UiPath REFramework Manual

Author: Mihai Dunareanu
Peer reviewer: Andrei Cioboata
Revision 1.0

About the framework and its purpose

このフレームワークは、最小限の設定で、プロジェクト構成データの格納、読み取り、および変更の容易な方法、堅牢な例外処理スキーム、すべての例外および関連トランザクションのイベント・ロギングを提供するプロセスをユーザーが設計できるようにするためのテンプレートです。

各プロセスで生成されたログはレポート生成の重要なコンポーネントであるため、フレームワークはビジネストランザクションを理解するために関連する各ステップでメッセージを記録し、それらのログを Orchestrator サーバーに送信します。これらは大量のデータ保存と様々なデータ表現を可能にする ELK Stack（Elasticsearch、logstash、kibana プラットフォーム）に格納されます。

ツールを構築する際には、最初に目的を定義します。このシナリオでは、フレームワークの目的は一連のビジネストランザクションを実行することです。最も単純なビジネスプロセスを除くすべてのビジネスプロセスは、典型的には複数の、範囲と目的で異なるビジネストランザクションの集合で構成されているため、ビジネスプロセスを記述しないことに注意してください。このようなリレーショナルビジネストランザクションの集合を、ビジネス・プロセス・コンポーネントと呼ぶことにしましょう。ビジネス・プロセス・コンポーネントは、完全なビジネスプロセスの一部です。

したがって、ビジネス・プロセス・コンポーネントは、一連のトランザクションに必要なデータを取得し、処理し、IT リソースに入出力するアクションのセットとして定義できます。

このようなコンポーネントは、実行するマシンに簡単にデプロイできるなければなりません（Orchestrator サーバーは、すべてのランタイムマシンでバージョン管理とデプロイが簡単です）。スケーラビリティが必要で、外部メディアとの出力データを通信する必要があるため、ビジネスプロセスの中で、中断した部分をピックアップすることがあります。このようなメディアは、共有フォルダ、データサーバー、FTP サーバー、電子メール、Orchestrator サーバーキューなどです。

Understanding a business process

次のビジネスプロセスを考えます：週1回ウェブリソース1（外部企業のウェブサイト）で燃料価格をチェックし、新しい値でファイルを更新する必要があります。別のユーザーは、ウェブサービス2（社内のウェブサイト）を利用して、勤務先の車両が移動した距離に関する情報を取得し、この情報を新しい燃料費と関連付ける。その後、Webリソース3（外部企業のWebサイト）を使用して配信料金を支払います。

この例では、3つのビジネス・プロセス・コンポーネントを考えます：

- 最初は、週に一回、燃料価格ファイルをチェックして更新するためにリソース1からデータを読み込みます。
- 2番目は、リソース2から距離に関する情報をダウンロードし、前のサブプロセスで得られた値を参照して、そのデータをフィルタリングし、さらに細分化します。完了後、データを保存します。
- 3番目のコンポーネントは、プロセス2によって生成された情報を読み込み、それのデータをリソース3に入力します。

もちろん、このビジネスプロセスは、サブプロセス1とサブプロセス2をまとめてグループ化するなど、3つではなく2つのビジネス・プロセス・コンポーネントとして表現することもできます。
もちろん、サブプロセス2は、リソース2から情報をダウンロードする2つの部分と、リソース1と2の両方の情報を読み込んで処理する2つの部分に分割することもできます。
問題を簡単に定義可能な単純なコンポーネントに分割するこの手法は、どんなに複雑であっても、ビジネスプロセスを解決するうえで最適な方法です。
それらはまた、現実、時間の基本的な側面に対処するのにも役立ちます。

正確に理解するために、上記のビジネスプロセスを変更してみましょう。燃料価格を含むファイルが更新された後、ファイルを開いて値の有効性を確認すし。サインをします。この場合、リソース2から情報を取得する前に、まず燃料コストファイルにサインするのを待つ必要があります。3つのコンポーネントでてこれを実装した場合、サブプロセス番号2にチェックを１つ追加するだけでよいため、変更は小さくなります。
ファイルにサインされているか、されていれば続ける、そうでなければ、終了し、後で再実行する。こうすれば、その間、ロボットが他のタスクを処理できます。
一方、コンポーネント1とコンポーネント2をまとめてグループ化した場合、データを再フェッチしたり、燃料ファイルを追加でチェックしたりする必要があります。これは望ましいことではありません。
実務は時間の経過と共に変化するものですから、こうした小さな変更は長い間発生しそうです。

Introduction

ステートマシン（state machines）について

ご存じのように、UiPath Studioには、シーケンス（sequence）、フローチャート（flowchart）、ステートマシン（state machine）の3種類のデータフロー表現があります。

フレームワークには3つのデータフロー表現がすべて含まれていますが、目的のデータフローを表現するためのよりクリーンなソリューションが提供されたため、プログラム本体にはステートマシンを選択しました。

以下はWikipediaのfinite state machineについての記述です:

“有限オートマトン（ゆうげん-、英: finite automaton)または有限状態機械（ゆうげんじょうたいきかい、英: finite state machine, FSM）とは、有限個の状態と遷移と動作の組み合わせからなる数学的に抽象化された「ふるまいのモデル」である。デジタル回路やプログラムの設計で使われることがあり、ある一連の状態をとったときどのように論理が流れるかを調べることができる。有限個の「状態」のうち1つの状態をとる。ある時点では1つの状態しかとらず、それをその時点の「現在状態」と呼ぶ。何らかのイベントや条件によってある状態から別の状態へと移行し、それを「遷移」と呼ぶ。それぞれの現在状態から遷移しうる状態と、遷移のきっかけとなる条件を列挙することで定義される。“

ステートマシンを使うときの基本ルール:

- システムは一度に1つのステート（状態）しかとれないため、ステートの内部の条件、外部条件のいずれかまたは両方で、別の状態に遷移できるようにしなければなりません。
- 各ステートからの遷移条件は排他的でなければなりません（2つの遷移条件は同時に真になることができないので、あるステートから2つの遷移条件を設定することができます)。
- 遷移アクションでは重い処理を実行してはいけません。すべての処理はステートの内部で行う必要があります。

このテンプレートで解決する問題は次の通りです:

1. プロジェクト設定データの保存と読み込み
2. ITリソースの開始、利用、終了を分離します
   (a) 再試行されたすべてのトランザクションについて、ITリソースをリスタートします
3. 堅牢な例外処理とトランザクション再試行スキームを実装する
   (a) タイプ別の例外を補足する
   (b) 例外タイプを利用して、Application Exceptionで失敗したトランザクションを再試行する
4. すべての例外と関連するトランザクション情報のログをキャプチャして送信する

Framework component functions

Table 1にフレームワークの呼び出し構造を示します。どのワークフローが呼び出されるのか、呼び出される順序、ワークフローが呼び出されるステートマシンの状態です。

Table 1 - Component call tree structure	
Component file names and locations	State where it is called
Main.xaml	
Framework¥InitAllSettings.xaml	Init
Framework¥KillAllProcesses.xaml	Init
Framework¥InitAllApplications.xaml	Init
Framework¥GetTransactionData.xaml	GetTransactionData
Process.xaml	Process
Framework¥SetTransactionStatus.xaml	Process
Framework¥TakeScreenshot.xaml	Process
Framework¥CloseAllApplications.xaml	Process
Framework¥KillAllProcesses.xaml	Process
Framework¥CloseAllApplications.xaml End	Program
Framework¥KillAllProcesses.xaml End	Program

上記とは別に追加のワークフローがありますが、デフォルトでは呼び出されません。これら機能については、“Additional functions”の章を参照してください。

Global Variables

グローバル変数は、スコープがメインプログラムまたはメインワークフローである変数です。 Main.xaml ワークフローファイルで、メインステートマシン内の任意の場所をクリックして、変数ペインをクリックすることで確認できます。Table 2は、プロジェクトのグローバル変数のリストです。

これらは、プロセスのランタイムを通じて利用可能な情報を格納するために使用します。それぞれの変数がどこに書き込まれているか、どこで読み込まれているかを理解することが重要です。

背景が赤いセルは、変数が書き込まれたワークフローを、背景が緑のセルは、読み取られたワークフローを表します。

Table 2 - Global variables table			
Name	Data type	Is written in workflows	Is read in workflows
TransactionItem	QueueItem	GetTransactionData.xaml	Process.xaml SetTransactionStatus.xaml
TransactionData		GetTransactionData.xaml	GetTransactionData.xaml
SystemError	Exception	Main.xaml	Main.xlsx SetTransactionStatus.xaml
BusinessRuleException	BusinessRuleException	Main.xaml	Main.xlsx SetTransactionStatus.xaml
TransactionNumber	Int32	SetTransactionStatus.xaml	GetTransactionData.xaml
Config	Dictionary(x:String, x:Object)	InitAllSettings.xaml	InitAllApplications.xaml GetTransactionData.xaml Process.xaml SetTransactionStatus.xaml
RetryNumber	Int32	SetTransactionStatus.xaml	SetTransactionStatus.xaml
TransactionID	string	GetTransactionData.xaml	SetTransactionStatus.xaml
TransactionField1	string	GetTransactionData.xaml	SetTransactionStatus.xaml
TransactionField2	string	GetTransactionData.xaml	SetTransactionStatus.xaml

Init State

InitAllSettings.xaml ワークフロー

このワークフローは、プロジェクトで使用されるキーと値のペアを持つ設定ディクショナリを出力します。設定はローカル設定ファイルから読み込まれ、Orchestratorアセットから取得されます。アセットは設定ファイルの設定内容を上書きします。

Table 3 - InitAllSettings.xaml Arguments and values		
dataType and Name	Argument Type	Values
String: in_ConfigFile	Input	"Data¥Config.xlsx"
String[]: in_ConfigSheets	Input	{"Settings", "Constants"}
Dictionary(x:String, x:Object): out_Config	Output	Config

InitAllApplications.xaml ワークフロー

内容: 必要に応じてアプリケーションを開いて初期化します。

以前の状態: N/A

以降の状態: Applications opened

Table 4 - InitAllApplications.xaml Arguments and values		
dataType and Name	Argument Type	Values
String: in_Config	Input	Config

Init Transitions

Init Stateの終わりに、構成ファイルをグローバル変数であるディクショナリConfigに読み込み、起動時にのみKillAllApplications.xamlワークフローを呼び出して作業環境を整理し、使用するすべてのアプリケーションを初期化します。

Table 5 - Init Transitions			
Name	Condition	Transition to State	Description
SystemError	SystemError isNot Nothing	End Process	初期化フェーズで Application Exception が発生した場合、プロセスを開始するための重要な情報が不足しています。これが、End Processに移行する理由です。
Success	SystemError is Nothing	Get Transaction Data	

Get Transaction Data State

GetTransactionData.xaml ワークフロー

内容: スプレッドシート、データベース、電子メール、Web API、またはUiPathサーバーのキューからデータを取得します。新しいデータがない場合は、out_TransactionItemをNothingに設定します。

リニアプロセス（反復しない）の場合は、in_TransactionNumber を1に - 最初と最後のトランザクションに対してのみout_TransactionItemを設定します。

プロセスが反復可能な場合は、in_TransactionNumber 1に対してio_TransactionDataを1回設定し、in_TransactionNumberを使用して新しいout_TransactionItemを割り当てて、io_TransactionDataを索引付けします。 io_TransactionDataコレクションの最後には、out_TransactionItemをNothingに設定してプロセスを終了する必要があることに注意してください。

Table 6 - GetTransactionData.xaml Arguments and Values		
dataType and Name	Argument Type	Values
Int32: in_TransactionNumber	Input	TransactionNumber
Dictionary(x:String, x:Object): in_Config	Input	Config
QueueItem: out_TransactionItem	Output	TransactionItem
Datatable: io_TransactionData	Input/Output	TransactionData
String: out_TransactionID	Output	TransactionID
String: out_TransactionField1	Output	TransactionField1
String: out_TransactionField2	Output	TransactionField2

Get Transaction Data Transitions

GetTransactionData状態からは、2つの遷移が可能です。最初は、TransactionItem変数で新しいトランザクションデータを取得したので、トランザクション処理状態に移ります。もう1つは、収集するデータがなくなったか、TransactionItem変数をNothingに設定したか、GetTransactionData.xaml を処理しているときにApplication Exceptionを取得した場合です。このエラーにより、プロセス終了状態に移行します。

Table 7 - Get Transaction Data Transitions

Name	Condition	Transition to State	Description
No Data	TransactionItem is Nothing	End Process	TransactionItemがデータ収集の後に何もない場合は、「プロセスの終了」に進みます。
New Transaction	TransactionItem isNot Nothing	Process Transaction	TransactionItem にデータがあれば処理します。

Process Transaction State

Process.xaml ワークフロー

このファイルでは、他のすべてのプロセス固有のファイルが呼び出されます。Application Exceptionが発生した場合、現在のトランザクションを再試行できます。 BREがスローされた場合、トランザクションはスキップされます。フローチャートまたはシーケンスが利用できます。プロセスが単純な場合、開発者はプロセスを複数のサブプロセスに分割し、Process.xamlワークフローで一度に1つずつ呼び出す必要があります。

Table 8 - Process.xaml Arguments and values		
dataType and Name	Argument Type	Values
QueueItem: in_TransactionItem	Input	TransactionItem
Dictionary(x:String, x:Object): in_Config	Input	Config

SetTransactionStatus.xaml ワークフロー

このワークフローは、TransactionStatusを設定し、そのステータスと詳細を追加のログフィールドに記録します。フローチャートは、成功、ビジネス例外、Application Exceptionの3つの可能なトランザクションステータスに分岐します。

各ブランチは、TransactionItemのタイプを分析します。TransactionItemが空でなくQueueItemの場合は、Orchestratorキューを使用していることを意味します。したがって、"Set Transaction Status"アクティビティをコールして、トランザクションの結果をOrchestratorに通知する必要があります。TransactionItemがQueueItemでない場合、"Set Transaction Status"アクティビティはトリガされません。

その後、結果を簡単に検索できるように、カスタムログフィールドにトランザクションの結果を記録します。

このワークフローでは、io_TransactionNumber変数のインクリメントが行われます。Application Exceptionがあり、MaxRetryNumberに達していない場合は、io_TransactionNumber変数ではなくio_RetryNumber変数をインクリメントします。これは、フレームワークの再試行メカニズムを管理し、"システムエラー処理"シーケンスの一部である、ロボットの再試行のフローチャートで行われます。

Table 8 - Process.xaml Arguments and values		
dataType and Name	Argument Type	Values
Dictionary(x:String, x:Object): in_Config	Input	Config
Exception: in_SystemError	Input	SystemError
BusinessRuleException: in_BusinessRuleException	Input	BusinessRuleException
QueueItem: in_TransactionItem	Input	TransactionItem
Int32: io_RetryNumber	Input/Output	RetryNumber
Int32: io_TransactionNumber	Input/Output	TransactionNumber
String: in_TransactionField1	Input	TransactionField1
String: in_TransactionField2	Input	TransactionField2
String: in_TransactionID	Input	TransactionFieldID

TakeScreenshot.xaml ワークフロー

使用法: in_Folderを、スクリーンショットを保存するフォルダの名前に設定します。または、io_FilePathにファイル名を含む完全パスを指定します。

内容: このワークフローはスクリーンショットをキャプチャし、名前と場所を記録します。その後、それを保存します。 io_FilePathが空の場合、in_Folderに画像を保存しようとします。拡張子は.pngです。

Table 10 - TakeScreenshot.xaml Arguments and Values		
dataType and Name	Argument Type	Values
String: in_Folder	Input	in_Config("ExScreenshotsFolderPath").ToString
String: io_FilePath	InputOutput	

CloseAllApplications.xaml ワークフロー

ここでは、すべての動作中のアプリケーションがソフト・クローズされます。

以前の状態: N/A

以後の状態: Applications closed

KillAllProcesses.xaml ワークフロー

ここですべての作業プロセスを終了します。

以前の状態: N/A

以前の状態: N/A

Process Transaction Transitions

Process Transaction Stateは、すべてのトランザクションの処理作業が行われる場所です。Process.xamlファイルが実行された後、生成された例外（ビジネスルールまたはアプリケーション）を検索します。例外がキャッチされなかった場合は、成功したこととします。

SetTransactionStatus.xamlワークフローは、Process.xaml出力のロギングと、次のトランザクションの管理または現在のトランザクションの再試行の両方を管理します。このワークフローでは、TransactionNumberとRetryNumberが書き込まれ、Application Exceptionの場合に自動で再試行されます。

Table 11 - Process Transaction Transitions			
Name	Condition	Transition to State	Description
Success	BusinessRuleException is Nothing AND SystemError is Nothing	Get Transaction Data	ビジネスルール例外が発生した場合は、ログに記録して次のトランザクションを処理します。
Rule Exception	BusinessRuleException isNot Nothing	Get Transaction Data	ビジネスルールの例外が発生した場合は、ログに記録し、Get Transaction Data Stateに移動して次のトランザクションに移動します。
Error	SystemError isNot Nothing	nit	Application Exceptionが発生した場合は、すべてのプログラムを終了し、失敗した場合は終了します。例外が発生した瞬間にスクリーンショットを撮ってInitに進み、作業環境を再初期化し、失敗したトランザクションから新たに開始します（最大再試行限度に達するまで再試行します）

End Process State

CloseAllApplications.xaml ワークフロー
すべての動作中のアプリケーションをソフト・クローズします。
以前の状態: N/A
以後の状態: Applications closed.

KillAllProcesses.xaml ワークフロー
すべての作業プロセスを終了します。
以前の状態: N/A
以前の状態: N/A

End Process Transitions
遷移のない最終状態です。

Additional Functions

上記の機能とは別に、クレデンシャル管理を容易に実装して安全にするための便利なワークフローが含まれています。

GetAppCredentials.xaml ワークフロー
使用法: 前回作成したOrchestratorアセットまたはWindowsクレデンシャルでin_redredを変更し、out_Usernameとout_Passwordを出力します。

内容: このワークフローは、入力時に定義されたクレデンシャルのセットを安全に取得または作成して使用します。最初にOrchestratorからフェッチを試みます。それに失敗すると、Windowsの資格情報マネージャーからそれらをフェッチしようとします。存在しない場合は作成します。

Table 10 - TakeScreenshot.xaml Arguments and Values		
dataType and Name	Argument Type	Values
String: in_Credential	Input	"TestRobot-Credential"
String: out_Username	Output	
SecureString: out_Password	Output	

Logging

ログメッセージは、何が起きたかをレポートできるため、ビジネスプロセス設計にとって非常に重要です。

前述のように、ログメッセージは複数のログフィールドで構成され、それぞれに対応する値があります。重要なイベントが発生したときにロボットによって自動的にログが生成されますが、開発者は“Log Message”アクティビティを使ってOrchestratorサーバーにプッシュし、さらにそのログがElasticsearchデータベースにプッシュされるようコンポーネントを実装します。

Logged Messages

以下は、フレームワーク内のすべてのメッセージログ、対応する“Log message”アクティビティが呼び出される場所、メッセージおよびログのレベル（info, warn, error, fatal）のリストです。

Table 13 - Message logs		
Message	Workflow	Log Level
Stop process requested	Main.xaml	Info
Config("LogMessage_GetTransactionDataError").ToString+TransactionNumber.ToString+". "+exception.Message+" at Source: "+exception.Source	Main.xaml	Fatal
"SetTransactionStatus.xaml failed: "+exception.Message+" at Source: "+exception.Source	Main.xaml	Fatal
Config("LogMessage_GetTransactionData").ToString+TransactionNumber.ToString	Main.xaml	Info
"Applications failed to close normally. "+exception.Message+" at Source:"+exception.Source	Main.xaml	Warn
Process finished due to no more transaction data	Main.xaml	Info
"System error at initialization: " + SystemError.Message + " at Source: "+SystemError.Source	Main.xaml	Fatal
"Loading asset " + row("Asset").ToString + " failed: " + exception.Message	Framework¥InitAllSettings.xaml	Warn
No assets defined for the process	Framework¥InitAllSettings.xaml	Trace
Opening applications...	Framework¥InitAllApplications.xaml	Info
in_Config("LogMessage_Success").ToString	Framework¥SetTransactionStatus.xaml	Info
in_Config("LogMessage_BusinessRuleException").ToString +in_BusinessRuleException.Message	Framework¥SetTransactionStatus.xaml	Error
in_Config("LogMessage_ApplicationException").ToString+" Max number of retries reached. "+in_SystemError.Message+" at Source: "+in_SystemError.Source	Framework¥SetTransactionStatus.xaml	Error
in_Config("LogMessage_ApplicationException").ToString+" Retry: "+io_RetryNumber.ToString+". "+in_SystemError.Message+" at Source:	Framework¥SetTransactionStatus.xaml	Warn

"+in_SystemError.Source		
in_Config("LogMessage_ApplicationException").ToString+in_SystemError.Message+" at Source: "+in_SystemError.Source	Framework¥SetTransactionStatus.xaml	Error
"Take screenshot failed with error: "+exception.Message+" at Source: "+exception.Source	Framework¥SetTransactionStatus.xaml	Warn
"CloseAllApplications failed. "+exception.Message+" at Source: "+exception.Source	Framework¥SetTransactionStatus.xaml	Warn
"KillAllProcesses failed. "+exception.Message+" at Source: "+exception.Source	Framework¥SetTransactionStatus.xaml	Warn
"Screenshot saved at: "+io_FilePath	Framework¥TakeScreenshot.xaml	Info
Closing applications...	Framework¥CloseAllApplications.xaml	Info
Killing processes...	Framework¥KillAllProcesses.xaml	Info

ログのメッセージの多くは、静的文字列と変数に格納されている文字列の間の連結（+記号）で構成されていることがわかると思います。

例を一つ細かく見ていきましょう。他のログはすべて同じロジックです。メッセージは次の通りです。

in_Config("LogMessage_ApplicationException").ToString+" Retry:" + io_RetryNumber.ToString+"."+in_SystemError.Message+" at Source: " +in_SystemError.Source

メッセージの最初の部分 in_Config("LogMessage_ApplicationException").ToString は、Config ディクショナリから読み込まれるので、必要に応じて簡単に変更できます。これは、Condig Excel ファイルの Constants シートにあり、現時点では"System exception."です。
次に、定数文字列 "Retry : "を追加しています。この文字列には、io_RetryNumber が続いており、これは再試行した回数です。次に、in_SystemError メッセージとソースを追加します。例外が発生した場所とそのメッセージが表示されます。
以上は、一つのログメッセージで状況が分かるよう、連結されます。

Custom Log Fields

Elasticsearch は NO-SQL 型のデータベース（リレーショナルではありません）なので、特定の基準に基づいてログをグループ化する機能が必要です。これらの基準は、フレームワーク全体で追加した追加のログフィールドになります。

ほとんどは修正する必要はありませんが、いくつかは開発者が値を変更する必要があります。以下の表に、フレームワークに追加されたログフィールドのリスト、その値、フレームワークを使用して実装している開発者がこれらの値を変更する必要があるかどうか、およびプログラム内の場所を追加するかどうかを示します。

Table 14 - Custom log fields

Field Name	Values	Value Change required	Location field is added
logF_BusinessProcessName	"Framework"	Yes	Main.xaml, Init State
logF_TransactionStatus	“Success” “BusinessException” “ApplicationException”	No	SetTransactionStatus.xaml, Success branch SetTransactionStatus.xaml, Business exception branch SetTransactionStatus.xaml, Application exception branch
logF_TransactionNumber	io_TransactionNumber .ToString	No	SetTransactionStatus.xaml, Success branch SetTransactionStatus.xaml, Business exception branch SetTransactionStatus.xaml, Application exception branch
logF_TransactionID	in_TransactionID	Yes	SetTransactionStatus.xaml, Success branch SetTransactionStatus.xaml, Business exception branch SetTransactionStatus.xaml, Application exception branch
logF_TransactionField1	in_TransactionField1	Yes	SetTransactionStatus.xaml, Success branch SetTransactionStatus.xaml, Business exception branch SetTransactionStatus.xaml, Application exception branch
logF_TransactionField2	in_TransactionField2	Yes	SetTransactionStatus.xaml, Success branch SetTransactionStatus.xaml, Business exception branch

			SetTransactionStatus.xaml, Application exception branch

Field Descriptions and explanations

1. logF_BusinessProcessName
   このフィールドには、ビジネスプロセスの名前が格納されます。目的は、複数のビジネスコンポーネントを Elasticsearch 内の同じダッシュボードにグループ化することです。これは、ビジネスプロセスが複数のビジネスコンポーネントで構成されている場合に重要です。したがって、このフィールドでは、分離されたデータをグループ化する方法が提供されます。Table 13 から分かるように、値は変更する必要があります。たとえば、ビジネスプロセスが 3 つのビジネスコンポーネントで構成されているとします。ビジネスプロセスは「請求書管理」と呼ばれます。あなたがする必要があるのは、各フレームワーク構築コンポーネントを開き、このフィールドの値として「請求書管理」を入れることです。

2. logF_TransactionStatus
   トランザクションのステータスを保持しているので、このログは変更する必要はありません。例外コンテンツを保持する BusinessRuleException および SystemError という名前のグローバル変数を SetTransactionStatus.xaml に渡すことによって、トランザクションの結果が正確にわかり、このフィールドに値を設定できることを覚えておいてください。

3. logF_TransactionNumber
   このフィールドの値は、トランザクション索引番号 TransactionNumber です。この変数はシステムによって管理されるため、その値を変更する必要はありません。

4. logF_TransactionID
   この値は、グローバル変数から SetTransactionStatus.xaml ワークフローへの入力引数として渡される変数 TransactionID の値です。この変数は、GetTransactionData.xaml ワークフローに書き込まれます。つまり、新しいトランザクションアイテムを取得したら、そのアイテムの識別子を選択する必要があります。このフィールドの値を使用して、異なるトランザクションごとにトランザクションの結果を表示するので、一意でなければなりません。

5. logF_TransactionField1
   この値は変数 TransactionField1 の値で、グローバル変数から SetTransactionStatus.xaml ワークフローへの入力引数として渡されます。この変数は、GetTransactionData.xaml ワークフローに書き込まれます。つまり、新しい取引アイテムを取得すると、追加の情報を追加することができます。単一のフィールド logF_TransactionID では不十分な場合があります。

6. logF_TransactionField2
   この値は変数 TransactionField2 の値で、グローバル変数から SetTransactionStatus.xaml ワークフローへの入力引数として渡されます。この変数は、GetTransactionData.xaml ワークフロ

ーに書き込まれます。つまり、新しいトランザクションアイテムを取得すると、単一のフィールド logF_TransactionID では不十分な場合があるため、追加の情報を追加することができます。

この章では、プロセスの名前を指定する logF_BusinessProcessName フィールドとトランザクション固有のフィールド（logF_TransactionID、logF_TransactionField1、logF_TransactionField2）を除いて、ログに記録されているフィールドについて心配する必要はないことを覚えておいてください。次に処理されるトランザクションに関する識別情報は書き込む必要があります。

言い換えれば、ログに記録される特定のトランザクションに関する 3 つ以上の項目の情報が必要な場合は、メインプログラム、GetTransactionData.xaml と SetTransactionStatus.xaml ワークフローで追加の変数を作成し、 SetTransactionStatus.xaml 内の“Add Log Fields”および“Remove Log Fields”アクティビティには、新しいフィールドの名前と、直前に宣言した変数を介して入力される値が格納されています。

Getting started, examples

フレームワークのデプロイ

フレームワークをデプロイするには、以下の手順に従います。

- Step 1: フォルダをプロジェクトの場所にコピーし、プロジェクト名を表す名前に変更します。
- Step 2: プロジェクトフォルダに移動し、メモ帳などのテキストアプリケーションを使用して、project.json ファイルを開きます。手順 1 で定義したプロジェクト名を「id」フィールドに書き込みます。「説明」フィールドにプロジェクトの説明を書き込みます。ファイルを保存して閉じます。
- Step 3: Main.xaml を開き、Init State に移動し、logF_BusinessProcessName フィールドの値をデフォルトの "Framework"からビジネスプロセス名に変更します。

プロセスコンポーネントのスコープ定義とフレームワークの準備

はじめにやることは、グローバル変数 TransactionItem と TransactionData のデータ型を選択することです。TransactionItem には、単一のトランザクションを完了するために必要なデータが格納されています。そのため、TransactionData はコレクション、リスト、データテーブルなどでなければならず、TransactionItems の「コレクション」です。フレームワークは、トランザクションデータから新しい TransactionItem をフェッチするインデックスとして TransactionNumber を使います。

次のステップは、これらの変数が渡されるワークフローを確認することです。 Main.xaml ワークフローとそれが引数として渡される他のフローの両方で、データ型を変更する必要があります。

- Step 1: メインプログラムの TransactionItem および TransactionData のデータ型を変更します。
- Step 2: Table 2 - グローバル変数テーブルを見ると、両方の変数が GetTransactionData.xaml、Process.xaml、および SetTransactionStatus.xaml ワークフローに渡されていることがわかります。
- Step 3: GetTransactionData.xaml と Process.xaml を開き、引数のタイプを必要なものに変更します。ワークフローを保存して終了します。
- Step 4: Figure 1 を使用する - コンポーネント呼び出しツリー構造は、Main.xaml で GetTransactionData.xaml と Process.xaml が呼び出される場所を見つけます。呼び出しのポイントに移動し、各ワークフローで、”引数をインポート”をクリックします。手順 3 で保存した新しい引数の型が表示されます。値のセクションで、変更された型の変数を main から渡します（Step 1）。
- Step 5: SetTransactionItem.xaml ワークフローの引数を変更する必要はありませんが、TransactionItem の QueueItem データ型を選択しない場合は、値フィールドから削除し、そのフィールドを空のままにするか、NULL を渡します。

これで、ニーズに応じて設定されたフレームワークがセットアップできました。

開発時には、次の簡単なルールに従ってください：

常に InitAllApplications.xaml ワークフローでアプリケーションを開きます。

- 常に KillAllApplications.xaml ワークフローでアプリケーションを終了してください。
- TransactionNumber は、TransactionData をループして新しい TransactionItem を取得するために使用するインデックスです。ループは Get Transaction Data State と Process State の間で発生し、システムはインデックスの増分を管理します。開発者が行う必要があるのは、新しい Item を取得するためにそれを使用することだけです。
- プロセスは、TransactionItem が Nothing になると終了します。したがって、プロセスの最後に null ポインタ Nothing を TransactionItem に割り当てるのは開発者の責任です。

Usage example 1

GetTransactionData.xaml の変更
Figure 1 のように、TransactionItem がより大きなデータ構造に含まれている場合、TransactionData がデータテーブルである場合（メモリに Excel ファイルを読み込んだ場合など）、TransactionData を一度読み取ってから TransactionNumber を使用する必要があります。現在のトランザクションのインデックスで、データを取得します。

Figure 1 では、最初のトランザクションで、Excel ファイル全体を読み込んで、グローバル変数 TransactionData に渡します。これはデータテーブルです。この場合、TransactionItem はデータ全体の一部である DataRow なります。

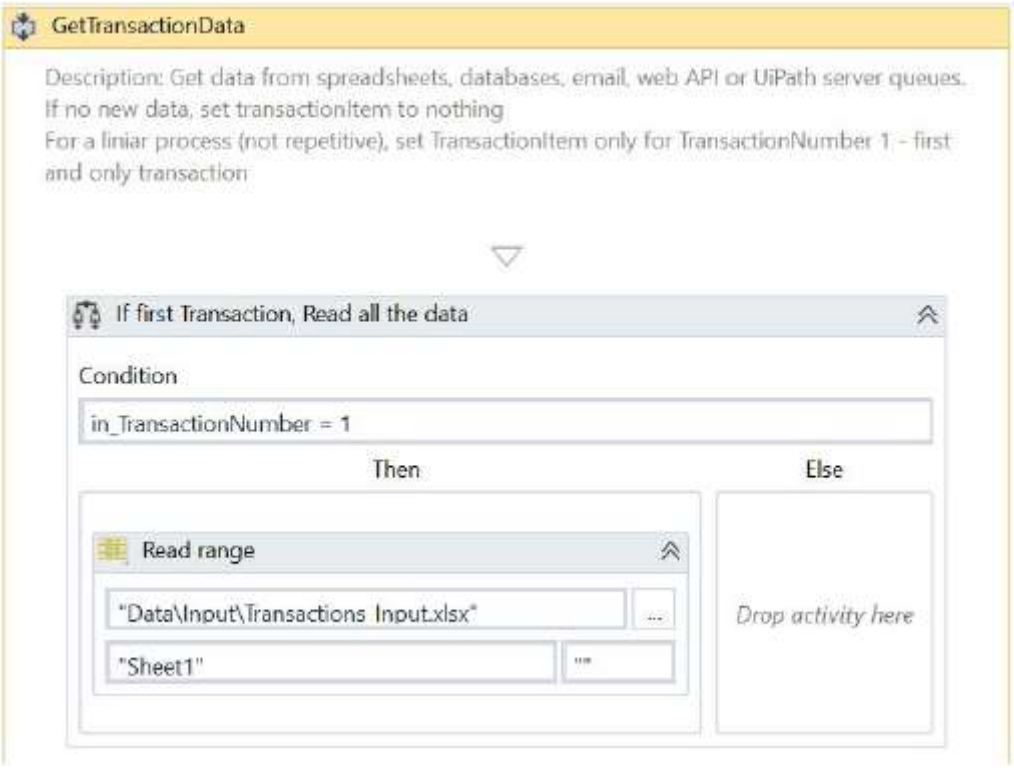

Figure 1 - read TransactionData once and output it to Global variables

次に、TransactionItem を取得するために、TransactionNumber というインデックスを使用します。副次的なことに、"For Each Row"アクティビティを使用して、データテーブルのデータローを 1 つずつ読み取ることができますが、TransactionNumber インデックスを使用して処理したトランザクションを記憶する必要があり。再試行は単にインデックスをインクリメントしないだけです。

したがって、Figure 2 では、if を使用してループ停止条件を定義しています。TransactionNumber はフレームワークによってインクリメントされるので、それを DataTable 内の行の数と比較することができます。行の数よりも大きくなったら、ループを停止します。Figure 7 Get Transaction Data Transitions では、End Process State で終了するために必要な条件が「TransactionItem is Nothing」であるため、行がなくなった場合は TransactionItem を Nothing に設定します。そうでない場合は、out_TransactionItem = io_TransactionData.Rows（in_TransactionNumber - 1）を設定します。TransactionNumber - 1 は初期値が 1 で、行のインデックスが 0 から始まるため、TransactionNumber - 1 を使用します。

Figure 2 - while we still have rows, read the current one based on

Table 2 - Global variables table と Figure 3 - Argument list for GetTransactionData.xaml により、これらの変数がグローバルスコープになっていることがわかります。

Name	Direction	Argument type	Default value
in_TransactionNumber	In	Int32	Enter a VB expression
in_Config	In	Dictionary<String,Object>	Enter a VB expression
out_TransactionItem	Out	DataRow	Default value not supported
out_TransactionField1	Out	String	Default value not supported
out_TransactionField2	Out	String	Default value not supported
io_TransactionData	In/Out	DataTable	Default value not supported
out_TransactionID	Out	String	Default value not supported
Create Argument			

Figure 3 - Argument list for GetTransactionData.xaml

次に、これらの値を SetTransactionStatus.xaml ファイルに引き渡す Log Field 変数に値を割り当てます。

トランザクションの識別情報を最もよく表すよう out_TransactionItem から 1 つまたは複数のフィールドを選択します。

例えば：
out_TransactionID=out_TransactionItem.Item(“Identifier Column name”).ToString

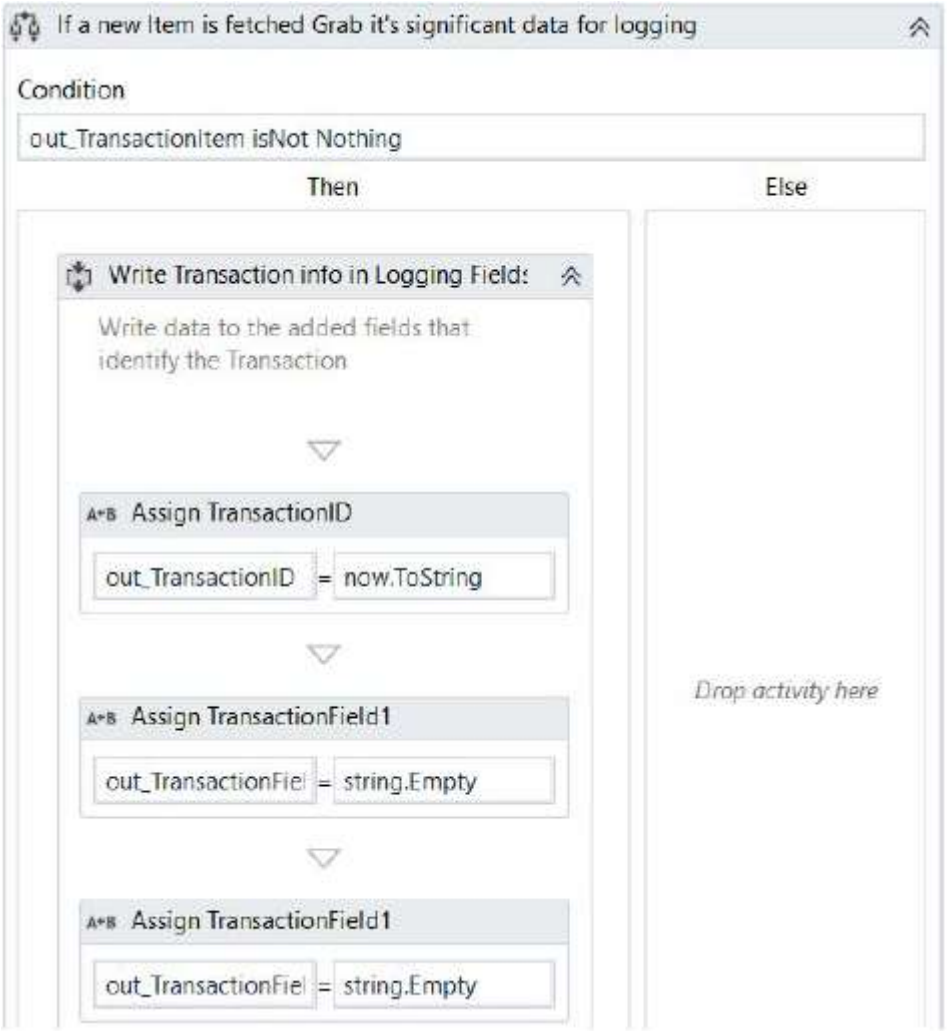

Figure 4 - Transaction Log field value assignments in GetTransactionData.xaml

Process.xaml の変更
in_TransactionItem 変数に格納された 1 つのトランザクションのデータを取得するステップを追加し、それを使用してプロセスを実行します。アプリケーションがすでに開いており、データが利用可能なので、process.xaml ファイルで作業を開始できます。このケースでは、in_TransactionItem は DataRow 型であるため、 “column named A”に含まれる値を取得するには、in_TransactionItem.Item(“column named A”)または、in_TransactionItem.Item(3)と記述します。

InitAllApplications.xaml の変更
すべてのアプリケーションを開き、ログインして環境をセットアップします。 “Log Message”アクティビティを変更し、使用しているアプリケーションに関する情報を設定します。

CloseAllApplications.xaml の変更
ログアウトし、すべてのアプリケーションを閉じます。 “Log Message”アクティビティを変更し、使用しているアプリケーションに関する情報を設定します。

KillAllApplications.xaml の変更
CloseAllApplications.xaml を呼び出したときに反応がない、あるいは、クローズできないアプリケーションがある場合、それらを強制終了します。“Log Message”アクティビティを変更し、使用しているアプリケーションに関する情報を設定します。

Usage example 2

この例では、トランザクションに必要なデータがすでに取得されており、Orchestrator Queue に格納されているものとします。

GetTransactionData.xaml の変更
データはOrchestratorサーバーキューに格納されるため、TransactionItemはQueueItem型です。キュー項目取得アクティビティを使用して次の項目を取得するだけです。Orchestrator サーバーはキューからアイテムを 1 つずつ処理するため、TransactionData を使用してすべてのトランザクションの合計を格納する必要はありません。その結果、TransactionNumber を TransactionData のインデックスとして使用することについて心配する必要はありません。キューが空になると、Orchestrator サーバーから Nothing ポインタか Nothing を受け取ります。これにより、プログラムは終了プロセス状態に移行します。

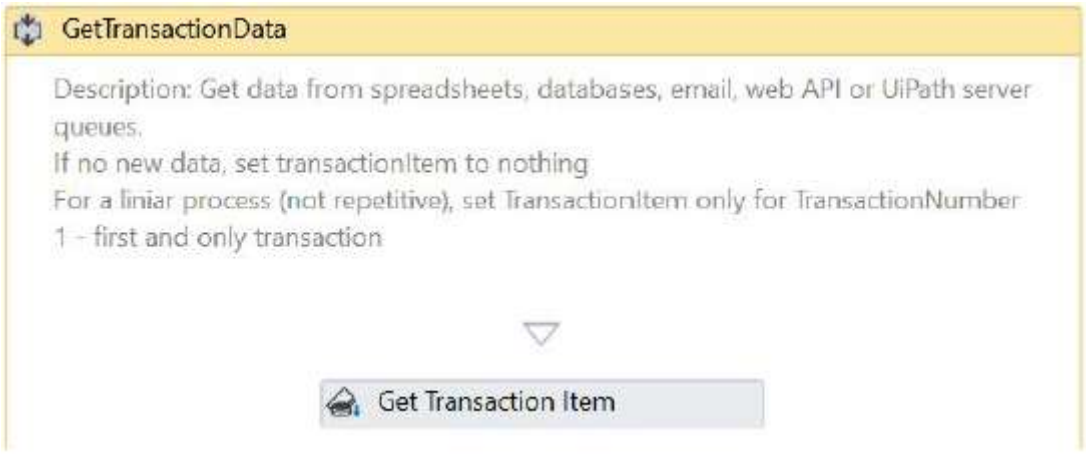

Figure 5 - Get QueueItem activity to get the next TransactionItem

次に、これらの値を SetTransactionStatus.xaml ファイルに格納する Log Field 変数に値を割り当てます。トランザクションの識別情報を out_TransactionItem から 1 つまたは複数のフィールドから選択します。

例えば：
out_TransactionID=out_TransactionItem.Item(“Identifier Column name”).ToString

Process.xaml の変更
in_TransactionItem 変数に格納されたトランザクションデータを 1 つ取得するステップを追加し、それを使用してプロセスを実行します。アプリケーションがすでに開いており、データが利用可能なので、process.xaml ファイルで作業を開始できます。
このケースでは、in_TransactionItem は QueueItem 型であるため、フィールド " field named A" に含まれる値を取得するために、in_TransactionItem.SpecificContent（“field named A”）を書き込みます

InitAllApplications.xaml の変更
すべてのアプリケーションを開き、ログインして環境をセットアップします。 “Log Message”アクティビティを変更し、使用しているアプリケーションに関する情報を設定します。

CloseAllApplications.xaml の変更
ログアウトし、すべてのアプリケーションを閉じます。 “Log Message”アクティビティを変更し、使用しているアプリケーションに関する情報を設定します。

KillAllApplications.xaml の変更
CloseAllApplications.xaml を呼び出したときに反応がない、あるいは、クローズできないアプリケーションがある場合、それらを強制終了します。“Log Message”アクティビティを変更し、使用しているアプリケーションに関する情報を設定します。

用語集（marked in italics)

ビジネス・プロセス・コンポーネント: ビジネスプロセスの別個の部分を表すサブプロセス。

IT リソース: 情報技術情報源。プログラムまたはデータファイルのことです。

UiPath Orchestrator: 高度なスケーラビリティを備えたサーバープラットフォームで、1 台のロボットから数十、または数百に至るまで、迅速なデプロイが可能です。アクティビティを監査およびモニターし、すべてのタイプのプロセスをスケジュールし、ワーク・キューを管理することができます。 Elasticsearch と Kibana ツールから高度なのレポートを作成します。リリース管理、コラボレーションツール、集中型ロギング、ロールベースアクセスもサポートされています。

ELK Stack: オープンソースツールのプラットフォームで、ユーザーは任意のソースから任意の形式でデータを確実かつ安全に取得し、リアルタイムで検索、分析、視覚化することができます。
（ELK: **E**lasticsearch, **L**ogstash, **K**ibana）

トランザクションデータ: 情報の視点から見て、類似する範囲または目的を持つ一連のトランザクションの詳細データの集合。

トランザクションアイテム: 情報の観点から、単一のトランザクションを完全に表すデータ。これは、コレクションに含まれるデータのサブセットです。

Business Rule Exception or **BRE**: 開発者が"Throw"アクティビティを使って手動でトリガした例外。アクティビティの構文は：new *UiPath.Core.BusinessRuleException（"this is my reason message"）*。開発者は、プロセスを続けるために特定の情報を入手する必要があるときに BRE を投げるべきですが、テストでは利用できません。

Application Exception: アプリケーションの状態が期待通りでない場合に、失敗したアクティヴィや開発者が実行したアクティビティによってトリガされる例外（例えば、プログラムが正しいフォーマットのデータを必要とする場合、正しくないフォーマットを渡すとエラーが発生します。再試行で問題が解決する場合は、メッセージをキャプチャして Application Exception を発行することができます）。
"Throw" アクティビティの基本的な構文は：*new System. Exception("this is my reason message")*で例外の種類はいろいろあります。

Workflow: UiPath アプリケーションの基本ビルディングブロック。シーケンス、フローチャート、またはステートマシンを使用してデータを表現できます。引数を持ち、他のワークフローから呼び出すことができます。