Workflow Inspector

Contents

Overview

Guaranteeing the quality of workflows in an RPA implementation project can directly impact its success. Although experienced developers and practices like peer reviewing can improve the quality of workflows created, having an automated way to inspect many XAML files can be helpful to do a preliminary automatic evaluation.

This creates the need for a tool that can aid RPA developers and Solution Architects in the process of workflow review, checking whether most commonly used development best practices are being followed.

In addition, since development conventions vary with each implementation project, it is essential that users of the this tool can easily understand it, and that the tool itself can be more easily extended to meet the necessities and standards of each implementation.

The tool described in this document, Workflow Inspector, addresses the mentioned needs by having the following characteristics:

- It is offered as a regular UiPath Studio project, making it easy for users to understand and make modifications without having to learn a different programming language or tool.
- It uses an Excel file to specify checks to be done and their configurations. This file also contains explanations about checks and suggestions for improvements, contributing to the education of developers in an RPA implementation.
- It traverses the structure of XAML files by using XPath expressions, which is a standard to query XML and XAML documents. This provides the necessary flexibility to adapt the tool to specific needs of an implementation.

How to Use

The most straightforward way to use the Workflow Inspector is to open it on UiPath Studio and run the Main workflow file. The user will then be asked to specify a UiPath Studio project folder, and the tool will proceed to verify whether the XAML files in that folder pass the selected best practices checks.

After all checks are performed, any problems found are reported in a newly generated file in the Reports folder.

Alternatively, the Main workflow also accepts two arguments: the path to the project to be checked and the path to where the new report should be output. This enables the tool to be included as a component of other solutions - for example, as part of a CI/CD pipeline. It also makes it possible to perform checks in batches of projects.

Interpreting Results

Unless an output path was specified as argument of Main.xaml, reports generated by the Workflow Inspector can be found in the folder Reports.

Each report is a table whose rows represent problems found in the project and whose columns give more details about such problems:

- *Workflow Filename* indicates in what workflow the problem was found.
- *Internal Path* considers the hierarchy of activities in a workflow and specifies how to arrive to the problem by starting from the first activity (i.e., sequence, flowchart or state machine). This makes it easier to find the problem, especially in workflows containing many activities and levels.
- *Target* represents the element in which a problem was found. Targets usually are activities, but can also be variables, arguments and individual workflows.
- *Issue* describes the problem found. This description is retrieved from the configuration checklist.
- *Action* refers to what should be done regarding this problem. If a problem is marked as *Fix*, it means that the problem should be fixed. On the other hand, *Double-check* means that it might not be possible to be fixed, but this should be confirmed. For example, using a default click instead of *SimulateClick* or *SendWindowMessages* can be reported as a problem, but since not all controls support *SimulateClick* or *SendWindowMessages*, it might not be possible to use those options at all times. In this case, it is recommended to include an annotation explaining that the target control only supports default clicks. The specification of *Fix* and *Double-check* for a problem is done in the configuration checklist.
- *Suggestion* gives an idea of how to fix the problem. The suggestion's content is also defined in the configuration checklist.

Configuration

The selection of checks to be performed can be done by using the checklist files, which are in the Config folder and divided into folders corresponding to different languages.

A checklist is a table whose rows represent checks and whose columns give details about each check:

- *Run* indicates whether a check should be used by the Workflow Inspector during its execution.
- *Issue* refers to the issue's title and should give a clear description of what it is.
- *Check Filename* determines the name of the workflow file that implements a given check. This column accepts both relative and absolute paths.
- *Arguments* can be used to pass configuration parameters to a specific check. The values of this column must be defined as JSON strings, which are then parsed by the Common\ParseArguments.xaml workflow and passed to individual checks.
- *Action* specifies what should be done regarding the problem. If it is a clear violation of a best practice, it should be fixed. If there can be cases in which the rule does not apply, then it should be double-checked.
- *Explanation* describes the issue and clarifies why it is a problem.
- *Suggestion* recommends how to fix the problem.

Note that some columns are common between the checklist and the generated report: *Issue*, *Action*, *Explanation* and *Suggestion*. This is done to make it easier for users to understand the report without having to consult the checklist frequently.

In addition, each checklist provides a separation between project-level checks and workflow-level checks. Although they have the same columns, they are used differently in the implementation of the Workflow Inspector: project-level checks receive the path to the project to be examined, and workflow-level checks additionally receive the contents of each workflow to be inspected.

It is also possible to change the layout of the generated reports. The report templates (ReportTemplate.xslx) are found in the same folders as the checklists and can be formatted according to specific needs. However, adding or removing new columns, as well as changing the order of columns, require modifications to workflows that implement the Workflow Inspector.

Architecture

The Workflow Inspector accepts a project path as input and generates a report file with issues found in said project. The execution can be divided in three main steps, as shown in Figure 1 and described as follows:

- Initialize: Based on the system's locale settings, chooses and reads the appropriate checklist and initializes variables that will be used throughout the execution, like the table of issues.
- Check: Runs the selected checks on the project (project-level checks) and on individual workflow files (workflow-level checks), adding issues found to the overall table of issues.
- Report: Outputs the issues find to a report based on a template.

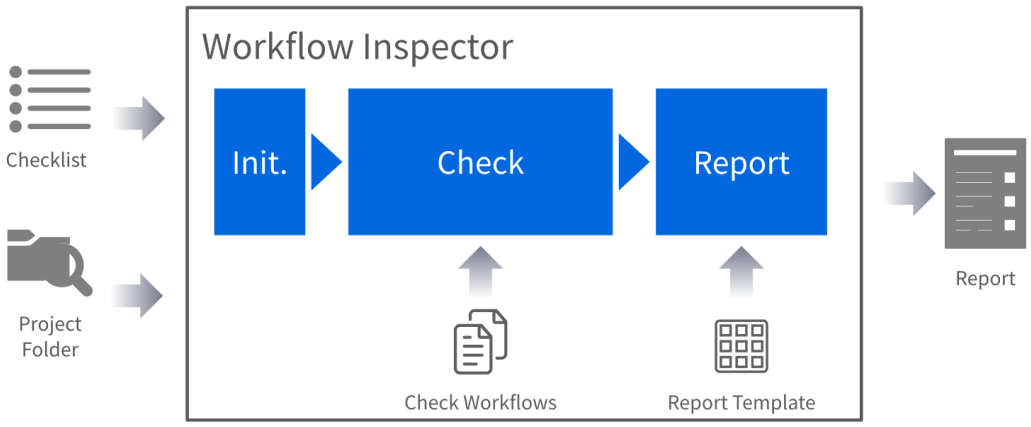

Figure 1 - Architecture of Workflow Inspector

Table 1 summarizes information about the files that implement such steps. Files in the *Core* folder perform basic operations of the Workflow Inspector, and the ones in *Common* implement procedures used in different parts of the tool.

The checks themselves are located in the Checks folder and divided in Standard checks, representing universally accepted best practices, and Custom checks, which were created by customer, partners or other community members and might be less generic than the Standard ones. The Checks folder also contains the Template subfolder, which contains templates to be used when creating new checks.

Table 1 - Main files of the Workflow Inspector

Workflow Filepath	Purpose
Core\Initialization.xaml	Initializes the issues table and reads data about checks from checklist (chosen according to the system's culture settings).
Core\ProjectChecks.xaml	Performs project-level checks and adds eventual issues to the issues table.
Core\WorkflowChecks.xaml	Performs the workflow-level checks and adds eventual issues to the issues table.
Core\Reporting.xaml	Outputs the issues table to a report based on a template.
Common\GetInternalPath.xaml	Retrieves the path from the root activity in the workflow to the activity that is passed as parameter. This is useful to localize the issue in workflows that have many activities.
Common\IsVariableUsedInStringValue.xaml	Verifies whether the given variable name is used in a string.
Common\ParseArguments.xaml	Converts the JSON string provided in the Argument column of the checklist file to a dictionary for easier manipulation.
Common\ParseXAML.xaml	Parses a XAML file's string representation into a tree, returning the tree and the associated namespace manager.

Workflows created in UiPath Studio are saved as XAML (Extensible Application Markup Language) files, which are based on XML (Extensible Markup Language). Because of this, it is possible take advantage of the inherent structure of XML files to check whether certain patterns are found in the file. In this case, such patterns represent best practices for workflow development.

The parsing of XAML files and the management of namespaces is implemented in the Common\ParseXAML.xaml file and it is done for all workflow files found in the input project folder. After that, the XML tree and the namespace manager is passed to each workflow file, along with other arguments such as the path to the project, the path to the workflow being inspected and a dictionary with arguments that were specified for that check in the checklist.

To do a search for specific XML nodes in a workflow file, the Workflow Inspector uses XPath, which is a standard way to query XML and XAML documents. Note that many elements that represent activities are included in the default namespace of the XAML file, and the namespace manager provides access to these elements through the prefix *xaml* (for example, *xaml:If* and *xaml:Catch*). Since other activities might be defined in other namespaces (e.g., *ui:SendHotkey*), it is recommended to check the text representation of the workflow to confirm whether an element is in a namespace other than the default one.

Each check is implemented as a separated workflow for easier maintenance, but they usually have a similar structure: initialization of a table to hold issues of a workflow being inspected, definition of a XPath expression to be used in the check and a *For Each* activity that iterates through the nodes retrieved by the XPath expression. Inside the body of the *For Each* activity, the check's logic is included and an *If* activity is used to decide whether an issue should be added to the issues table.

When adding a new issue, it is necessary to specify the internal path that leads to that point in the workflow. In other words, the internal path provides a way to find the location of the issue, considering the hierarchy of activities in the workflow. The extraction and formatting of the internal path is done by the file Common\GetInternalPath.xaml.

How to Create a New Check

In most cases there is no need to change any of the files used to implement the core components of the Workflow Inspector, but it might be necessary to extend the tool and create new checks based on specific needs of an RPA implementation project.

Based on this structure, the general steps to create a new workflow check are as follows:

1. Specify the check's purpose, the explanation behind the related best practice and suggestions to solve the issue. It is also helpful to list steps to perform the check, including XPath expressions to retrieve the desired nodes in the XML tree structure.
2. Prepare a sample workflow to be used for testing. Adopting a test-driven approach makes it easier to plan for edge cases in that might be included in the check.
3. Make a copy of the file WorkflowCheckTemplate.xaml, located in the Checks\Templates folder. This copy is used to implement the new check, so change its name according to the check. It is recommended to choose a descriptive name that makes it easy to understand what is being checked (e.g., MissingScreenshot.xaml and VariableNamingConvention.xaml).
4. In the workflow corresponding to the new check:
    a. Include the check name and description in the main sequence, as well as the list of steps to implement the check. This helps document the check so that it is more easily understood.
    b. If the check needs to retrieve certain nodes from the XML tree structure, specify the XPath expression using the *Assign* activity.
    c. In the *For Each* activity's body, include all the logic to perform the check. If the logic is complex, it is recommended to divide it in subcomponents and use the *Invoke Workflow File* activity to call them. If the check has arguments defined in the checklist file (Checklist.xlsx), these arguments will be available through the *in_CheckArguments* input argument of the workflow.
    d. The existing *Invoke Workflow File* activity entitled "Get activity's internal path" is used to retrieve the path to the activity. It receives the

current node as input and returns a string that denotes the internal path to that node. This information is included in the report to make it easier to find the reported issue.

e. Optionally, use the existing *If* activity to verify conditions related to the check.

f. The following *If* activity, named "If activity has DisplayName", returns the name of the activity as it appears in UiPath Studio - for example, "Click Login Button", instead of just "Click". However, some activities do not have *DisplayName* attributes, so the node's local name might be used instead.

g. Use the existing *Add Data Row* activity entitled "Add issue to issues table" to include the information about the found issue to the aggregated issue table.

5. Add the new check to the checklist file corresponding to the target language - for example, Config\EN\Checklist.xslx. It is necessary to specify the issue name, the check filepath, the explanation and the suggestion. Optionally, use the Argument column to pass external arguments to the check in the form of a JSON string.
6. Test the check by using the previously prepared sample workflow and verify whether it added the intended issues to the report file.

Example of Check Creation

For a more concrete explanation of the steps to create a new check, consider the steps used to create the UndocumentedDefaulClick.xaml check:

1. Specify the check's purpose, the explanation behind the related best practice and suggestions to solve the issue. It is also helpful to list steps to perform the check, including XPath expressions to retrieve the desired nodes in the XML tree structure.
    - Purpose: Verify whether *Click* activities are using default click, instead of *SimulateClick* or *SendWindowMessages*.
    - Explanation: 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 target control does not support such properties. For more about input methods, refer to the *Click* activity's documentation.

- Suggestion: Use *SimulateClick* or *SendWindowMessages* if the target control supports it.
- Steps:
    i. Get all nodes that have the attribute *SimulateClick.*
    ii. Check whether the attributes *SimulateClick* and *SendWindowMessages* are set to False, and whether the activity has an annotation.
- XPath expression to retrieve nodes with *SimulateClick*: `//*[@SimulateClick]`. This expression looks for all nodes having the *SimulateClick* attribute and will return *Click* activities. It considers *Click* and *Double Click* activities, since both have the *SimulateClick* attribute.

2. Prepare a sample workflow to be used for testing (Figure 2).
    - Create a new workflow.
    - Include four *Click* activities: one using *SimulateClick*, one using *SendWindowMessages*, one using default click with annotation and one using default click without annotation. In this test case, only the last *Click* activity should be reported as an issue.

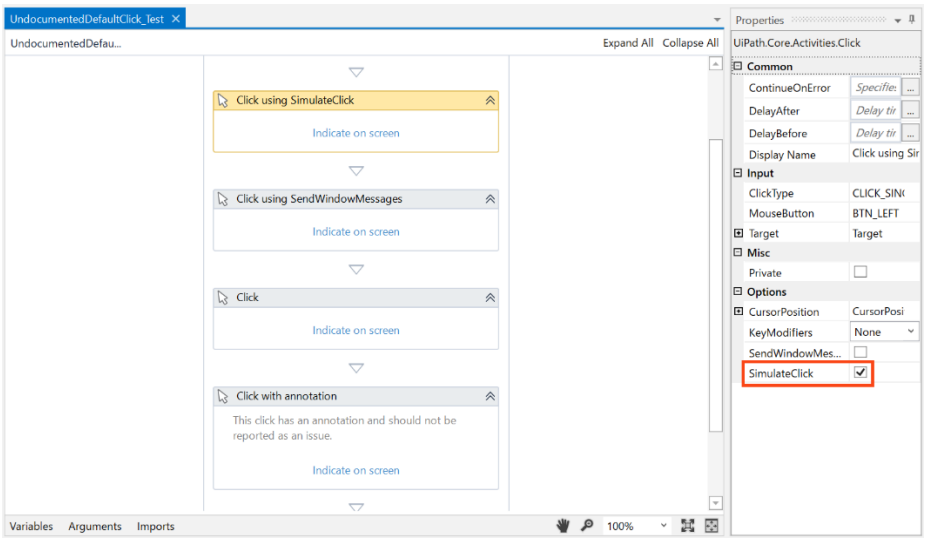

Figure 2 - Creation of Test Case

3. Make a copy of the file WorkflowCheckTemplate.xaml, located in the Checks\Templates folder. This copy is used to implement the new check, so change its name according to the check: UndocumentedDefaultClick.xaml (Figure 3).

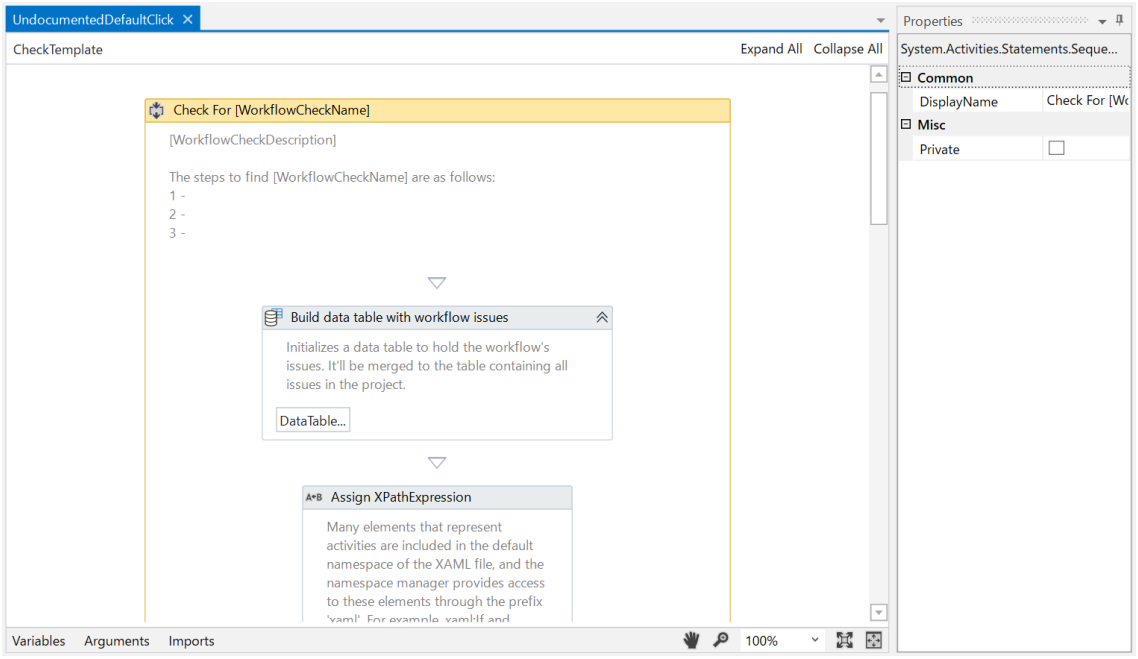

Figure 3 - Copied and Renamed Check Template

4. In the workflow corresponding to the new check:

a. Include the check name and description in the main sequence of the template, as well as the list of steps to implement the check (Figure 4).

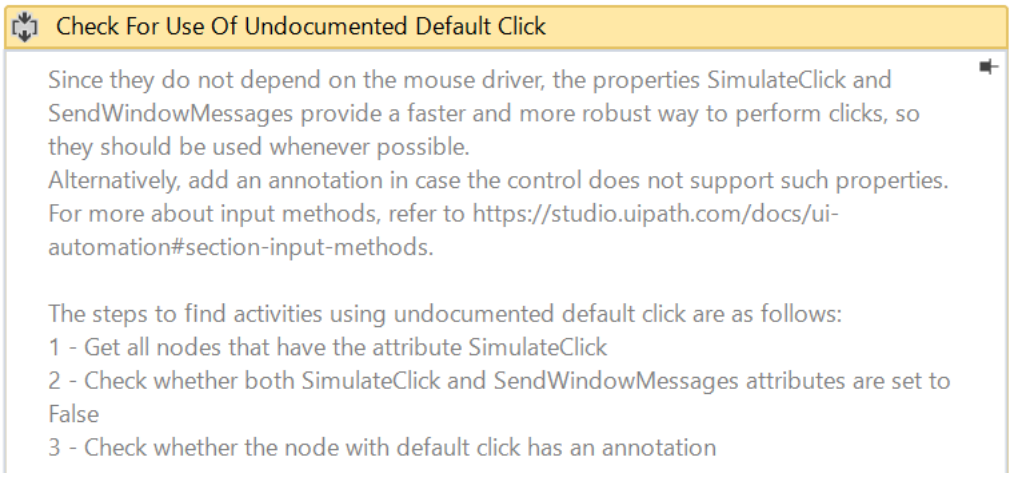

Figure 4 - Check's Name and Description

b. If the check needs to retrieve certain nodes from the XML tree structure, specify the XPath expression using the *Assign* activity (Figure 5).

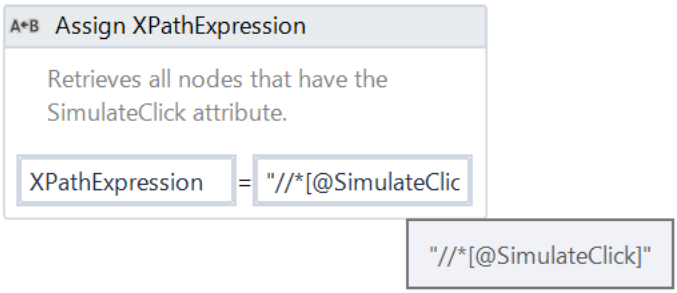

Figure 5 - XPath Expression Specification

c. In the *For Each* activity's body, include all the logic to perform the check (Figure 6). In this case, the check verifies whether the *SimulateClick* and the *SendWindowMessages* attributes of the XML node representing the *Click* activity have the value False, which indicate those properties are not being used. In addition, it also confirms whether the node has the attribute *sap2010:Annotation.AnnotationText*, which means that the activity has an annotation.

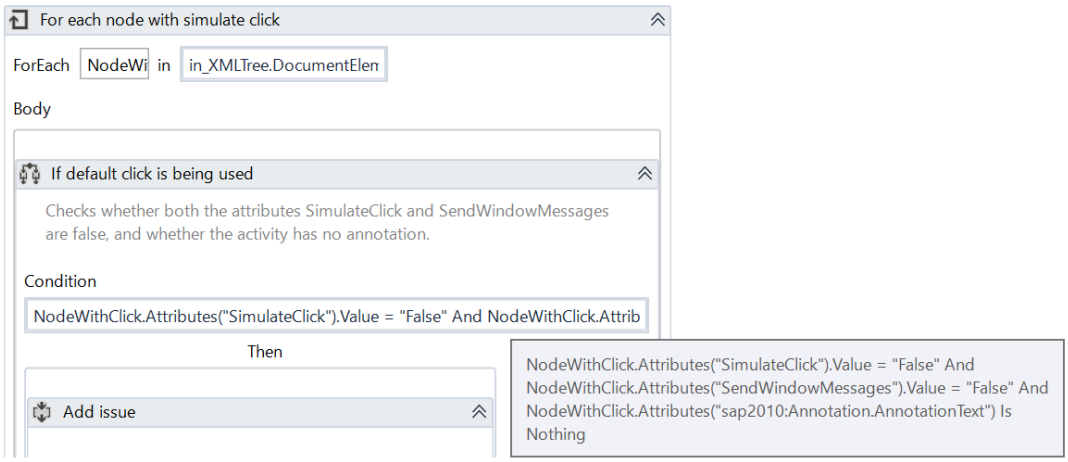

Figure 6 - Check's Logic

d. The existing *Invoke Workflow File* activity entitled "Get activity's internal path" is used to retrieve the path to the activity. There is no need to change it for this particular check.

e. The following *If* activity, named "If activity has DisplayName", returns the name of the activity as it appears in UiPath Studio. There is no need to change it for this particular check.

f. Use the existing *Add Data Row* activity entitled "Add issue to issues table" to include the information about the found issue to the aggregated issue table. There is no need to change it for this particular check.

5. Add the new check to the checklist file corresponding to the target language - for example, Config\EN\Checklist.xslx (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 Checklist

6. Test the check by using the initially prepared sample workflow and verify whether it added the intended issues to the report file.

As shown by this example, the most important part when developing a check is the creation of a proper XPath expression that will make it possible to search for the desired nodes in the XML tree. After that, the template offers a basis for easier implementation of other part: iteration through nodes retrieved, calculation of internal path and addition to the issues table.

Distribution

The tool can be downloaded as a ZIP file from UiPath Go! (https://go.uipath.com/). It contains the following main components:

- Core and common workflows: Workflows that provide the backbone of the tool, including initialization of configurations, XAML parsing procedures and output of results.
- Check workflows (standard and custom): Workflows that implement individual checks and that are called from the main execution flow. Standard checks refer to rules considered to be applicable to any RPA implementation, and custom checks cover less generic but still useful rules.
- Configuration files: List of checks to be done and report templates. These files can be localized to better adapt to regional needs.

Support Policy

Workflow Inspector is available for distribution to any customer or partner interested in it. It is offered under the UiPath Open Platform Activity License Agreement (https://www.uipath.com/hubfs/legalspot/UiPath_Activity_License_Agreement.pdf), thus support is provided on a best-effort basis.

Regarding extensions, it is highly encouraged that customers and partners understand the steps for extension and implement such modifications by themselves. For more about how to create additional checks, please refer to the section “How to Create a New Check” of this document.