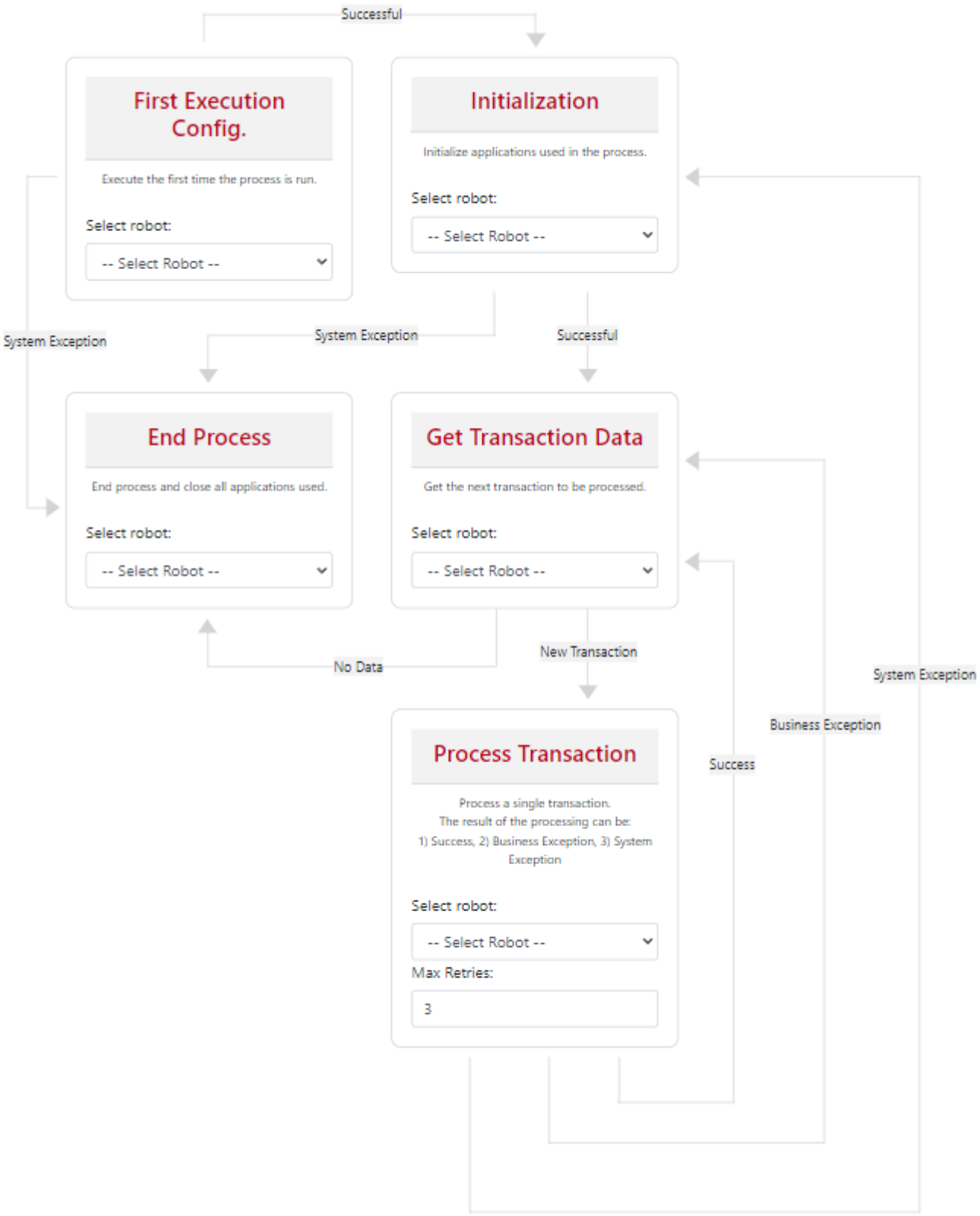
Rocketbot Transaction Framework

Para acceder al comando, deben dirigirse a la sección nativa **Frameworks**, y dentro seleccionar **Rocketbot Transaction Framework** (RTFramework para abreviar). Al abrir se les mostrará la configuración del comando, tal cual la imagen de arriba, donde les permitirá seleccionar el robot que representará cada estado del proceso, ahondaremos en ello más adelante.

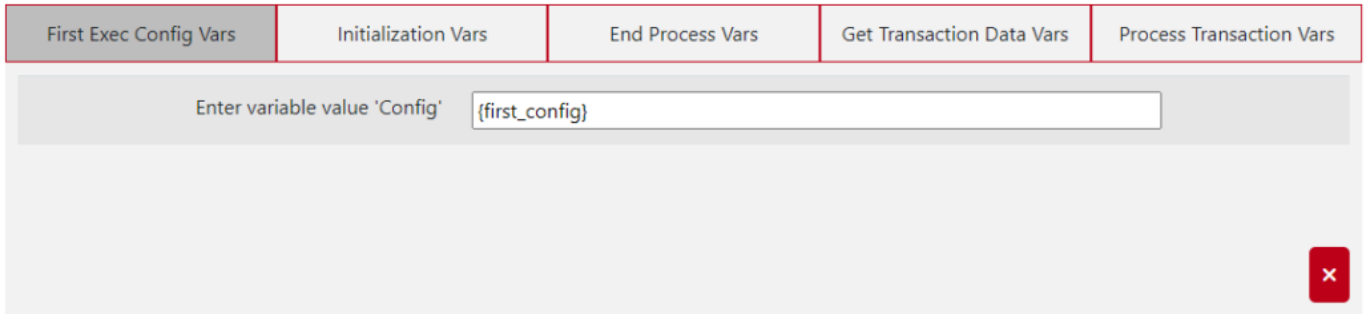
Los archivos que estaremos revisando en la documentación se pueden descargar desde este link:
http://docs.rocketbot.com/wp-content/uploads/2024/07/RTFramework.zip

Durante la documentación estaremos revisando la base de datos **RTFramework_example.db**. El zip también incluye la base de datos **RTFramework_template.db** la cual es la estructura base, y el archivo **Products_Table_framework.xlsx** el cual se utiliza en la base de datos de ejemplo. Para poder probar la ejecución de ejemplo se debe dejar el archivo xlsx en la raíz de Rocketbot.

Vista general



Vista general del RTFramework



Vista de la asignación de variables de cada robot

Los robots utilizados en el Framework son ejecutados siguiendo la misma metodología que el comando **Ejecutar robot Rocketbot (expose)**. Esto quiere decir que los robots no heredarán variables del padre, sino que se les asignará el valor desde el framework, además retornará el resultado de la ejecución, esto último deberá ser obligatorio en algunos de los robots que ejecutaremos para que todo funcione correctamente.

Robots del RTFramework

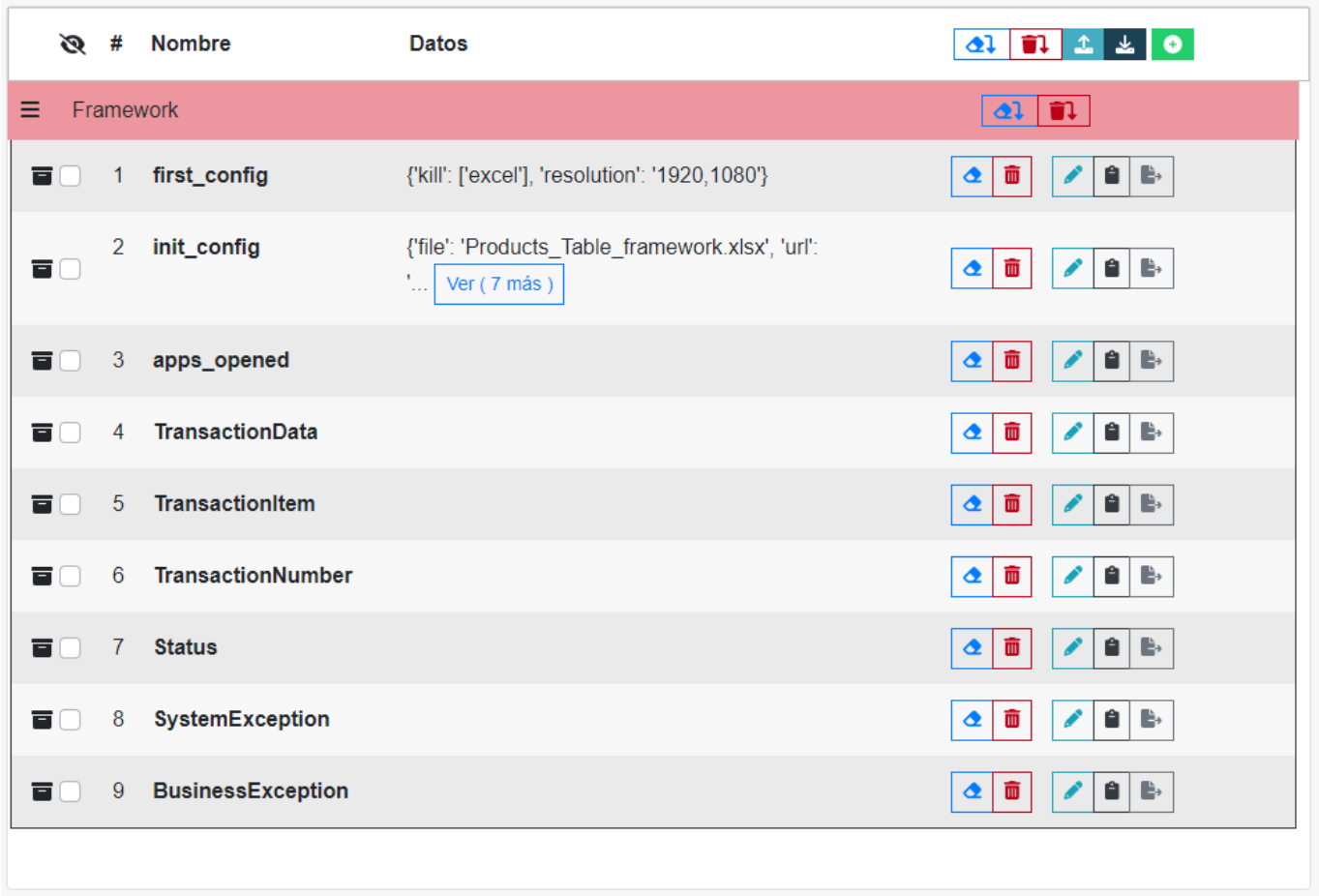
Main Robot

El robot principal es el que va a contener el comando del RTFramework.

Vista del comando del robot principal en la base de datos de ejemplo

Variables

RTFramework hace uso de diferentes variables durante su ejecución, en tu robot principal deberás crear las siguientes:



first_config: Variable opcional. En la db de ejemplo se envía como valor a la variable {Config} del robot **First Execution Config**. Su contenido es un diccionario con las llaves kill y resolution, datos que serán utilizados en el robot mencionado para matar procesos y para modificar la resolución de la pantalla respectivamente.

init_config: Variable opcional. En la db de ejemplo se envía como valor a la variable {Config} del robot **Initialization**. Su contenido es un diccionario con las llaves file y url, donde file es el archivo excel que se utilizará y url la página que se automatizará.

apps_opened: Variable opcional. En la db de ejemplo no tiene ningún valor por defecto, es utilizada para recibir parte del retorno de la ejecución de **Initialization**. Se utiliza como parámetro en el robot **End Process** para saber qué aplicaciones deben cerrarse al finalizar la ejecución.

TransactionData: Variable obligatoria. Se utiliza en el robot de **Initialization** y **Get Transaction Data**. En el retorno de información del robot **Initialization** debe asignarse la lista de información que será procesada en cada transacción.

TransactionItem: Variable obligatoria. Se utiliza en el robot **Get Transaction Data** para asignar un valor individual de la variable **TransactionData**. Este contenido que se obtenga se utilizará en el robot **Process Transaction**.

TransactionNumber: Variable obligatoria. Se utiliza en el robot **Get Transaction Data** y **Process Transaction** para indicar el número de transacción

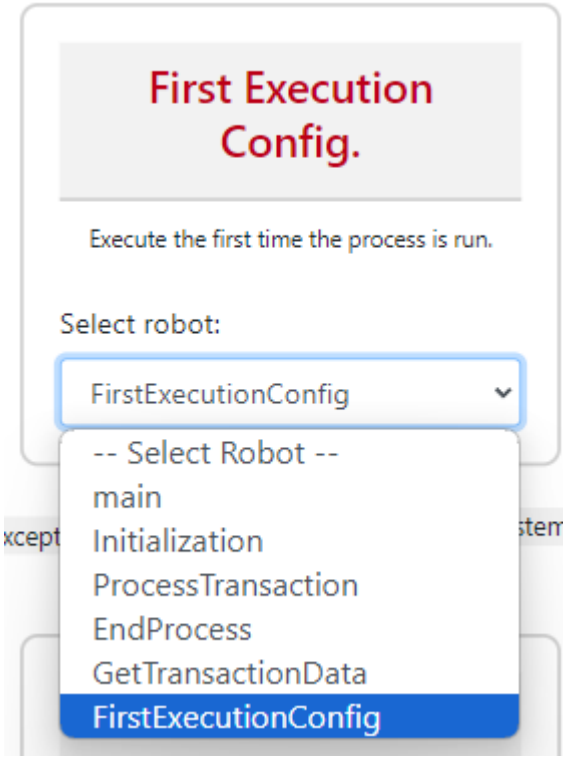
que se procesará. Su valor se generará automáticamente en base al número de transacción en curso durante la ejecución.

Status: Variable obligatoria. Se utiliza como variable de control en caso de que la ejecución falle en algún punto. Se debe utilizar en el catch de todos los robots del RTFramework, ahí se asignará el mensaje de error cuando un comando falla.

SystemException: Variable opcional. Acá se asignará el texto de cualquier excepción de sistema que ocurra durante la ejecución del RTFramework.

BusinessException: Variable opcional. Acá se asignará el texto de cualquier excepción de negocio que ocurra durante la ejecución del RTFramework.

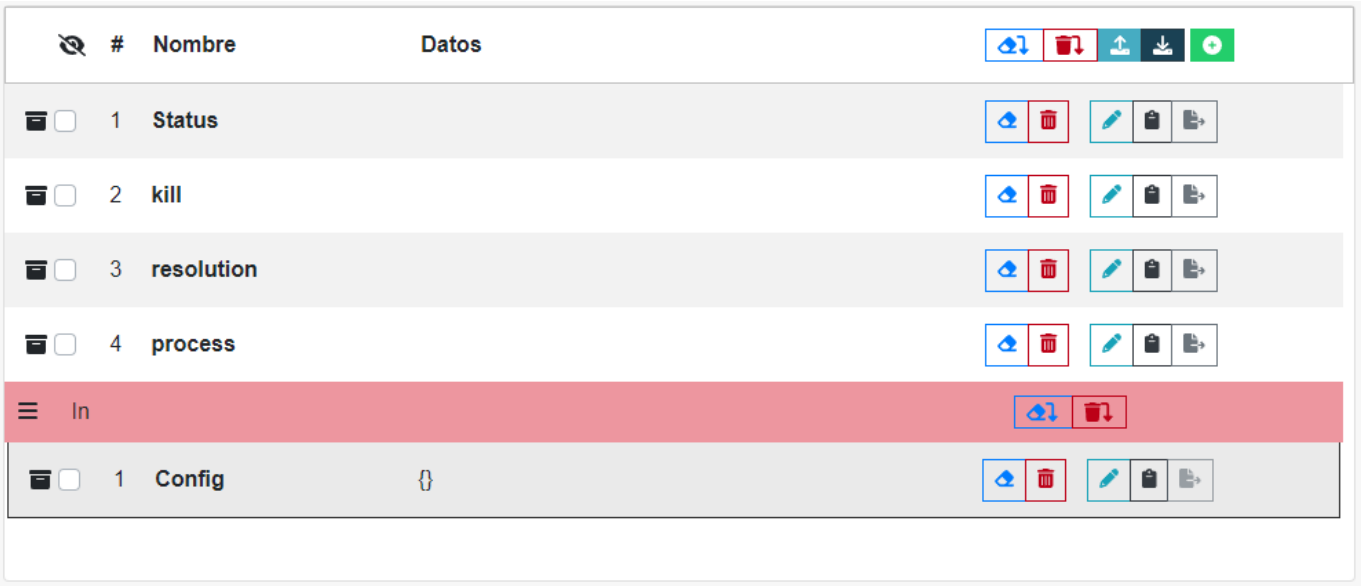
First Execution Configuration



Vista de la selección de robot First Execution Config en base de datos de ejemplo

El primer robot que debemos seleccionar en el RTFramework es el de la primera configuración de ejecución. Este robot se encargará de preparar el ambiente donde ejecutaremos nuestro proceso.

Variables

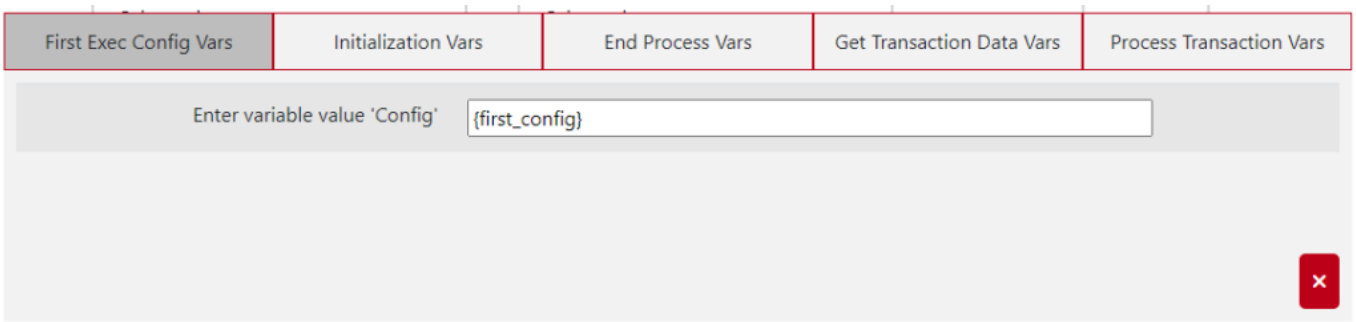


Variables del robot en la db de ejemplo

Status: Variable obligatoria. Se utiliza como variable de control en caso de que la ejecución falle en algún punto. Se debe utilizar en el catch de todos los robots del RTFramework, ahí se asignará el mensaje de error cuando un comando falla.

kill resolution process: Variables opcionales. Son aquellas que se van a utilizar internamente dentro del robot de la db de ejemplo.

Config: Variable opcional expuesta que recibirá la información en la configuración del RTFramework.



En el ejemplo se le asigna el contenido de la variable {first_config} del robot principal

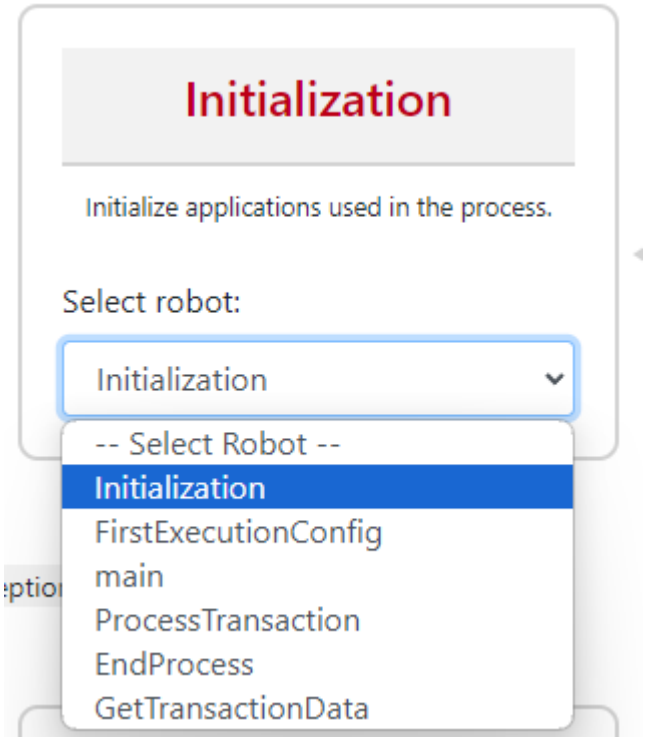
Estructura

El robot **First Execution Config**. debe seguir una estructura determinada para que el RTFramework funcione de manera correcta, a continuación la detallamos:

```
[Bloque TryCatch]
    Try:
        # Acá van los comandos que se ejecutarán como primer configuración
        # de la ejecución
        [Comandos de configuración]

        # Al final de manera opcional puede ir un comando de
        # Retornar informacion en caso de que necesitemos devolver
```

```
        # algún dato al robot principal
        [Retornar Informacion: Datos {var_retorno}]
    Catch:
        # El primer comando del Catch debe ser Último Estado.
        # De esta manera obtenemos el código del error que pueda suceder y
        # lo asignamos a la variable Status
        [Último Estado: Variable Status]

        # Luego pueden ir los comandos que podamos llegar a requerir para
        # el manejo del error
        [Comandos de manejo de error]

        # Al igual que con el bloque try, debajo de todo el catch debe ir
        # un comando de Retornar Informacion indicando en el input la
        # variable Status
        [Retornar informacion: Datos {Status}]
```

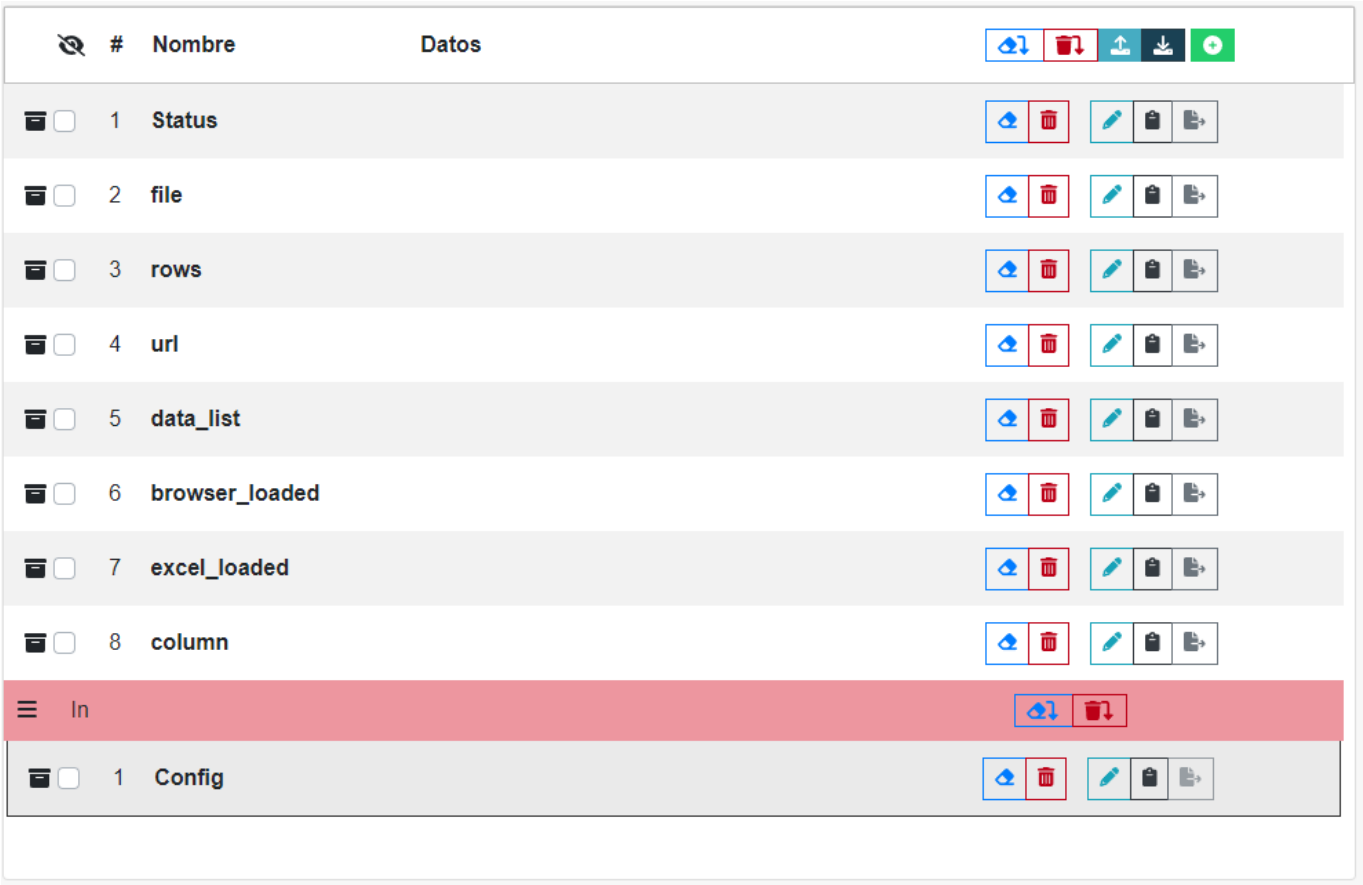
Initialization



Vista de la selección de robot Initialization en base de datos de ejemplo

El segundo robot que debemos seleccionar es el de la inicialización. Este robot se encargará de abrir las aplicaciones que se utilizarán en el RTFramework y obtener la información de las transacciones que se utilizarán durante el proceso.

Variables



Variables del robot en la db de ejemplo

La variable **Status** es la que recibirá el mensaje de error en caso de que algún comando falle. **Config** es una variable expuesta que recibirá la información en la configuración del RTFramework. **data_list** guardará la lista de información que se procesará y deberá ser devuelta como valor bajo la llave TransactionData en el último comando dentro del try en el robot. El resto de variables son específicas del ejemplo.

En el ejemplo se le asigna el contenido de la variable {init_config} del robot principal

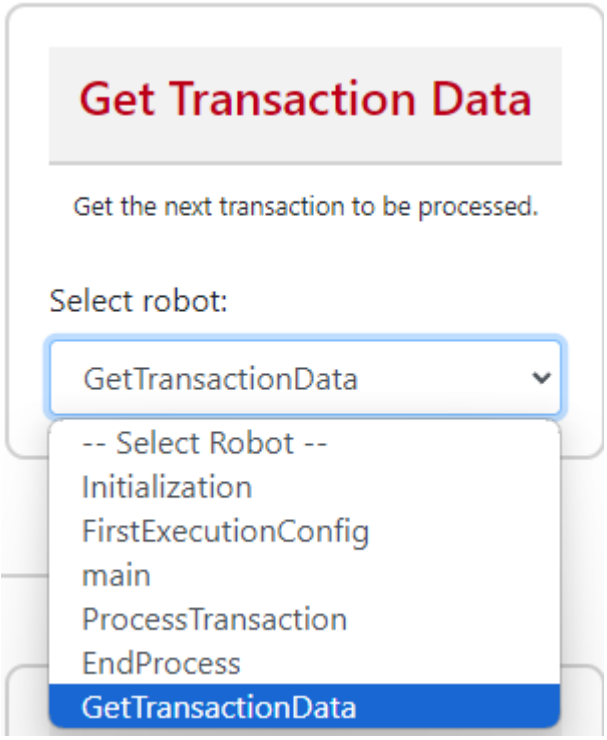
Estructura

El robot **Initialization** debe seguir una estructura determinada para que el RTFramework funcione de manera correcta, a continuación la detallamos:

```
[Bloque TryCatch]
    Try:
        # Acá van los comandos que se ejecutarán como inicialización
        [Comandos de inicializacion]
```

```
        # Al final de manera obligatoria debe ir un comando de
        # Retornar informacion que incluya un diccionario con, al menos,
        # la llave TransactionData que contenga una lista de los datos que
        # se utilizarán en el proceso. De manera opcional puede incluir
        # otras llaves con valores que deseemos asignar en el padre.
        [Retornar Informacion: Datos {"TransactionData": "{data_list}"}]
    Catch:
        # El primer comando del Catch debe ser Último Estado.
        # De esta manera obtenemos el código del error que pueda suceder y
        # lo asignamos a la variable Status
        [Último Estado: Variable Status]

        # Luego pueden ir los comandos que podamos llegar a requerir para
        # el manejo del error
        [Comandos de manejo de error]

        # Al igual que con el bloque try, debajo de todo el catch debe ir
        # un comando de Retornar Informacion indicando en el input la
        # variable Status
        [Retornar informacion: Datos {Status}]
```

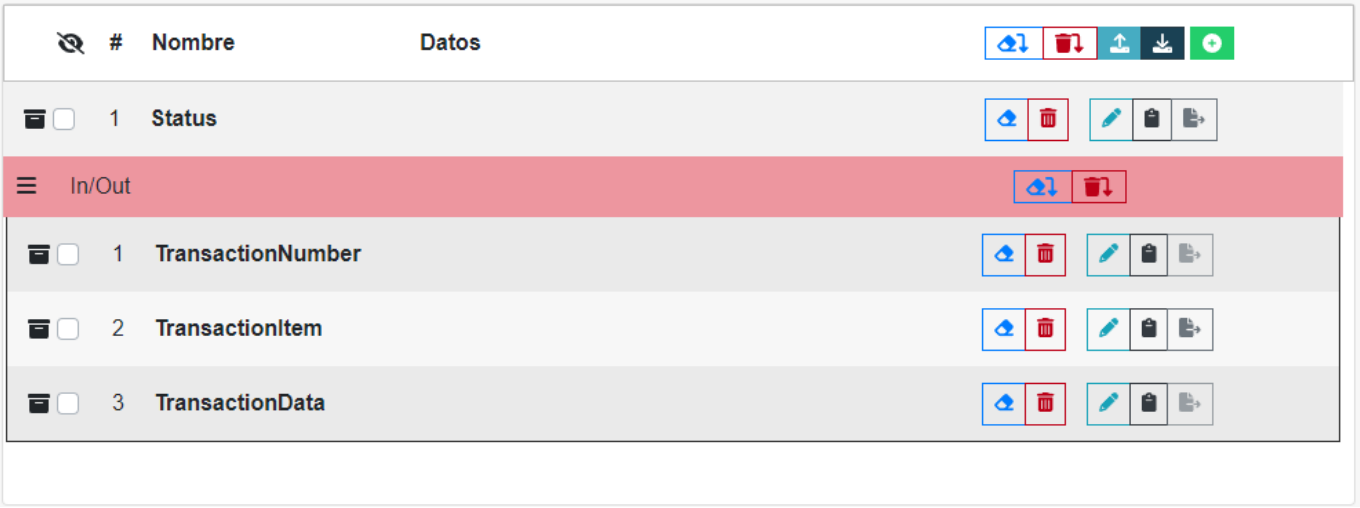
Get Transaction Data



Vista de la selección de robot Get Transaction Data en base de datos de ejemplo

El tercer robot que debemos elegir es el encargado de obtener información de la transacción. Su tarea será identificar la próxima transacción a procesar, la cual será enviada al robot Process Transaction.

Variables



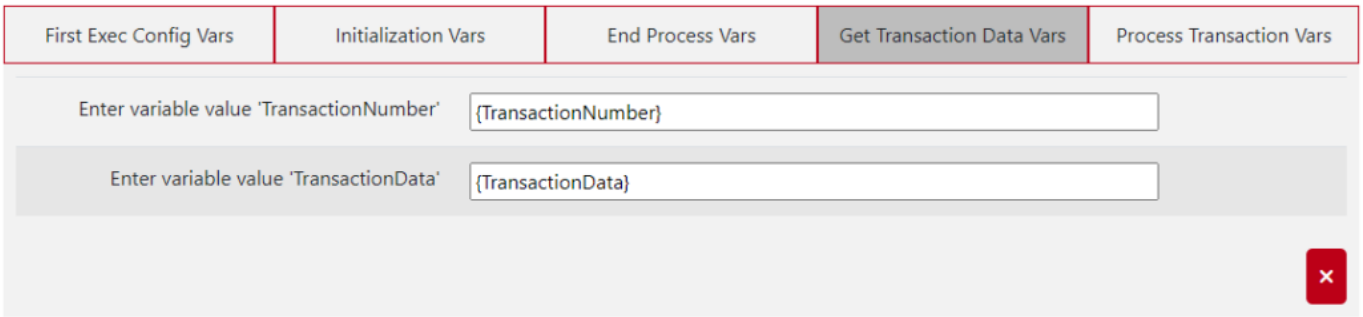
Variables del robot en la db de ejemplo

Status: Variable obligatoria. Se utiliza como variable de control en caso de que la ejecución falle en algún punto. Se debe utilizar en el catch de todos los robots del RTFramework, ahí se asignará el mensaje de error cuando un comando falla.

TransactionNumber: Variable obligatoria expuesta. Se utilizará internamente para saber qué numero de transacción se procesará a continuación.

TransactionItem: Variable obligatoria. Durante el proceso incluirá el contenido de la transacción que se procesará.

TransactionData: Variable obligatoria expuesta. Es la variable que contiene todas las transacciones que se utilizan durante el proceso.



En la db de ejemplo se asignan a ambas variables su valor correspondiente desde el padre.

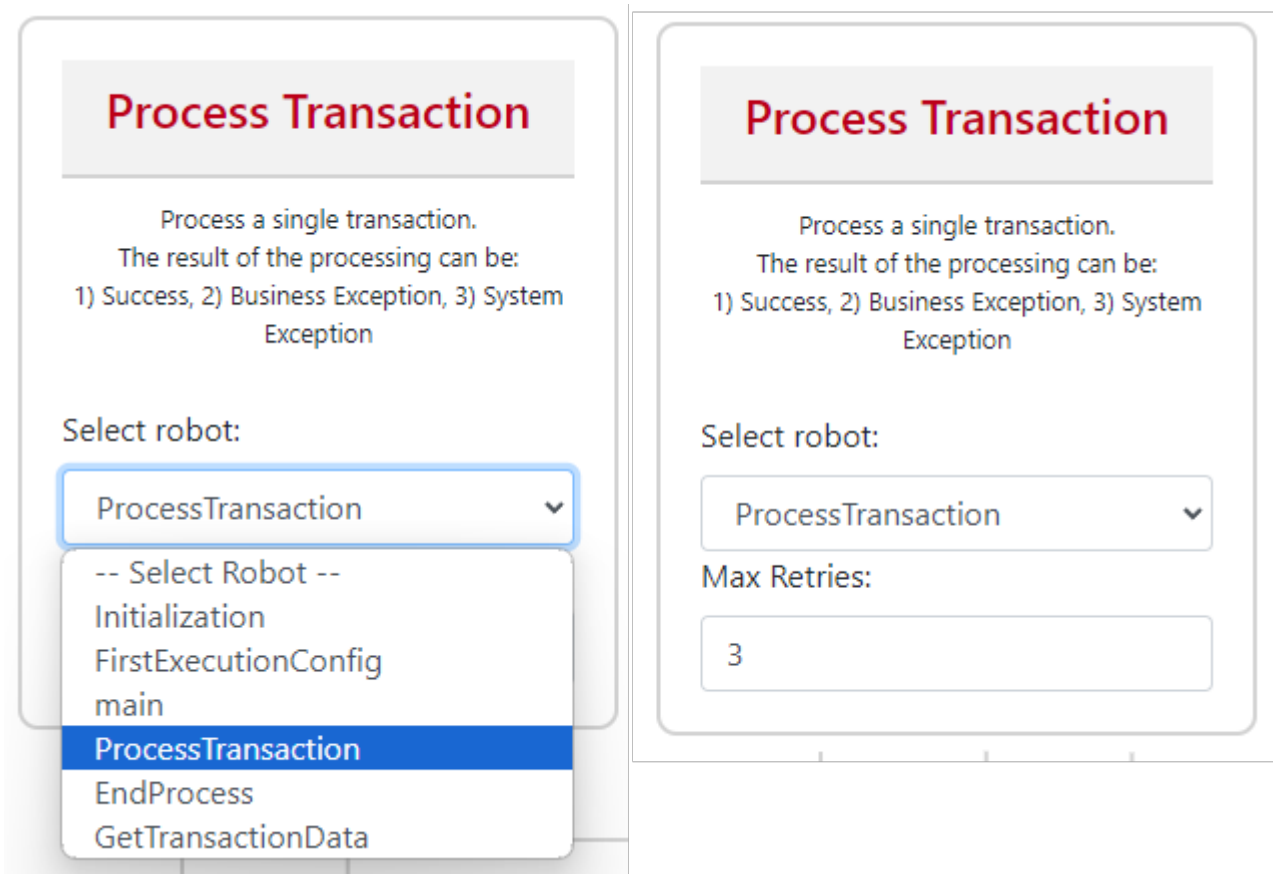
Estructura

El robot **Get Transaction Data** debe seguir una estructura determinada para que el RTFramework funcione de manera correcta, a continuación la detallamos:

```
[Bloque TryCatch]
    Try:
        # Grupo de comando destinado a contener la lógica que obtiene la
próxima
        # transacción a procesar
```

```
        Group:
            # Si el número de transacción es menor al largo de la lista de
            # transacciones, obtengo el siguiente item
            IF: {TransactionNumber} < len({TransactionData})
                [Asignar Variable: {TransactionData}[{TransactionNumber}
                 Dest.: TransactionItem]
            # Si no se cumple significa que se procesaron todas las
transacciones,
            # entonces devolvemos None para dar paso a la finalización de la
            # ejecución.
            ELSE:
                [Asignar variable: None
                 Dest.: TransactionItem]

        # Acá aplicamos los comandos que necesitemos ejecutar (opcional)
        [Comandos opcionales al obtener la información de la transacción]

        # Al final de manera obligatoria debe ir un comando de
        # Retornar informacion que incluya un diccionario con, al menos,
        # la llave TransactionItem que contenga el valor de su respectiva
variable.
        # De manera opcional puede incluir otras llaves con valores que
deseemos
        # asignar en el padre.
        [Retornar Informacion: Datos {"TransactionItem":
"{TransactionItem}"}]
    Catch:
        # El primer comando del Catch debe ser Último Estado.
        # De esta manera obtenemos el código del error que pueda suceder y
        # lo asignamos a la variable Status
        [Último Estado: Variable Status]

        # Luego pueden ir los comandos que podamos llegar a requerir para
        # el manejo del error
        [Comandos de manejo de error]

        # Al igual que con el bloque try, debajo de todo el catch debe ir
        # un comando de Retornar Informacion indicando en el input la
        # variable Status
        [Retornar informacion: Datos {Status}]
```

Process Transaction

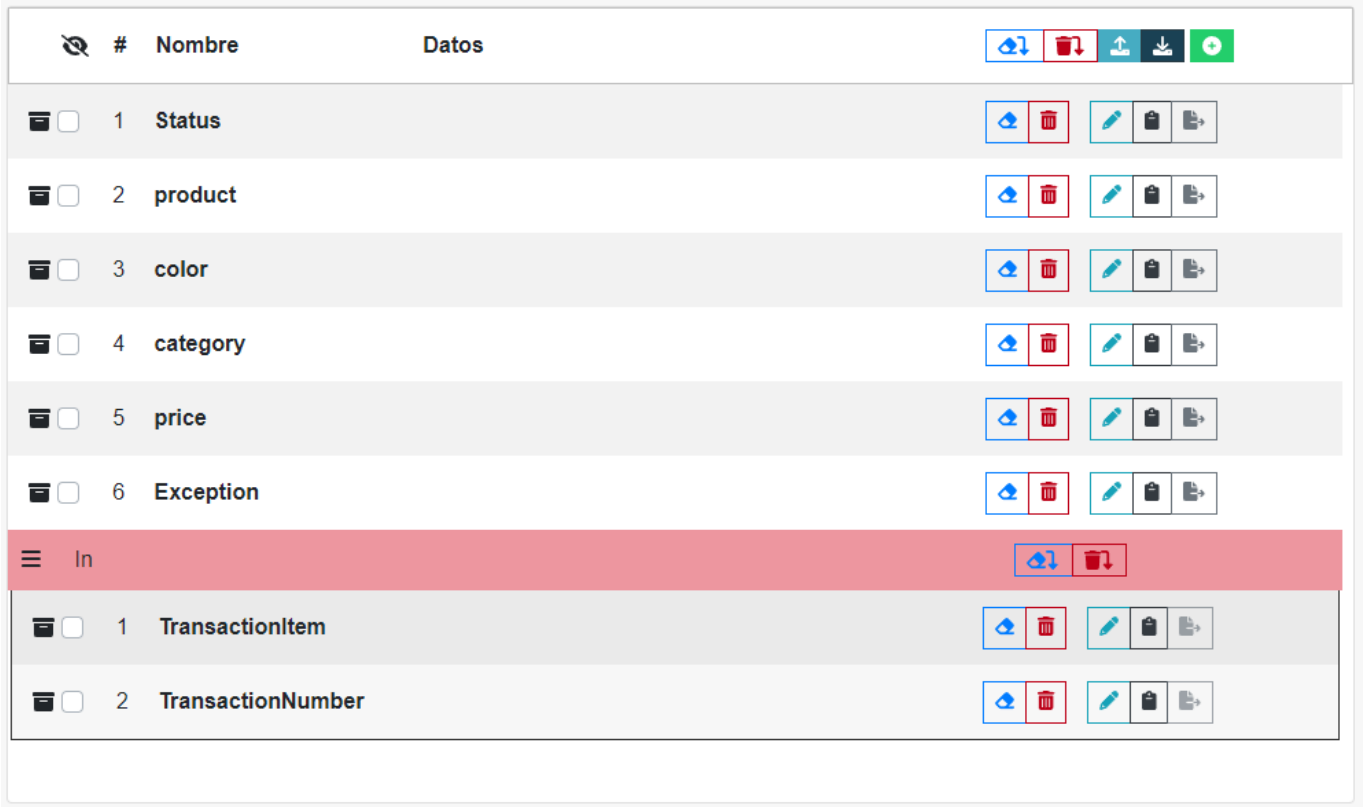


Vista de la selección del robot y la cantidad de reintentos del robot Process Transaction

El cuarto robot que debemos seleccionar es el que procesa una única transacción. Este robot recibirá como dato la transacción obtenida en el paso anterior, realizará los pasos que necesitemos y finalmente deberá devolver el item de transacción y el número de la misma. En la ejecución de cada transacción pueden haber tres posibles resultados:

- **Éxito**: Este resultado ocurre cuando todo el proceso se ejecutó correctamente, sin errores ya sean controlados o no controlados. En este caso, se continúa con la siguiente transacción a procesar.
- **Business Exception**: Este resultado ocurre cuando el proceso se encuentra con un error controlado. Son errores predecibles basados en las reglas de negocio que se conocen y se definen con antelación (ver cómo configurarlas en la sección **Estructura > Catch**). Si un proceso cae bajo este resultado, la transacción se reintentará el número de veces que le indiquemos en el input **Max Retries**.
- **System Exception**: Este resultado ocurre cuando el proceso se encuentra con un error **no** controlado. Pueden ser causados por problemas en el entorno de ejecución o en el sistema. Si un proceso cae bajo este resultado, el proceso se cancela y se ejecuta el robot **Initialization**, para reintentar toda la ejecución del RTFramework nuevamente.

Variables



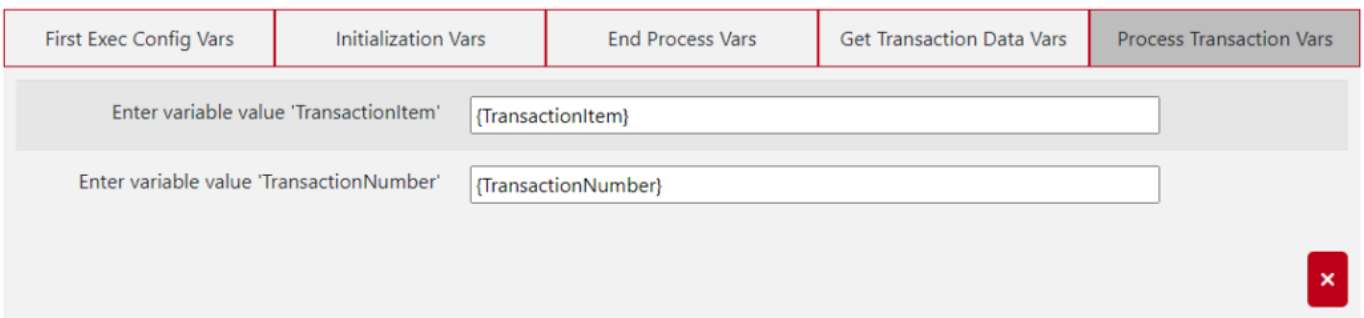
Status: Variable obligatoria. Se utiliza como variable de control en caso de que la ejecución falle en algún punto. Se debe utilizar en el catch de todos los robots del RTFramework, ahí se asignará el mensaje de error cuando un comando falla.

product, color, category, price: Variables opcionales de ejemplo que son utilizadas para manejar dentro del proceso la diferente información que contiene mi transacción.

Exception: Variable obligatoria. Cuando la ejecución no funciona de manera esperada, esta variable recibirá el mensaje de error. Esta variable es muy importante ya que forma parte del manejo de errores para decidir si es una **excepción de negocio** o **excepción del sistema**.

TransactionItem: Variable obligatoria (expuesta). Durante el proceso incluirá el contenido de la transacción que se procesará.

TransactionNumber: Variable obligatoria (expuesta). Es la variable que indica el número de transacción que se realiza.



En el ejemplo se asigna a cada una su variable correspondiente del bot padre.

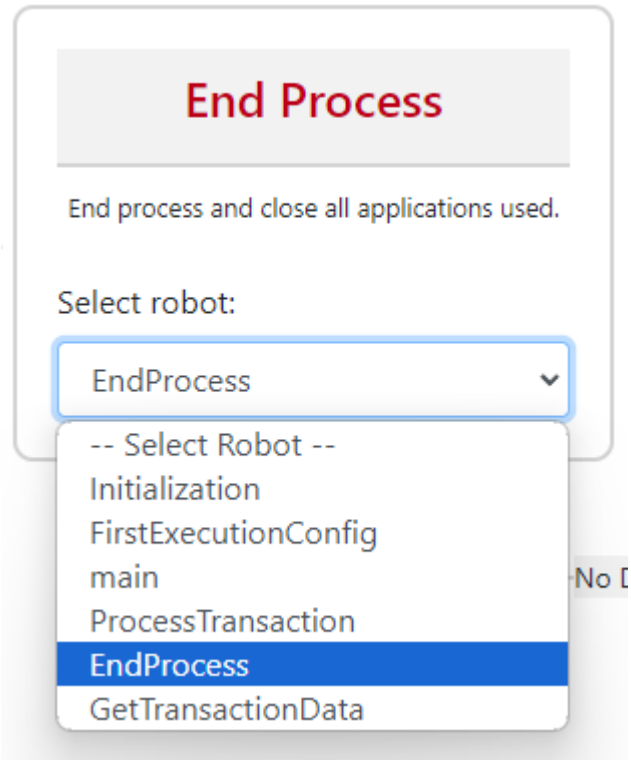
Estructura

El robot **Process Transaction** debe seguir una estructura determinada para que el RTFramework funcione de manera correcta, a continuación la detallamos:

```
[Bloque TryCatch]
    Try:
        # Acá van los comandos que realizarán el proceso de la transacción
        [Comandos de manejo de transación]

        # Al final del try de manera obligatoria deben ir dos comandos:
        # Asignar variable que sume uno a TransactionNumber
        # El comando Retornar Información que devuelva TransactionItem y
        # TransactionNumber, junto con cualquier otro contenido opcional
        [Retornar Informacion: Datos {"TransactionItem": "{TransactionItem}",
         "TransactionNumber": "{TransactionNumber}"}]

    Catch:
        # El primer comando del Catch debe ser Último Estado.
        # De esta manera obtenemos el código del error que pueda suceder y
        # lo asignamos a la variable Status
        [Último Estado: Variable Status]

        # Luego asignamos a la variable Exception el mensaje del error
        [Asignar Variable: {Status}.get('message')
         Dest: Exception]

        # En este punto se deben establecer las reglas de negocio. En la db
de
        # ejemplo se utiliza un IF que comprueba si en el mensaje del error
que
        # ocurrió se encuentra el texto "Unable to locate element". Este
error es
        # muy común en automatizaciones web cuando un elemento no puede ser
ubicado
        # en la página. A veces se soluciona con un reintento, entonces
quiero
        # establecer este caso como un Business Exception.
        IF: "Unable to locate element" in """{Exception}"""
            True:
                # En este punto debemos retornar un diccionario con, al
menos, la
                # llave BusinessException que contenga el texto del error.
                [Retornar informacion: Datos {'BusinessException':
'{Exception}'}]
            False:
                # Si llegamos a este punto es porque es un error no
controlado.
                # Por lo tanto, debemos retornar todo el contenido de la
variable
                # Status, generando un System Exception.
```

[Retornar informacion: **Datos** {Status}]

End Process

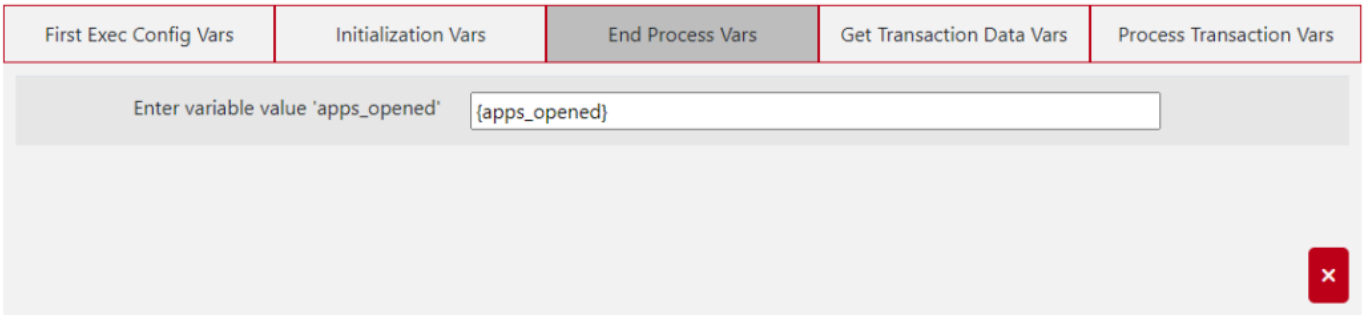


Vista de la selección del robot End Process en la base de datos de ejemplo.

El quinto y último robot que debemos seleccionar es el que finaliza la ejecución. Este robot se deberá encargar de enviar los reportes, cerrar los procesos que utilicemos durante la ejecución, etc…

Variables

No recibe ninguna variable obligatoria por parte del padre. En la db de ejemplo, se le comparte la variable apps_opened que indica las aplicaciones abiertas, de esta manera el robot se puede encargar de cerrarlas correctamente.



Estructura

Al igual que el resto de robots, **End Process** debe utilizar try catch, sin embargo dentro del try tendremos la libertad de utilizarlo como lo necesitemos, sin seguir una estructura predefinida ni devolviendo nada

específico al padre. Por otro lado, en el catch debemos colocar como primer comando Último Estado y al final de todo el Retornar Información que devuelva dicha variable.

```
[Bloque TryCatch]
    Try:
        # Acá van los comandos que se ejecutarán para finalizar la ejecución
        # (cierre de apps, envio de reportes, etc...)
        [Comandos de finalización]
    Catch:
        # El primer comando del Catch debe ser Último Estado.
        # De esta manera obtenemos el código del error que pueda suceder y
        # lo asignamos a la variable Status
        [Último Estado: Variable Status]

        # Luego pueden ir los comandos que podamos llegar a requerir para
        # el manejo del error
        [Comandos de manejo de error]

        # Debajo de todo el catch debe ir un comando de
        # Retornar Informacion indicando en el input la
        # variable Status
        [Retornar informacion: Datos {Status}]
```