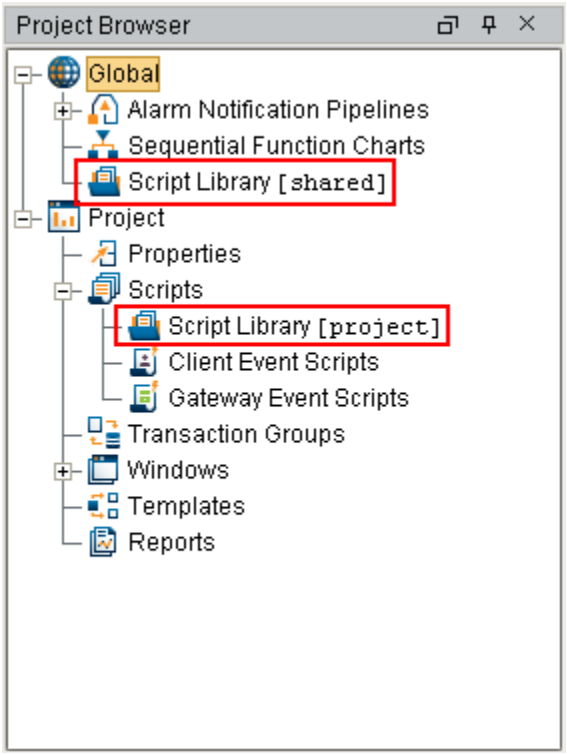
Project and Shared Scripts

Script Modules

Ignition's **Shared** and **Project Script Modules** are global libraries of scripts that can be called from anywhere within their scope. Shared Script Modules are available to the whole Gateway including every project, and Project Script Modules are available anywhere in the project that they were created in. These scripts are organized as named modules that all live under the **shared** or **pr oject** Script Library. To open the Script Module Editor, go to the **Project Browser** and double click on the **shared** or **project** Script Library.



On this page

...

Rule of Thumb: Never Blindly Copy-and-Paste

If you're unsure of when to put scripts in a script module vs. writing the script directly in an event handler, follow this simple rule: If you ever find yourself copying a script from one event handler to another, stop and refactor the script into a global script module! Then simply call your new module from the event handler. This rule will help prevent code duplication across your project which can be a major maintenance liability. Consolidating scripts into one location makes them easier to manage.

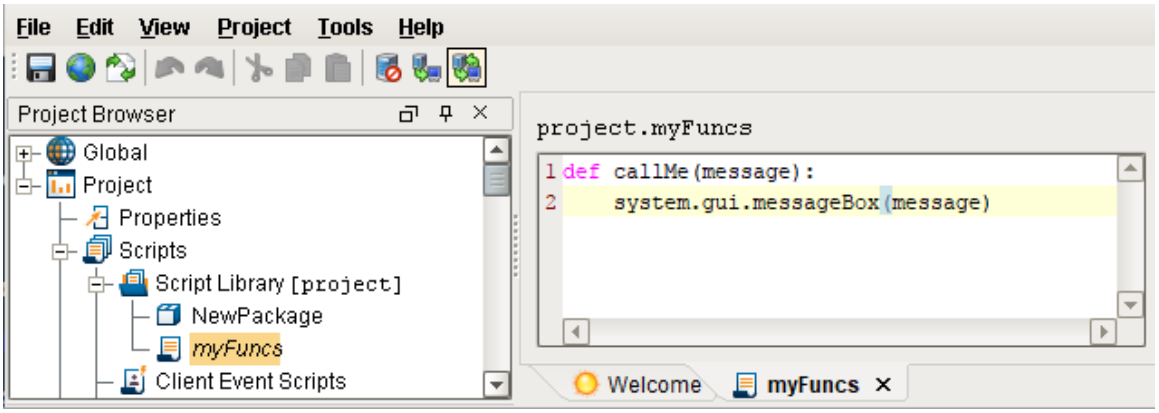
How to Use Script Modules

To add a script module, simply right click the **Script Library [project]** package and click the **New Script** option. Each module is a python script that can define many functions. You can also keep your scripting modules organized in **Packages** by creating additional nested subpackages. Packages act a little like folders except you can still add scripts to them. To create a Package, click on the Script Library **[project]** and click the **New Package** option.

For example, let's suppose you added the following script module named **myFuncs**, whose body is shown below.

Python - Project Script

```
def callMe(message):
	system.gui.messageBox(message)
```

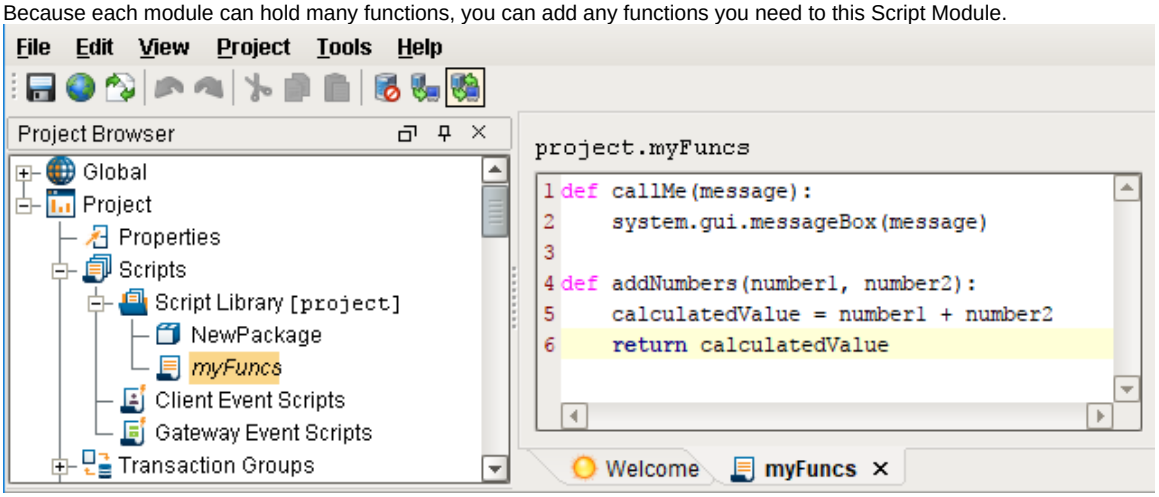


Now, anywhere in your project you can call this function.

Python - Calling the Project Script

```
project.myFuncs.callMe('Hello World')
```

Because each module can hold many functions, you can add any functions you need to this Script Module.



Just like before, you can call this function using:

Python - Calling Different Functions

```
result = project.myFuncs.addNumbers(14, 87)
```

Note, that the **addNumbers** function also has a return value. This will be returned to the script that called the function, allowing you to use it within that script.

System Import

In Ignition, certain libraries are automatically imported into any scripts you create to make everything easier. For example, the System Library is automatically imported, giving you access to all of the "`system.` functions" without having to add in "`import system`" at the beginning of your script. The Project and Shared Libraries are also imported, giving you access to any of the Script Modules you may have written. However, prior to version 7.7.0, Script Modules did not automatically import the System Library. This means that if using a version older than version 7.7.0, you will need to add "**`import system`**" at the beginning of your script to be able to use the System Library.

Project Scripts vs. Shared Scripts

There are a few differences between Project and Shared Script Modules. The first difference is the way each script type is called. Shared Scripts are called with "**shared.**" preceeding the event handler function, while Project Scripts are called with "**project.**" The other difference is where Shared and Project Script Modules can be used. Project Script Modules are limited to the Project Scope, and can only be used within the project they are located in. This is useful when you are using a script many times within a project, but do not want it to be used by any other project or elsewhere in your Gateway.

Shared Script Modules, on the other hand, are not limited by scope and can be used anywhere within any project or Gateway. However, this means the Shared scripts can be used by projects that possibly shouldn't have access to them. Since Shared Script Modules are stored on the Gateway, they are not part of a normal project export. So moving a project to a new Gateway will require that Shared scripts also be included in the transfer.

A Simple Example

The Script Modules are very easy to get started with, and we can quickly put together a simple example to that demonstrates how they work.

1. Navigate to the **Script Library [project]**, right click and select New Script. Name the new script **Example**.
2. In the script area, add this script:

Python - Message Box Input

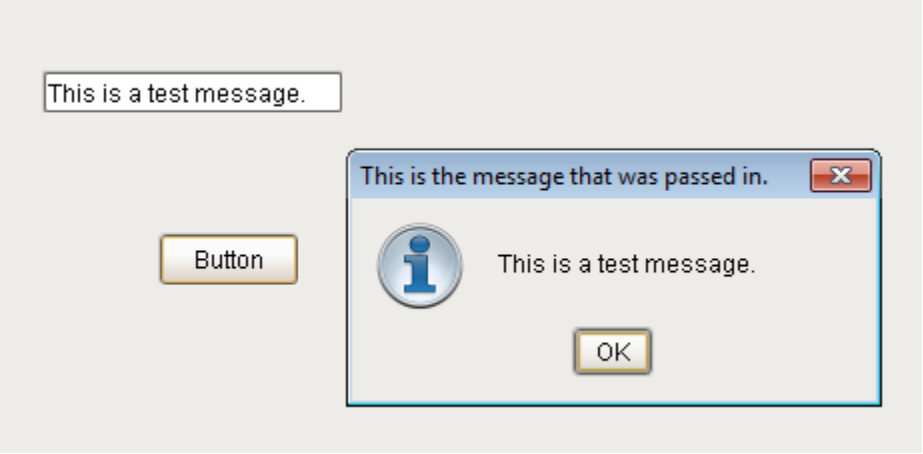
```
def myMessage(message):

        # Will take the message string that we pass in and display it in a message box.
        system.gui.messageBox(message, "This is the message that was passed in.")
```

3. On a new window, drag a **Button** and **Text Field** components onto the screen.
4. Right click the button and navigate to Scripting.
    a. On the `actionPerformed` Event Handler, add this script:

Python - Calling the Script with the Input

```
# Will call the project script that was just created, passing in the value of the text
field for the message.
project.Example.myMessage(event.source.parent.getComponent('Text Field').text)
```

5. Save the project. The project script that we created in step 2 will not be available to use until it is saved.
6. Try it out! Type a message into the text field and then click the button. You should see a message box popup with your message inside.



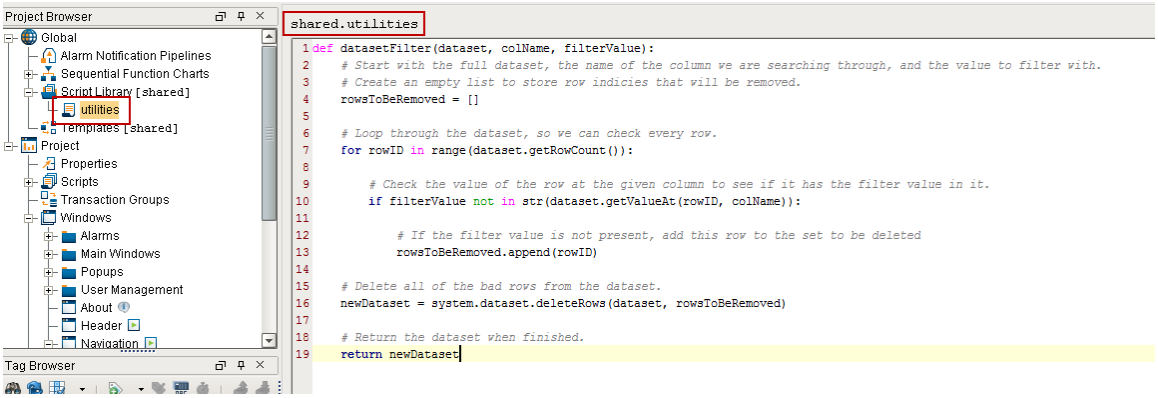
Filtered Dataset Example

In this example, we can use a shared script to filter a dataset in any project. The script takes in a dataset, a column name, and a filter string. It will then remove any of the rows that do not contain the filter string in them.

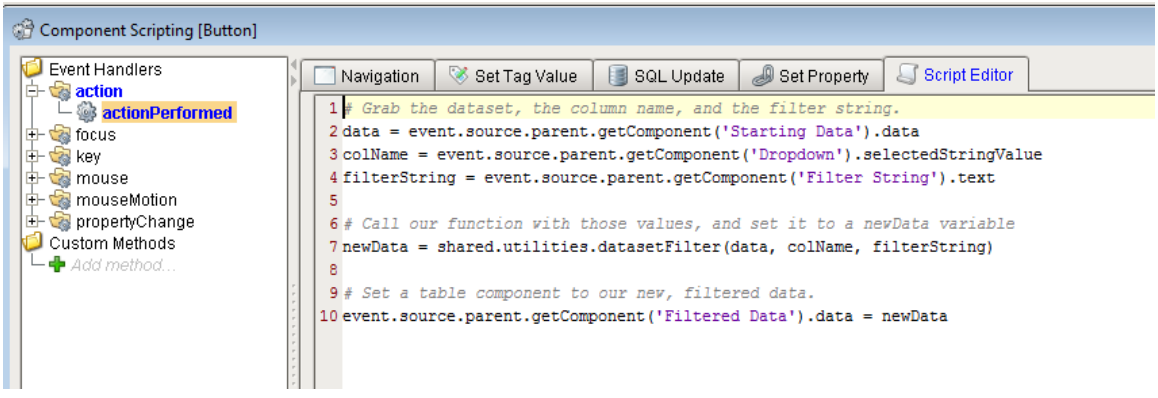
Python - Dataset Filter

```
def datasetFilter(dataset, colName, filterValue):
	# Start with the full dataset, the name of the column we are searching through, and the value to
filter with.
	# Create an empty list to store row indicies that will be removed.
	rowsToBeRemoved = []

	# Loop through the dataset, so we can check every row.
	for rowID in range(dataset.getRowCount()):

		# Check the value of the row at the given column to see if it has the filter value in it.
		if filterValue not in str(dataset.getValueAt(rowID, colName)):

			# If the filter value is not present, add this row to the set to be deleted
			rowsToBeRemoved.append(rowID)

	# Delete all of the bad rows from the dataset.
	newDataset = system.dataset.deleteRows(dataset, rowsToBeRemoved)

	# Return the dataset when finished.
	return newDataset
```

By placing the script in a shared script called utilities, it can be used by any project, allowing us to filter a dataset no matter what the data is.



It can then be called in a project on something like a button component using **shared.utilities.datasetFilter()**.



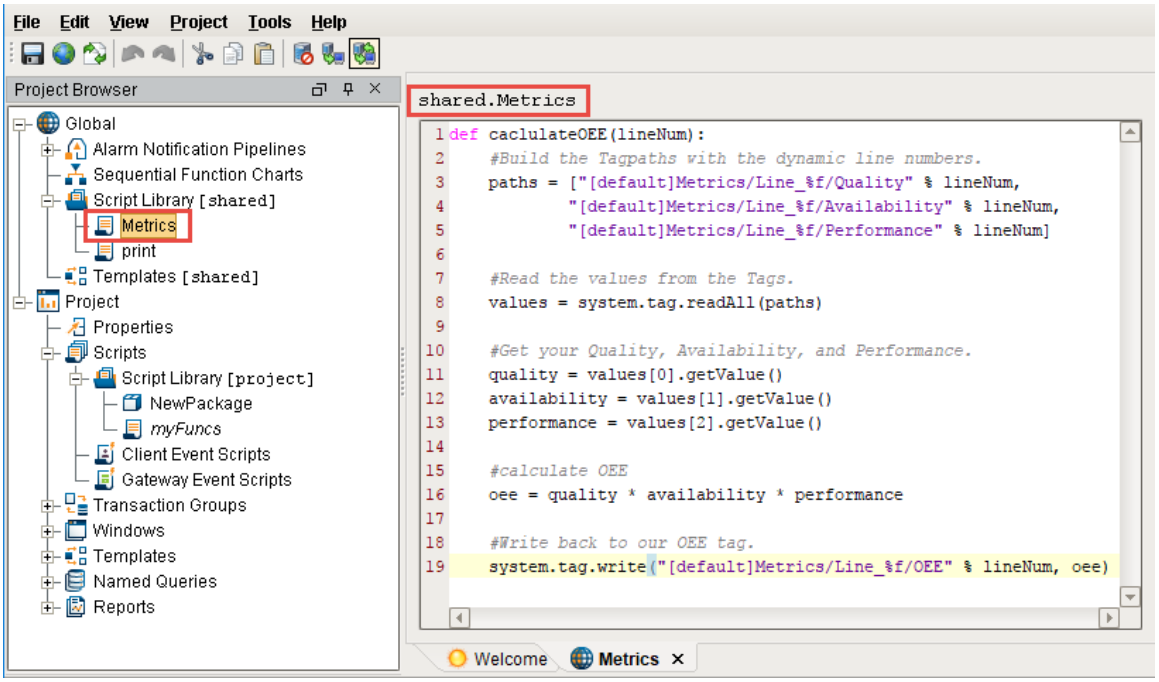
Calculate OEE Example

In our projects, we use a fairly simple script that calculates OEE, and writes it to a Tag as shown below.

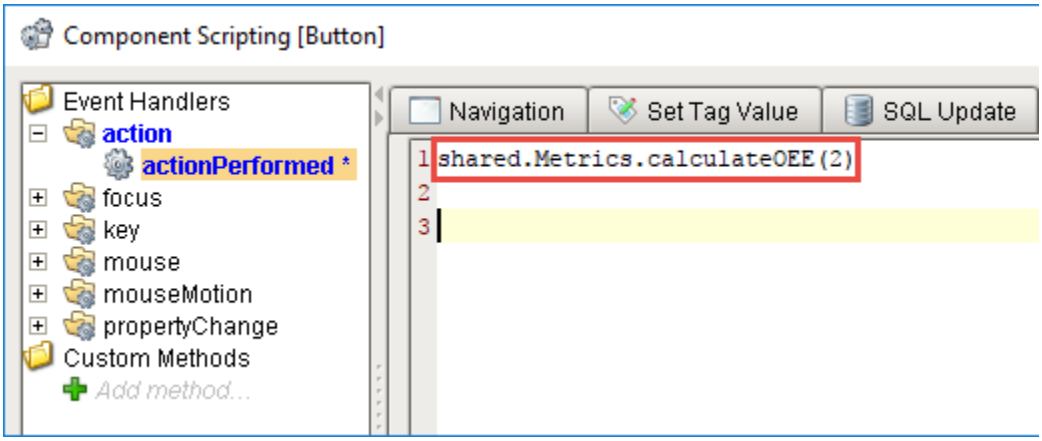
Python - Calculate OEE

```
def caclulateOEE(lineNum):
        #Build the Tagpaths with the dynamic line numbers.
        paths = ["[default]Metrics/Line_%f/Quality" % lineNum,
                        "[default]Metrics/Line_%f/Availability" % lineNum,
                        "[default]Metrics/Line_%f/Performance" % lineNum]

        #Read the values from the Tags.
        values = system.tag.readAll(paths)

        #Get your Quality, Availability, and Performance.
        quality = values[0].getValue()
        availability = values[1].getValue()
        performance = values[2].getValue()

        #calculate OEE
        oee = quality * availability * performance

        #Write back to our OEE tag.
        system.tag.write("[default]Metrics/Line_%f/OEE" %(lineNum), oee)
```

Instead of copying this script between projects and throughout all of the windows that need it, a Shared Script Module was created in the Shared Script Library called "Metrics." The code was copied and pasted into the scripting area in the Shared Script Library.



This could then be called from a script in any project by calling **shared.metrics.calculateOEE()**:



Related Topics ...

- User Defined Functions
- Scripting in Ignition