Automation Workbench

- Introduction
- Visual Scripting

Introduction

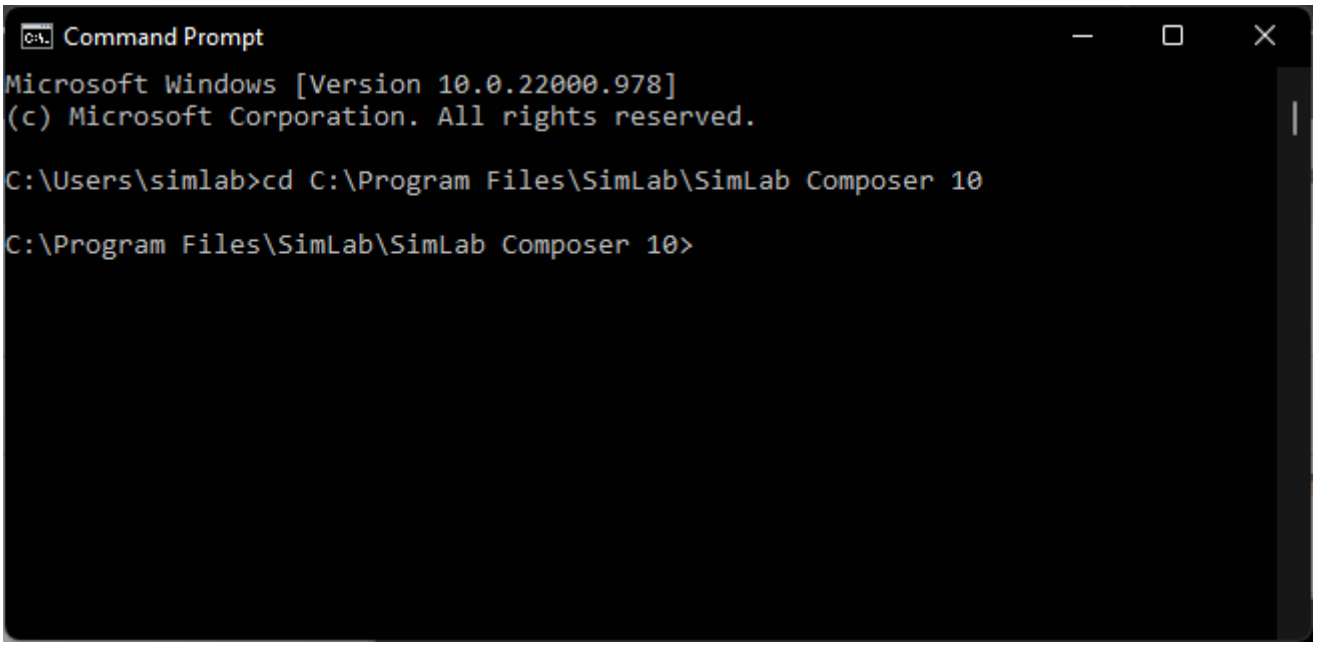
SimLab Composer allows using Python, and Java Scripts to automate processes, both scripts automation is supported in two modes:

1. Command line (for batch processing for a large number of files). This includes command line without scripting using -ie command, and with scripting using Python, and Java scripts.
2. Interactive mode (from inside the GUI of SimLab Composer)

Scripting is supported in the Ultimate edition of SimLab Composer

Command line without scripting

Open the command line window, by typing "cmd" in Start. Go to the directory where SimLab Composer was installed, the default installation directory is "C:\Program Files\SimLab\SimLab Composer 10" to go there type **cd C:\Program Files\SimLab\SimLab Composer 10**



Now to run import/export functions in SimLab Composer, type Sim.. then start clicking the Tab button, until SimLabComposer.exe appears.

Type in the code -ie <import_file> <export_file>

With actual file locations, the below line will convert RubikCube.obj 3D models into RubikCube.skp in the indicated folders. Don't forget " "

-ie "C:\Users\simlab\Desktop\Delete\RubikCube.obj"
"C:\Users\simlab\Desktop\Delete\RubikCube.skp"

Check this article for more about the command line-based methods, also for commands on Mac.

Command line Python Scripts

Python scripts can be run from the command line using the following command

```
SimLabComposer.exe -py “File.py”
```

So if the user named a script as example.py, and saved it in folder C:\Scripts, The user should use the following command

```
SimLabComposer.exe -py “C:\Scripts\example.py”
```

Passing arguments to Python Scripts

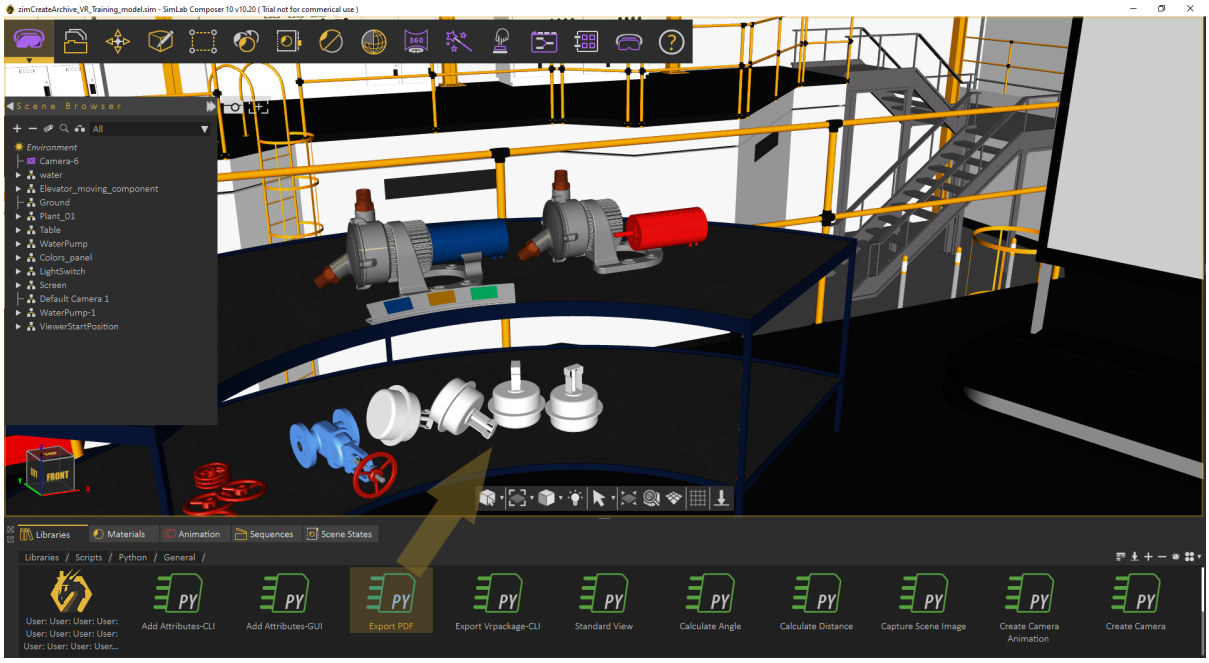
Passing arguments to a script makes it dynamic, and reusable without the need to change its code.

```
scene =Scene()
runtime =RunTime()
scene.reset()
fileName= runtime.args.getAsString("-path")
scene.importFile(fileName)
```

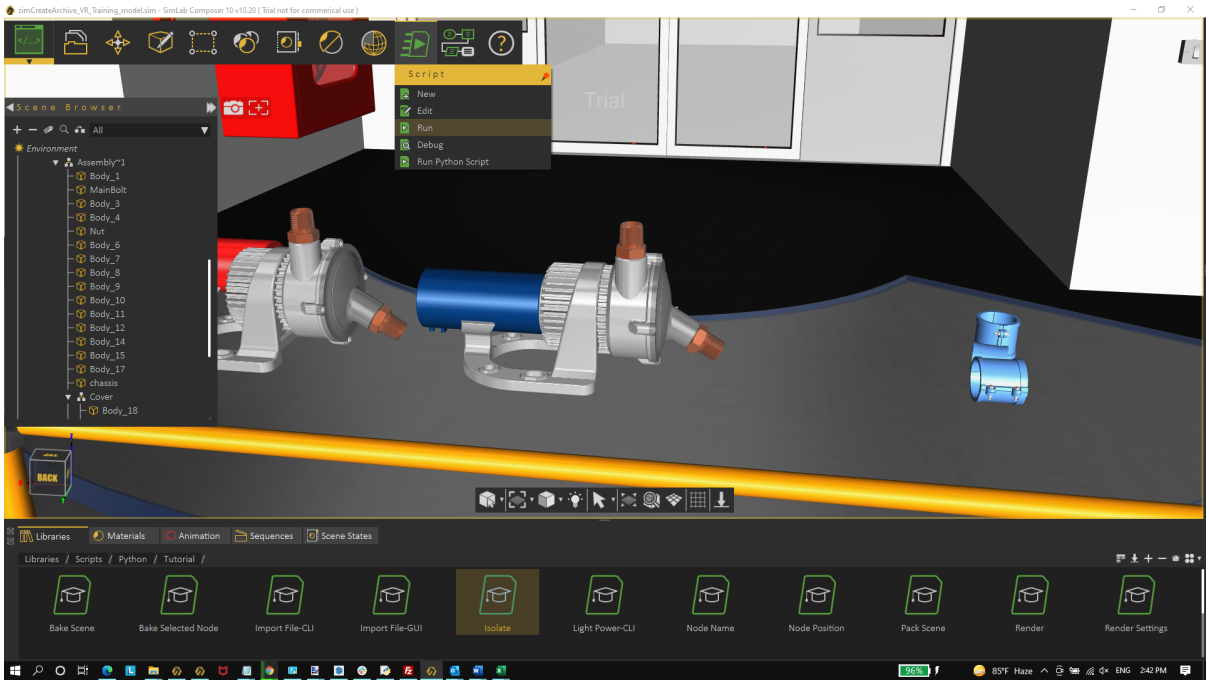
Interactive Scripting - Running Python script interactively

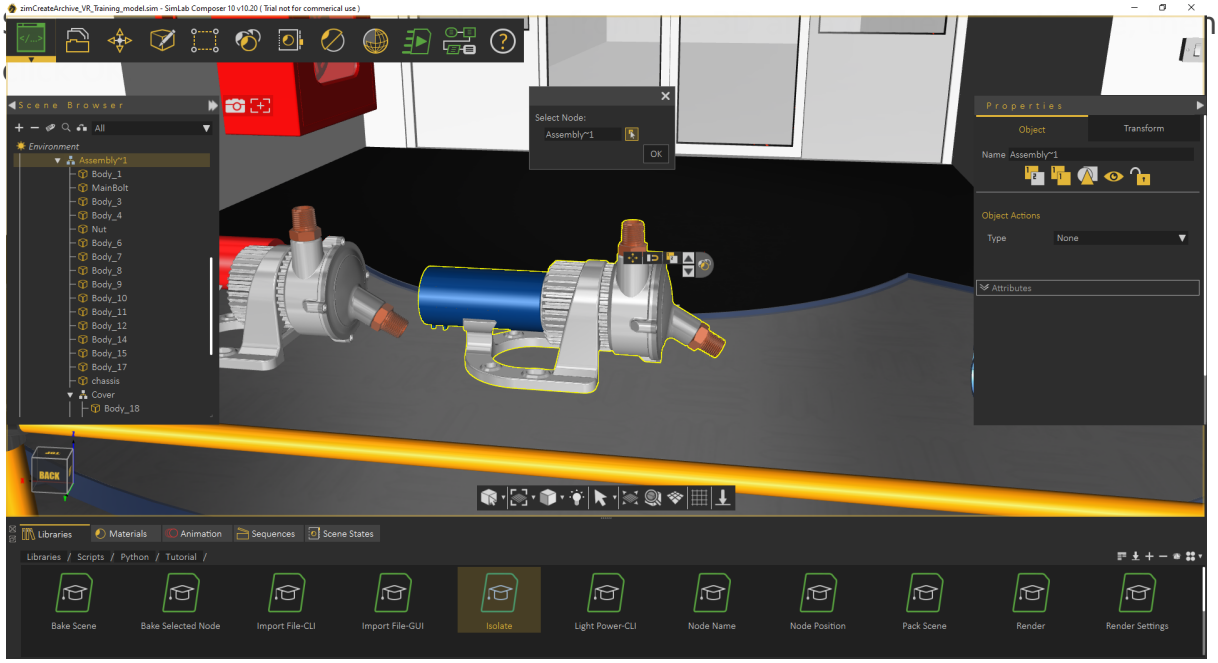
The user can run Python Scripts interactively in different ways:

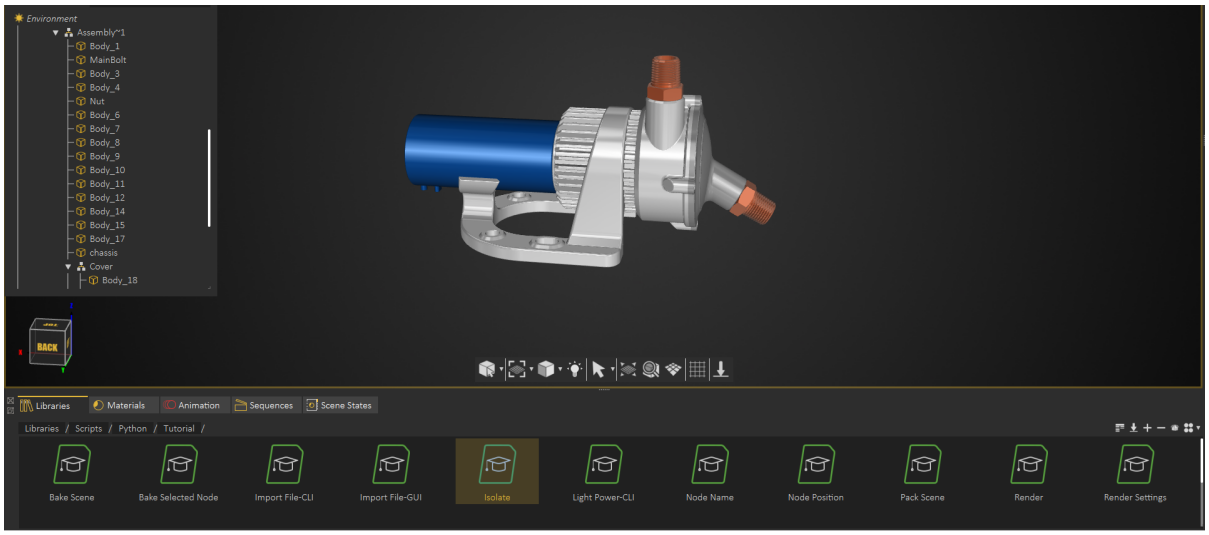
1. Select a script from the scripting library, drag it, and drop it on the 3D area



2. Select a script from the library, then from the Script menu, click Run


Script
New
Edit
Run
Debug
Run Python Script
Scene Browser
Environment
Assembly^1
Body_1
MainBolt
Body_3
Body_4
Nut
Body_6
Body_7
Body_8
Body_9
Body_10
Body_11
Body_12
Body_14
Body_15
Body_17
chassis
Cover
Body_18
Libraries
Materials
Animation
Sequences
Scene States
Bake Scene
Bake Selected Node
Import File-CLI
Import File-GUI
Isolate
Light Power-CLI
Node Name
Node Position
Pack Scene
Render
Render Settings

3. 
Select Node:
Assembly^1
OK
Properties
Object
Transform
Name Assembly^1
Object Actions
Type
None
Attributes
Scene Browser
Environment
Assembly^1
Libraries
Materials
Animation
Sequences
Scene States
Bake Scene
Bake Selected Node
Import File-CLI
Import File-GUI
Isolate
Light Power-CLI
Node Name
Node Position
Pack Scene
Render
Render Settings



My first Python script / Python Scripts using GUI input

The following script gets the location to save the rendered image, using a GUI dialog. Renders the current scene, saves the resulting image in the selected location, and finally displays a dialog indicating that rendering is done.

```
from simlabpy import *


scene = Scene()
runtime = RunTime()
render_path = runtime.ui.getSaveFileName("Exported rendered image location:", "",
"*.jpg;;*.png")
scene.render(render_path)
runtime.ui.alert("Rendered image was created.")
```

For a list of supported Python scripting commands visit this page

> Check out a blog about the approaches to automatically do things with SimLab Composer.


SimLab SOFT
Approaches to
automatically
do things with
SimLab Composer
Python
Node-based
Conversion
Schedule
Trigger
Scripting
Effecient
-py
Non-GUI-mode
CADVRter
JavaScript
-fl
Batch
Servers
Scripts
Faster
CMD
Smarter
CLI
SimLab
-ie
Companies
Terminal
-js
Quick
GUI-made
12
3
6
9
SimLab SOFT

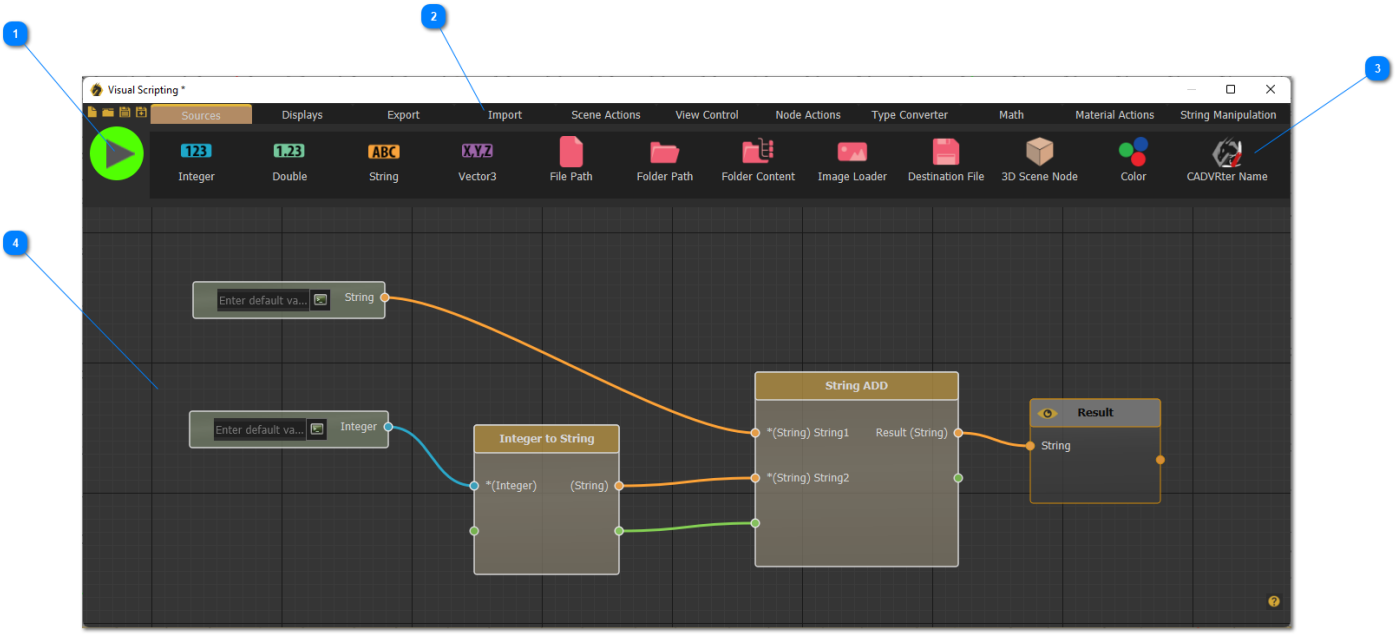
Visual Scripting

Visual Scripting in SimLab Composer is a tool created to minimize the programming effort for non-technical users with no coding skills. Therefore, instead of writing the computer code in a text editor, the Visual Scripting tool allows the user to develop her\his desired programs via block diagrams using a graphical user interface. This makes the code easier to be written as well as to be understood: Any designer, artist, or animator reading the diagrams can quickly grasp the flow of logic therein.

Getting Started

Select Visual Scripting from the automation workbench, the window shown below will appear with the following main parts:

1) Options to ‘Run’ the existing flow, ‘Create’ a new flow, ‘Open’, or ‘Save’ flows.
2) The Main Object Groups Bar, which includes the category titles of all the available sources and functions.
3) For each Object Group, corresponding ‘Sources’ and ‘Functions’ are available.
4) The ‘Work Space’, where blocks can be dragged, dropped, and linked to create the desired flow charts.



Any flow in the Visual Scripting should consist of three main components: source, smart block, and connections.

A) Source: Sources are used to define file paths and/or 3D nodes to be used as inputs for different blocks in the diagram. Users can find several source types in the sources tab.

B) Block: Blocks are basically functions that (can) take inputs and produce outputs. Each block performs a single process, for example, the block on the left side in the previous figure takes an integer number as input and converts it to a string as an output. Users can find several block types categorized into different tabs throughout the Main Object Groups Bar.

C) Connections: Connections are used to make the whole flowchart meaningful by linking sources and blocks. They thus define which sources and blocks are connected, accordingly, they arrange the execution order of the block diagram. The green parts on the blocks are not essential but they are very important to ensure the right execution order and to create dependencies between the blocks. To make the process easier for users. Ports on each block are colored based on the data type that should be passed through them.

After completing the flow diagram, hit the run button to start the execution, and a pop-up message will appear once the execution completes. Flowcharts can be saved and shared to be used in different projects.

To learn more about Visual Scripting in SimLab Composer, check the tutorial below, and visit the SimLab Visual Scripting web page

https://www.youtube.com/embed/yROPc90VFGk