Workflow Inspector

目次

概要

RPA 実装プロジェクトにおけるワークフローの品質担保は、プロジェクトの成功に大きな影響を与えます。経験豊富な開発者やピアレビューなどの実践により、ワークフローの品質を改善できますが、多くの XAML ファイルを自動的に検査できれば、予備の自動評価として便利です。

つまり、ワークフローレビューの過程で RPA 開発者とソリューションアーキテクトを支援でき、一般的に使用される開発のベストプラクティスに従っているかどうかを確認できるツールが求められているということです。

さらに、開発規約は実装プロジェクトごとに異なるため、ツールのユーザーが簡単に内容を理解できること、および実装ごとの要求と標準を満たすためにツールを簡単に拡張できることが求められます。

本書で説明する Workflow Inspector は、上記のニーズに対応するため、以下の特徴を備えています：

- UiPath Studio プロジェクトとして提供されるため、ユーザーが別のプログラミング言語やツールを学ばなくても、理解しやすく、修正することも容易です。
- Excel ファイルを使用してチェックの内容と設定を指定します。このファイルにはチェックについての説明と改善の推奨も含まれており、RPA 開発者の教育に役立ちます。
- XML および XAML のドキュメントのクエリーを行う標準的な方法である XPath 式を使用して、XAML ファイルの構造をトラバースします。これにより、ツールの拡張に必要な柔軟性を提供します。

使い方

Workflow Inspector のもっとも簡単な利用方法は、UiPath Studio で開き、メインワークフローを実行することです。ユーザーはチェックする UiPath Studio プロジェクトのフォルダーを指定するよう求められ、フォルダー内の XAML ファイルが指定されたベストプラクティスチェックに合格するかどうかをチェックします。

すべてのチェックを実行すると、見つかった問題は、Reports フォルダーに新しく作られたファイルにレポートされます。

メインワークフローは 3 つの引数を渡すこともできます。レポートの言語の 2 文字コード（ISO-639）とチェックすべきプロジェクトへのパスとレポートを出力するパスです。これにより、ツールを他のソリューションのコンポーネントとして、例えば、CI/CD パイプラインの一部として

実行することができるようになります。また、プロジェクトのバッチでチェックを実行することもできます。

結果の解釈

Main.xaml の引数で出力パスを指定しない場合は、Workflow Inspector のレポートは Reports フォルダーに作成されます。

レポートはテーブルの形式で、各行が見つかった問題、各カラム（下記）はその詳細です：

- ワークフローファイル（*Workflow Filename*）は、問題が見つかったワークフローのファイル名です。
- 内部パス（*Internal Path*）は、ワークフローのアクティビティの階層で、最初のアクティビティ（例：sequence, flowchart または state machine）から問題個所に到達する方法を示します。これにより、多くのアクティビティや階層のあるワークフローで問題個所を見つけやすくなります。
- 対象（*Target*）は、問題が見つかった要素です。通常、対象はアクティビティですが、変数や引数、またはワークフローになることもあります。
- 事象（*Issue*）は、見つかった問題の説明です。この説明は、設定のチェックリストから転記されます。
- 対応（*Action*）は、問題への対応方法です。*Fix* とマークされていれば、その問題は修正する必要があることを示します。*Double-check* は修正できない可能性があるが、確認する必要があることを示します。例えば、*SimulateClick* や *SendWindowMessages* の代わりにデフォルトのクリックを使用する場合、これを問題として報告しますが、すべてのコントロールが *SimulateClick* や *SendWindowMessages* をサポートしているわけではないため、オプションを使うことができないかもしれません。この場合、「ターゲットコントロールはデフォルトのクリックのみをサポートする」ことを説明するアノテーション（注釈）記載することを勧めます。問題に対する *Fix* と *Double-check* の指定は、設定のチェックリストで行います。
- メッセージ（*Suggestion*）は、問題の修正方法に関するアイディアを提供します。この内容も、設定のチェックリストで指定します。

設定

実行するチェックの選択は、チェックリストファイルで行います。チェックリストファイルは、Config フォルダーにあり、言語ごとのサブフォルダーに分けられています。
チェックリストはテーブルの形式で、各行がチェック、各カラム（下記）はその詳細です：

- 実行（*Run*）は、Workflow Inspector がチェックを行うかどうかを指定します。

- 事象（*Issue*）は、タイトルで、問題点の内容を明確に説明する必要があります。
- チェックファイル（*Check Filename*）は、チェックを実装するワークフロー名を指定します。 相対パス、絶対パスのいずれでも設定できます。
- 引数（*Arguments*）は、チェックに設定パラメータを渡すために使うことができます。JSON 文字列で指定する必要があり、Common¥ParseArguments.xaml ワークフローで解析後チェックに渡されます。
- 対応（*Action*）は、問題にどう対処すべきかを指定します。ベストプラクティスの明らかな違反がある場合、修正（*Fix*）とし、規則が適用されない可能性がある場合、確認（*Double-check*）とします。
- 説明（*Explanation*）は、問題点の説明で、なぜ問題であるのかを明確にします。
- メッセージ（*Suggestion*）は、問題点の修正方法を推奨します。

カラムには、チェックリストと生成されるレポートで共通のものがあるので注意してください：事象（*Issue*）、対応（*Action*）、説明（*Explanation*）およびメッセージ（*Suggestion*）です。これは、ユーザーがチェックリストを見なくても、レポートを理解しやすくするためです。

さらに、チェックリストは、project-level チェックと workflow-level チェックのタブに分かれています。各タブには同じ列がありますが、Workflow Inspector での使われ方が違います：project-level チェックは、検査すべきプロジェクトのパスを受け取り、workflow-level チェックは、さらに、検査すべきワークフローの内容を受け取ります。

生成されたレポートのレイアウトを変更することもできます。レポートテンプレート（ReportTemplate.xlsx）は、チェックリストと同じフォルダーにあり、必要に応じてフォーマットを変更できます。ただし、新しい列の追加、削除、および列の順序の変更を行うには、Workflow Inspector の修正が必要になります。

アーキティクチャ

Workflow Inspector は、入力としてプロジェクトパスを受け入れ、そのプロジェクトで見つかった問題点を含むレポートファイルを生成します。実行は、Figure 1 のとおりで、以下のように 3 つの手順に分割することができます：

- Initialize：システムのロケール設定基づいてチェックリストを読み込み、issues テーブルのような、必要な変数の初期化を行います。
- Check：選択されたチェックをプロジェクト（project-level checks）やワークフロー（workflow-level checks）に対して実行し、見つかった問題を issues テーブルに追加します。

- Report: 見つかった問題点をテンプレートに基づいてレポートを生成します。

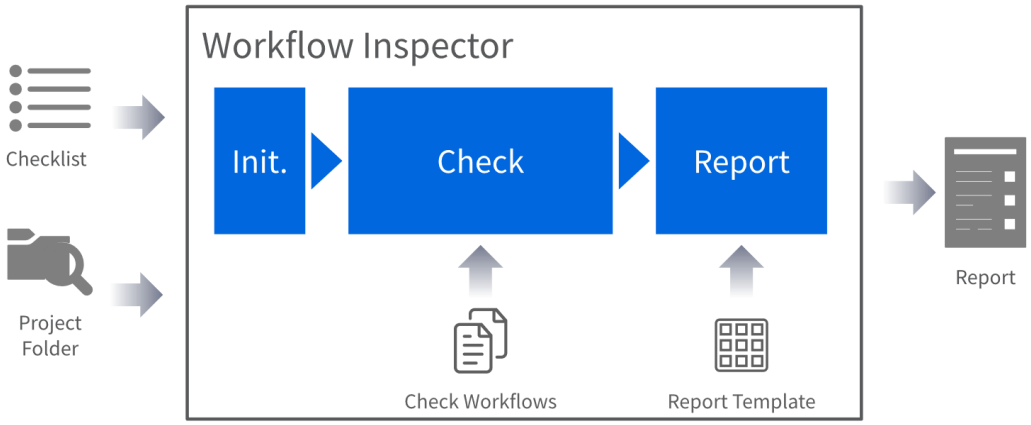

Figure 1 - Architecture of Workflow Inspector

Table 1 はこのステップを実装したファイルの情報をまとめたものです。Core フォルダーのファイルは Workflow Inspector の基本操作を実装し、Common フォルダーのファイルはツールで使われるパーツを実装しています。

チェック自体は、Checks フォルダーにあり、広く受け入れられているベストプラクティスを表す Standard checks と、顧客、パートナーまたは他のコミュニティーメンバーによって作成された Custom checks に分かれています。Custom checks は、Standard check よりも一般的ではないかもしれません。また、Checks フォルダーには、Template サブフォルダーも含まれています。このサブフォルダーには、新しいチェックを作成する際に使用されるテンプレートが含まれています。

Table 1 - Main files of the Workflow Inspector

Workflow Filepath	Purpose
Core¥Initialization.xaml	issues テーブルを初期化し、（システムのカルチャ設定に基づいて選択された）チェックリストからデータを読み取ります。
Core¥ProjectChecks.xaml	project-level チェックを実行し、見つかった問題を issues テーブルに追加します。
Core¥WorkflowChecks.xaml	workflow-level チェックを実行し、見つかった問題を issues テーブルに追加します。
Core¥Reporting.xaml	テンプレートに基づいて issues テーブルをレポートに出力します。
Common¥GetInternalPath.xaml	ワークフローのルートアクティビティから、パラメータとして渡されたアクティビティまでの内部パスを取得します。この内部パスは、多くのアクティビティがあるワークフローの問題個所を特定するのに役立ちます。
Common¥IsVariableUsedInStringValue.xaml	指定された変数名が文字列の中で使われているかをチェックします。
Common¥ParseArguments.xaml	簡単に扱えるよう、チェックリストの Argument カラムで指定された JSON

	文字列を辞書型に変換します。
Common¥ParseXAML.xaml	XAML ファイルを文字列として解析し、XML ツリーを作成し、関連する namespace manager とともに返却します。

UiPath Studio で作成されたワークフローは、XML（Extensible Markup Language）に基づく、XAML（Extensible Application Markup Language）として保存されます。このため、ファイルの中に特定のパターンが見つかるかどうかを確認するために XML ファイル固有の構造を生かすことが可能です。この場合、パターンは、ワークフロー開発のベストプラクティスを表します。

XAML ファイルの解析とネームスペースの管理は、Common¥ParseXAML.xaml に実装され、入力プロジェクトフォルダー内で見つかったすべてのワークフローファイルに対して実行されます。その後、XML ツリーおよび namespace manager は、プロジェクトへのパス、対象となっているワークフローへのパス、およびチェックリストのチェックのために指定された Arguments の辞書などの他の引数と共に、各ワークフローファイルに渡されます。

ワークフローファイル内の特定の XML ノードを検索するために、Workflow Inspector は、XML と XAML のドキュメントのクエリーを行う標準的な方法である XPath を使用します。アクティビティを構成する多くのエレメントは、XAML ファイルのデフォルトネームスペースに含まれており、namespace manager は、prefix xaml（例：xaml:If や xaml:Catch）を介してこれらの要素へのアクセスを提供することに注意してください。他のアクティビティは異なるネームスペース（例：ui:SendHotkey）で定義される可能性があるため、エレメントがデフォルトの名前空間以外のネームスペースにあるかどうかを確認するためにワークフローのテキスト表現をチェックしてください。

メンテナンスを簡単にするために、各チェックは分離したワークフローとして実装されますが、それらは通常次のような似た構造を持っています。チェックの対象となっているワークフローの問題点を保持するテーブルの初期化、チェックで使用される XPath 式の定義、および XPath 式によって読み出されるノードを通じて繰り返し実行する *For Each* アクティビティです。*For Each* アクティビティの body には、チェックのロジックがあり、*If* アクティビティは、問題点が issues テーブルに追加すべき決めるために使用されます。

新しい問題点を追加する場合、ワークフロー内の位置を示す内部パスを指定する必要があります。つまり、内部パスは、ワークフローのアクティビティ階層を考慮して、問題点の場所を見つける方法を提供します。
内部パスの抽出と書式設定は、Common¥GetInternalPath.xaml で行われます。

新しいチェックの作成方法

ほとんどの場合、Workflow Inspector のコアコンポーネントを変更する必要はありませんが、ツールを拡張し、RPA 実装プロジェクトの特定のニーズに基づいてツールを拡張や、新しいチェック作成する必要があるかもしれません。

これを踏まえ、新しいワークフローのチェックを作成する手順は以下のとおりです：

1. チェックの目的、関連するベストプラクティスの背景の説明、および推奨する問題の解決方法を明確にします。XML ツリー構造から目的のノードを読み出す XPath 式やチェックを実行する手順のリストを作成することも役に立ちます。
2. テストに使用するサンプルワークフローを準備します。テスト駆動型アプローチを採用することにより、チェックに含まれる可能性があるエッジケースの設計がより簡単になります。
3. Checks¥Templates フォルダーにある WorkflowCheckTemplate.xaml のコピーを作ります。このファイルは新しいチェックの実装に使用するため、チェックの内容に合わせて名前を変更します。どんなチェックを行うかがわかるよう名前を付けてください。（例：MissingScreenshot.xaml や VariableNamingConvention.xaml）
4. 新しいチェックを実装するワークフローは次のとおりです：
    a. メインのシーケンスにチェックの名前と説明、およびチェックの実装手順を記載します。これは作成するチェックの文書化に役立ち、理解しやすいものになります。
    b. チェックが XML ツリー構造から正確なノード情報を読み出す必要がある場合、*Assign* アクティビティを使用して XPath 式を指定します。
    c. *For Each* アクティビティの body に、チェックを実行するためのすべてのロジックを含めます。ロジックが複雑な場合は、ロジックをサブコンポーネントに分割し、*Invoke Workflow File* アクティビティを使用して呼び出すことを勧めます。チェックがチェックリストファイル（Checklist.xlsx）内で定義された引数（Arguments）を持つ場合、これらの引数は、ワークフローの in_CheckArguments input argument を通じて利用できます。
    d. “Get activity's internal path”というタイトルの *Invoke Workflow File* アクティビティは、アクティビティへのパスを読み出すために使用されます。現在

のノードを入力として受け取り、そのノードの内部パスを示す文字列を返します。この情報は、報告された問題点を見つけやすくするために、レポートに含まれます。

e. 任意で、*If* アクティビティを使用してチェックに関連する条件をチェックします。

f. “If activity has DisplayName”というタイトルの *If* アクティビティは、UiPath Studio に表示されるアクティビティの名前を返します。例えば、”Click”ではなく、”Click Login Button”というようにです。ただし、一部のアクティビティには DisplayName 属性がないため、代わりにノードのローカル名が使われる場合があります。

g. “Add issue to issues table”というタイトルの *Add Data Row* アクティビティを使用して、見つかった問題点に関する情報を issues テーブルに追加します。

5. 新しいチェックをターゲット言語のチェックリスト（例：Config¥EN¥Checklist.xslx）に追加します。事象（Issue）、チェックファイル（Check Filename）、説明（Explanation）、およびメッセージ（Suggestion）を指定する必要があります。任意で引数（Argument）列を使用して、外部引数を JSON 文字列の形式でチェックに渡します。

6. 前もって準備しておいたサンプルワークフローを使用してチェックをテストし、意図した問題点がレポートファイルに追加されたかをチェックします。

チェックの作成例

UndocumentedDefaulClick.xam チェックを作成する手順を使ってより具体的な説明をします：

1. チェックの目的、関連するベストプラクティスの背景の説明、および推奨する問題の解決方法を明確にします。XML ツリー構造から目的のノードを読み出す XPath 式やチェックを実行する手順のリストを作成することも役に立ちます。

   - 目的：*Click* アクティビティが *SimulateClick* や *SendWindowMessages* の代わりに default click を使用しているかをチェックします。.
   - 説明：プロパティの *SimulateClick* や *SendWindowMessages* はマウスドライバに依存しないため、クリックを実行するための高速で堅牢な方法を提供します。そのため、これらのプロパティが有効である場合は利用するべきです。また、ターゲットコントロールがこれらのプロパティをサポートいない場合、

アノテーションを追加します。インプットメソッドの詳細については、*Click* アクティビティのドキュメントを参照してください。

- 推奨：ターゲットコントロールがサポートしていれば、*SimulateClick* や *SendWindowMessages* を使用します。
- 手順：
    i. 属性に *SimulateClick* を持つすべてのノードを取得します。.
    ii. 属性の *SimulateClick* と *SendWindowMessages* が False になっているか、アクティビティにアノテーションはあるかをチェックします。
- *SimulateClick* を持つノードを読み出す XPath 式：//*[@SimulateClick]。この式は *SimulateClick* 属性を持つすべてのノードを探し、*Click* アクティビティを返します。*Click* と *Double Click* アクティビティはどちらも *SimulateClick* 属性を持つため、対象となります。

2. テストに使用するサンプルワークフローを用意します（Figure 2）。
    - 新しいワークフローを作成します。
    - ４つの *Click* アクティビティを追加します： １つは *SimulateClick* を使用、１つは *SendWindowMessages* を使用、１つはアノテーション付き default click を使用、そして１つはアノテーションなしの default click を使用します。このテストケースでは、最後の *Click* アクティビティのみを問題としてレポートする必要があります。

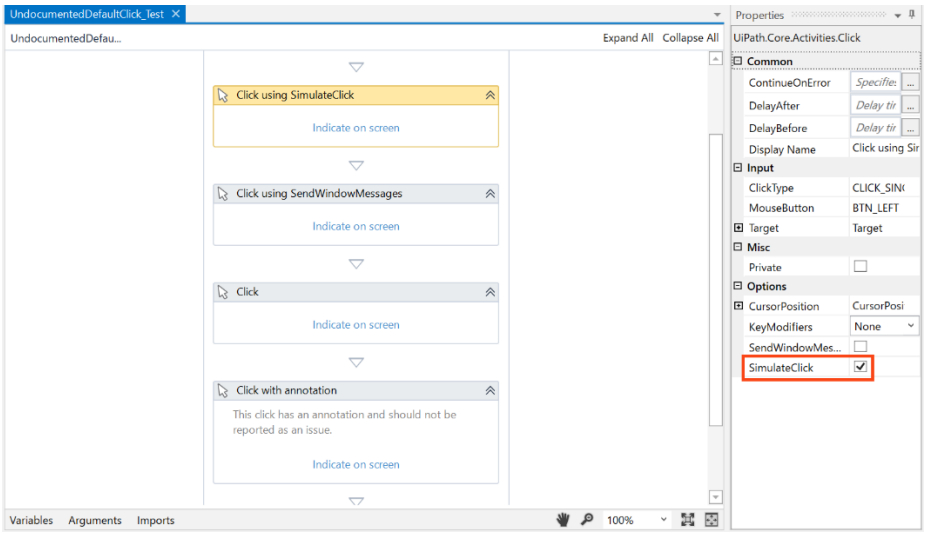

Figure 2 - Creation of Test Case

3. Checks¥Templates フォルダーの WorkflowCheckTemplate.xaml ファイルのコピーを作ります。このコピーは新しいチェックの実装に使うため名前を変更します：UndocumentedDefaultClick.xaml （Figure 3）.

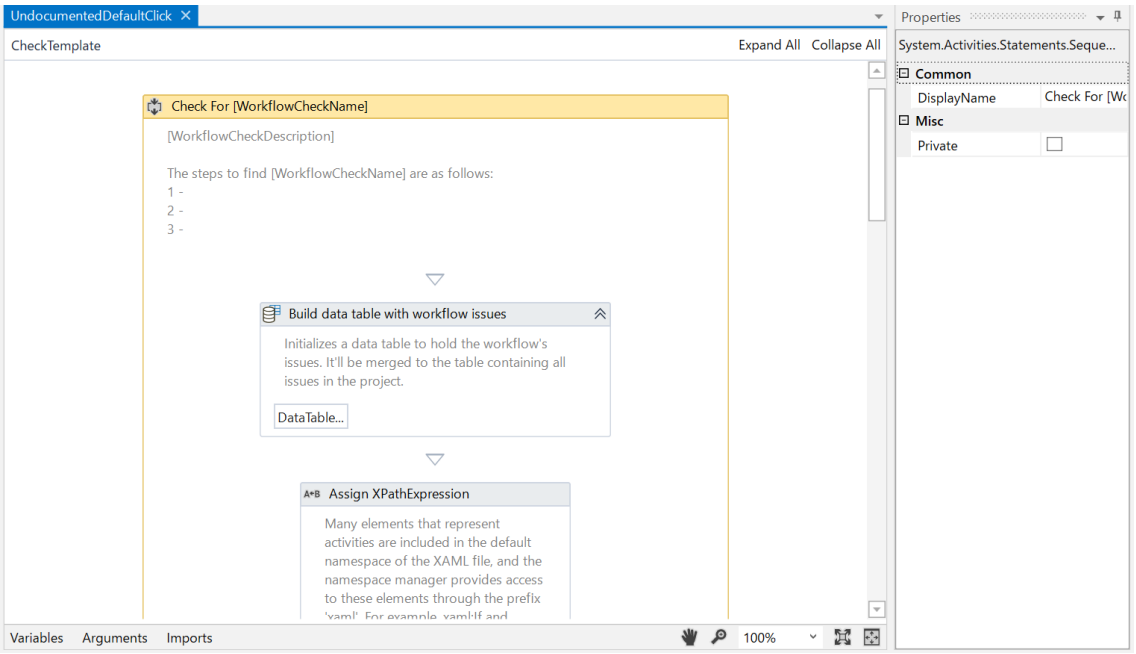

Figure 3 - Copied and Renamed Check Template

4. 新しいチェックを実装するワークフローは次のとおり：

    a. テンプレートのメイン sequence にチェックの名前と説明、およびチェックを実装する手順のリストを記述します（Figure 4）。

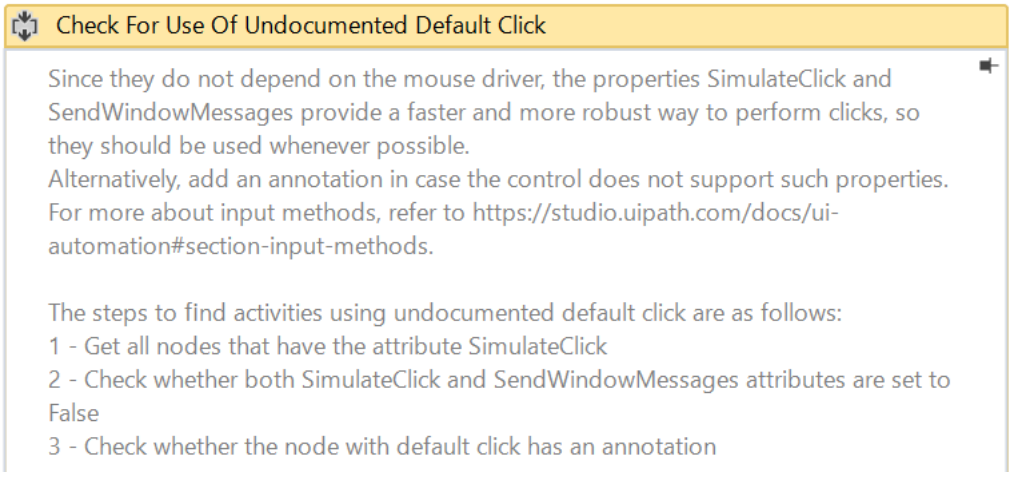

Figure 4 - Check's Name and Description

    b. チェックが XML ツリー構造から特定のノードを読み出す必要がある場合、*Assign* アクティビティを使用して XPath 式を指定します（Figure 5）。

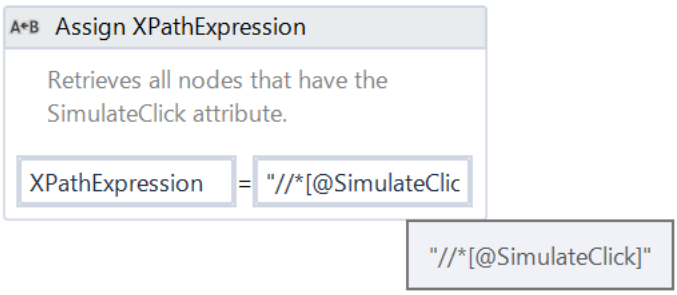

Figure 5 - XPath Expression Specification

c. *For Each* アクティビティの body に、チェックを実行するためのすべてのロジックを含めます（Figure 6）。このケースでは、チェックは *Click* アクティビティの XML ノードの *SimulateClick* と *SendWindowMessages* 属性が False かをチェックします。値 False は、これらのプロパティが使用されていないことを意味します。
さらに、ノードに *sap2010:Annotation.AnnotationText* 属性があるかもチェックします。この属性があれば、アクティビティにアノテーションがあることを意味します。

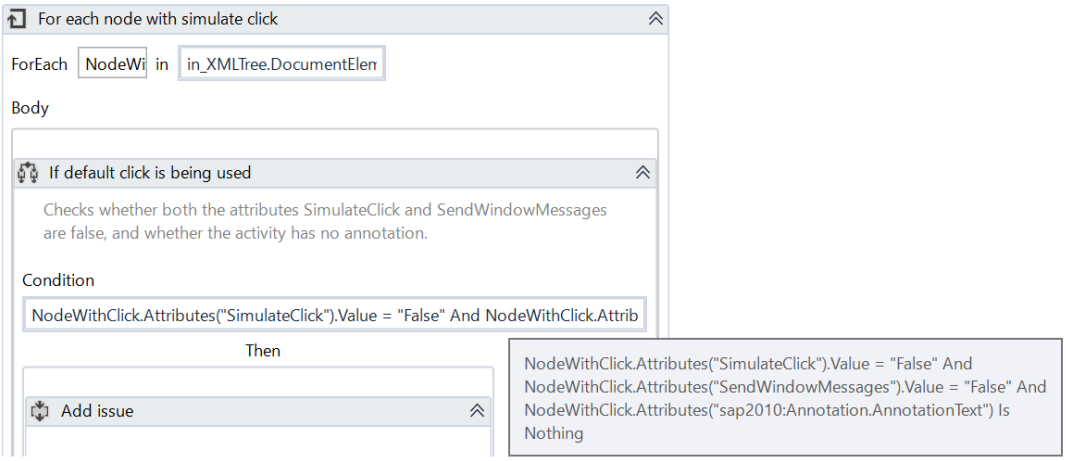

Figure 6 - Check's Logic

d. “Get activity's internal path”というタイトルの *Invoke Workflow File* アクティビティは、アクティビティへのパスを読み出すために使用されます。このチェックでは変更の必要はありません。

e. “If activity has DisplayName”というタイトルの *If* アクティビティは、UiPath Studio に表示されるアクティビティの名前を返します。このチェックでは変更の必要はありません。

f. “Add issue to issues table”というタイトルの *Add Data Row* アクティビティを使用して、見つかった問題点に関する情報を issues テーブルに追加します。このチェックでは変更の必要はありません。

5. 新しいチェックをターゲット言語のチェックリスト（例：Config¥EN¥Checklist.xslx）に追加します（Figure 7）。

Yes	Undocumented default click	Checks\Standard\UndocumentedDefaultClick.xaml		Double check	Since they do not depend on the mouse driver, the properties SimulateClick and SendWindowMessages provide a faster and more robust way to perform clicks, so they should be used whenever possible. Alternatively, add an annotation in case the control does not support such properties. For more about input methods, refer to https://studio.uipath.com/docs/ui-automation#section-input-methods	Use SimulateClick or SendWindowMessages if the target control supports it.

Figure 7 - New Row in チェックリスト

6. 最初に準備したサンプルワークフローを使用してチェックをテストし、意図した問題点がレポートファイルに追加されたかをチェックします。

この例によって示されるように、チェックを開発する際に最も大切なことは、XML ツリーから必要なノードを検索するための適切な XPath 式を作成することです。そうすれば、テンプレートは読み出したノードのイテレーション、内部パスの計算、issues テーブルへの追加といった実装を簡単にするための基盤を提供します。

配布

このツールは、UiPath Connect （https://connect.uipath.com/）から ZIP ファイルとしてダウンロードできます。ツールには次の主要コンポーネントが含まれています：

- Core and common workflows： initialization of configurations、XAML parsing procedures および output of results を含む、ツールのバックボーンを提供するワークフロー。
- Check workflows （standard and custom）： 個々のチェックを実装し、主要な実行フローから呼び出されるワークフロー。標準チェックは、あらゆる RPA 実装に適用できると考えられる規則を指しており、カスタムチェックはあまり一般的ではないものの依然として有益な規則を網羅します。
- Configuration files： 実行するチェックとレポートテンプレートのリスト。これらのファイルは、ローカライズして各地域のニーズに適応させることができます。

サポートポリシー

Workflow Inspector は、関心のあるあらゆるカスタマーまたはパートナーに配布いただけます。配布条件は the UiPath Open Platform Activity License Agreement （ https://www.uipath.com/hubfs/legalspot/UiPath_Activity_License_Agreement.pdf）に基づいており、サポートはベストエフォートベースで提供されます。

拡張に関しては、カスタマーおよびパートナーの方々がご自身で拡張の手順を理解した上で実装することを強くお勧めします。追加のチェックを作成する方法の詳細については、本書の「新しいチェックの作成方法」セクションを参照してください。