Working with Parallel Branches

Derek Pascarella
Updated 1 minute ago

Unfollow

Applies To: Ayehu NG

Description

When designing workflows in **Ayehu NG**, many of the classic logic controls of traditional programming languages is available to the developer. These are referred to as "Common Controls" and can be explored further in the following support article:

https://support.ayehu.com/hc/en-us/articles/115003136329-Understanding-Common-Controls

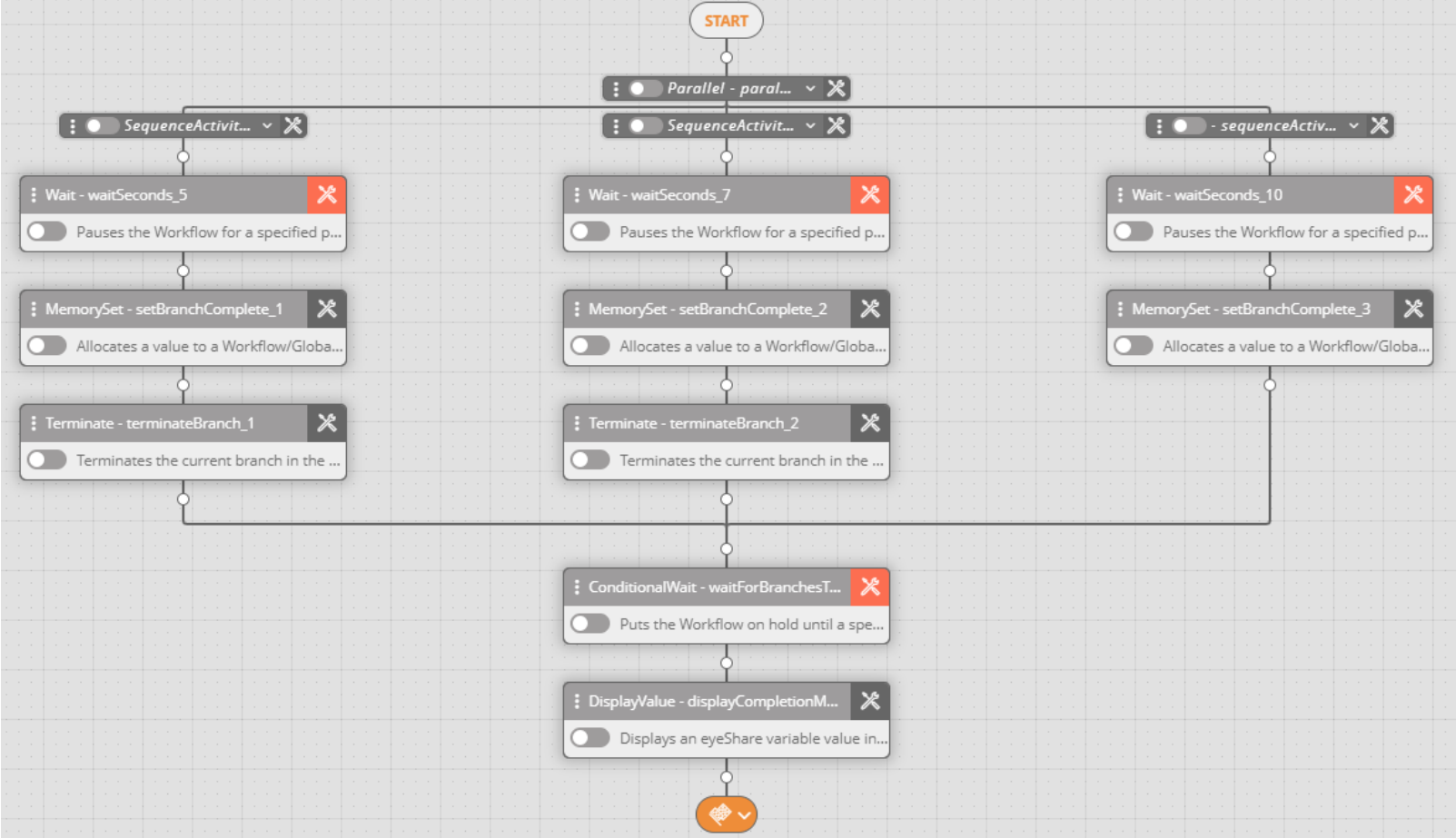
This article focuses on **Parallel Branches** and how they can be utilized to execute a series of activities simultaneously. The typical behavior of **Parallel Branches** is such that any and all activities that follow them will be executed separately after each **Parallel Branch** has completed its execution. In other words, the workflow diverges entirely at the point of the **Parallel Branches** and any activities that follow them are considered part of their own separate thread.

However, the goal of this tutorial is to implement a series of **Parallel Branches** that will result in only a singular stream of execution after their completion.

Please note that the workflow developed in this tutorial is also attached to this article (Conditional Wait with Parallel Branches (EXAMPLE).xml). This workflow is also available on our GitHub Community Repository (https://github.com/Ayehu/custom-workflows/blob/master/Parallel%20Branches/Conditional%20Wait%20with%20Parallel%20Branches%20(EXAMPLE).xml).

Workflow Overview

Consider the following workflow where there are three separate **Parallel Branches** that are all being executed simultaneously. Our goal is to wait for all three branches to complete their execution and then proceed with running the rest of the activities in the workflow that follow it.

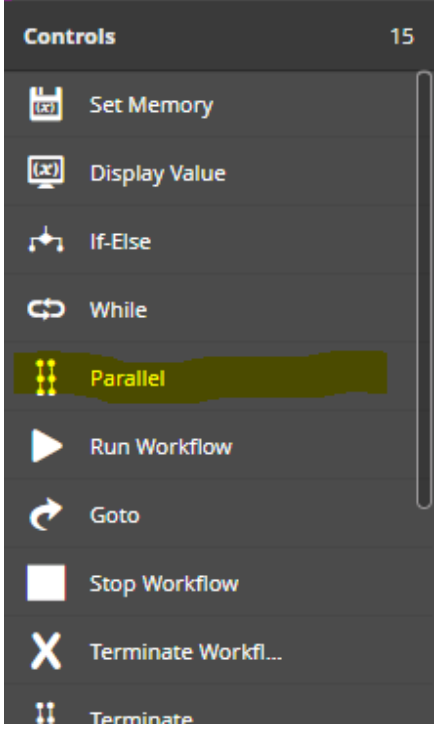


We can see that the activities involved in this example are **Parallel Branch**, **Wait**, **MemorySet**, **Terminate**, and **Conditional Wait**. The **DisplayValue** activity at the end serves the purpose of notifying us once all three branches have completed their separate but simultaneous execution.

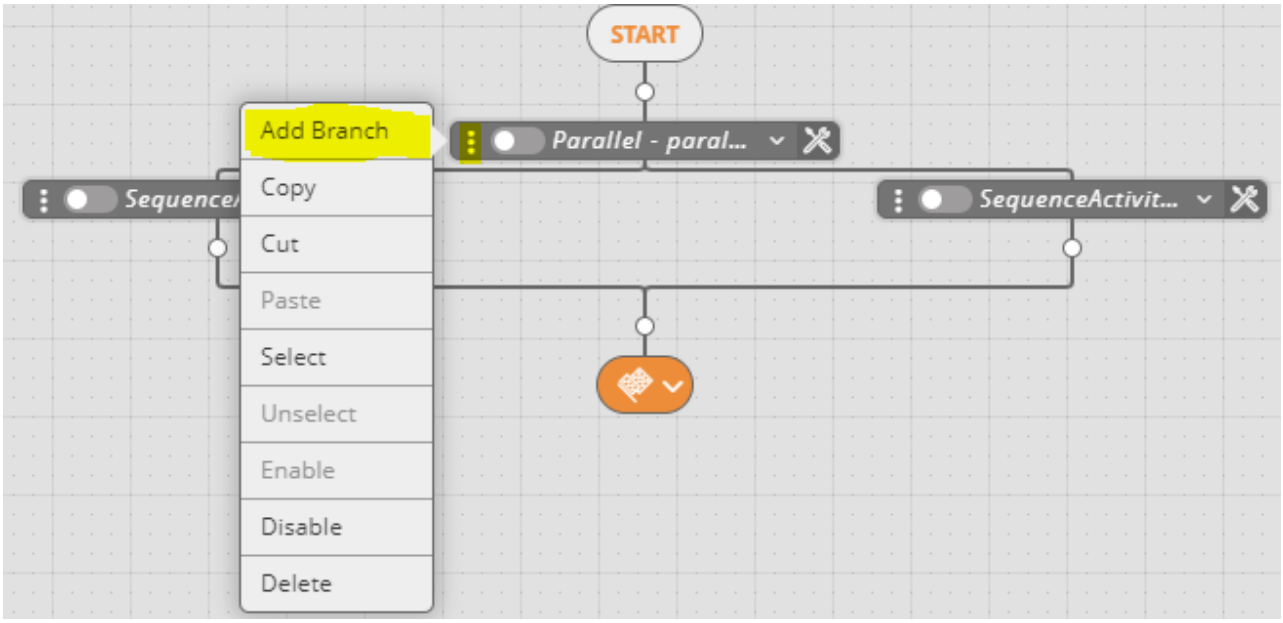
To learn how to leverage **Parallel Branches** for your own workflow development, follow along with this tutorial as we develop the above workflow from scratch.

Creating the Parallel Branches

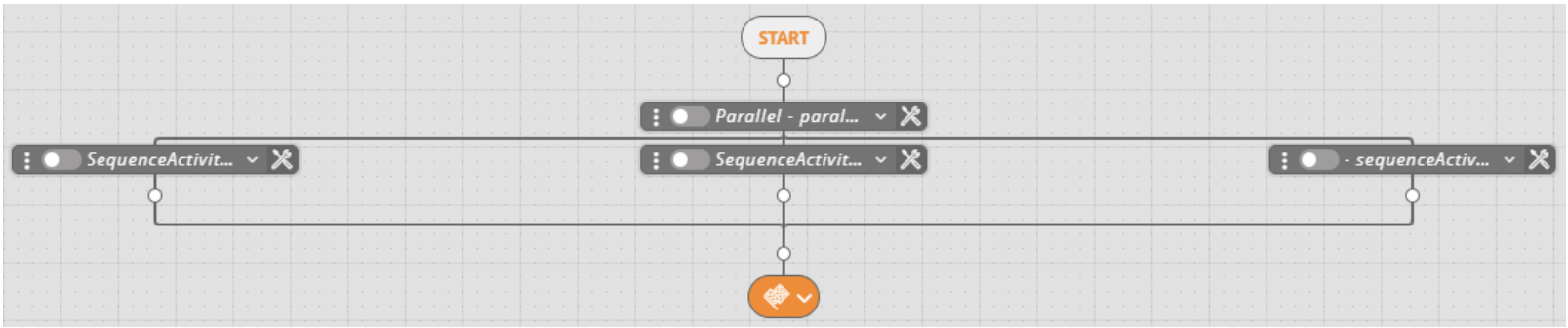
First, start by dragging the **Parallel Branch** activity to your workflow, which can be found at the bottom-right of the **Workflow Designer** as seen in the screenshot below.



Then, click the hamburger icon (the three dots on top of each other) to the left of the **Parallel Branch** activity to add a third branch.

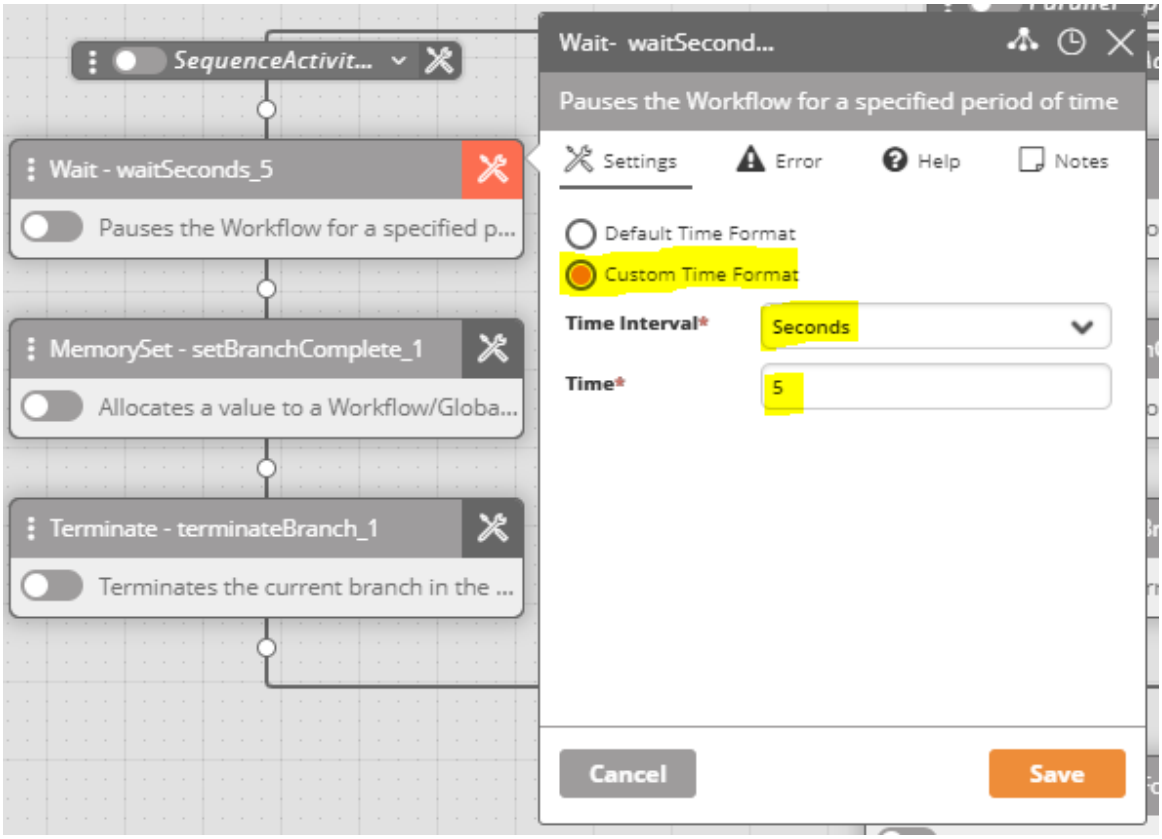


With all three empty **Parallel Branches** now ready, we will begin adding and configuring each activity.

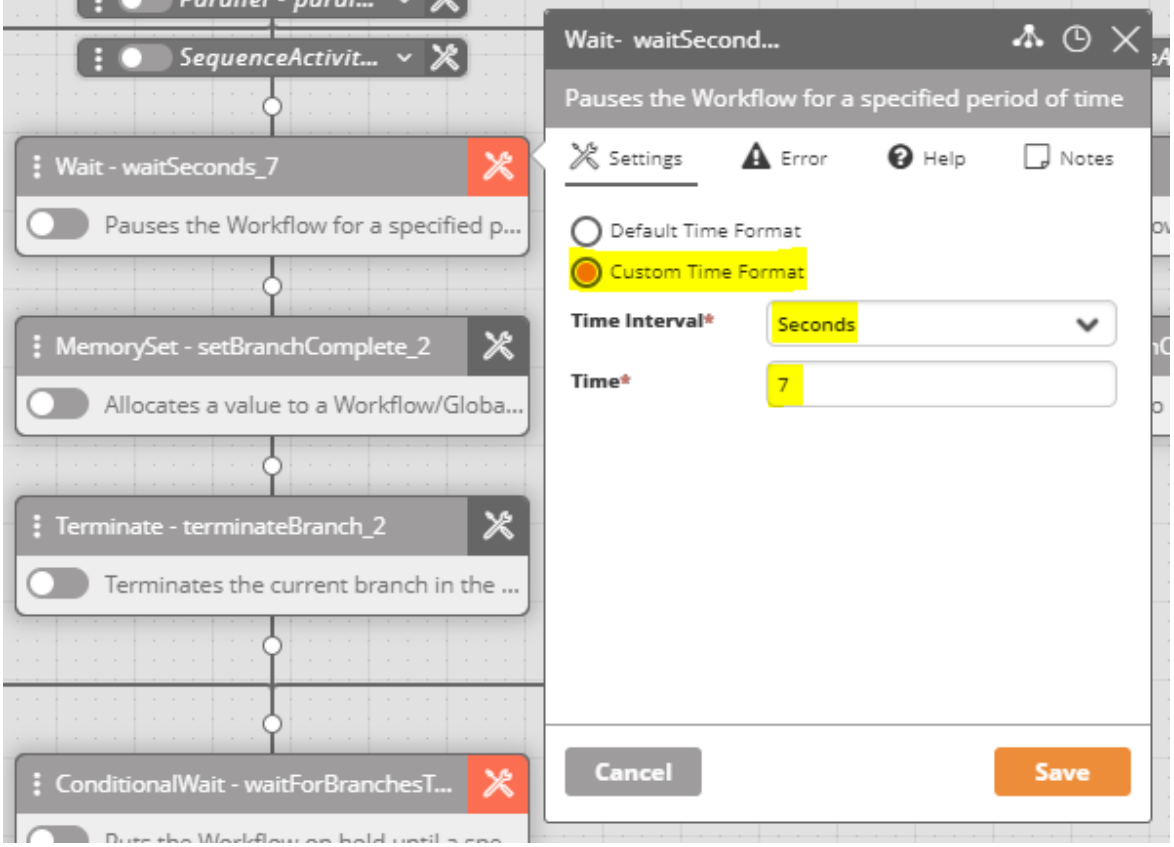


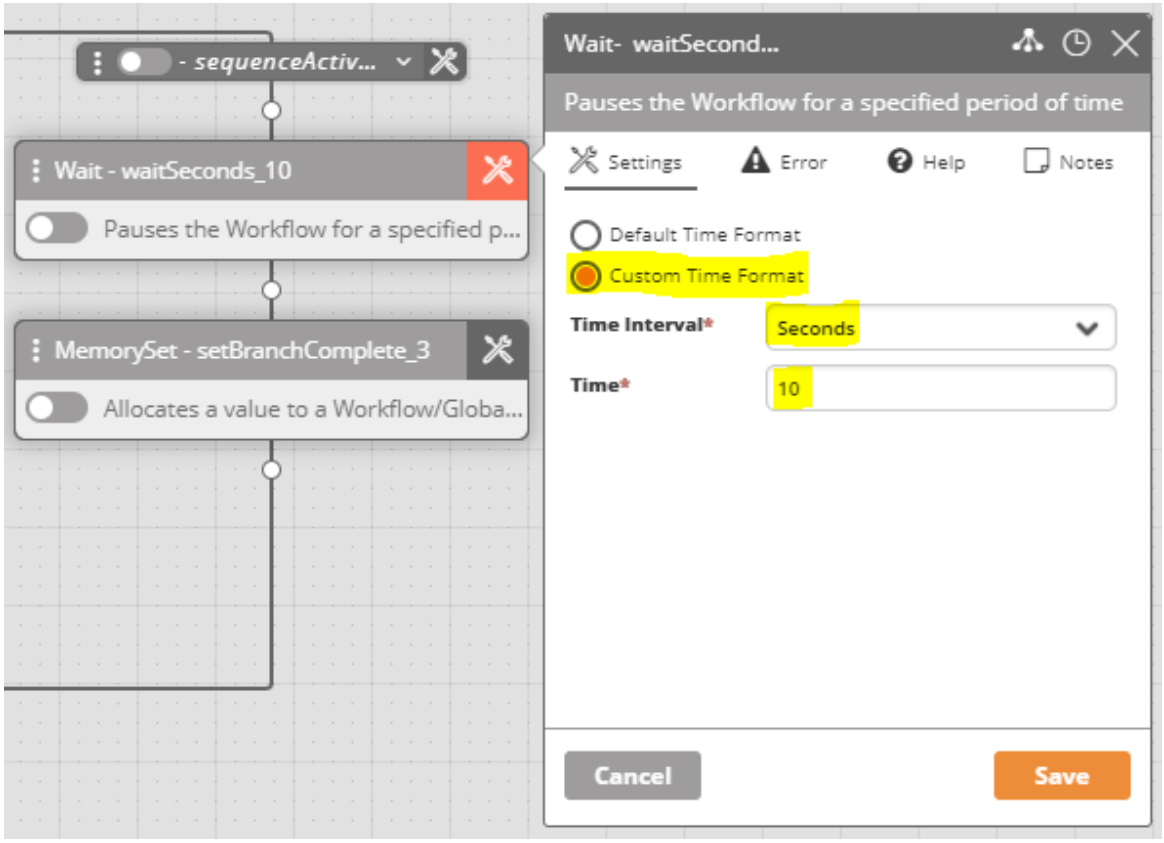
Configuring the Activities - Wait

In each branch, add a **Wait** activity. Select "Custom Time Format" and then "Seconds" for the "Time Interval" field. On the left branch, name the **Wait** activity **waitSeconds_5** and enter "5" in the "Time" field. See the screenshot below.



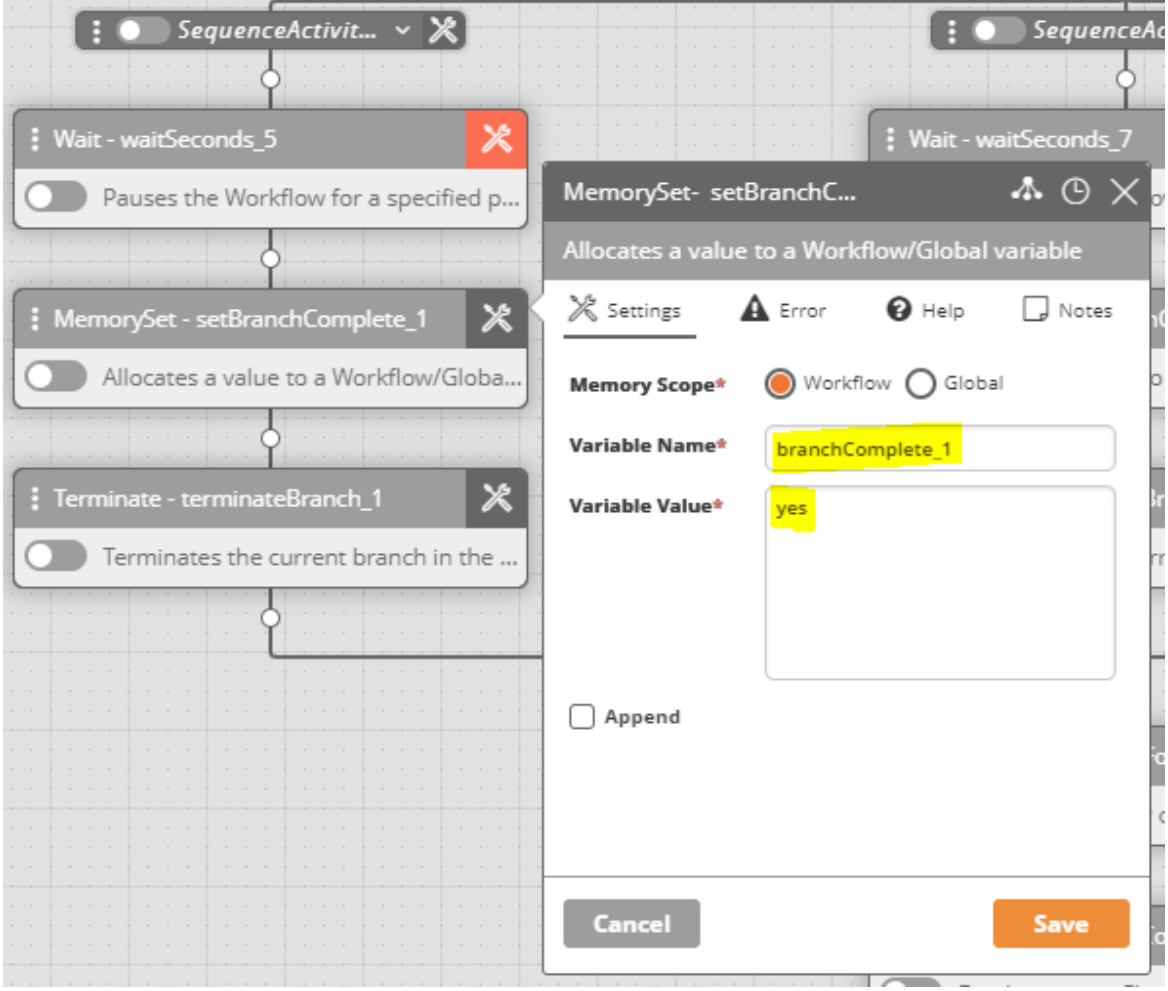
For the middle and right branches, use the **Wait** activity again. Name the middle branch's **waitSeconds_7** and the right branch's **waitSeconds_10**, as seen below.



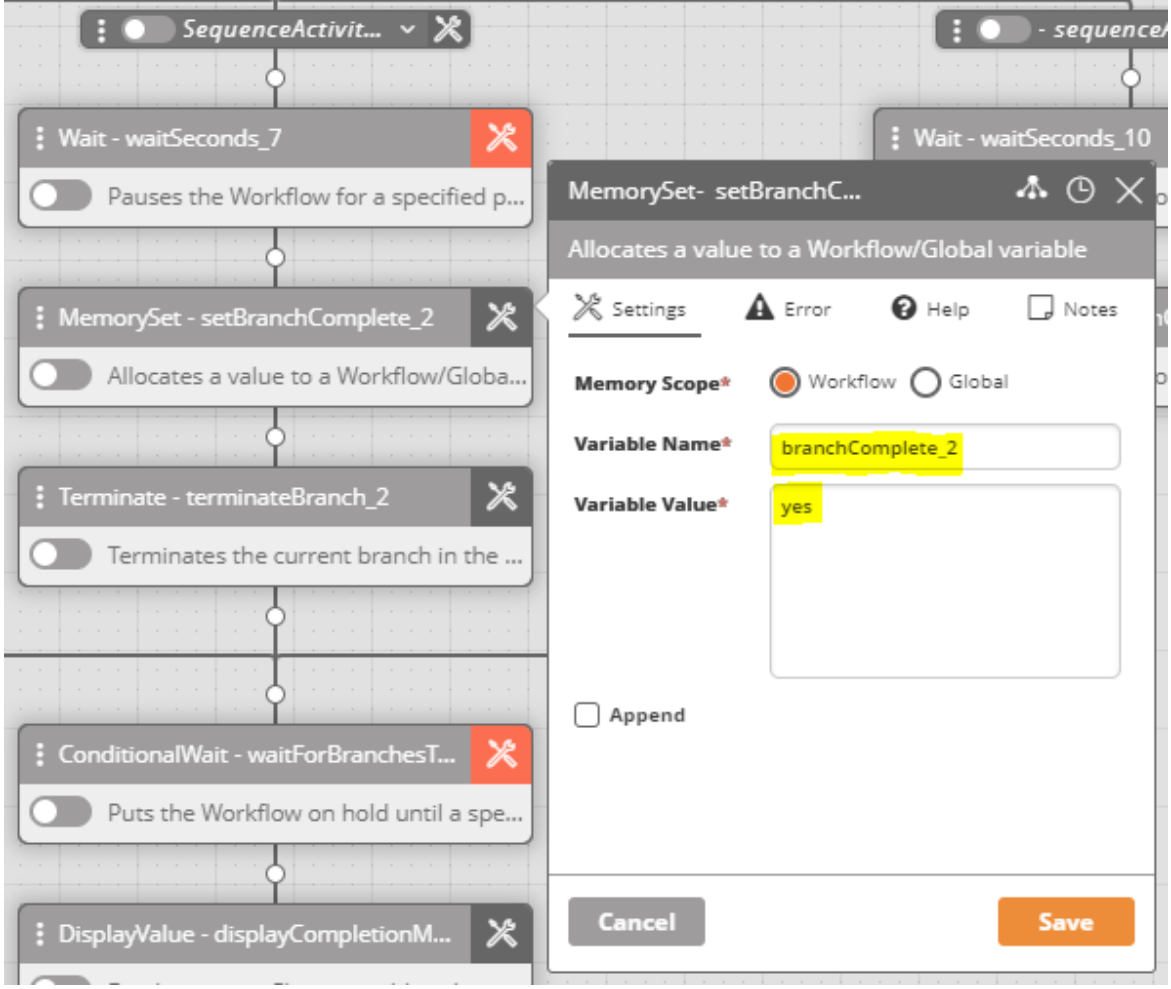


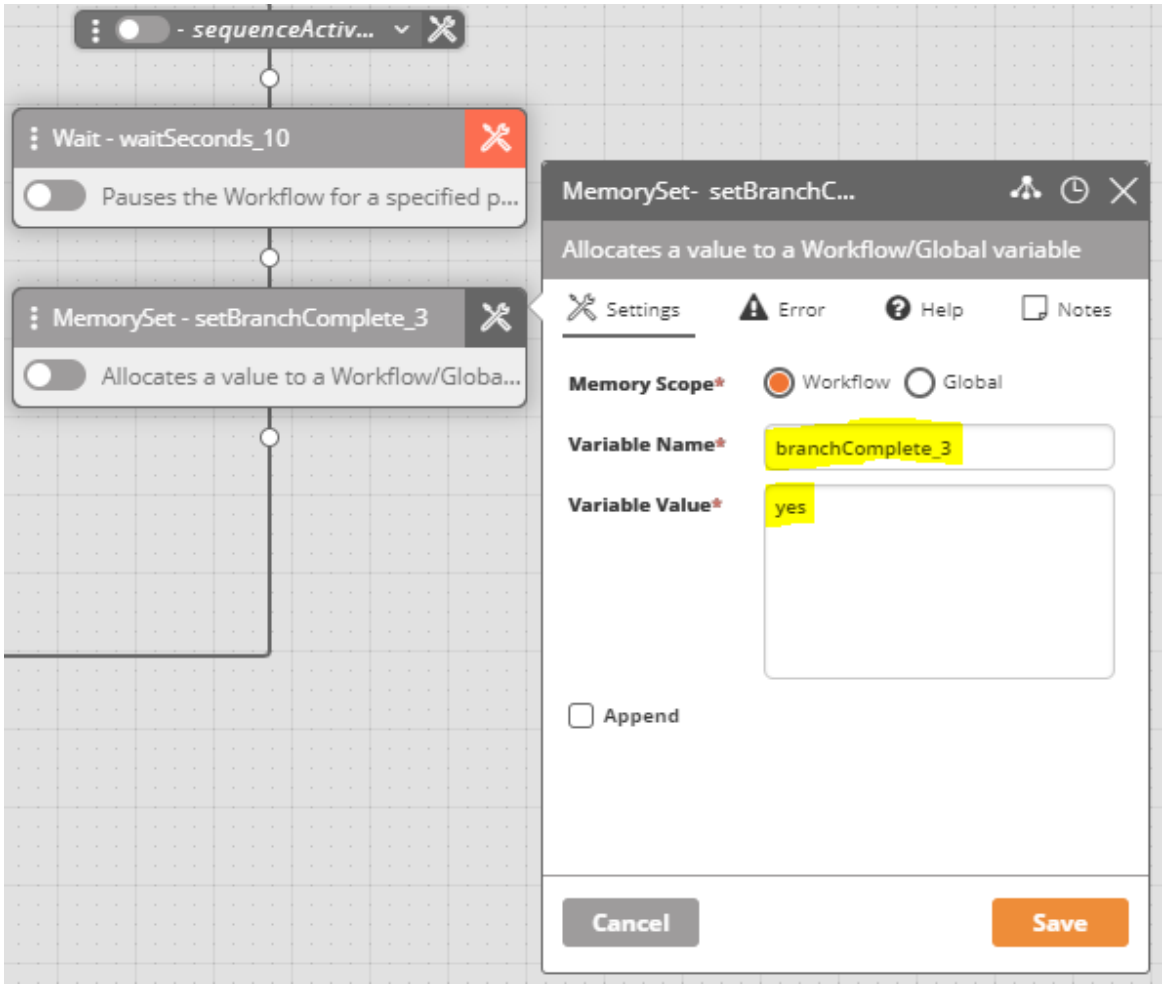
Configuring the Activities - MemorySet

In each branch, create a **MemorySet** activity with the "Variable Value" field set to "yes". For the left branch, use "branchComplete_1" for the "Variable Name" field, as seen below.



For the middle and right branches, use a **MemorySet** activity configured the same way, except use "branchComplete_2" and "branchComplete_3" for the "Variable Name" field, respectively. See below.

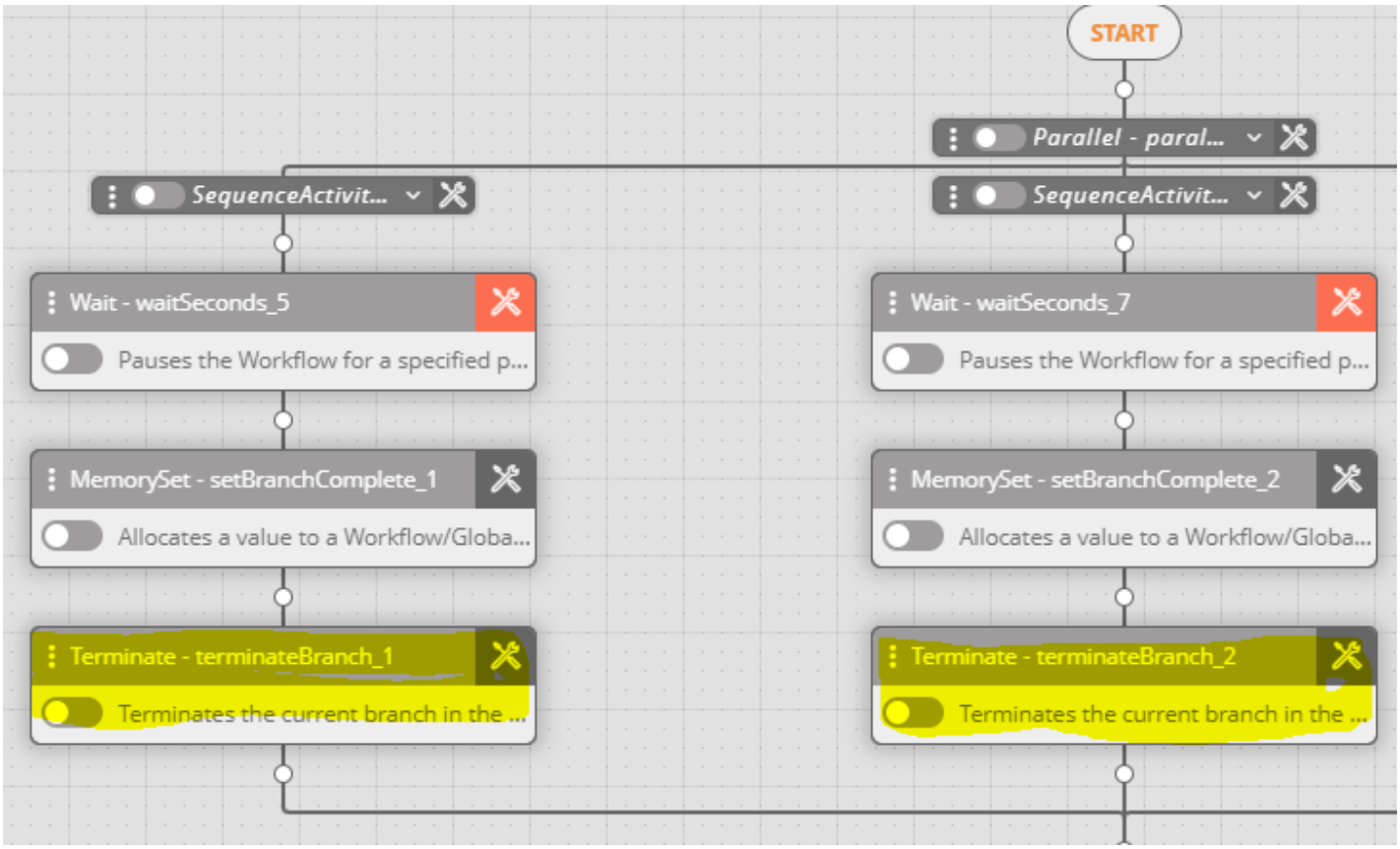

SequenceActivit...
Wait - waitSeconds_7
Pauses the Workflow for a specified p...
MemorySet - setBranchComplete_2
Allocates a value to a Workflow/Globa...
Terminate - terminateBranch_2
Terminates the current branch in the ...
ConditionalWait - waitForBranchesT...
Puts the Workflow on hold until a spe...
DisplayValue - displayCompletionM...
Wait - waitSeconds_10
MemorySet- setBranchC...
Allocates a value to a Workflow/Global variable
Settings
Error
Help
Notes
Memory Scope*
Workflow
Global
Variable Name*
branchComplete_2
Variable Value*
yes
Append
Cancel
Save


- sequenceActiv...
Wait - waitSeconds_10
Pauses the Workflow for a specified p...
MemorySet - setBranchComplete_3
Allocates a value to a Workflow/Globa...
MemorySet- setBranchC...
Allocates a value to a Workflow/Global variable
Settings
Error
Help
Notes
Memory Scope*
Workflow
Global
Variable Name*
branchComplete_3
Variable Value*
yes
Append
Cancel
Save

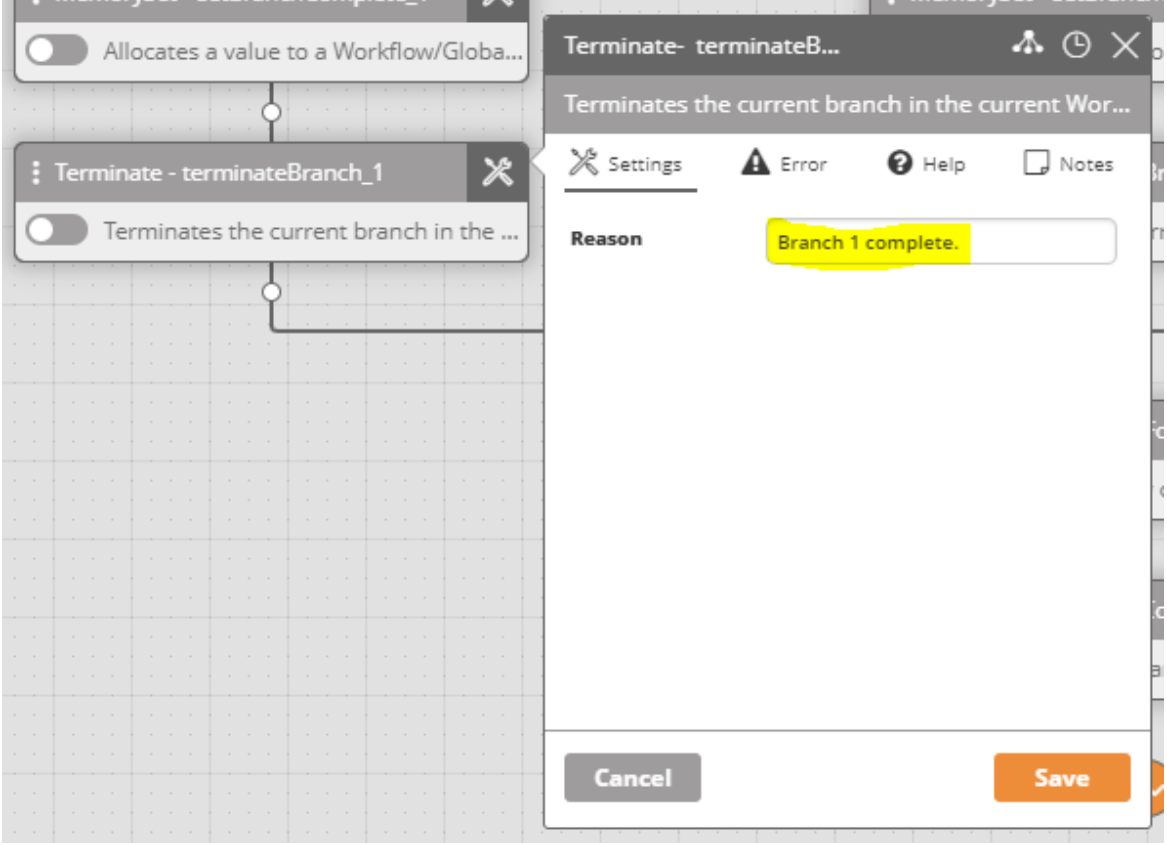
Configuring the Activities - Terminate

Due to the nature of how **Parallel Branches** behave, we want to include a **Terminate** activity in all but one of the branches. For your own workflow development, always be sure to leave one of the **Parallel Branches**without a **Terminate** activity. Without using a **Terminate** activity in all but one branch, all of the activities that follow a series of **Parallel Branches** will be executed separately for each and every branch that completes its execution. Without leaving one of the branches without a **Terminate** activity, none of the activities below a series of branches will be executed.

In the screenshot below, we can see a **Terminate** activity in the left and middle branches.



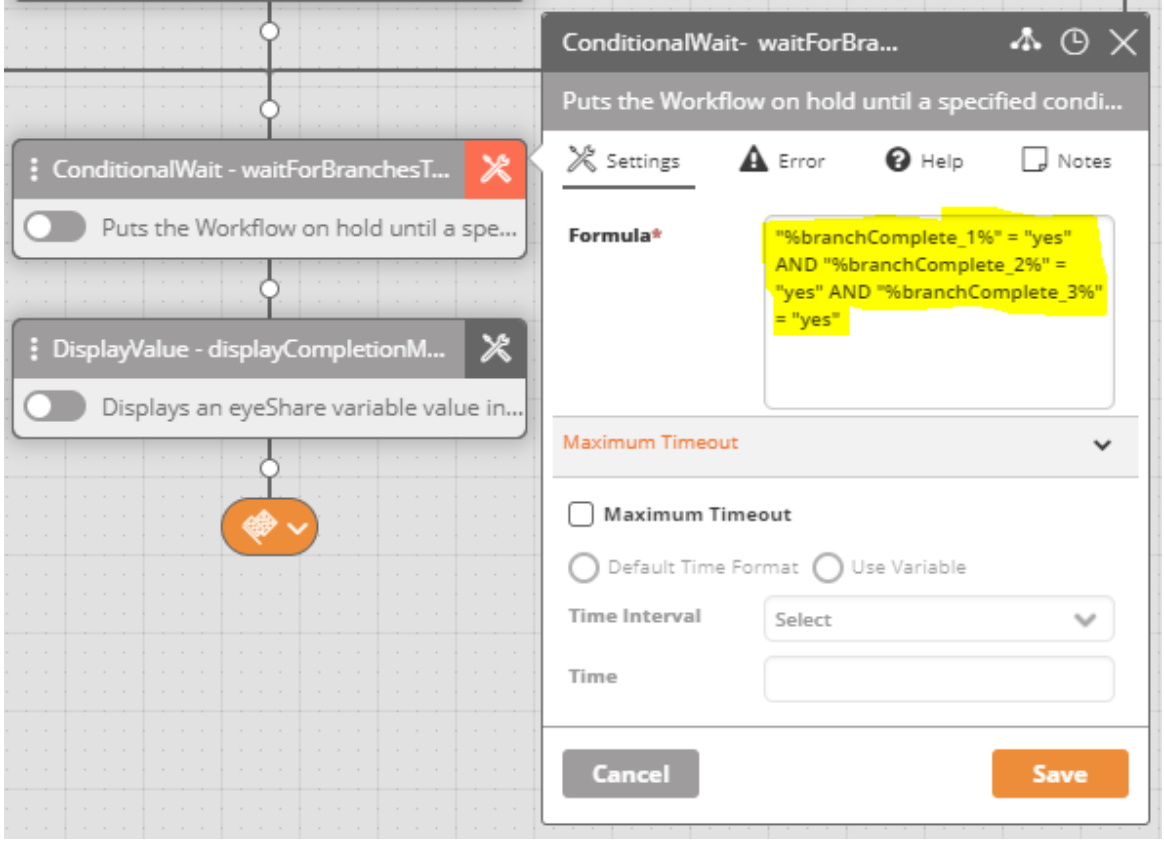
For your own development and debugging purposes, it may be helpful to include a termination reason in the **Terminate** activity's configuration, as seen in the example below.



Configuring the Activities - ConditionalWait

This final activity is what ties the entire process together. Using the **ConditionalWait** activity, we are telling the workflow that it must pause until a certain condition, or series of conditions, is met. There are other methods that can be used to achieve this type of functionality in a workflow (i.e. wait for a condition to be true), but this is the simplest and most straight-forward option for this example.

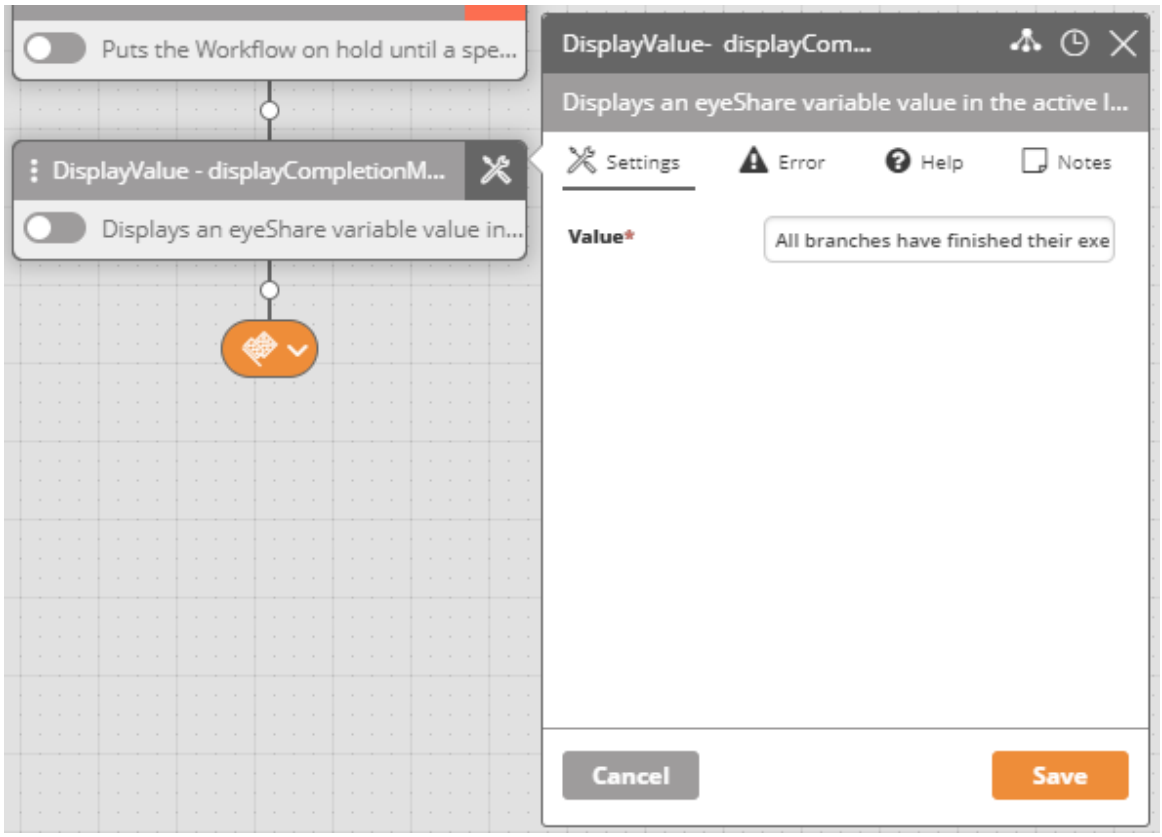
As seen in the screenshot below, a custom formula is used to evaluate the contents of the three variables we created in our **Parallel Branches**. Those variables are **%branchComplete_1%**, **%branchComplete_2%**, and **%branchComplete_3%**.



The formula utilized for this example ensures that all three of those branch completion variables are set to "yes" before proceeding with the rest of the workflow. That formula is as follows:

```
"%branchComplete_1%" = "yes" AND "%branchComplete_2%" = "yes" AND "%branchComplete_3%" = "yes"
```

For this example, the final **DisplayValue** activity is present in order to notify us that all three branches have completed their execution. In your own workflows, this would be the point where the rest of the automated process would continue.



Successful Workflow Execution Example

In the screenshot below, we can see the **Workflow Execution Log** for a successful run of this example workflow.

Branch Name	Event Type	Activity Name	Status	Result	Remark
	Incoming event				
sequenceActivity1	Wait	waitSeconds_5	Executed		5 Seconds
sequenceActivity2	Wait	waitSeconds_7	Executed		7 Seconds
sequenceActivity3	Wait	waitSeconds_10	Executed		10 Seconds
sequenceActivity1	MemorySet	setBranchComplete_1	Executed	Success	yes
sequenceActivity1	Terminate	terminateBranch_1	Executed		Branch 1 complete.
sequenceActivity2	MemorySet	setBranchComplete_2	Executed	Success	yes
sequenceActivity2	Terminate	terminateBranch_2	Executed		Branch 2 complete.
sequenceActivity3	MemorySet	setBranchComplete_3	Executed	Success	yes
Workflow Root	Conditional Wait	waitForBranchesToFinish	Executed		"yes" = "yes" AND "yes" = "yes" AND "yes" = "yes" No timeout
Workflow Root	Display value	displayCompletionMessage	Executed	All branches have finished their execution.	All branches have finished their execution.
	Terminate		Executed		

Reviewing this log, we can see that all three **Wait** activities are executed at the same time, and then each branch's **MemorySet** and **Terminate** activities are executed as soon as that wait time is over. Lastly, we see the **ConditionalWait** activity shows that all three **%branchComplete_X%** variables equal "yes" before the workflow proceeds to its next activity, which for this tutorial is a **DisplayValue** that prints out "All branches have finished their execution".