Video Transcript

Hello and welcome to a new demo on invoke workflow and arguments! In this demo we’ll learn how to invoke workflows in our automation project and use arguments.

So far in this course, we’ve worked with only one workflow, Main.xaml. However, breaking our processes into smaller workflows is paramount to good project design. Dedicated workflows allow independent testing of components while encouraging team collaboration by developing working on separate files.

How do you think we pass data between the parent and the child workflows?

When working with multiple workflows, instead of variables, we use arguments to pass data between the parent and the child workflows.

Before we dive into using arguments, let’s understand what arguments are.

In a global sense, arguments resemble variables, as they store data dynamically and pass it on from one project to another.

Variables pass data between activities, while arguments pass data between automations. As a result, they enable you to reuse certain projects time and again.

Let’s start!

First, we'll create a new workflow with the Sequence layout that we can later invoke in Main.

The purpose of this workflow will be to compute the sum of two given double variables.

Let’s rename the sequence and add a descriptive annotation.

Let’s create an argument that we'll call Sum for now.

An interesting point to note here is that an argument is a way of providing more information. What kind of information? This information is about the direction of an argument, that tells the application where the information stored in them is supposed to go.

Now, there are three directions that we can choose for an argument: In, Out, and In/Out.

"In" arguments can be used only to pass data into the invoked workflow, "out" only to extract data from the invoked workflow, and "in/out" works both ways.

The "Property" option is used to indicate that an argument isn't currently in use, without having to delete it from the list.

We'll use the Sum argument to pass the result of the calculation back to Main. Thus, we'll choose the direction Out.

An argument direction should always prefix the name of the argument. Since we chose "out" for the sum, let's use that as a prefix.

The types that are available to choose from for variables are also available for arguments. So besides the list that we have here, we can click on "browse for types" and choose the one that we need. In our case, let's go with Double.

To add the terms, let's create another pair of Double arguments. We'll name the first one as “in first value” and the type as double. And the second one let's name as “in second value” and of type double.

Alright, for arguments with direction "in" we can choose a default value which can be used. But any value that we pass to the argument from the invoked workflow will overwrite this one.

Lastly, let's also create an "in/out" string argument that we'll use to display the current step of the calculation. We'll name this one “IO execution Info”.

Let's log the value of the “IO execution info” argument, and then add the actual calculation step.

We'll assign the result of the first value plus the second value to the out Sum argument.

**Out\_Sum = in\_FirstValue + in\_SecondValue**

Let’s add another Assign activity to set the IO execution info argument to “calculation completed”

Okay, we're done here. Let’s click Save for this workflow.

Back to main, let’s invoke the Calculate Sum workflow that we just created.

To do that we can add an Invoke workflow activity from the Activities panel and then configure it…

Or,

We can drag and drop the workflow that we want to invoke from the Project panel, thus having the activity automatically configured.

Now, let’s rename the sequence and add annotations to the sequence and the invoke workflow activity

The next step is to configure the mapping of the arguments. Let’s click Import Arguments.

We’ll press Control + K in the Value field for out Sum to create a new variable in Main for it. Here, we'll need a variable because the argument is of type "out" and it needs a place to store the value retrieved from the invoked workflow.

The same applies for the "in out" argument, so let's create another variable as well here.

For arguments of the type "in" we can pass both variables or hardcoded values. Let’s assign two hardcoded values for this example.

Once done, click ok to save the configuration.

Before we move to the next step, let’s check the log properties of the invoke workflow.

This is a property for logging in invoke workflow where the log messages are integrated in the invoke workflow

The three types of logs are:

**Log Entry**– this tells us what type of information to automatically log when entering the invoked resource.

**Log Exit** – talks about what type of information to automatically log when exiting the invoked resource... and the

**Log Level** - The logging level applied to the messages generated by Log Entry and Log Exit.

We'll explore logging in the Introduction to Logging course.

Next, let's log the ExecutionInfo variable as well as the result. For this we'll need a log message activity where we'll have Execution Info plus “Result is:” plus “Sum dot to string” as the message. Great!

Before we run the workflow, let's assign an initial value to the ExecutionInfo variable that will be passed on to the invoked workflow.

Now, let’s run the project in debug mode and see what happens.

Let’s view the Output panel. The workflow is executed successfully!

We see that the IO argument has been logged and passed the “Before calculation” value of ExecutionInfo to the CalculateSum workflow.

The value “Calculation completed” replaces the Executioninfo. This is passed back via the IO argument and logged in to Main.

The two hardcoded values are passed to the CalculateSum workflow via two In arguments. Their sum is assigned to the  Out Sum argument, then passed back to Main and logged.

Ever wondered what happens when arguments are configured in invoke workflow and then renamed?

Let’s check it out. To do this:

We’ll create a second workflow and call it Renaming arguments

Let’s create arguments here just like how we did for the previous workflow.

For demo purposes, let's create three arguments and call them,

In FirstNumber , in SecondNumber  and out Result all three of the type double.

Let’s add an Assign activity and enter the To field and the Value field

**out\_Result = In\_FirstNumber  +  in\_SecondNumber**

Let’s save this and invoke this second workflow in Main.

Let’s go to Main and click on the Project Renaming Arguments

Great, let’s configure the mapping of the arguments. We’ll click on Import Arguments. And provide a variable for out Result in Main using the Ctrl+K method and save

The arguments are configured, now let’s go back to our second workflow and change the name of an argument here.

For demonstration, let’s change in FirstNumber to in LastNumber.

We see that the renamed argument updates itself in all occurences .

Alright, let's save this and go to Main.

We’ll click on Import Arguments of the second workflow and see if the invoke workflow is configured.

And it is. Awesome!

Let’s list out the key takeaway from the demo, we learned how to work with multiple workflows and use arguments to pass data between the parent and the child workflows. We also learned what happens when arguments are configured in invoke workflow and then renamed.

For more practice on this topic, try to explore the robustness of arguments and their directions to pass data between multiple workflows.

And this is it for this demo! Thank you for watching!