Instructions:

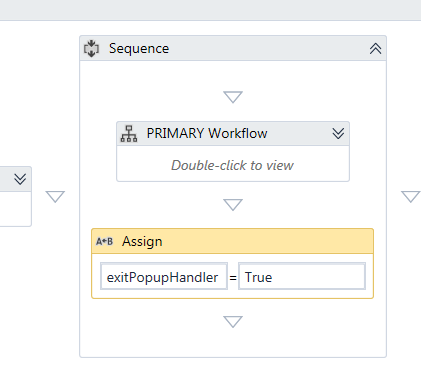
POP-UP HANDLER

Summary:

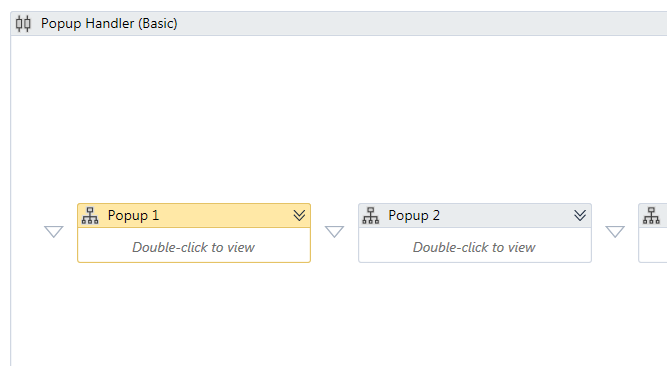
The Pop-Up Handler is a parallel framework for handling unwanted pop-ups during a user-assisted or fully-automated RPA process. It is designed to be simple and easy to use/modify for even the most entry-level RPA user.

Step-by-Step Instructions:

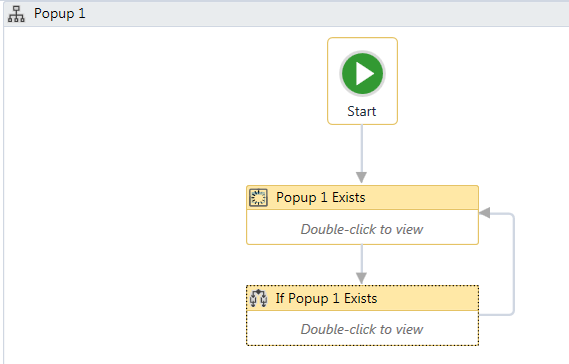
1. The pop-up handler consists of a single Parallel statement containing the user’s primary workflow alongside short, eternally-looping “catches” for specific, user-defined pop-ups.
2. The Parallel statement should contain a Boolean variable (e.g. *exitPopupHandler*) to determine when the pop-up handler should finish running. The “CompletionCondition” for the Parallel statement should be when *exitPopupHandler* = TRUE. This Boolean should be predefined as NOTHING or FALSE, and only assigned to TRUE as the very last step following the user’s primary workflow:



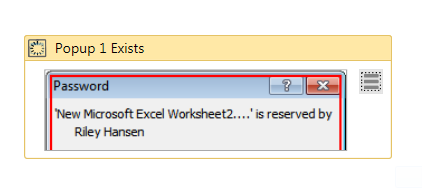
1. For each potential/anticipated pop-up, the user should program a separate “catch” flowchart:



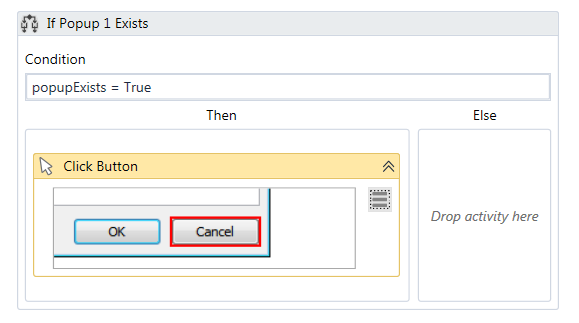
1. Each “catch” flowchart contains an infinite loop designed to catch one specific pop-up:



1. In order to set the pop-up to be handled by a specific “catch”, the user need simply…
   1. Double-click on the “Element Exists” activity and use the “Indicate on Screen” feature to select a defining UI element (body, text, etc.) of the pop-up to be handled:



* 1. Using the same method, the user should select which pop-up button the handler should press for the “Click” activity within the looped IF statement:



1. The user may program as many “catches” as necessary, by simply copying and pasting identical “catch” flowcharts and putting these in parallel with the others, then changing the selectors.

Alternative Applications and Alterations:

* The pop-up handler can be set to run indefinitely or as part of a user-assisted process. In such situations, an additional activity should be added within the parallel (e.g. a “Monitor” event waiting for a specific key combination, or a timed-delay) to determine when the process is complete. (An example of such a workflow is included in the documentation.)

Issues Addressed:

* Popups tend to pull focus away from whichever element is currently being interacted with. The pop-up handler cannot prevent this behavior. For this reason, the user should use activities and settings that restore object focus (e.g. “Activate”, “ClickBeforeTyping”), and/or use activities that do not require object focus (e.g. “SendWindowMessages”, “SimulateClick”) whenever possible during any portion of the primary workflow where pop-ups are anticipated.
* By default, the hander uses “SimulateClick” to close pop-ups in order to avoid interrupting the user’s primary workflow. Depending on the pop-up involved, “SimulateClick” may not work, and a different option (“SendWindowMessage”, regular click, or even sending a hotkey) may be necessary.

Performance Notes:

* Users can be as specific or not-specific as they please when choosing the selectors for the pop-ups to be handled. For instance, if a user wants to handle all Microsoft Excel pop-ups by pressing “Cancel”, it may be possible to set a very generic selector within the “Element Exists” activity that will identify all Excel pop-ups and handle them the same. On the other hand, it may be necessary to select part of the text within a pop-up as the selector element in order to differentiate it from other, similar pop-ups if different actions are required in each situation.
* Avoid using scopes (Excel scope, Browser scope, etc.) within the primary workflow inside your pop-up handler if pop-ups are expected within that scope. Scopes “block” access to outside activities (including the handler’s parallel “catches”), meaning that pop-ups within a scope cannot be accessed by the pop-up handler. If a scope is absolutely necessary, one way around this is to place the handler inside the scope rather than visa versa, although certain pop-ups during the initiation stages of the scope may still be unhandled.

Sample Use Cases:

* An automated workflow navigates Google Forms on the web. Pop-ups may occasionally appear asking if the user is sure they would like to navigate away from the current page. The pop-up handler “catches” these, allowing the process to continue.
* A user-assisted automation involves an application where it can be easy to accidentally delete important items. The user programs the handler to instantaneously “catch” and cancel any potential deletes to keep the user from accidentally pressing the wrong button.