自動化の手引き

年次レポートの出力

自動化の手引き– 年次レポートの出力

このトレーニングモジュールの目的は、Orchestrator のキューを使用することで、キューの機能への理解を深めることがです。

事例をもとに、例えば、次の方法を学習していきます。

・複数のロボットを使用したデータ処理方法

・キューへの追加処理を実行した際に、自動化対象のアプリケーションにおいてシステムエラーが発生した場合に一番初めのトランザクションに戻るのではなく、システムエラーが発生した時点のトランザクションから開始する方法

また、キュー処理のプロセスは 2 つのプロセスに分けます。

1 つはキューにトランザクションの追加を行う**ディスパッチャー**と呼ばれるプロセスです。

もう 1 つは、キューに追加されたトランザクションに対するプロセスである**パフォーマー**です。

上記の方法を採用することで、ディスパッチャーを使用しキューにトランザクションを一括で読み込み、ディスパッチャーでキューに追加されたトランザクションを複数の ”パフォーマー専用のロボット” を使用して処理します。

ディスパッチャープロセス

- REFramework テンプレートより作成を開始します。
    - ディスパッチャーはキューにトランザクションの追加を行う役割を果たします。トランザクション一つひとつの識別に使用する「**WIID**」をキューにアップロードする必要があります。
    - 例えば、次のページに移行するためのボタンがないため、データを取得できず、結果として全てのデータを取り出せないというケースを仮定します。

- さらに、複数のページに跨るデータを全て取得するためにページを移行している際に、何らかの事情でエラーが起きた場合、ディスパッチャーはエラーが発生した時点のトランザクションから復旧し、作業を再開します。
  ここでは、WorkItems の全ての対象処理項目を 1 つのトランザクションとし、それぞれのページ番号自体を"トランザクションアイテム"とします。

- つまり、トランザクションアイテムは、現在処理中のページ番号を示す文字列型のデータです。

- 現在のプロセスに合わせて Config ファイルを編集します。
  - 「**Settings**」シート上で、**QueueName** を追加し、Value に **InHouse_Process4** と追加します。同じ名前を使用（InHouse_Process4）し、Orchestrator にキューを追加します。

  - 「**Settings**」シートで、ACME System 1 の URL と、ACME System 1 の資格情報パラメーターを追加します。

  - 「**Constants**」シートで、**MaxRetryNumber** の値を 2 に設定します。

- Framework 上で次の変更を確認します：
  - Main ファイルの変数「**TransactionItem**」を System.String 型に変更します。また、ワークフロー「**GetTransactionData**」「**Process**」「**SetTransactionStatus**」のそれぞれの引数タイプを「**TransactionItem**」と同様の変数の型になるように変更します。

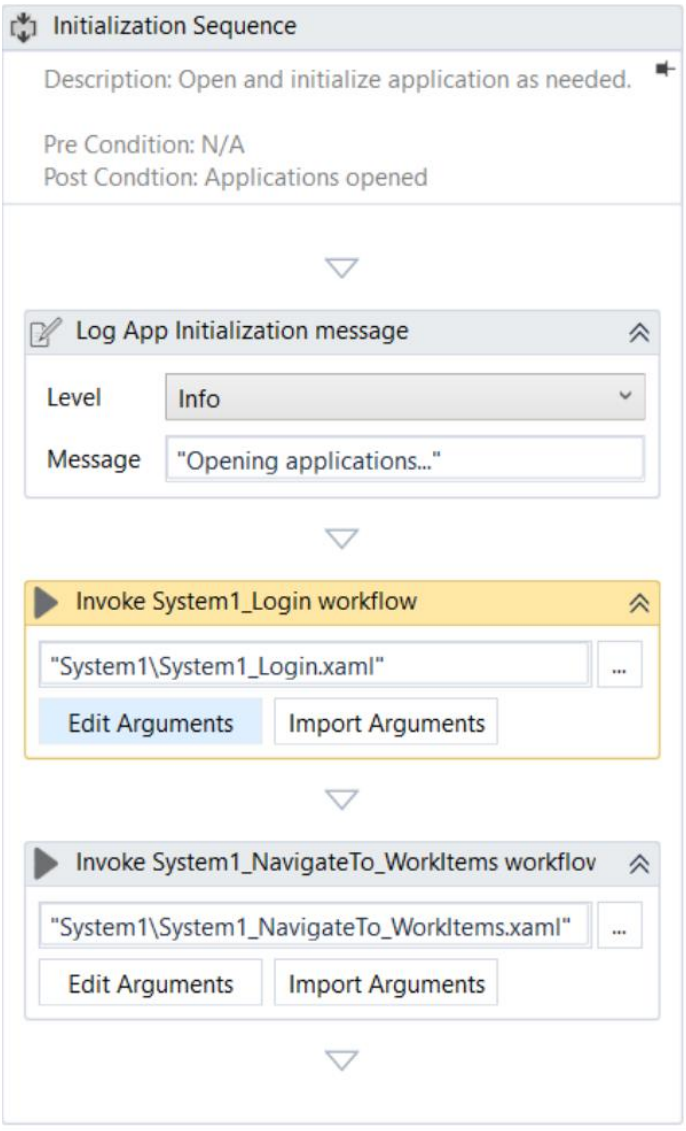
- この演習では、**ACME System 1** のみを使用します。ソリューションのルートディレクトリに **System1** という名前のフォルダーを作成します。

  - 今回のプロジェクトに次の共通部品（再利用可能なコンポーネント）を使用してください。これらは **System1** フォルダーにコピーします。
    - **System1_Login.xaml**

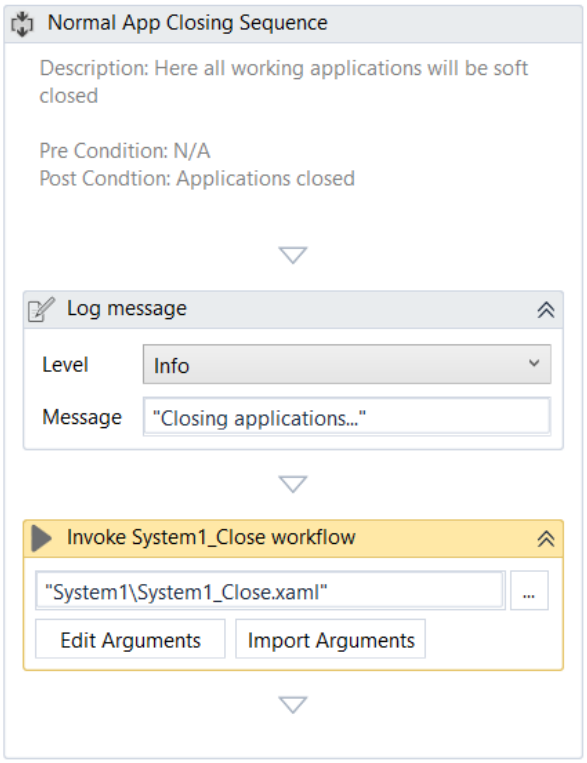
- **System1_Close.xaml**
- **System1_NavigateTo_WorkItems.xaml**
- **SendEmail.xaml** ファイルは、**Common** フォルダーにコピーします。

- 「**InitAllApplication**」ワークフローを開きます。

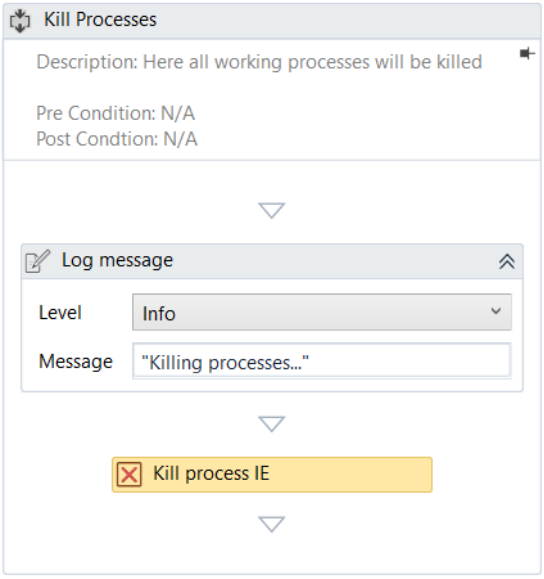
  - 「**System1¥System1_Login.xaml**」ワークフローを呼び出します。
  - 「**System1¥System1_NavigateTo_WorkItems.xaml**」ファイルを呼び出します。「**InitAllAplications**」ワークフローは以下のスクリーンショットの通りになります。



- 「**CloseAllApplications**」ワークフローを開きます。
    - 「**System1¥System1_Close.xaml**」ファイルを呼び出します。
    - 「**CloseAllApplications**」ワークフローは以下のスクリーンショットの通りになります。



- Framework フォルダーの「**KillAllProcesses.xaml**」ワークフローを開きます。
    - **Kill Process** アクティビティを追加し、名前を「Kill process IE」に変更します。
    - プロパティ上で、**ProcessName** を「iexplore」に設定します。
    - 「**KillAllProcesses.xaml**」プロジェクトは以下のスクリーンショットの通りになります。



- Framework フォルダーの「**GetTransactionData.xaml**」プロジェクトを開きます。

- ワークフローの目的を説明する短めの説明文を注釈（Annotation）として追記します。TransactionNumber ではページ番号を表す内容を追記しましょう。

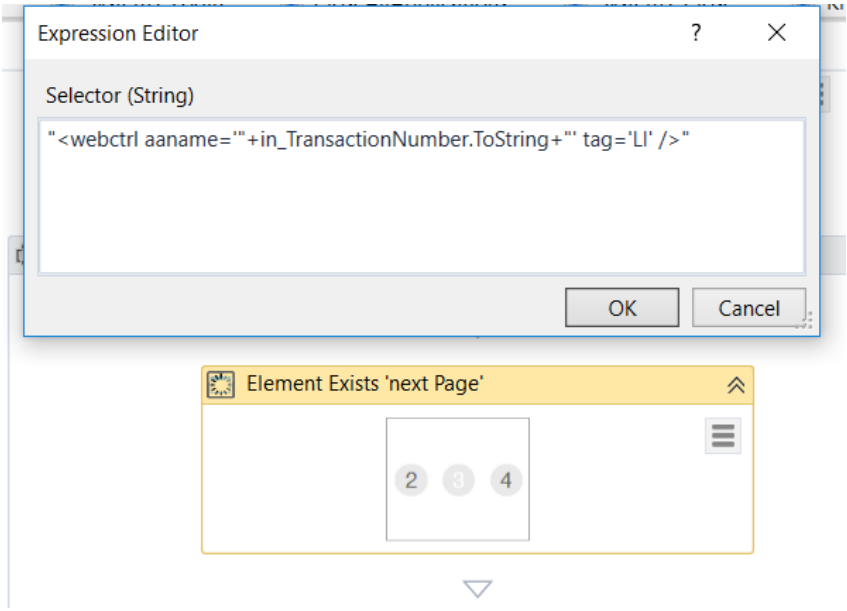
    - キューへのアップロードにはディスパッチャーのプロセスを使用するため、**Get Transaction Item** アクティビティを削除します。

    - 「**Write Transaction info in Logging Fields**」シーケンスの前に、**Attach Browser** アクティビティを追加し、「WorkItems」ページをスクリーンで指定します。

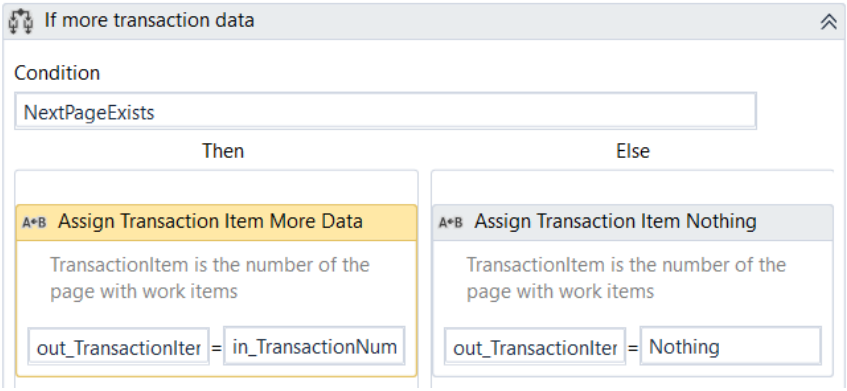
    - "次のページ" が利用可能かどうかをチェックする **Element Exists** アクティビティを追加します。ページ番号を示し、そのページ番号に関連する属性を使用するようにセレクターを編集します（注：前述の説明の通り、**in_TransactionNumber** 引数はディスパッチャープロセス用のページ番号となります。）

    - 出力パラメーターで、**NextPageExists** という名前の Boolean 型変数を作成します。

    - **Element Exists** アクティビティは次のスクリーンショットの通りになります。



- **If** アクティビティを追加し、全てのトランザクションデータを取得するように設定します。
- 次のページが存在している場合は、出力引数 **out_TransactionItem** に現在のページの値を設定します（ここでは **in_TransactionNumber** と設定）。
- 次のページが存在しない場合は、**out_TransactionItem** に対して “Nothing” いう値を入力します。
- **If** アクティビティは以下のスクリーンショットの通りになります。



- Process Transaction ステート内の「**Process.xaml**」ファイルを開きます。

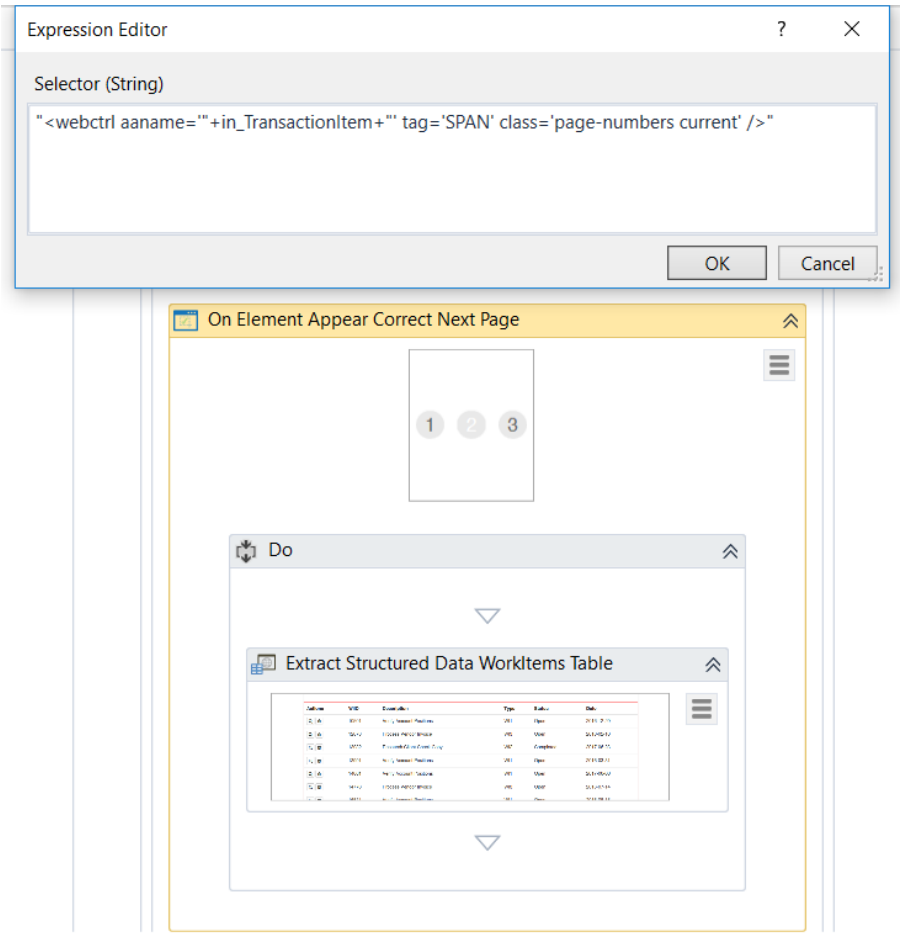
    - **Attach Browser** アクティビティを追加して、System1 ACME の「**WorkItems**」ページを参照します。

    - **Click** アクティビティを使用し、同じダイナミックセレクターを使用している処理中のページの番号を選択し、現在のページを識別します。
（**in_TransactionNumber** 引数を設定します。）

    - 次に、**On Element Appear** アクティビティを追加し、処理中のページが既に開かれているかどうかを確認します。クラス属性を使うとアクティブまたは非アクティブページの識別が可能です。これには UiExplorer を活用してください。

    - **On Element Appear** アクティビティで、WorkItems 上のテーブルを取得します。**Output** プロパティに、取得したテーブルを格納する **dt_WorkItems** という変数を作成します。

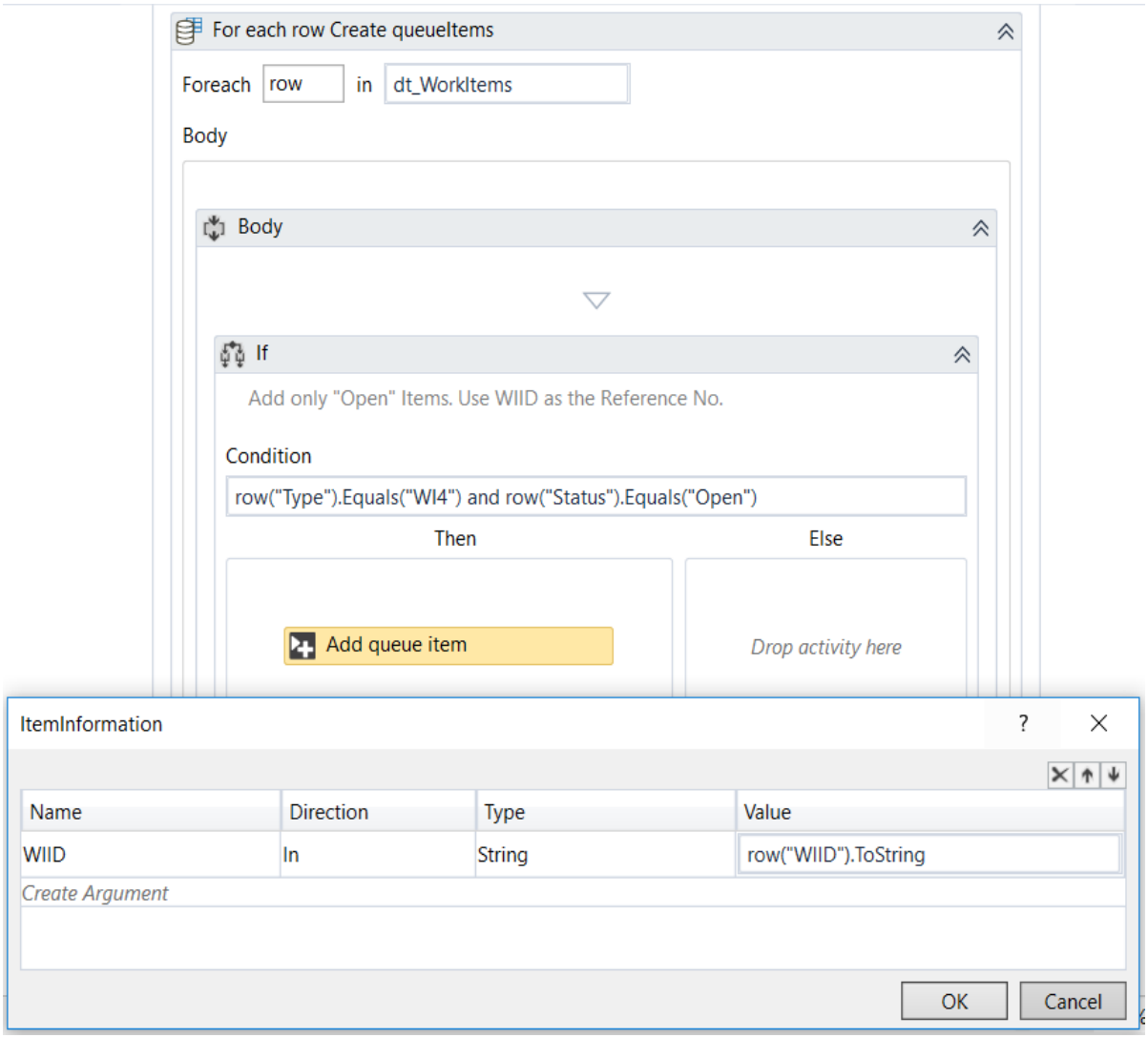
    - **On Element Appear** アクティビティは以下のスクリーンショットの通りになります。



- 次に、取得したデータテーブル内にある行のうち、Type が WI4 かつステータスが Open である全てのトランザクションの **WIID** を Orchestrator 上のキューにアップロードします。

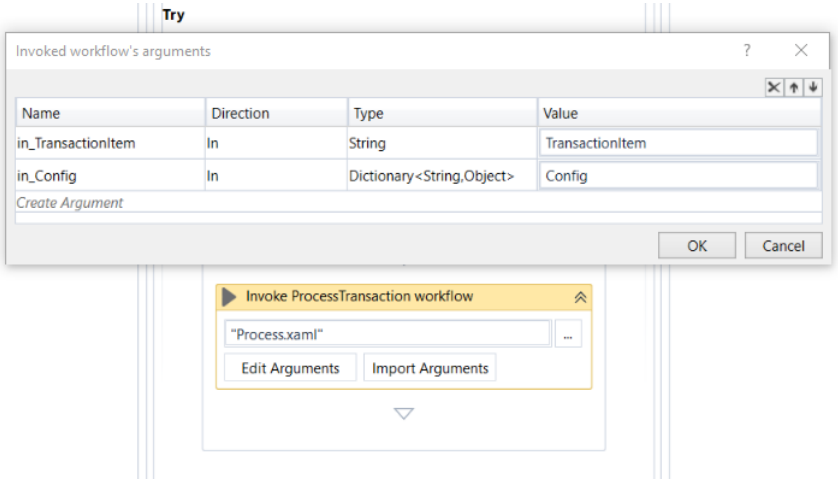
- キューにアップロードするためには、「**Add Queue Item**」アクティビティを使用します。**Properties** パネルで、**in_Config** ディクショナリの値を使用し **QueueName** フィールドに値を設定します。**ItemInformation** フィールドに、**WIID** という引数を作成し、引数に対応する値を設定します。

- **Add Queue Item** アクティビティは以下のスクリーンショットのようになります。



- **Main.xaml** ファイルを開きます。
    - **Process.xaml** から呼び出したプロジェクトの引数が正しく設定されていることを確認してください。

- 引数は以下のスクリーンショットの通りになります。



- 以上でプロセスの実装は完了です。次は、プロセスの動作検証を実施し、値がキューに正しくアップロードされることを確認します。

パフォーマープロセス

パフォーマーは、ディスパッチャーがキューにアップロードしたトランザクションをすべて処理するプロセスです。

そのため、TransactionItem の型は **QueueItem** である必要があります。

- REFramework テンプレートから始めます。
  - **TransactionItem** 引数は **QueueItem** 型にしてください。これは、REFramework の TransactionItem の既定の変数の型です。

- 現在のプロセスに合わせて **Config** ファイルを以下の通り編集します。

  - 「**Settings**」シート上の、「**QueueName**」パラメーターに「InHouse_Process4」値を追加します。Orchestrator 上のキューの名前も、同じ名前を使用します。
  - 「**Settings**」シートで、**System1 URL** と **System1 Credential パラメーター**用の設定を追加します。

- 「**Constants**」シートの、**MaxRetryNumber** の値は 0 のままにします。これは、リトライ機能は Orchestrator で処理されるためです。

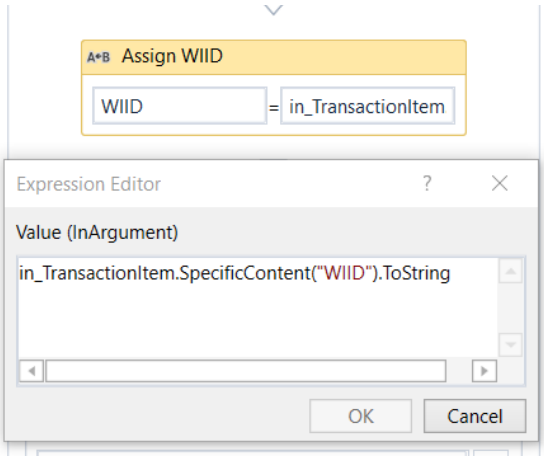
- この演習で使用するアプリケーションは ACME System 1 のみです。ルートフォルダーに **System1** という名前のフォルダーを作成します。

  - **パフォーマー**プロセスでは、次の “共通部品” を再利用してください。
    - **System1_Login.xaml**
    - **System1_Close.xaml**
    - **System1_NavigateTo_WorkItems.xaml**
    - **System1_NavigateTo_WIDetails.xaml**
    - **System1_UpdateWorkItem.xam l** ファイルを **System1** フォルダーにコピーします。
    - **SendEmail.xaml** ファイルを **Common** フォルダーにコピーします。

- **InitAllApplications** ファイルを開きます。
  - **System1¥System1_Login.xaml** ファイルを呼び出します。
- **CloseAllApplications** ファイルを開きます。
  - **System1¥System1_Close.xaml** ファイルを呼び出します。
- **KillAllProcesses.xaml** プロジェクトを開きます。**Kill Process** アクティビティを追加し、名前を **Kill process IE** に変更します。
  - **ProcessName** プロパティを **iexplore** に設定します。
- 「**Process Transaction**」ステート内の **Process.xaml** プロジェクトを開きます。

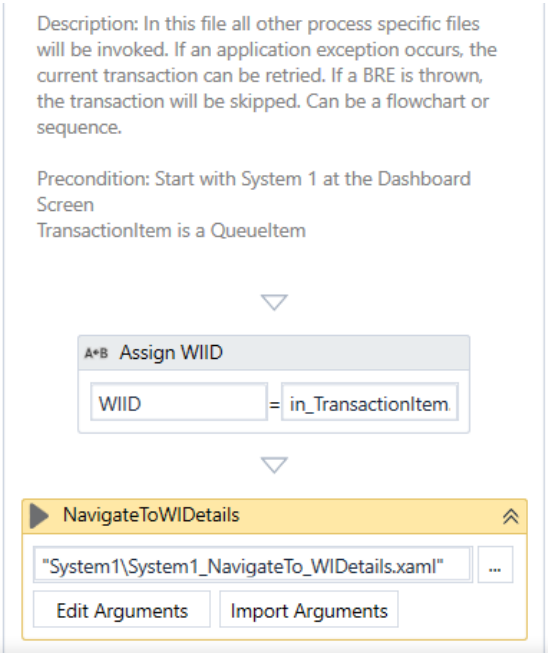
- 現在の WIID を設定する String 型の変数を作成します。
  - **Assign** アクティビティを追加し、今作成した変数をキューの値に設定します。それには、SpecificContent メソッドを使用し、in_TransactionItem.SpecificContent("WIID").ToString のように使用します。

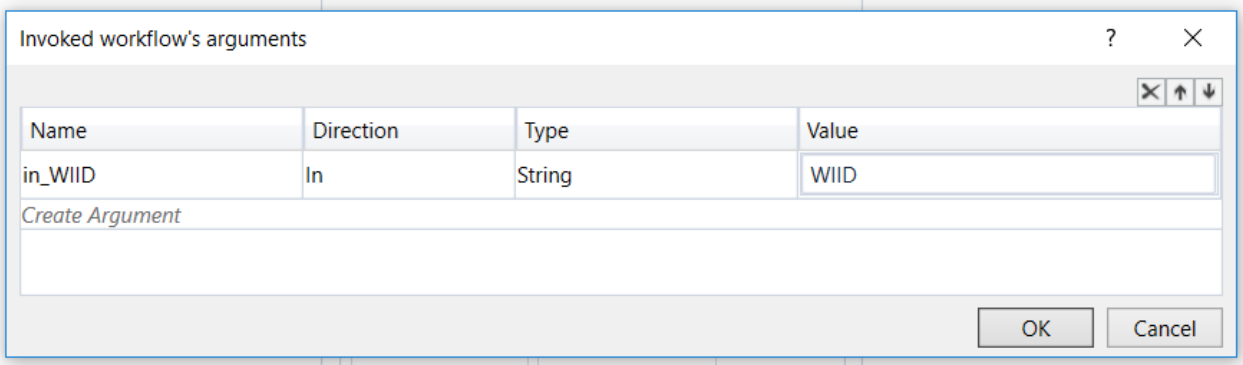
  - **Assign** アクティビティは以下のスクリーンショットの通りになります。

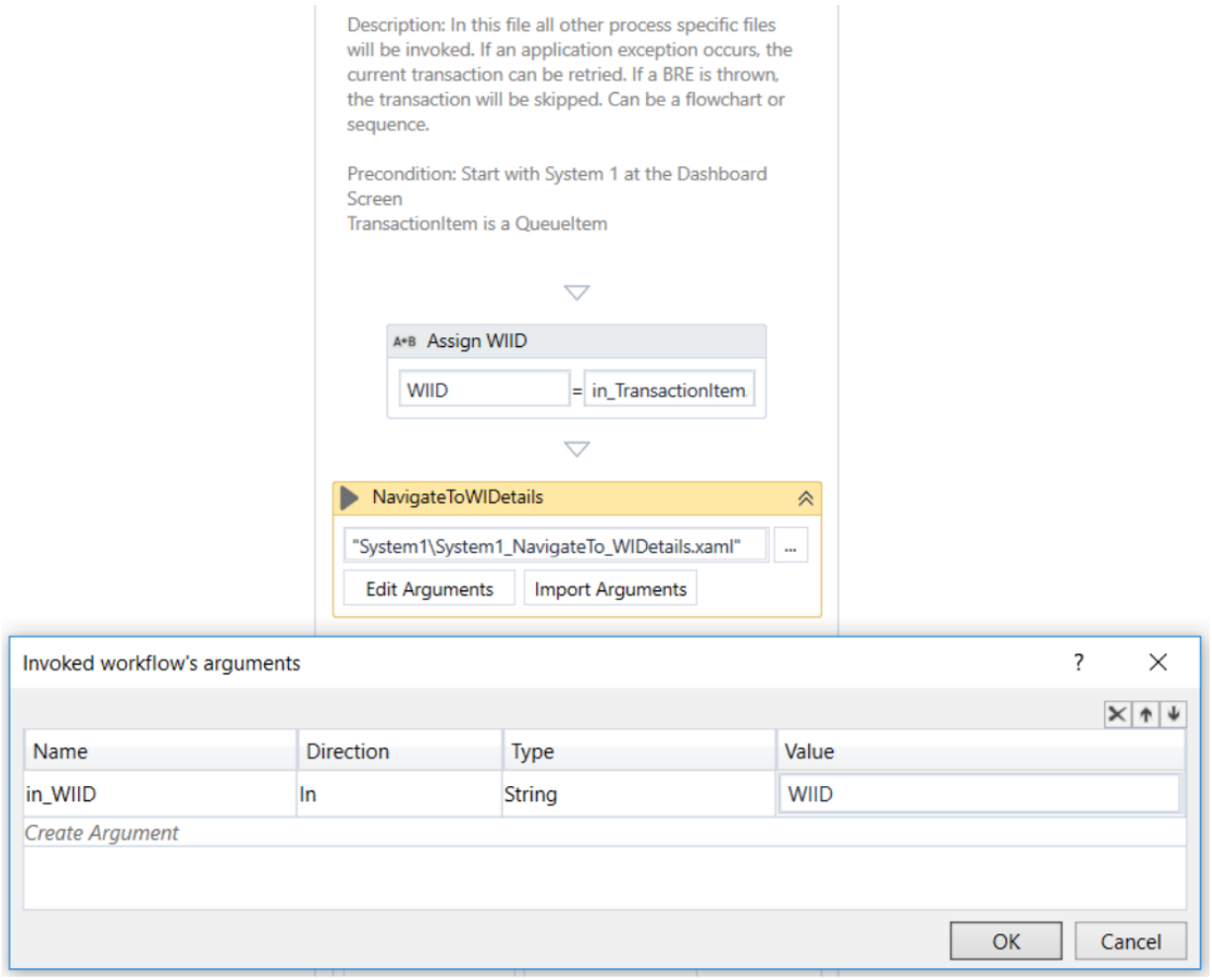


- **Process.xaml** ワークフローを開きます。
  System1¥System1_NavigateTo_WIDetails.xaml を呼び出します。
  SpecificContent メソッドを前述の通りに設定し、キューから取り込む必要のある引数を読み込みます。

- **Invoke** アクティビティと WIID 引数は以下のスクリーンショットの通りになります。







- 次に、System1 フォルダーに空のシーケンスを作成します。このシーケンスは、「Work Item Details」ページから TaxID 値を取得するために使用します。このワークフローに System1_ExtractVendorInformation.xaml という名前を付けましょう。
  - 説明文（注釈）として記載される前提条件は「『Work Item Details』ページがすでに開かれていること」です。

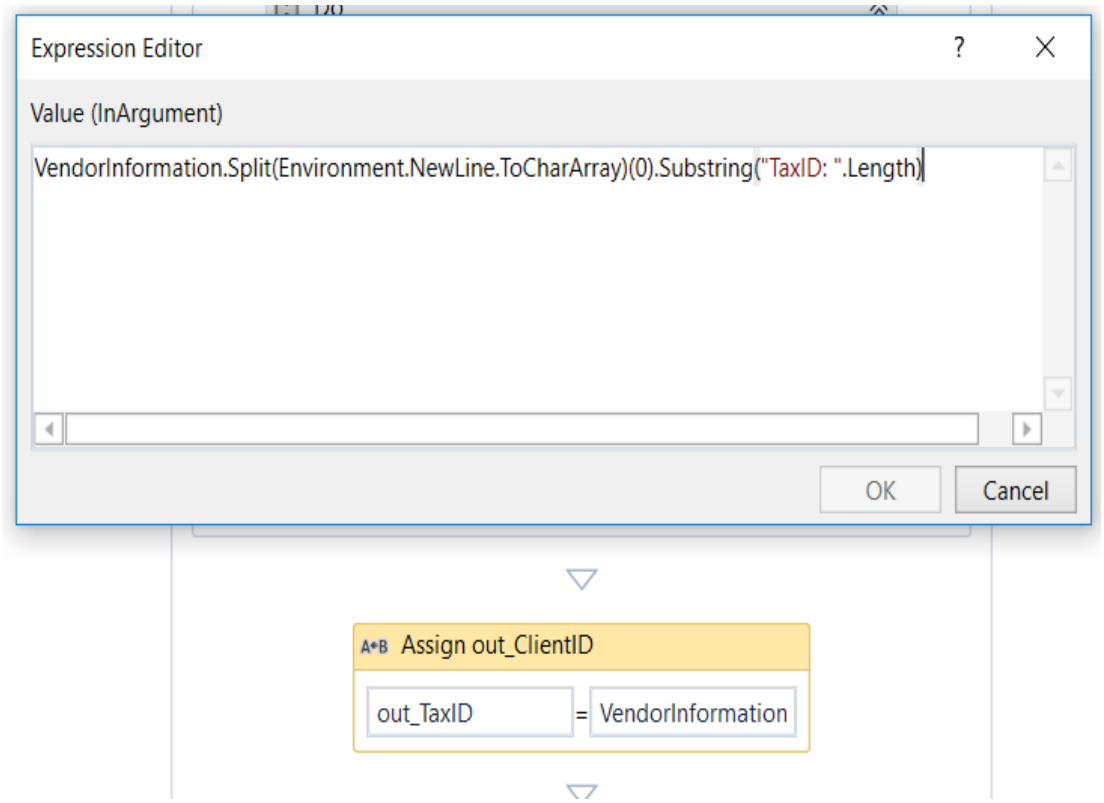
- out_TaxID という String 型の出力引数を作成します。この引数はプロジェクトで後から使います。
- **On Element Appear** アクティビティを追加し、「Work Item Details」ページの Vendor Information を参照します。**RepeatForever** プロパティを False に設定します。

- **Get Text** アクティビティを **On Element Appear** の一部である **Do** シーケンスに追加します。前述のパラグラフを参照し、そこからテキストを取得します。

- **Get Text** アクティビティの **Properties** パネルで、**Output** フィールドに VendorInformation という名前の変数を作成します。

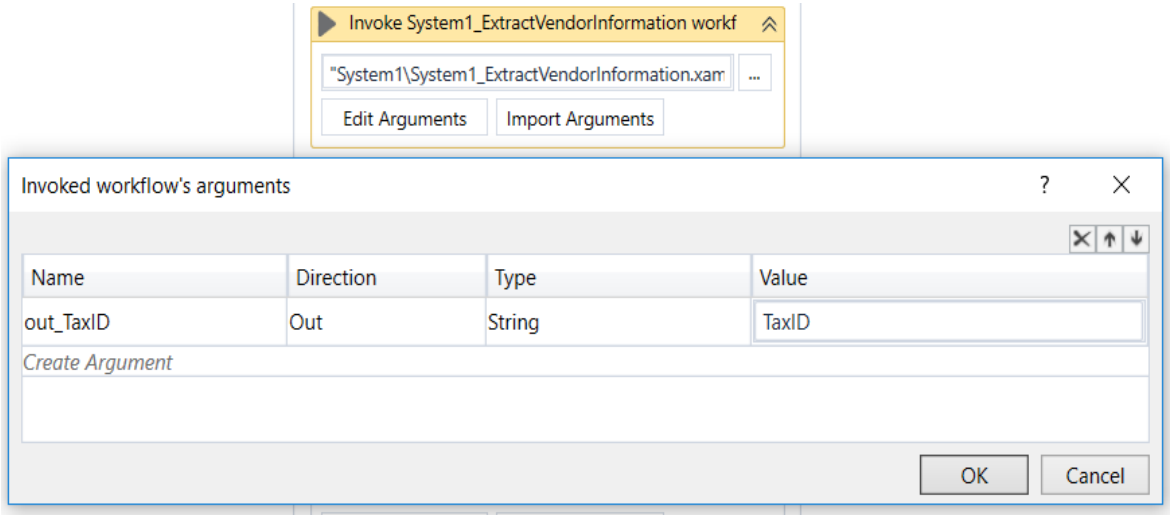
- 次に、**On Element Appear** アクティビティの後に **Assign** アクティビティを追加し、out_TaxID 引数の値（先ほど作成した VendorInformation 変数を使って取得できます）を TaxID 値に設定します。

- **Assign** アクティビティは以下のスクリーンショットの通りになります。



- **Process.xaml** を開き、以前作成したワークフローファイルから取り込んだ出力引数の値を格納する、TaxID という名前の String 型変数を作成します。**System1¥System1_ExtractVendorInformation.xaml** を読み出し、引数を取り込みます。

- 呼び出されたワークフローは以下のスクリーンショットの通りになります。



- 次に、「Dashboard」ページに移動するためのワークフローを新規に作成します。このワークフローに System1_NavigateTo_Dashboard.xaml という名前を付けましょう。このワークフローには、「Dashboard」ページを選択する **Click** アクティビティを追加します。

- **Process.xaml** ファイルでワークフローを呼び出します。
- TaxID を取得後、**Download Monthly Report** ページに移動します。
- まず、**System1_NavigateTo_MonthlyReport** という名前で空のシーケンスを作成します。

    - このシーケンスに合った説明文を追加します。「Dashboard」ページがすでに開かれていることが前提条件となります。

    - **Attach Browser** アクティビティを追加して、「Dashboard」ページを参照します。

    - **Click** アクティビティを 2 つ追加し、**Reports** に移動し、続いて **Download Monthly Report** に移動します。

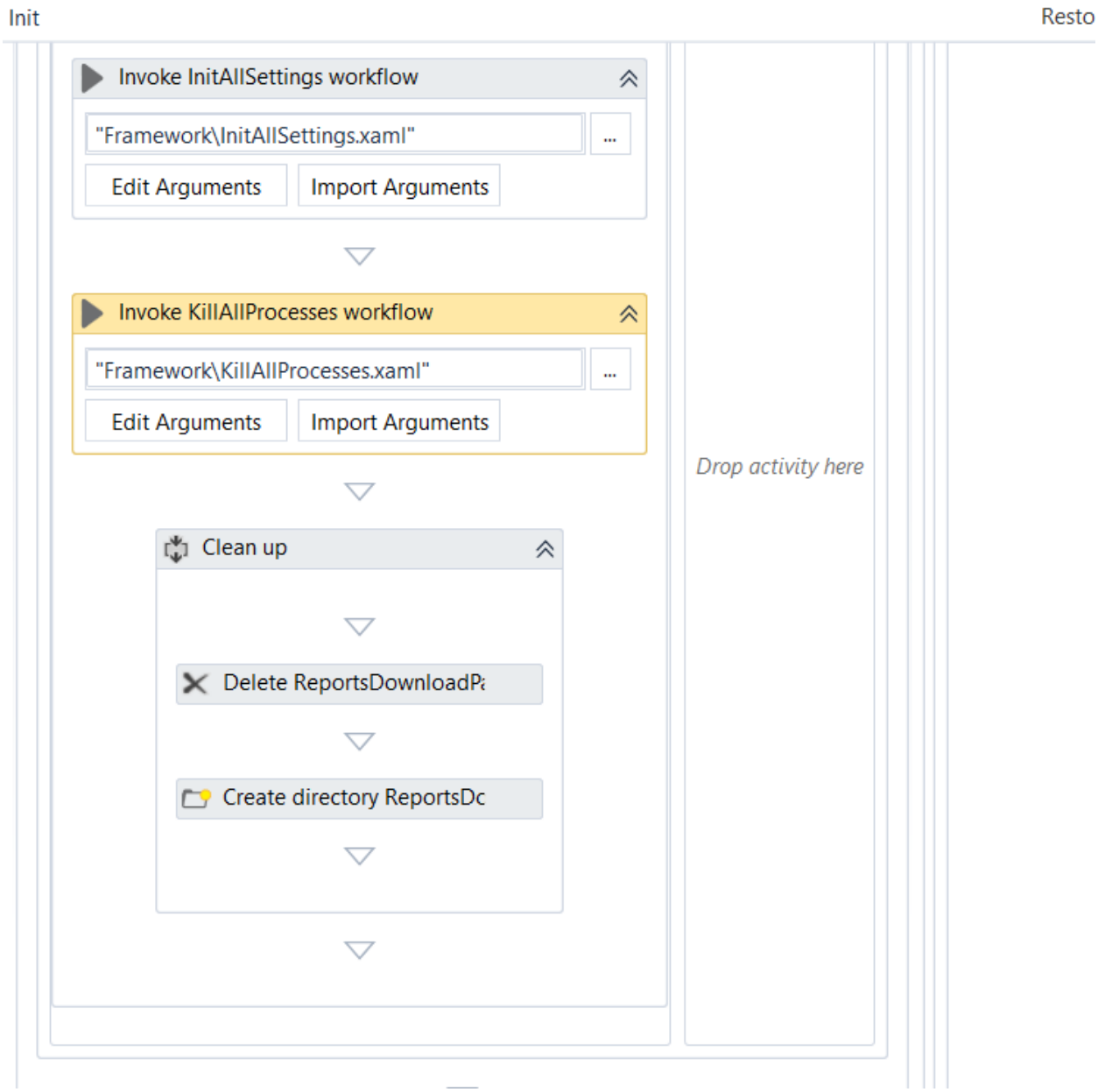
    - 2 つのアクティビティの **Properties** パネルで、**Simulate Click** フィールドを有効化します。**Reports** メニューは、「**Reports**」ボタンにカーソルを重

ねた場合にのみ表示されます。**Download Monthly Report** でクリックをするには、UiExplorer を使います。「**F2**」キーを使って 3 秒間一時停止し、ボタンが表示される前に要素が画面上に表示されるのを確認します。

- 既存の請求書をダウンロードすると問題が発生することがあります。

- この演習では例外を回避するために、ロボットの動作を開始させるたび、環境の状態をリセットしておく必要があります。そのためには、**Config** ファイルに記述されている **Download Reports** フォルダーを一旦削除し、作成し直します。次に、Main.xaml ファイルで Init State を開きます。呼び出された KillAllProcesses.xaml ファイルの後に、**Clean Up** というシーケンスを追加します。ここでは、**Delete** と **Create Directory** という 2 つのアクティビティを使用します。

- **Delete** アクティビティの **Properties** パネルで、**Path** フィールドに **Config** ファイルの値を設定します。このように設定することで、ロボットが開始されると必ず Data¥Temp が新しく作成されるようになります。Clean Up シーケンスは以下のスクリーンショットの通りになります。



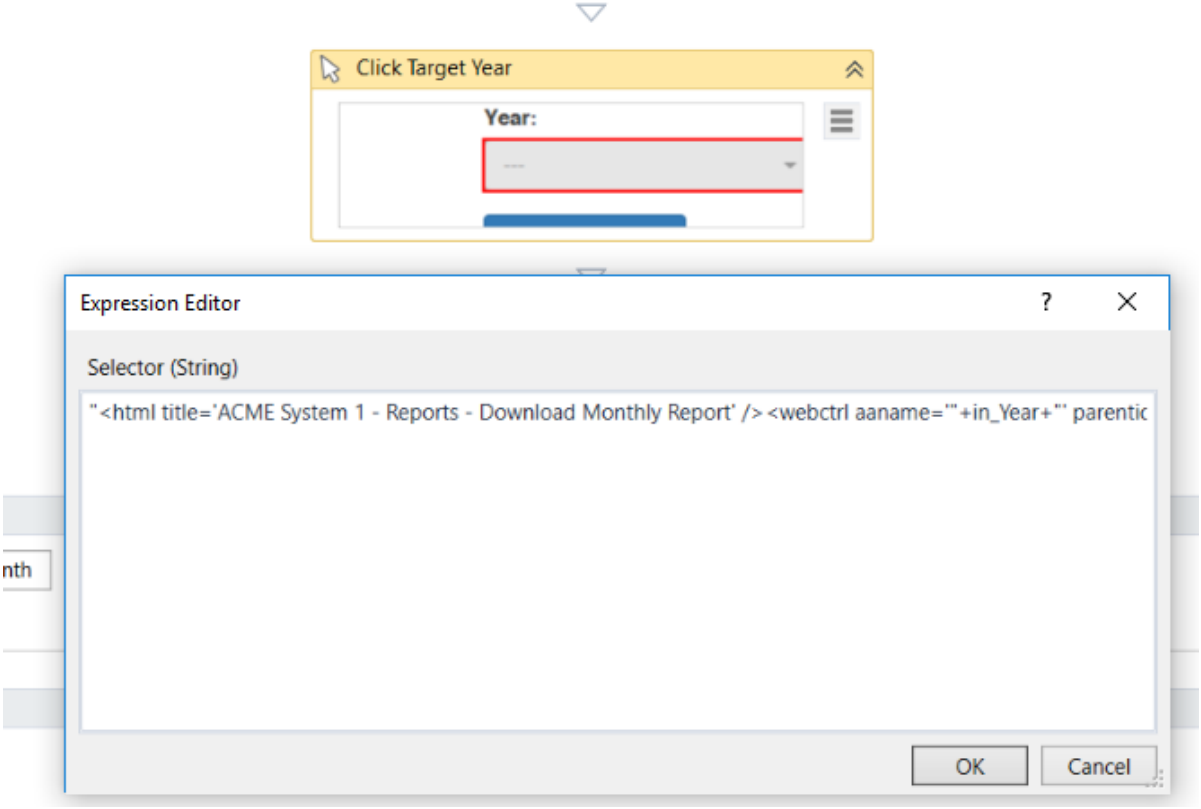
- 「**Process.xaml**」ファイルに戻り、先ほど作成した **System1¥System1_NavigateTo_MonthlyReport.xaml** ファイルを呼び出します。

- **ReportYear** という名前で新しい変数を作成します。**Assign** アクティビティを使用し、値を前年に設定します。

- 次の手順は年次レポートの作成です。まず、**System1_CreateYearlyReport.xaml** という名前で空のシーケンスを作成します。
    - 前提条件として ACME System 1 アプリケーションに Monthly Report Page がすでに開かれている、という内容の注釈（Annotation）を追記します。
    - 以下の 3 つの入力引数を作成します。
        - **in_TaxID** – メインファイルにより提供されます。TaxID 値を格納します。
        - **in_Year** – メインファイルにより提供されます。レポートの作成対象となる年を格納します。
        - **in_ReportsDownloadPath** – 月次レポートがダウンロードされるフォルダーです。
    - **out_YearlyReportPath** という名前の出力引数を 1 つ作成します。月次レポートを 1 つにまとめた後に作成される年次レポートファイルのパスを格納します。
    - Config ファイルの Settings シートに、レポートがダウンロードされるフォルダーパスを参照する項目を新しく追加します。Name フィールドには **ReportsDownloadPath** と入力し、Value フィールドには **Data¥Temp** と入力します。
    - dt_YearlyReport という名前の Data Table 変数を新しく作成します。この変数を使ってすべての月次レポートを 1 つにまとめます。**Assign** アクティビティを使って値に**新しい Datatable** を設定します。
    - 次に、**Type Into** アクティビティを使用し、ACME System 1 アプリケーションに TaxID 値を入力します。「Monthly Report」ページはこの時点ですでに開かれているはずです。

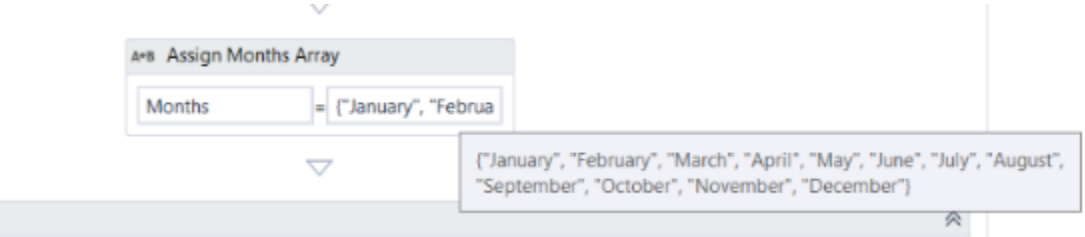
- **Click** アクティビティを追加して当該年を選択します。**Simulate Click** プロパティを有効にし、次に対象の年を選択します。これにより、ドロップダウンメニューが表示されず要素が非表示であったとしても、該当するアクティビティをバックグラウンドで実行させることができます。

- セレクターの aaname 属性を **in_Year** 引数に変更します。**Click** アクティビティとセレクターは以下のスクリーンショットの通りになります。



  - 別の方法として、**Click** アクティビティを使わずに **Select** アクティビティを使う方法があります。この場合は、該当する年度を参照するように UiExplorer を使用し、セレクターを更新する必要があります。

- **Months** という文字列配列の変数を作成します。**Assign** アクティビティを使用し、**Month** ドロップダウンリストに従って値を設定します。**Assign** アクティビティは以下のスクリーンショットの通りになります。



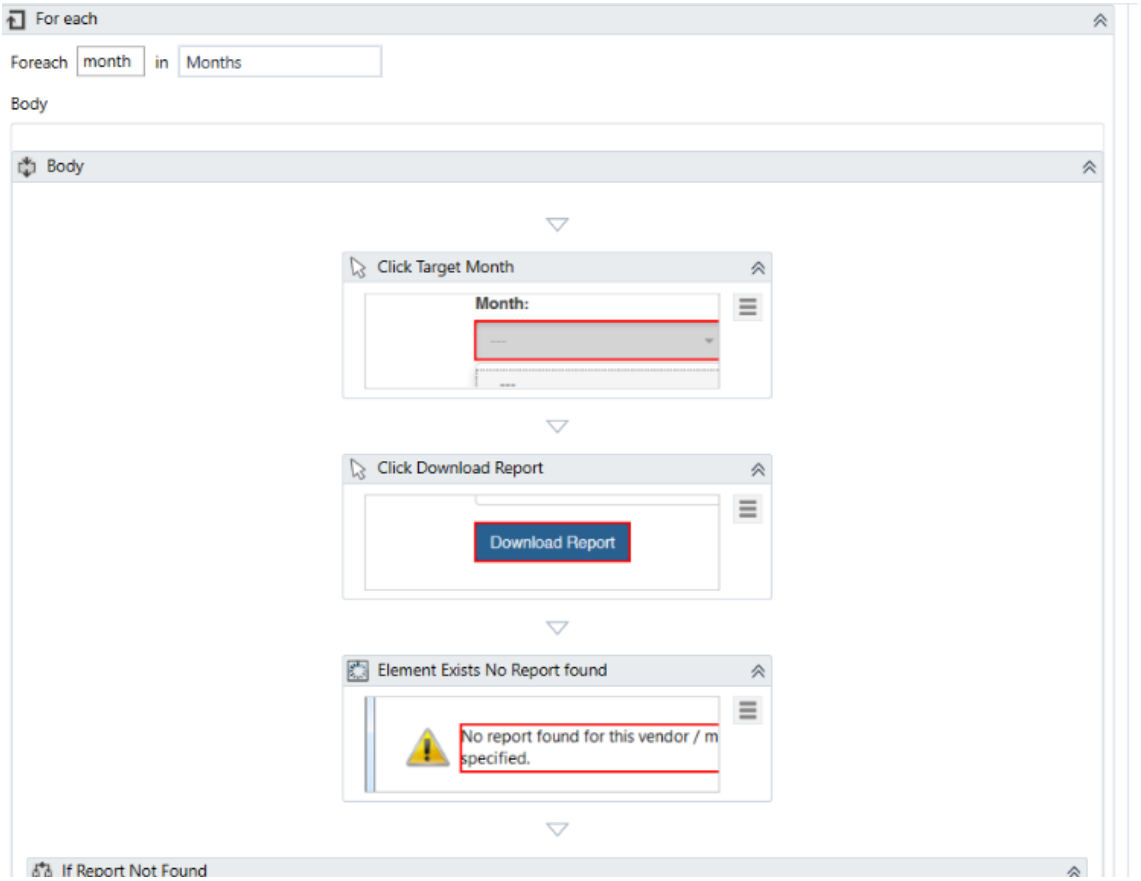
- 次に、月ごとのレポートをダウンロードするために、Months 配列をループ処理で読み取り、指定された月をドロップダウンボックスから選択し、レポートをダウンロードする **For Each** アクティビティを追加します。

- **For Each** アクティビティ内に対象の月を選択する **Click** アクティビティを追加します。Year ドロップダウンのときと同じように、ダイナミック aaname 属性を使ってセレクターを編集します。**Click** アクティビティの代わりに **Select** アクティビティを使用することも可能です。

- 次に、**Click** アクティビティを追加し、「Download」ボタンを参照します。その後 **Simulate Click** を有効化します。

- レポートの中には存在しないものもあるため、**Element Exists** アクティビティを使用し、レポートが存在しない場合に表示されるポップアップウィンドウのラベルを示します。**Output** プロパティに ReportNotFound という名前の Boolean 型変数を作成します。

- **For Each** アクティビティは、以下のスクリーンショットの通りになります。

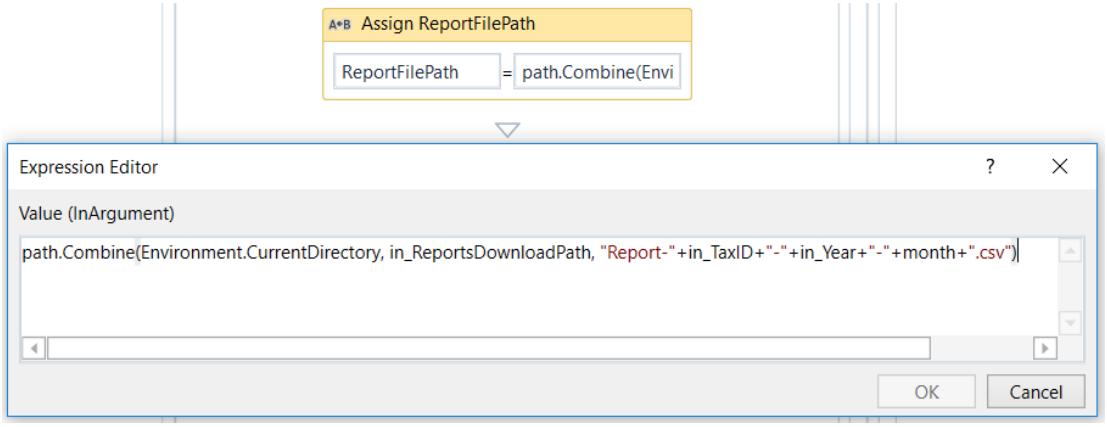


- 次に、レポートが存在しているかを確認する **If** アクティビティを追加します。**Condition** として ReportNotFound 変数を使います。**Then** セクションに **Click** アクティビティを追加し、ターゲットとしてドロップダウンウィンドウの「**OK**」ボタンを設定したら、次の月に進みます。

- Else セクションで、以下のアクティビティを使用し、Report をダウンロードします。

- **Click** アクティビティを追加し、「**Save as**」ボタンに移動させます。次に **Simulate Click** プロパティを有効にします。

- **Assign** アクティビティを追加します。ReportFilePath という名前で新しい変数を作成し、値に .csv 形式でダウンロードされた月次レポートのパスとファイル名を設定します。



- **Type Into** アクティビティを使用し、「Save As」ウィンドウの「File Name」フィールド内で、ReportFilePath に値を入力します。次に **Simulate Type** プロパティを選択します。

- **Click** アクティビティを追加し、「**Save**」ボタンに移動させます。次に **Simulate Click** プロパティを有効にします。

- ダウンロードしたばかりの csv ファイルを読み取ります。出力プロパティに、dt_MonthlyReport という名前のデータテーブル変数を作成します。

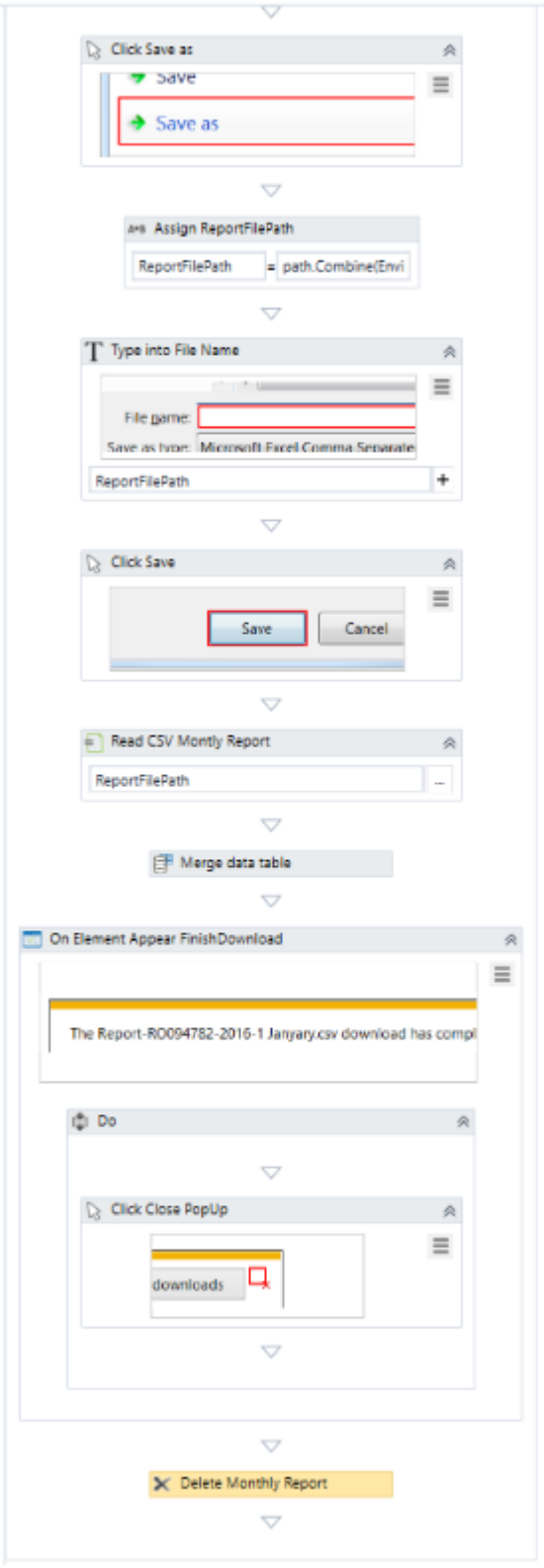
- 次に、**Merge Data Table** アクティビティを使って、**dt_MonthlyReport** の値を **dt_YearlyReport** データテーブルに追加します。

- 月次レポートのダウンロード時間は状況によって異なることから、次のレポートのダウンロードが始まる前にファイルのダウンロードが完了したことを確認するため、**On Element Appear** アクティビティを追加し、ダウンロードポップアップを参照します。ダイナミック属性値のワイルドカードを使ってセレクターを更新します。次に **Wait Visible** プロパティを有効にします。

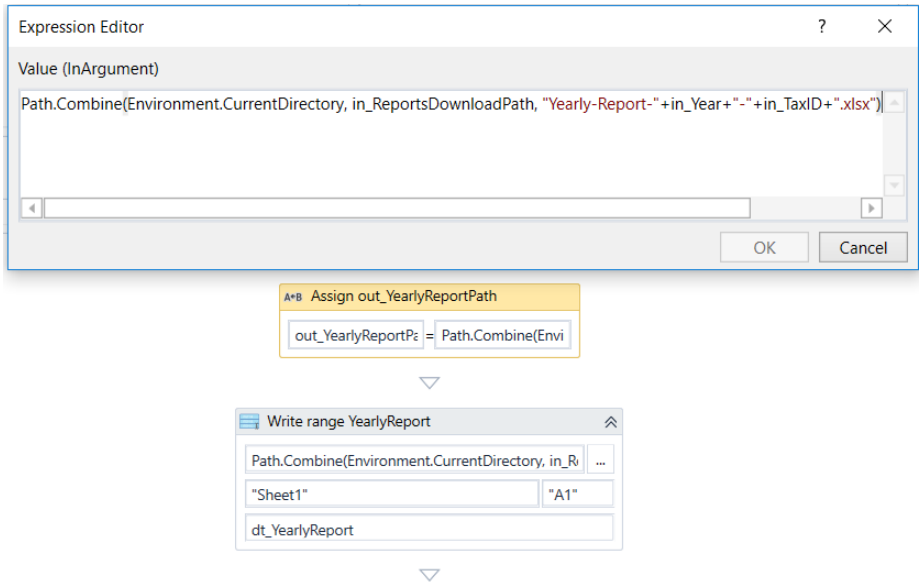
- **On Element Appear** アクティビティの **Do** セクションで、ポップアップを閉じる **Click** アクティビティを追加します。次に **Simulate Click** プロパティを選択します。

- ダウンロードを開始する前に Monthly Report（月次レポート）ファイルを削除する **Delete File** アクティビティを追加します。

- **Else** セクションは以下のスクリーンショットの通りになります。


Click Save as
Save
Save as
Assign ReportFilePath
ReportFilePath
= path.Combine(Envi
Type into File Name
File name:
Save as type:
ReportFilePath
Click Save
Save
Cancel
Read CSV Montly Report
ReportFilePath
Merge data table
On Element Appear FinishDownload
The Report-RO094782-2016-1 Janyary.csv download has compl
Do
Click Close PopUp
downloads
Delete Monthly Report

- このシーケンスにおける次のアクティビティは **Assign** です。**out_YearlyReportPath** 引数の値を設定します。Yearly Report Excel ファイルの名前が PDD ファイルのモデルと一致しており、さらにパスが Config ファイルのパスと一致していることを確認してください。



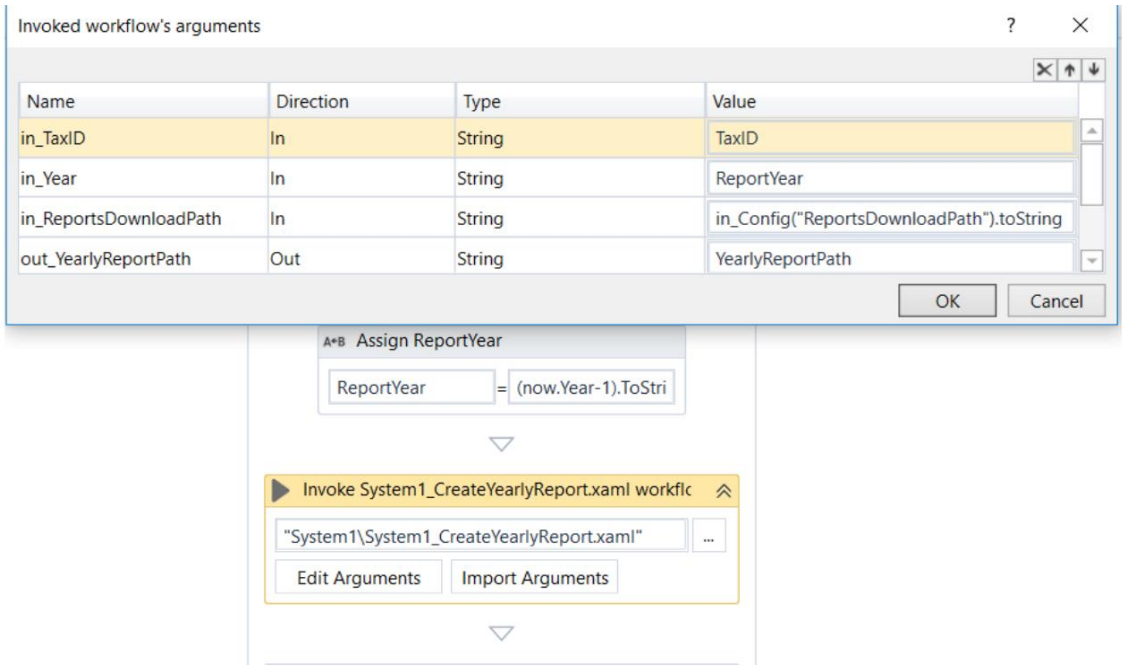
- 入力引数に既定値を設定し、ワークフローをテストします。

- **Process** ワークフローに戻ります。

  - 先ほど作成した **System1¥System1_CreateYearlyReport.xaml** ファイルを呼び出します。引数を読み込んで取り込みます。

  - YearlyReportPath という名前で String 型変数を作成します。前述のワークフローで **out_ yearlyReportPath** 引数の値の取得に使います。

  - 呼び出されたワークフローと引数は津語のスクリーンショットの通りになります。



- 次に **System1¥System1_NavigateTo_Dashboard.xaml** を呼び出して、「Dashboard」ページに戻ります。

- 以上で、Yearly Report ファイルは完成です。ACME System 1 アプリケーションの Reports - Upload Yearly Report ページにファイルを移動しましょう。まず、**System1_NavigateTo_UploadYearlyReport** という名前で空のシーケンスを新規に作成する必要があります。

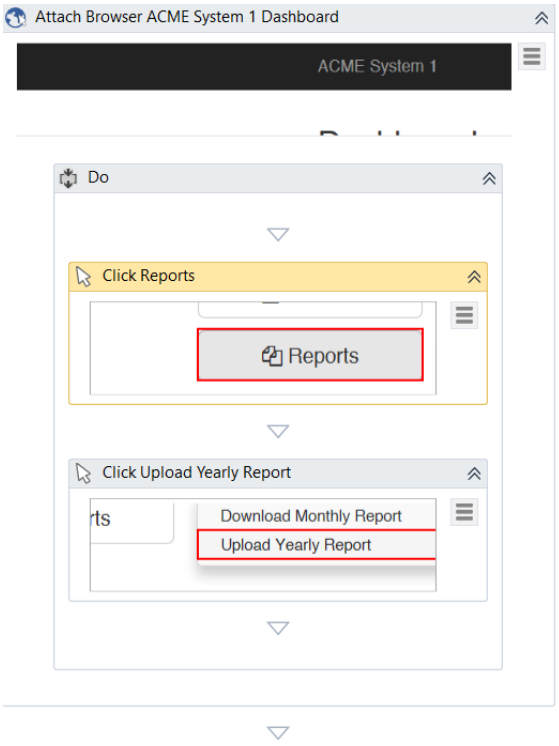
    - 説明文を追加して新しいワークフローを開始します。
      「Dashboard」ページがすでに開かれていることが前提条件となります。

    - **Attach Browser** アクティビティを追加して、「Dashboard」ページを参照します。

    - 「**Reports**」ボタンを選択する **Click** アクティビティを追加します。次に **SimulateClick** プロパティを有効にします。

- 次に、「**Upload Yearly Report**」ボタンを選択する **Click** アクティビティを新しく追加します。
  System1_NavigateTo_MonthlyReport.xaml ワークフローと同じように、**UiExplorer** を使ってこのボタンをクリックします。次に **SimulateClick** プロパティを有効にします。

- ワークフローは以下のスクリーンショットの通りになります。



- **Process** ワークフローに戻ります。

  - **System1¥System1_NavigateTo_UploadYearlyReport.xaml** ファイルを呼び出します。

  - 「**Reports - Upload Yearly Report**」ページに移動したら、年次レポートを更新する必要があります、まず、**System1_UploadYearlyReport** という名前で空のシーケンスを作成します。

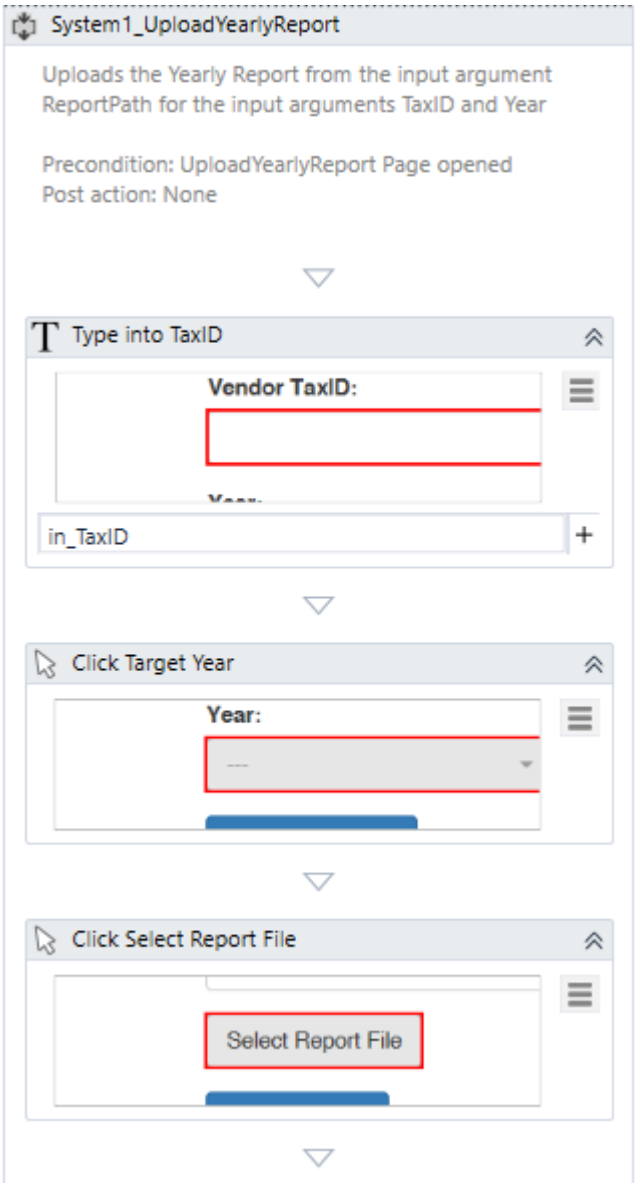
- 説明文を追加して新しいシーケンスを開始します。「Reports - Upload Yearly Report」ページがすでに開かれていることが前提条件となります。

- 年次レポートファイルのアップロードに必要な情報は、taxID、ファイルパス、対象年です。アップロードが実行されると、確認 ID が出力されます。そのため、このワークフローでは、**in_TaxID**、**in_ReportPath**、**in_Year** という 3 つの String 型入力引数と、**out_UploadID** という 1 つの String 型出力引数を使う必要があります。

- 以上で、準備はすべて整いました。**Type Into** アクティビティを追加し、**Vendor TaxID** フィールドを表示します。テキストとして **in_TaxID** 引数を使います。
- **Year** ドロップダウンメニューで **Click** アクティビティを使います。**in_Year** 引数を使ってセレクターを更新します。

- 「**Select Report File**」ボタンで **Click** アクティビティを使います。

- シーケンスは、以下のスクリーンショットの通りになります。
-



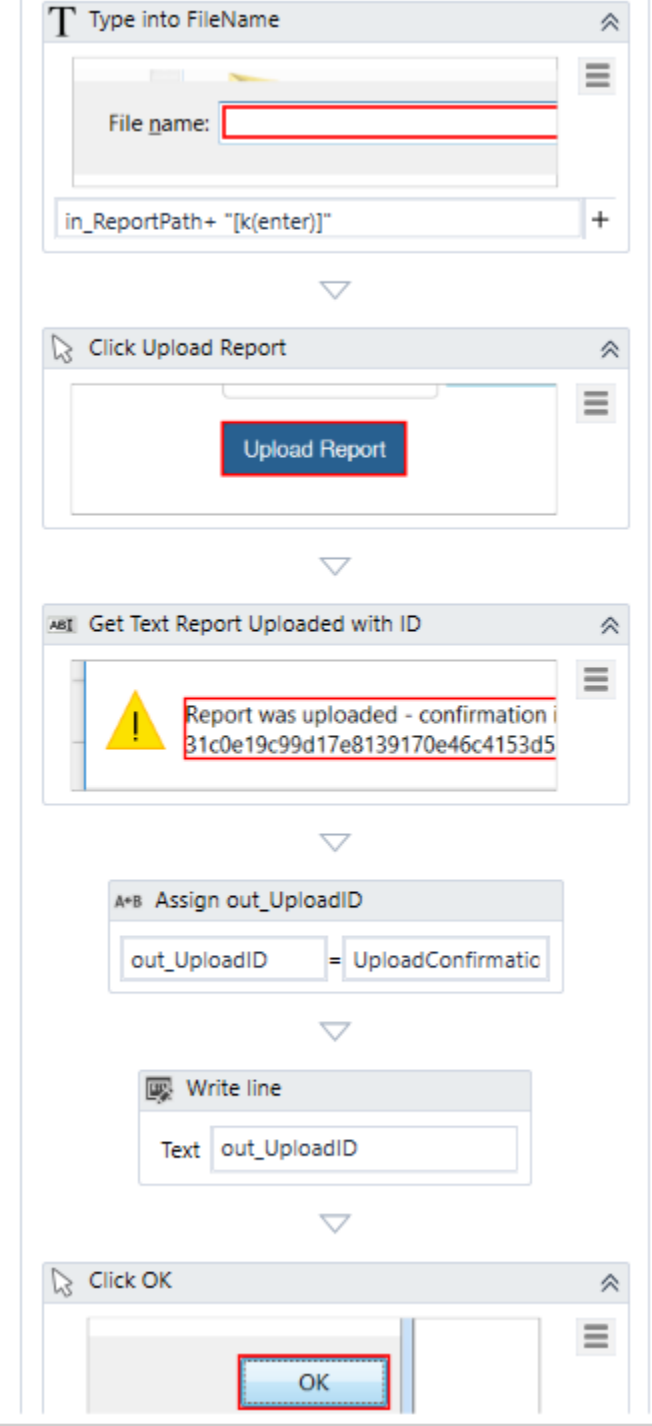
- 「**Choose file to Upload**」ウィンドウが表示されるので、Type Into アクティビティを使って年次レポートファイルのパス(**in_ReportPath**)を設定し、Enter キーを押しましょう。

- **Click** アクティビティを使って「Upload」ボタンを選択します。
- アップロード確認 ID の付いたポップアップウィンドウが表示されるので、**Get Text** アクティビティを使ってその値を取得します。**Output** プロパティで、UploadConfirmation という名前の変数を作成します。

- **Assign** アクティビティを使って、**out_UploadID** 引数の値を確認 ID の値に設定します。Substring メソッドを使って値を取得します: UploadConfirmation.Substring("Report was uploaded - confirmation id is ".Length)

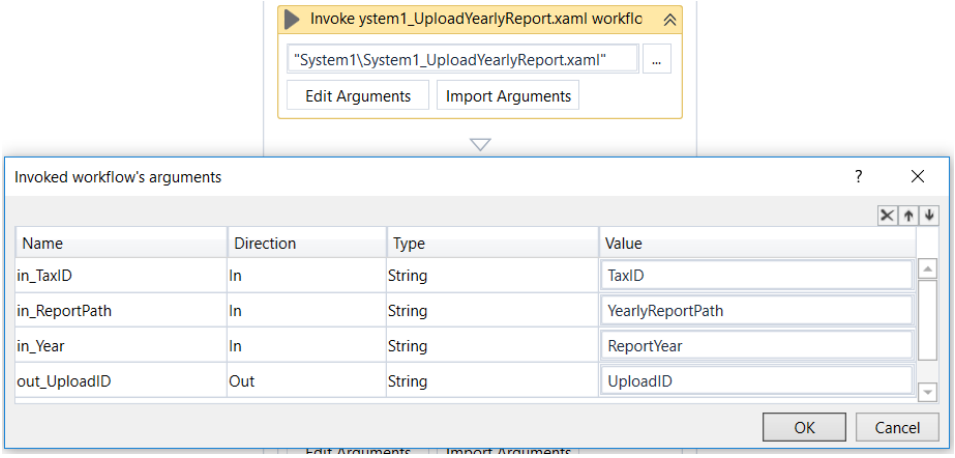
- 「**OK**」ボタンで **Click** アクティビティを使います。作業は以上です。

- ワークフローの最後の部分は以下のスクリーンショットの通りになります。

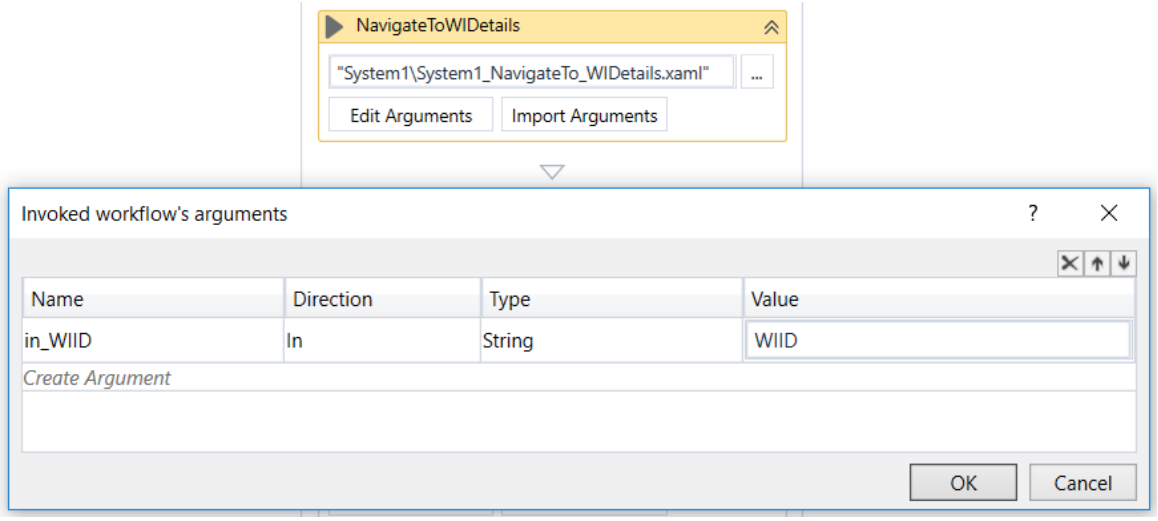


- **Process** ワークフローに戻ります。
    - **System1¥System1_UploadYearlyReport.xaml** ファイルを呼び出し、対応する引数を取り込みます。**Process** ワークフローに、**out_UploadID** 引数の値を格納する変数を作成します。変数に UploadID という名前を付けます。

- 呼び出し部分は次のスクリーンショットの通りになります。



- この時点で、年次レポートファイルのアップロードは完了しているので、Work Items のステータスを更新する必要があります。
- **System1¥System1_NavigateTo_Dashboard.xaml** ファイルを呼び出して「Dashboard」ページに移動します。
- **System1¥System1_NavigateTo_WIDetails.xaml** ファイルを呼び出して、特定の作業項目の Details ページに移動します。ご覧の通り、このワークフローには WIID という引数が 1 つあります。



- **System1¥System1_UpdateWorkItem.xaml** ファイルを呼び出し、作業項目のステータスを **Complete** に更新します。**Comment** と **Status セクション**には必ず正しい引数を設定してください。PDD ファイルで説明したように、ステータ

スは**「Completed（完了）」**に、Comment の値は **Uploaded with ID [uploadID]**に設定します。

- 最後に、次の項目を処理できるよう、アプリケーションを初期状態にしておく必要があります。そのためには、**System1¥System1_NavigateTo_Dashboard.xaml** ファイルを呼び出して「Dashboard」ページに戻るように設定します。

- 以上でプロセスの実装は完了です。最後に、プロセス全体の動作検証が必要となります。
  個々のワークフローについては作成直後に、引数の既定値を使用して動作検証を行います。

  - **Main** ワークフローを何度か実行し、その都度正しく実行されることを確認します。うまくいかない場合は、問題を修正して再度実行します。

  - User options メニューの **Reset test data** オプションを使うと、新しいテスト用データ一式が作成されます。

  - 必要に応じて、**ディスパッチャー**を使用し、新しいトランザクションアイテムをアップロードします。

まとめ

今回のセッションでは、まず**ディスパッチャー**プロセスを使用し、Orchestrator のキューにトランザクションをアップロードし、次に、**パフォーマー**を使用し、キュー内のトランザクションごとに処理を行いました。処理完了後は、キュー上のトランザクションのステータスが変化していることに注目してください。トランザクションアイテム自体は独立しており、依存関係はありません。
そのため、複数のパフォーマー専用のロボットにおいて、同時に処理を行うことが可能です。