingsLoad.xaml	Main.xaml GetTransactionItem.xaml ProcessTransactionItem.xaml SetTransactionItemStatus.xaml
dt_processesToKill	DataTable	Settings load	KillAllProcesses.xaml
dt_processesToStart	DataTable	Settings load	StartAllProcesses.xaml
ex_BusinessRuleException	Exception	Data Processing Data Output Exception Handler	Data Processing Data Output Exception Handler SetTransactionItemStatus.xaml
ex_FlowException	Exception	Main.xaml	Main.xaml SetTransactionItemStatus.xaml
int_RecoverRetries	Int32	Initialization Cleaning Process	Cleaning Process
stopwatchState	Stopwatch	Main.xaml	Main.xaml
stopwatchStateMachine	Stopwatch	Main.xaml	Main.xaml
str_executionTime	String	Main.xaml	Main.xaml

Volver al Contenido

Settings Load (Carga de configuraciones)

En este estado se genera el diccionario de datos compuesto por el conjunto de configuraciones necesario para que el proceso pueda ejecutarse; este diccionario se genera a partir del archivo *Settings.xlsx*.

El proceso finalizará de manera forzada (a través de la actividad *Terminate Workflow*) si el diccionario de configuraciones (*dicConfig*) no se encuentra inicializado y no pasará al estado *Exception Handler*; el detalle del error será notificado a través de la actividad *Log Message* con el nivel *Fatal.*

El template de este Framework tiene una carpeta llamada Settings File Model que cuenta con una plantilla modelo de dicho archivo.

Adicionalmente, dentro de la secuencia *Sequence - Define Process Folder* se definen dos cuestiones esenciales del proceso:

- **¿De dónde se obtiene la ruta del directorio raíz del proceso?**: A partir de la variable *bool_processConsumesAssets* se define si la ruta del directorio de raíz se obtiene localmente o a través de un *Asset* de *Orchestrator.*

En la variable *str_ProcessRootDirectory* se define el nombre del *Asset* que contiene la ruta completa del directorio raíz o se indica directamente la ruta completa en caso de que el directorio raíz se defina de manera local.

- **¿El proceso consume *QueueItems* de una *Queue* de Orchestrator?**: A partir de la variable *bool_IsTransactionItemConsumer* se define si el proceso debe consumir *QueueItems* de una *Queue.*

Para más información acerca de estas tres operaciones, dirigirse al apartado Configuración inicial del Framework.

Nombre de la variable	Tipo de variable	Leída en
bool_processConsumesAssets	Boolean	SettingsLoad.xaml
str_ProcessRootDirectory	String	SettingsLoad.xaml
bool_IsTransactionItemConsumer	Boolean	Data Mapping – Data Processing

El estado *Settings load* invoca el siguiente flujo de trabajo:

SettingsLoad.xaml

En este flujo de trabajo se lee el archivo *Settings.xlsx* y se generan tres argumentos de salida a raíz de dicha lectura, el primero de ellos es el diccionario de configuraciones que será utilizado a lo largo del proceso (*out_dicConfig*), el segundo es una tabla de datos con todos los procesos que deben ser cerrados de manera forzada antes de iniciar la toma de datos (*out_processesToKill*) y el tercero es una tabla de datos con todos los procesos que deben ser iniciados antes de iniciar la toma de datos (*out_processesToStart*).

El flujo de trabajo cuenta con los siguientes argumentos:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	out_dicConfig	Out	dicConfig
DataTable	out_processesToKill	Out	dt_processesToKill
DataTable	out_processesToStart	Out	dt_processesToStart
Boolean	bool_processConsumesAssets	In	bool_processConsumesAssets
String	in_str_ProcessRootDirectory	In	str_ProcessRootDirectory

TryToCloseSettingsFile.xaml

Para una ejecución adecuada de la robotización, el archivo *Settings.xlsx* no debe encontrarse en uso por parte de ninguna aplicación en el instante en el que se realiza la lectura del archivo por parte del robot.

Dentro del flujo de trabajo *SettingsLoad.xaml* se ejecuta este flujo el cual valida si el archivo se encuentra en uso, en caso de presentarse dicha situación, se le pregunta al usuario si desea cerrar el archivo de configuración *Settings.xlsx* en caso de que se encuentre abierto.

El proceso finalizará de manera forzada si el usuario no confirma el cierre del archivo; por el contrario, si el usuario confirma el cierre, el archivo será cerrado y los cambios que se hayan efectuado sobre el mismo no serán guardados.

No cuenta con argumentos de entrada ni de salida.

CloseSettingsFile.xaml

Este workflow puede ser reemplazado por el workflow *TryToCloseSettingsFile.xaml* dentro del flujo de trabajo *SettingsLoad.xaml*; a diferencia de este último, *CloseSettingsFile.xaml* procede a cerrar de manera forzada al archivo de configuración *Settings.xlsx* en caso de que se encuentre en uso.

No cuenta con argumentos de entrada ni de salida.

Excepción controlada

El flujo lanza una excepción a través de la actividad *Throw - Settings File doesn't exist exception* si el archivo *Settings.xlsx* no se encuentra en la ruta correspondiente *[Directorio Raíz]\Settings*.

Se debe tener en cuenta que es una excepción con un mensaje genérico que, en caso de ser necesario, puede ser modificado según las necesidades del proceso.

Por otro lado, la existencia del archivo *Settings.xlsx* es esencial para el funcionamiento del Framework y su inexistencia provoca el lanzamiento de excepciones no controladas que no pueden ser atrapadas por este último.

Transiciones

A continuación, se detallan las transiciones que el estado *Settings load* puede tener:

Nombre	Condición	Estado a transicionar	Descripción
Successful Configuration	bool_StateStatus	Inicialization	Si en el estado no se genera ningún tipo de excepción, es posible pasar al siguiente estado.
Settings Load Exception	ex_flowException IsNot Nothing	Exception Handler	Si se genera una excepción de cualquier tipo se transiciona al estado *Exception Handler*.

Volver al Contenido

Inicialization (Inicio del entorno de trabajo)

En este estado se realizan todas las actividades necesarias para garantizar que el ambiente en el que el robot debe operar se encuentra en las condiciones adecuadas; en otras palabras, en este estado se validan las precondiciones necesarias para que el proceso se pueda ejecutar sin inconvenientes. Algunas operaciones que

se pueden realizar en este estado son: la validación de la existencia de directorios locales, la conexión a internet, el acceso a un servidor remoto, inicios de sesión a determinadas aplicaciones, entre otras.

Recuperación del robot ante excepciones durante la Inicialización

Si se emite una excepción durante la ejecución de alguna actividad definida dentro de este estado, el robot transicionará a los estados *Exception Handler* (para controlar la excepción) y *Cleaning Process* (para limpiar el ambiente) y procederá a regresar al estado *Initialization* para ejecutar nuevamente todas las actividades.

La cantidad de reintentos de recuperación del robot se define en el campo *InitMaxRetries* de la solapa *Constants* del archivo *Settings.xlsx*; por defecto, la cantidad máxima es de tres (3).

El estado *Inicialization* invoca los siguientes flujos de trabajo:

ValidateDictionaryConstants.xaml

Este flujo de trabajo valida los valores definidos en las cuatro constantes del diccionario de configuraciones:

- **InitMaxRetries:** Es una constante utilizada para definir la cantidad de reintentos de recuperación que el robot debe realizar en caso de que alguna actividad dentro del estado de inicialización falle. Solo admite valores numéricos de tipo *Integer.*
- **ReturnToDataMappingOnGenericException:** Es una constante utilizada para definir si el proceso debe regresar al estado *Data Mapping* en caso de que se emita una excepción no controlada en los estados *Data Mapping, Data Processing* o *Data Output.* Solo admite valores de tipo *Boolean (True – False).*
- **ReturnToDataMappingOnBRE:** Es una constante utilizada para definir si el proceso debe regresar al estado *Data Mapping* en caso de que se emita una *Business Rule Exception* en los estados *Data Mapping, Data Processing* o *Data Output.* Solo admite valores de tipo *Boolean (True – False)*
- **ReturnToDataMappingOnSuccess:** Es una constante utilizada para definir si el proceso debe regresar al estado *Data Mapping* en caso de que se hayan ejecutado con éxito todas las actividades del estado *Data Output.* Solo admite valores de tipo *Boolean (True – False)*

El flujo de trabajo cuenta con el siguiente argumento de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	in_dicConfig	In	dicConfig

ValidateInternetAvailability.xaml

Este flujo de trabajo valida que en la máquina en la que se ejecuta el proceso se cuente con conexión a internet.El flujo de trabajo cuenta con el siguiente argumento de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	in_dicConfig	In	dicConfig

Excepción controlada

El flujo de trabajo lanza una excepción en la actividad *If - Internet is available* cuyo mensaje puede ser modificado según las necesidades del proceso, claramente, la excepción es lanzada si la máquina en la que se ejecuta el proceso no cuenta con Internet.

Para modificar el mensaje, se debe acceder a la solapa *Exceptions* del archivo *Settings.xlsx,* el mensaje debe ser indicado en la columna *Value* bajo la clave *ValidateInternetAvailabilityException.*

ValidateFoldersExistance.xaml

Este flujo de trabajo valida la existencia de los subdirectorios *Logs, Input* y *Output,* si los subdirectorios no existen, el proceso procede a crearlos. La creación y validación de subdirectorios se realiza a través del flujo *CreateFolder.xaml*; el desarrollador puede reutilizar este flujo para añadir otros subdirectorios que requieran ser utilizados por el robot.

El flujo de trabajo cuenta con el siguiente argumento de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	in_dicConfig	In	dicConfig

CreateFolder.xaml

Este flujo de trabajo valida la existencia de un subdirectorio, si no existe, el proceso procede a crearlos; adicionalmente, en caso de ser necesario, el flujo permite eliminar todos los archivos contenidos en el subdirectorio a partir de la variable *in_bool_DeleteFiles* de tipo *Boolean.*

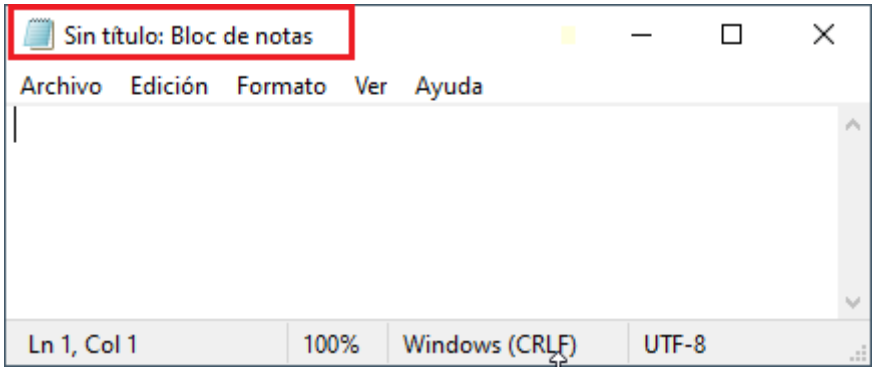
El flujo de trabajo cuenta con los siguientes argumentos de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
String	in_str_Folder	In	*Indicado por el developer*
Boolean	in_bool_DeleteFiles	In	True o False

KillAllProcesses.xaml

Este flujo de trabajo procede a forzar el cierre (matar) los procesos indicados en la solapa *ProcessesToKill* del archivo de configuración *Settings.xlsx.*

En la solapa *ProcessesToKill* se debe indicar el texto que aparece en la barra de título de la ventana de Windows del proceso.



Barra de título resaltada en el recuadro rojo de la imagen

El flujo de trabajo cuenta con el siguiente argumento de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
DataTable	in_dt_processesToKill	In	dt_processesToKill

StartAllProcesses.xaml

Este flujo de trabajo procede a abrir todos los procesos indicados en la solapa *ProcessesToStart* del archivo de configuración *Settings.xlsx* en caso de que los procesos no se encuentren en ejecución.

El flujo de trabajo cuenta con el siguiente argumento de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
DataTable	in_dt_processesToStart	In	dt_processesToStart

LogInWithCredentials.xaml

Este flujo de trabajo permite realizar el *Log In* a una aplicación a partir de unas credenciales que pueden ser obtenidas desde *Orchestrator* o desde el *Administrador de credenciales de Windows*.

El flujo deberá ser replicado en función a la cantidad de aplicaciones a las que se debe loguear el proceso.

El flujo de trabajo cuenta con el siguiente argumento de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Boolean	in_bool_processConsumesAssets	In	bool_processConsumesAssets
String	in_CredentialsName	In	*Definido por el desarrollador*
Dictionary<String, String>	in_dicConfig	In	dicConfig

Transiciones

A continuación, se detallan las transiciones que el estado *Initialization* puede tener:

Nombre	Condición	Estado a transicionar	Descripción
Successful Initialization	bool_StateStatus	Data Mapping	Si en el estado no se genera ningún tipo de excepción, es posible pasar al siguiente estado.
Initialization Exception	ex_flowException IsNot Nothing	Exception Handler	Si se genera una excepción de cualquier tipo se transiciona al estado *Exception Handler*.

Volver al Contenido

Data Mapping (Mapeo de datos)

En este estado se obtienen las transacciones a partir de los datos de entrada que pueden (y deberían) ser obtenidos del subdirectorio *Input* u otro repositorio de datos.

Es responsabilidad del desarrollador RPA modificar el booleano *bool_StateStatus* ya que, a partir de este booleano, se determina hacía qué estado se debe transicionar. Si el booleano se define en *True* se transicionará al estado *Data Processing* (cuando se ha obtenido una transacción) y transicionará al estado *Cleaning Process* cuando el booleano se define en *False*.

Obtención iterativa de nuevas transacciones

El Framework permite la obtención de nuevas transacciones de manera iterativa una vez finalizado el procesamiento de una transacción ya sea de manera exitosa o si durante su procesamiento se emitió una excepción no controlada o una *Business Rule Exception.*

Para definir el comportamiento del robot en estos tres casos, la solapa *Constants* del archivo *Settings.xlsx* cuenta con los siguientes campos:

- **ReturnToDataMappingOnGenericException:** Permite determinar si el robot debe regresar al estado *Data Mapping* en caso de que se emita una excepción no controlada en los estados *Data Mapping, Data Processing* o *Data Output.* Solo admite valores de tipo *Boolean (True – False).*
- **ReturnToDataMappingOnBRE:** Permite determinar si el robot debe regresar al estado *Data Mapping* en caso de que se emita una *Business Rule Exception* en los estados *Data Mapping, Data Processing* o *Data Output.* Solo admite valores de tipo *Boolean (True – False)*
- **ReturnToDataMappingOnSuccess:** Permite determinar si el robot debe regresar al estado *Data Mapping* en caso de que se hayan ejecutado con éxito todas las actividades del estado *Data Output.* Solo admite valores de tipo *Boolean (True – False)*

El estado *Data Mapping* invoca los siguientes flujos de trabajo:

GetTransactionItem.xaml

Este flujo de trabajo es ejecutado en el estado *Data Mapping* si la variable *bool_IsTransactionItemConsumer* es igual a *True*; en otras palabras, será ejecutado si el Framework se ha configurado para que el proceso obtenga *QueueItems* de una *Queue*.

El flujo de trabajo cuenta con los siguientes argumentos de entrada y salida:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	in_dicConfig	In	dicConfig
String	in_QueueName	In	dicConfig("QueueName").ToString
QueueItem	out_TransactionItem	Out	qi_transactionItem

Transiciones

A continuación, se detallan las transiciones que el estado *Data Mapping* puede tener:

Nombre	Condición	Estado a transicionar	Descripción
Successful Data Mapping	bool_StateStatus	Data Processing	Si en el estado se obtiene una transacción (de cualquier tipo), es posible transicionar al estado *Data Processing*. Es importante aclarar que es responsabilidad del desarrollador validar que se haya obtenido una transacción y, a su vez, asignarle la variable *bool_StateStatus* el valor *True.*
No Transaction Data	bool_StateStatus.Equals(False) and (ex_FlowException Is Nothing and ex_BusinessRuleException is Nothing)	Data output	Si en el estado NO se obtiene una transacción significa que el proceso debe proceder con el proceso de limpieza del ambiente. Es importante aclarar que es responsabilidad del desarrollador validar que NO se haya obtenido una transacción y, a su vez, asignarle a la variable *bool_StateStatus* el valor *False.*

Data Mapping Exception	bool_StateStatus.Equals(False) and (ex_FlowException IsNot Nothing Or ex_BusinessRuleException IsNot Nothing)	Exception Handler	Si se genera una excepción de cualquier tipo se transiciona al estado *Exception Handler*.

Volver al Contenido

Data Processing (Procesamiento de datos)

En este estado se procesan las transacciones que fueron obtenidas en el anterior estado.

El estado *Data Processing* invoca los siguientes flujos de trabajo:

ProcessTransactionItem.xaml

Este flujo de trabajo es ejecutado por el estado *Data Processing* si la variable *bool_IsTransactionItemConsumer* es igual a *True* y la variable *qi_transactionItem* no está vacía; en otras palabras, será ejecutado si el Framework ha obtenido un *QueueItem* a partir de una *Queue*.

El flujo de trabajo cuenta con los siguientes argumentos de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	in_dicConfig	In	dicConfig
QueueItem	in_TransactionItem	In	qi_transactionItem

SetTransactionItemStatus.xaml

Este flujo de trabajo es ejecutado en el bloque *Finally* del *Try Catch* incorporado en el estado *Data Processing*; este flujo será ejecutado si la variable *bool_IsTransactionItemConsumer* es igual a *True* y la variable *qi_transactionItem* no está vacía. En otras palabras, el proceso será ejecutado si el Framework ha procesado un *QueueItem* obtenido de una *Queue*.

El flujo de trabajo cuenta con los siguientes argumentos de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	in_dicConfig	In	dicConfig
QueueItem	in_TransactionItem	In	qi_transactionItem
Exception	in_flowException	In	ex_flowException
Exception	in_BusinessRuleException	In	ex_BusinessRuleException

Excepciones no controladas

El flujo de trabajo *SetTransactionItemStatus.xaml* se encuentra dentro de un *Try – Catch* para poder atrapar excepciones no controladas que podrían ocurrir durante el cambio de estado de un *QueueItem;* si una excepción no controlada llegase a ser lanzada durante la ejecución de este flujo, el mensaje de dicha excepción se loguea acompañado por un texto genérico que puede ser modificado dentro de la solapa *Exceptions* del archivo *Settings.xlsx,* el mensaje debe ser indicado en la columna *Value* bajo la clave *SetTransactionStatusException.*

Transiciones

A continuación, se detallan las transiciones que el estado *Data Processing* puede tener:

Nombre	Condición	Estado a transicionar	Descripción
Successful Data Processing	bool_StateStatus	Data Output	Si en el estado se procesa una transacción con éxito, se transiciona al estado *Data Output*.
Data Mapping Exception	ex_FlowException IsNot Nothing or ex_BusinessRuleException IsNot Nothing	Exception Handler	Si se genera una excepción de cualquier tipo se transiciona al estado *Exception Handler*.

Volver al Contenido

Data Output (Exportación de datos)

En este estado se exportan los datos de salida (output) obtenidos a partir de la transacción procesada.

El estado no invoca ningún flujo de trabajo.

Transiciones

A continuación, se detallan las transiciones que el estado *Data Output* puede tener:

Nombre	Condición	Estado a transicionar	Descripción
Data Output Total Completion	bool_StateStatus and bool_returnToDataMapping.Equals (False)	Cleaning Process	Los datos de salida fueron exportados con éxito y el proceso NO debe solicitar una nueva transacción.

Get next Transaction	bool_StateStatus and bool_returnToDataMapping	Data Mapping	Los datos de salida fueron exportados con éxito y el proceso DEBE solicitar una nueva transacción.
Data Output Exception	ex_FlowException IsNot Nothing or ex_BusinessRuleException IsNot Nothing	Exception Handler	Si se genera una excepción de cualquier tipo se transiciona al estado *Exception Handler*.

Volver al Contenido

Cleaning Process (Proceso de limpieza)

En este estado se realizan todas las tareas de limpieza necesarias para garantizar que el entorno quede en el estado en el que se encontraba antes de ejecutar el proceso; entre las tareas de limpieza más comunes se encuentra el cierre de aplicaciones, cierre de sesiones de dichas aplicaciones y limpieza de las colecciones para liberar memoria.

El estado no invoca ningún flujo de trabajo; sin embargo, cuenta con dos secuencias vacías, en la primera de ellas se deben agregar las actividades que limpien las colecciones de memoria y la segunda debe involucrar las actividades de cierre de sesión y cierre de aplicaciones.

Este estado se encuentra dentro de un *Try – Catch* para poder atrapar excepciones no controladas que podrían ocurrir durante el proceso de limpieza*;* si una excepción no controlada llegase a ser lanzada durante la ejecución de alguna actividad, el mensaje de dicha excepción se loguea acompañado por un texto genérico que puede ser modificado dentro de la solapa *Exceptions* del archivo *Settings.xlsx,* el mensaje debe ser indicado en la columna *Value* bajo la clave *CleaningProcessException.*

Ya sea que se procesen todas las acciones dentro de este estado o se lance una excepción no controlada, se transiciona al estado Final State.

Transiciones

A continuación, se detallan las transiciones que el estado *Cleaning process* puede tener:

Nombre	Condición	Estado a transicionar	Descripción
Cleaning process' Completion	bool_StateStatus And (int_RecoverRetries >= Integer.Parse(dicConfig("InitMaxRetries")) Or int_RecoverRetries.Equals(0))	Final State	Una vez se ejecuten todas las secuencias o se lance una excepción no controlada, se transiciona al estado final.

Recover	bool_StateStatus And Not int_RecoverRetries.Equals(0) And int_RecoverRetries < Integer.Parse(dicConfig("InitMaxRetries"))	Initialization	Si alguna actividad del estado Initialization falla y NO se han superado la cantidad de reintentos, se transiciona al estado *Initialization.*

Volver al Contenido

Exception Handler (Controlador de excepciones)

En este estado se reciben todas las excepciones emitidas por los otros estados; su propósito es tratar dichas excepciones realizando las acciones correctivas que requiera el proceso.

Este estado se encuentra dentro de un *Try – Catch* para poder atrapar excepciones no controladas que podrían ocurrir durante el manejo de excepciones*;* si una excepción no controlada llegase a ser lanzada durante la ejecución de alguna actividad, el mensaje de dicha excepción se loguea acompañado por un texto genérico que puede ser modificado dentro de la solapa *Exceptions* del archivo *Settings.xlsx,* el mensaje debe ser indicado en la columna *Value* bajo la clave *ExceptionHandlerException.*

Ya sea que se procesen todas las acciones dentro de este estado o se lance una excepción no controlada, el estado transicionará al estado Cleaning Process o Data Mapping según como se haya configurado el proceso.

El Framework recibe dos tipos de excepciones:

Nombre de la variable	Estado que genera la excepción	Tratamiento
ex_flowException	Settings load, Inicialization, Data Mapping, Data Processing y Data Output	Se genera un *Log Message* con nivel *Fatal* que incluye el mensaje de la excepción.
ex_BusinessRuleException	Data Mapping, Data Processing y Data Output	Se genera un *Log Message* con nivel *Error* que incluye el mensaje de la excepción.

El estado *Exception Handler* invoca los siguientes flujos de trabajo:

TakeScreenshot.xaml

Este flujo de trabajo realiza una captura de pantalla de la máquina en la que se ejecuta el robot luego de que una excepción haya sido lanzada, por defecto, las capturas de pantalla se alojan en la carpeta *Logs*.

El flujo de trabajo cuenta con el siguiente argumento de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	in_dicConfig	In	dicConfig

GetUiPathExecutionLog.xaml

Este flujo de trabajo realiza una copia del archivo *yyyy-mm-dd_Execution.log* (generado diariamente por el robot de UiPath) dentro del subdirectorio *Logs* del directorio raíz.

El flujo de trabajo cuenta con el siguiente argumento de entrada:

Tipo de variable	Nombre	Tipo de Argumento	Valor
Dictionary<String, String>	in_dicConfig	In	dicConfig

Transiciones

A continuación, se detallan las transiciones que el estado *Exception Handler* puede tener:

Nombre	Condición	Estado a transicionar	Descripción
Exception Handler Completion	bool_StateStatus and bool_returnToDataMapping.Equals(False)	Cleaning Process	Una vez se ejecuten todas las actividades o se lance una excepción no controlada y, además, el proceso NO tenga que solicitar una nueva transacción en *Data Mapping*, se transiciona al estado *Cleaning Process*.
Get next Transaction	bool_StateStatus and bool_returnToDataMapping	Data Mapping	Una vez se ejecuten todas las actividades o se lance una excepción no controlada y, además, el proceso TENGA que solicitar una nueva transacción, se transiciona al estado *Data Mapping*.

Volver al Contenido

Final State

En este estado se notifica el tiempo total de ejecución del proceso a través de un *Log Message*.

Telemetría

El Framework cuenta con dos secuencias que permiten medir el tiempo de ejecución del proceso por completo, cada uno de los estados y cada flujo que sea invocado en dichos estados.

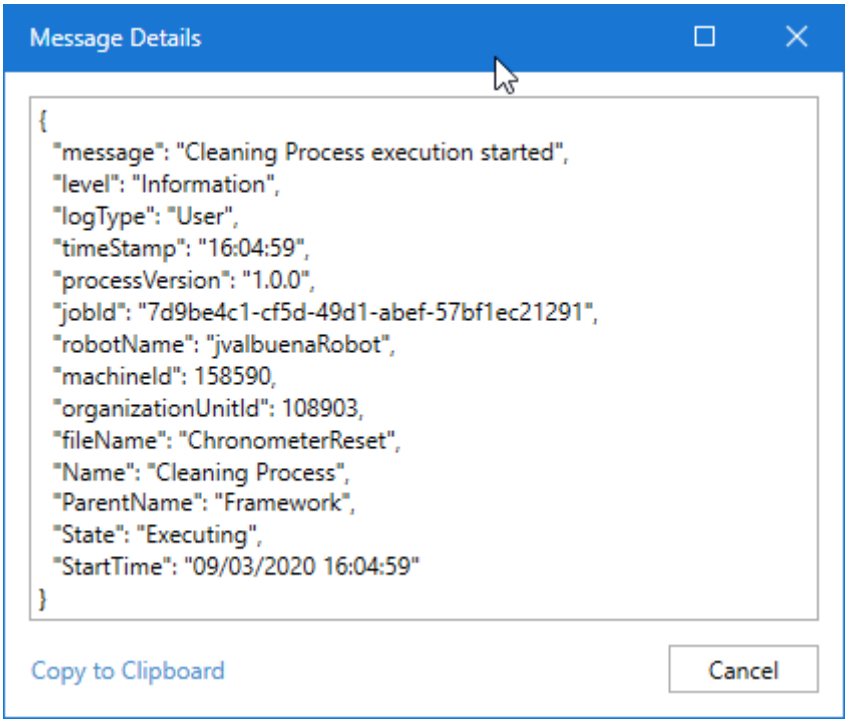
La primera secuencia inicia el cronómetro y genera un Log Message indicando el inicio del Estado / Secuencia; por otro lado, la segunda secuencia detiene el cronómetro y genera un Log Message indicando la finalización del Estado / Secuencia.

El desarrollador deberá decidir los flujos de trabajo en los que desea realizar la medición de tiempos de ejecución.

ChronometerReset.xaml

Este proceso inicia la medición del tiempo de ejecución de un Estado y/o Secuencia. El Log Message generado cuenta con los siguientes *Log Fields* personalizados:

- Name: Nombre del Estado o Flujo cuyo tiempo de ejecución será medido.
- ParentName: Nombre del Estado o Flujo padre que invoca al flujo cuyo tiempo será medido.
- State: Indica el estado del Flujo o Estado, en este caso, el estado se define como *Executing* (En Ejecución)
- StartTime: Indica la fecha en la que se inició la ejecución del Estado o Flujo.



El flujo de trabajo cuenta con los siguientes argumentos de entrada / salida:

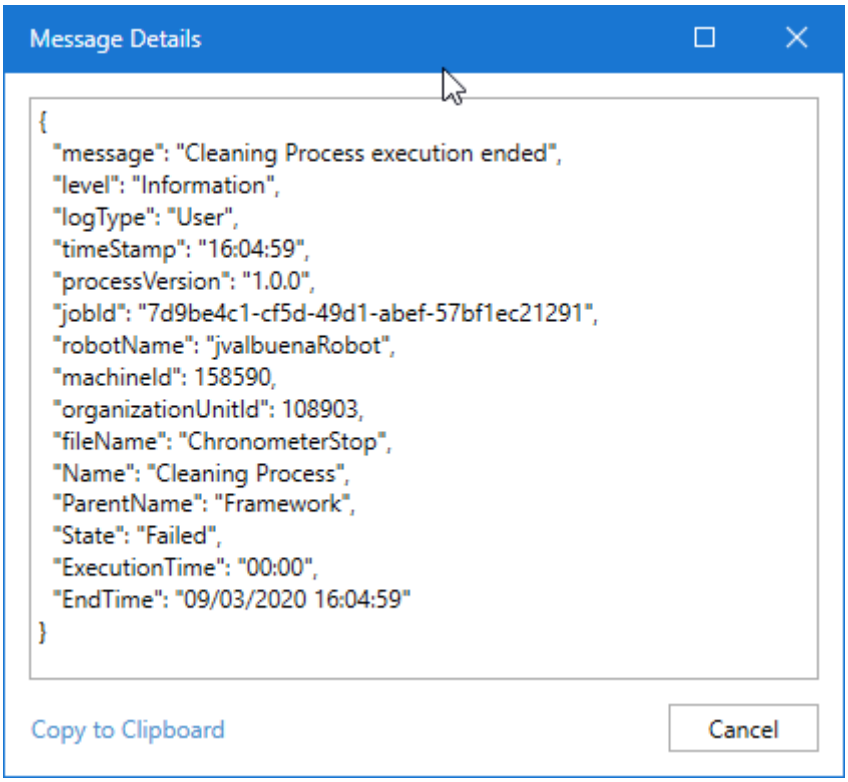
Tipo de variable	Nombre	Tipo de Argumento	Valor

String	in_str_Name	In	Nombre del Flujo / Estado
String	In_str_ParentName	In	Nombre del Flujo / Estado padre que invoca al flujo
String	In_str_State	In	Estado en el que se encuentra el flujo
String	Io_stopwatchState	In/Out	Cronómetro utilizado para la medición del tiempo de ejecución

ChronometerStop.xaml

Este proceso detiene el cronómetro utilizado para la medición del tiempo de ejecución de un Estado y/o Secuencia. El Log Message generado cuenta con los siguientes *Log Fields* personalizados:

- Name: Nombre del Estado o Flujo cuyo tiempo de ejecución será medido.
- ParentName: Nombre del Estado o Flujo padre que invoca al flujo cuyo tiempo será medido.
- State: Indica el estado del Flujo o Estado, en este caso, el estado se define como *Executing* (En Ejecución)
- ExecutionTime: Se define el tiempo de ejecución del Estado / Secuencia.
- EndTime: Indica la fecha en la que finalizó la ejecución del Estado o Flujo.



El flujo de trabajo cuenta con los siguientes argumentos de entrada / salida:

Tipo de variable	Nombre	Tipo de Argumento	Valor

String	in_str_Name	In	Nombre del Flujo / Estado
String	In_str_ParentName	In	Nombre del Flujo / Estado padre que invoca al flujo
String	In_str_State	In	Estado en el que se encuentra el flujo
String	Io_stopwatchState	In/Out	Cronómetro utilizado para la medición del tiempo de ejecución

Mensajes de log

En la siguiente tabla se detallan los *Log Messages* incorporados en el Framework cuyo mensaje puede ser modificado desde la solapa *Constants* del archivo *Settings.xlsx.*

Nombre	Mensaje	Ayuda
LogScreenshotSaved	Screenshot saved at:	Mensaje de log genérico que indica la ubicación en la que fue guardado un Screenshot
LogGetTransactionItem	The process has been stop from Orchestrator	Mensaje de error genérico al detener la ejecución del proceso desde Orchestrator
LogTransactionItemSuccessStatus	Transaction item successfully processed	Mensaje de log genérico que indica que un Transaction Item fue procesado con éxito
LogTransactionItemFailedStatus	Transaction item processing has failed	Mensaje de log genérico que indica que el procesamiento de un Transaction Item ha fallado

Volver al Contenido

Archivo de configuración *Settings.xlsx*

El archivo de configuración *Settings.xlsx* cuenta con las siguientes solapas, solapas necesarias para el funcionamiento adecuado del Framework.

- **Sheets:** En esta solapa se definen todas las solapas (salvo si misma) del archivo de configuración cuyos valores deben ser leídos en el flujo de trabajo *SettingsLoad.xaml* e incorporados en el diccionario de configuraciones que utiliza el proceso.
- **ProcessesToKill:** En esta solapa se indican todos los procesos cuyo cierre se debe forzar en el estado *Inicialization* dentro del flujo *KillAllProcesses.xaml*; esta solapa se excluye de la solapa *Sheets* ya que los procesos a cerrar se incorporan en una colección diferente a la del diccionario de configuraciones.
- **ProcessesToStart:** En esta solapa se indican todos los procesos que deben ser iniciados en caso de que no lo estén; en la columna *Name* se indica el nombre del proceso que será buscado en la lista de procesos en ejecución y en la columna *Value* se indica la ruta del archivo ejecutable (.exe) que inicia la ejecución de proceso.

- **Exceptions:** En esta solapa se indican todos los mensajes de las diferentes excepciones controladas que se pueden generar durante la ejecución del proceso. No se recomienda eliminar ninguna fila de esta solapa ya que puede afectar el funcionamiento adecuado del Framework.
- **Constants:** En esta solapa se define un conjunto de constantes que utiliza el Framework de manera determinada, no se recomienda eliminar ninguna fila de esta solapa ya que puede afectar el funcionamiento adecuado del Framework.
- **Folders:** En esta solapa se definen las rutas en las que se encuentra definido el directorio Raíz y los subdirectorios *Input, Output, Settings* y *Logs*; se debe tener en cuenta que, si se agrega una nueva carpeta dentro de esta Solapa, es necesario agregar la lógica que verifica su existencia y su posterior creación dentro del flujo de trabajo *ValidateFoldersExistance.xaml.*
- **Regex:** Esta solapa contiene todas las expresiones regulares que serán consumidas por el proceso que implemente el Framework.
- **eMails:** Esta solapa contiene todas las direcciones de correo electrónico0 que serán consumidas por el proceso que implemente el Framework.
- **Credentials:** Esta solapa contiene todas las credenciales definidas como Asset o definidas en el Administrador de credenciales de Windows que serán consumidas por el proceso que implemente el Framework.
- **UserSettings:** Esta solapa contiene todas las configuraciones y datos que serán consumidos por el proceso que implemente el Framework.

Volver al Contenido

Configuración inicial del Framework

Por defecto, al implementar el Framework, se debe definir el directorio raíz en el cual se ubicarán los siguientes subdirectorios:

- **Input**: En este directorio serán almacenados los archivos que el proceso requiere como datos de entrada.
- **Output**: En este directorio serán almacenados los archivos de salida producto de la ejecución del proceso.
- **Logs**: En este directorio serán almacenados los archivos de log producto de la ejecución del proceso.
- **Settings**: En este directorio se encuentra el archivo *Settings.xlsx* que contiene todos los datos de configuración necesarios para la ejecución del proceso.

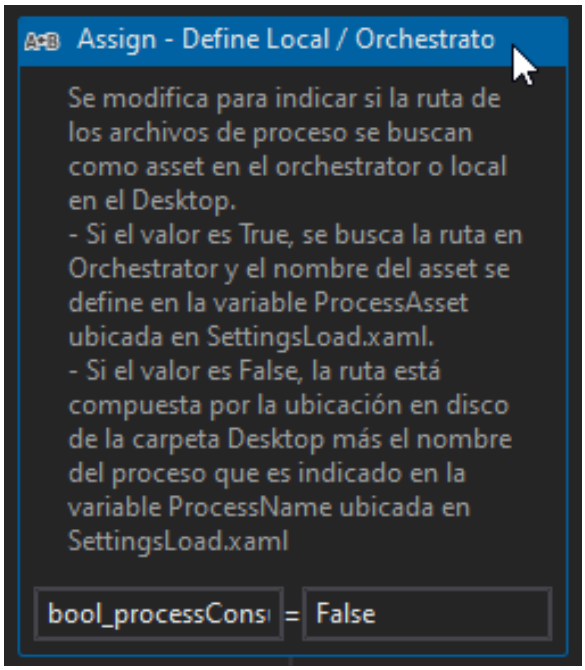
Antes de ejecutar por primera vez el proceso, es obligatorio que en el directorio raíz se haya creado el subdirectorio *Settings* y el archivo *Settings.xlsx* haya sido configurado con la información mínima requerida para que el Framework se ejecute sin inconvenientes; durante la primera ejecución del proceso, el Framework se encargará de crear los directorios que hagan falta.

La definición del directorio raíz se puede realizar de manera local o desde Orchestrator.

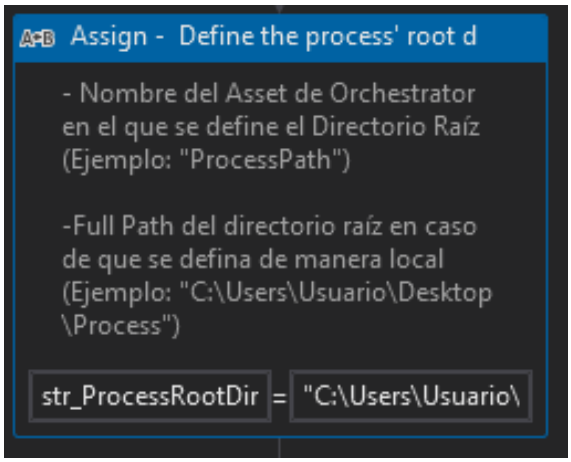
Definición local del directorio raíz

Para definir localmente el directorio raíz, se deben realizar los siguientes pasos:

- **Setear la variable *bool_processConsumesAssets* en *False:*** En el estado *Settings load*, se encuentra la actividad *Assign - Define Local / Orchestrator Folder*, en la cual definimos el valor de la variable *bool_processConsumesAssets;* si el valor es igual a *False* estamos indicando que el directorio raíz lo vamos a definir de manera local.



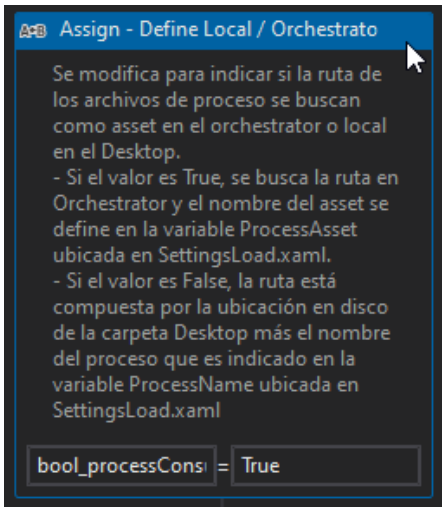
- **Indicar la ruta completa del *Directorio raíz*:** En la actividad *Assign - Define the process' root directory* se define la ruta completa del directorio raíz bajo la variable *str_ProcessRootDirectory.*



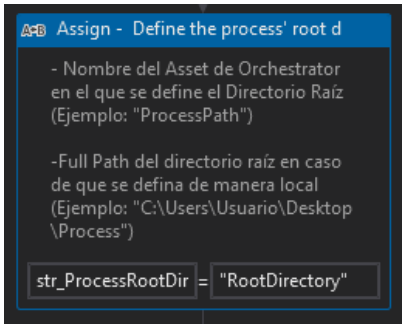
Definición del directorio raíz desde Orchestrator

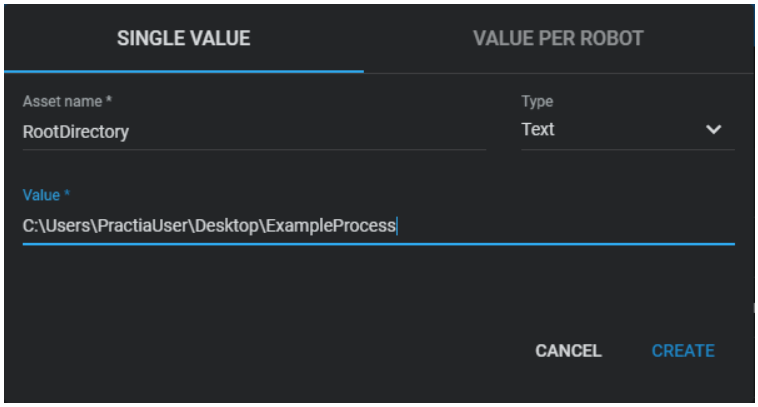
Para definir el directorio raíz desde Orchestrator, se deben realizar los siguientes pasos:

- **Setear la variable *bool_processConsumesAssets* en *True:*** En el estado *Settings load*, se encuentra la actividad *Assign - Define Local / Orchestrator Folder*, en la cual definimos el valor de la variable *bool_processConsumesAssets;* si el valor es igual a *True* estamos indicando que el directorio raíz lo vamos a obtener de un *Asset* de *Orchestrator*.



- **Indicar el *Asset* en el cual se definió el directorio raíz:** En la actividad *Sequence - Define Process Folder* se encuentra una actividad llamada *Assign - Define process' root directory asset* en la cual se define el nombre del *Asset* que contiene la ruta complete del directorio raíz.





Ejemplo del Asset definido en Orchestrator

Definir si el proceso consume QueueItems de una Queue

El Framework incorpora una lógica que permite obtener *QueueItems* de una *Queue*, si se desea utilizar está lógica en los procesos en los que se implemente el Framework, se deben seguir los siguientes pasos:

- **Setear la variable *bool_IsTransactionItemConsumer* en *True:*** En el estado *Settings load*, se encuentra la actividad *Assign - Define If Process consumes Transaction Items*, en la cual definimos el valor de la variable *bool_IsTransactionItemConsumer;* si el valor es igual a *True* estamos indicando que el proceso debe obtener *QueueItems* de una *Queue* de *Orchestrator*.
- **Indicar el nombre de la *Queue* en el archivo *Settings.xlsx*:** En la solapa *Constanst* del archivo de configuración se encuentra una clave bajo el nombre *QueueName* en la que se indica el nombre de la *Queue* a partir de la cual se van a obtener *QueueItems*.

Volver al Contenido