Import and export data with CSV files

Visual Components

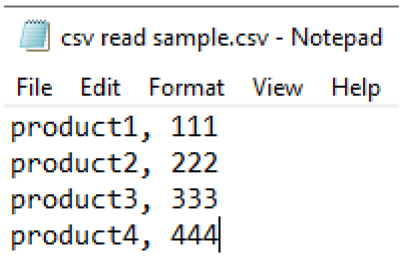
Contents

Working with CSV files using Python snippets

Before continuing this tutorial, make sure that you are using Visual Components Professional or Premium as you need to access the Modeling tab for creating Python scripts. In case you want to read data from CSV files or write data to CSV files within your python scripts, there are snippets already for these purposes. To understand how to use them, let's have a look at some simple examples.

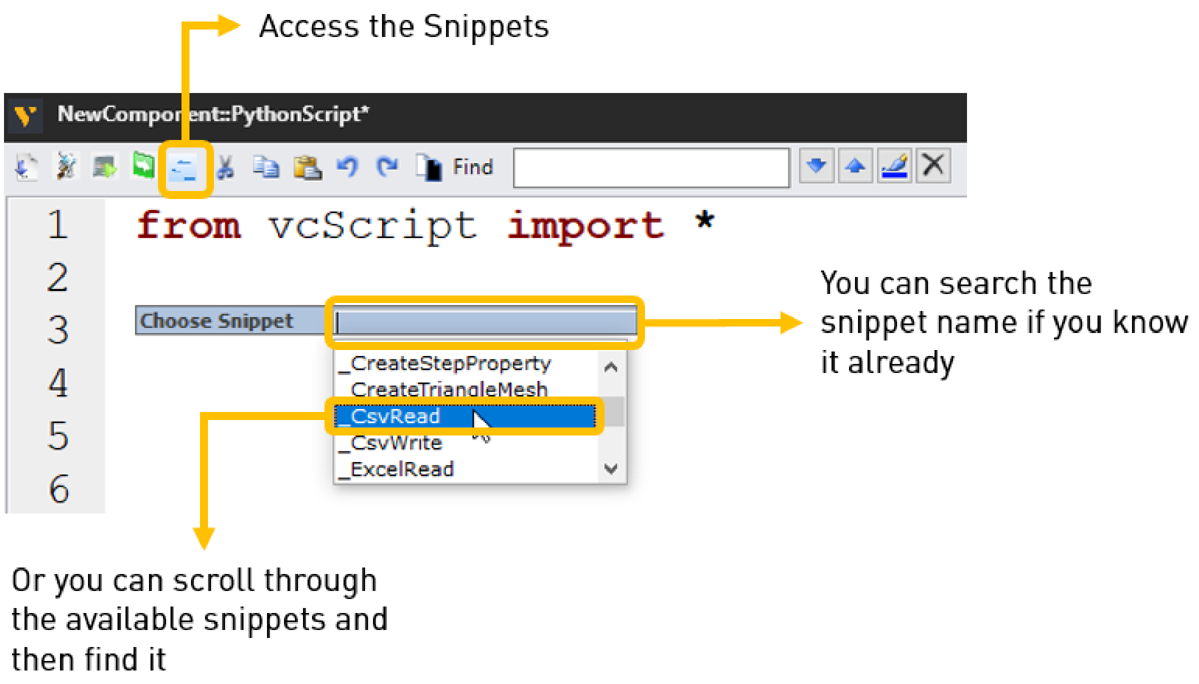
Reading from a CSV file and printing its data to the output panel

Consider the following data set in a Notepad which is saved as a CSV file. In there, there are four products with specified product IDs:



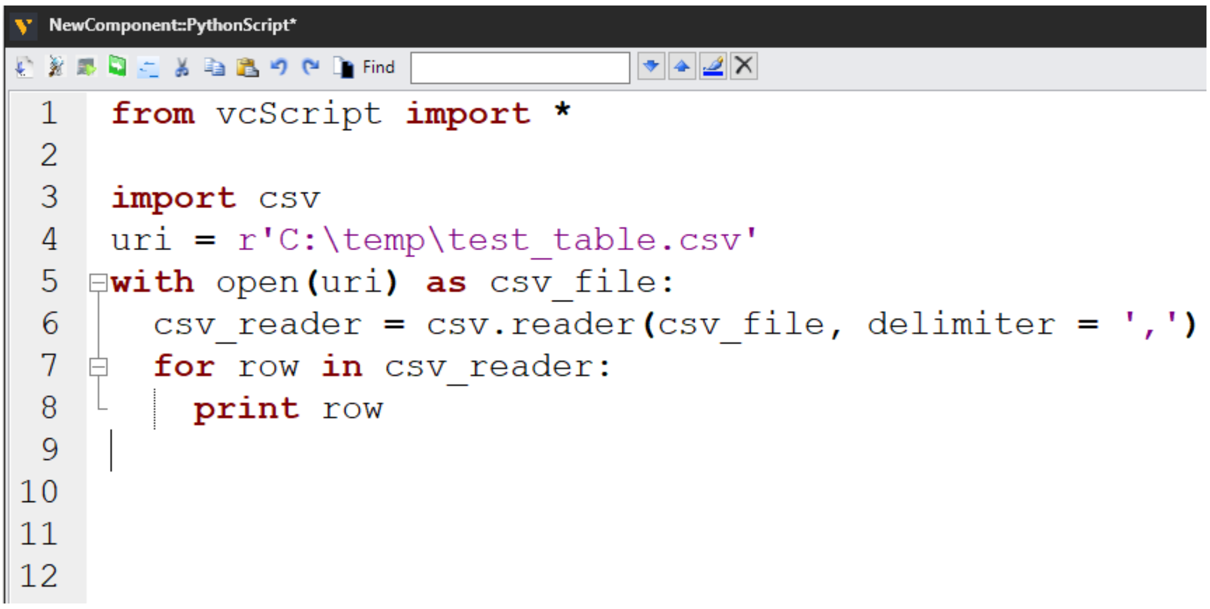
```
product1, 111
product2, 222
product3, 333
product4, 444
```

Now we want to read it within a python script using the existing snippet "_CsvRead". To access this snippet, just click on the Snippets, look for the _CsvRead, and then double-click on it:


Access the Snippets
NewComponent::PythonScript*
Find
Choose Snippet
_CreateStepProperty
_CreateTriangleMesh
_CsvRead
_CsvWrite
_ExcelRead
You can search the snippet name if you know it already
Or you can scroll through the available snippets and then find it

```
from vcScript import *
```

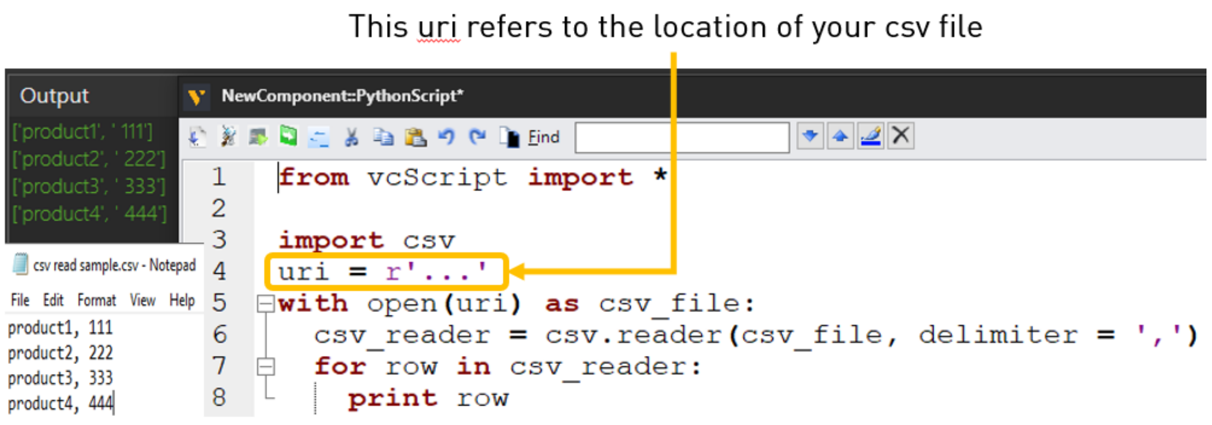
Here is what the snippet looks like:



```
from vcScript import *

import csv
uri = r'C:\temp\test_table.csv'
with open(uri) as csv_file:
  csv_reader = csv.reader(csv_file, delimiter = ',')
  for row in csv_reader:
    print row
```

In our example, we just want to read from our data set saved in a CSV format. At this step, we only need to replace the path to our CSV file in the uri and then compile the script. The content of the CSV file will be printed in the output panel:


This uri refers to the location of your csv file
Output
NewComponent::PythonScript*
Find
csv read sample.csv - Notepad
File Edit Format View Help

```
['product1', ' 111']
['product2', ' 222']
['product3', ' 333']
['product4', ' 444']
```

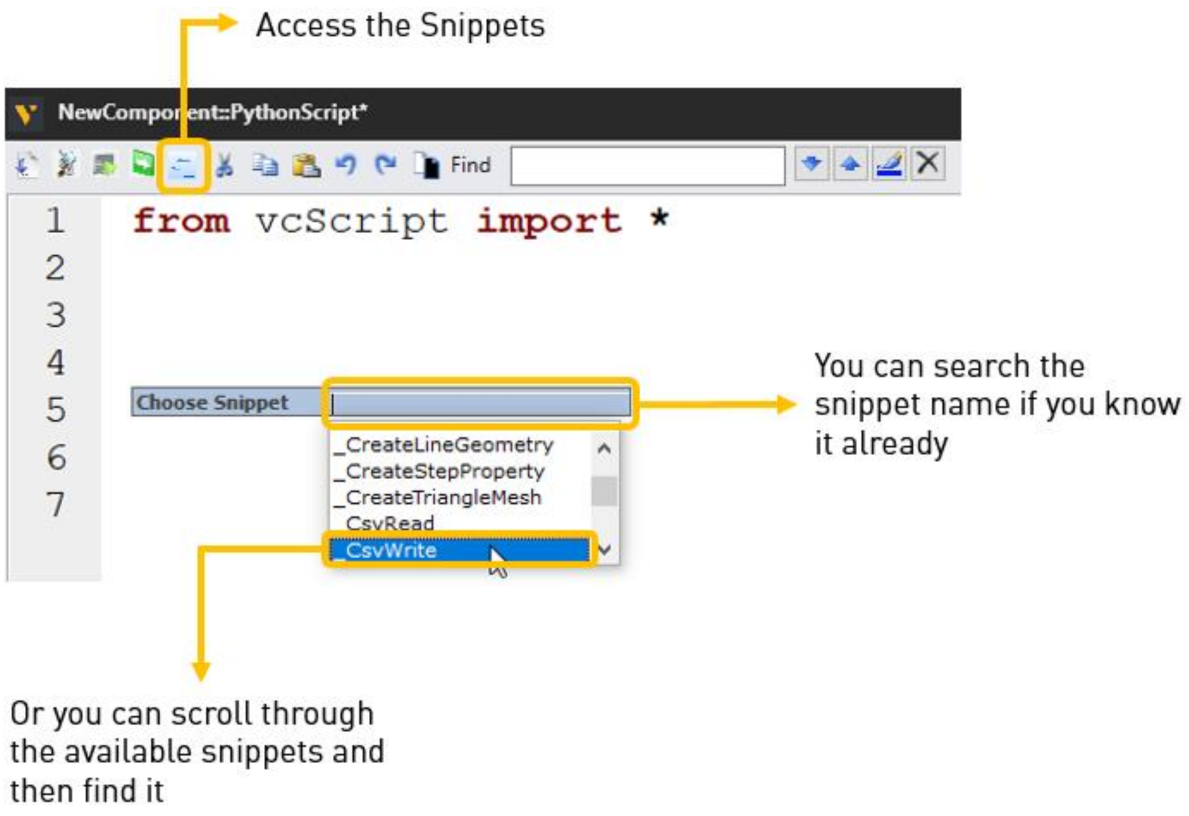
```
product1, 111
product2, 222
product3, 333
product4, 444
```

```
from vcScript import *

import csv
uri = r'...'
with open(uri) as csv_file:
  csv_reader = csv.reader(csv_file, delimiter = ',')
  for row in csv_reader:
    print row
```

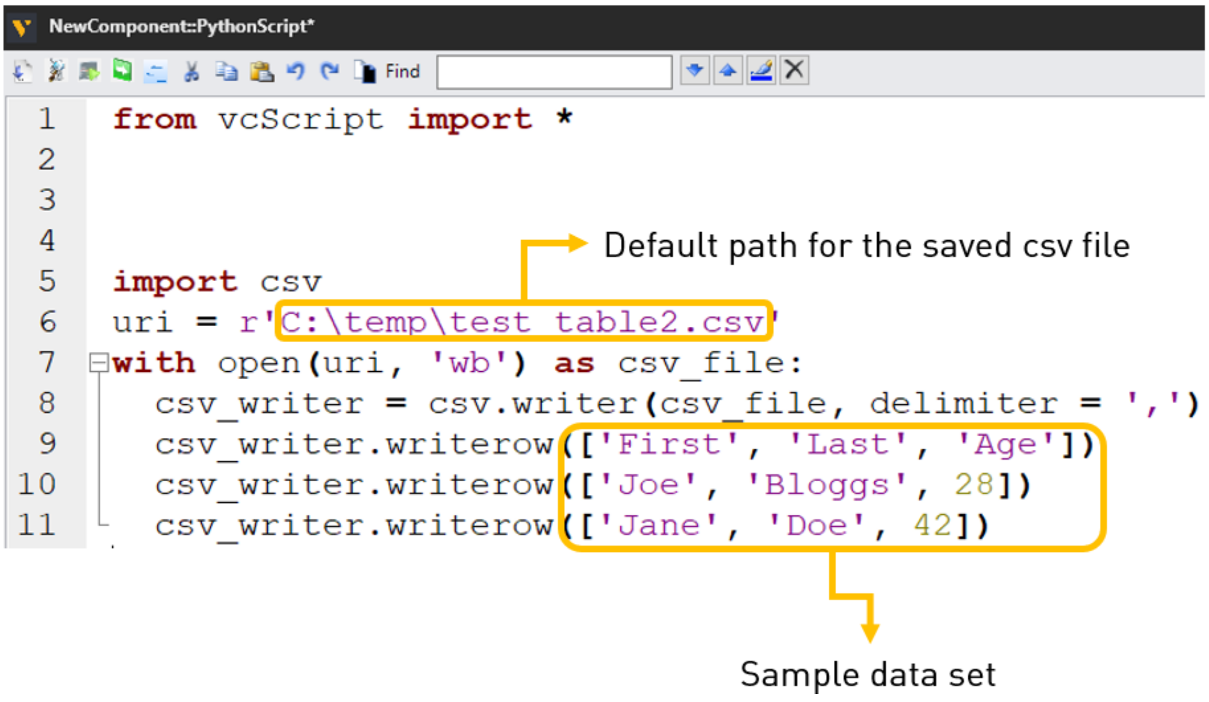
Writing to a CSV file

There is also a python snippet for writing data into a CSV file, which is called "_CsvWrite". You can similarly find this snippet by just searching for its name:


Access the Snippets
NewComponent::PythonScript*
Find
Choose Snippet
_CreateLineGeometry
_CreateStepProperty
_CreateTriangleMesh
_CsvRead
_CsvWrite
You can search the snippet name if you know it already
Or you can scroll through the available snippets and then find it

```
from vcScript import *
```

The snippet already has an example data set in it, that could be modified later:



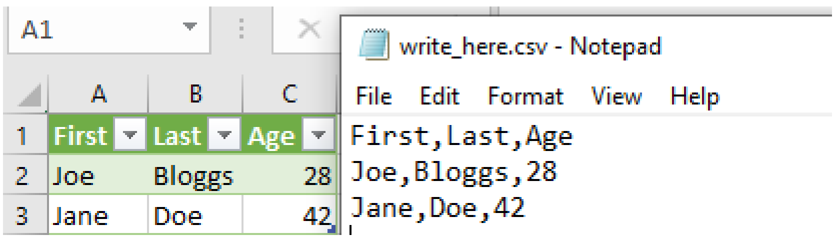
```
from vcScript import *



import csv
uri = r'C:\temp\test_table2.csv'
with open(uri, 'wb') as csv_file:
  csv_writer = csv.writer(csv_file, delimiter = ',')
  csv_writer.writerow(['First', 'Last', 'Age'])
  csv_writer.writerow(['Joe', 'Bloggs', 28])
  csv_writer.writerow(['Jane', 'Doe', 42])
```

NewComponent::PythonScript*
Find
Default path for the saved csv file
Sample data set

Once you compile the code, a CSV file will be generated with the given name in the given directory; in this example, both the file name and the file directory were changed. And here is how the CSV file looks if you open it in a Notepad or an Excel file:



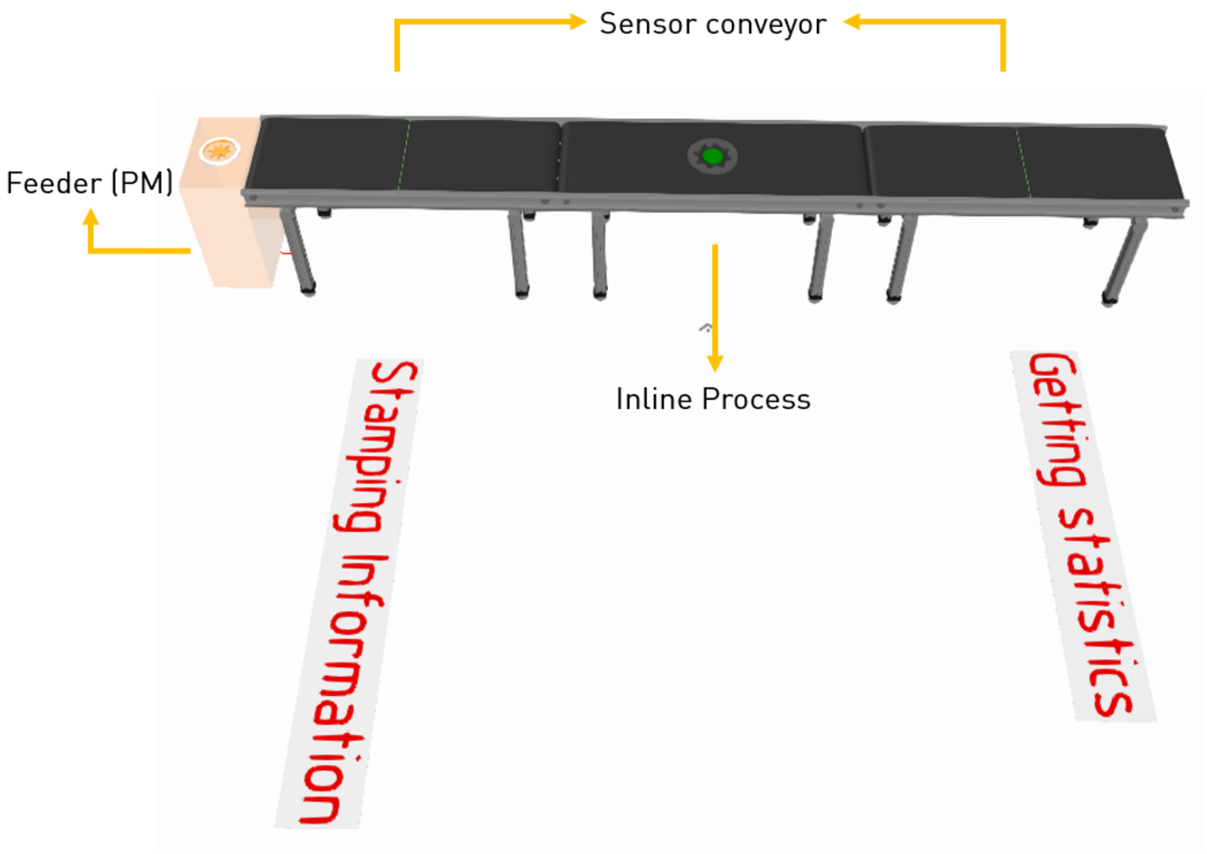
	A	B	C
1	First	Last	Age
2	Joe	Bloggs	28
3	Jane	Doe	42

```
First,Last,Age
Joe,Bloggs,28
Jane,Doe,42
```

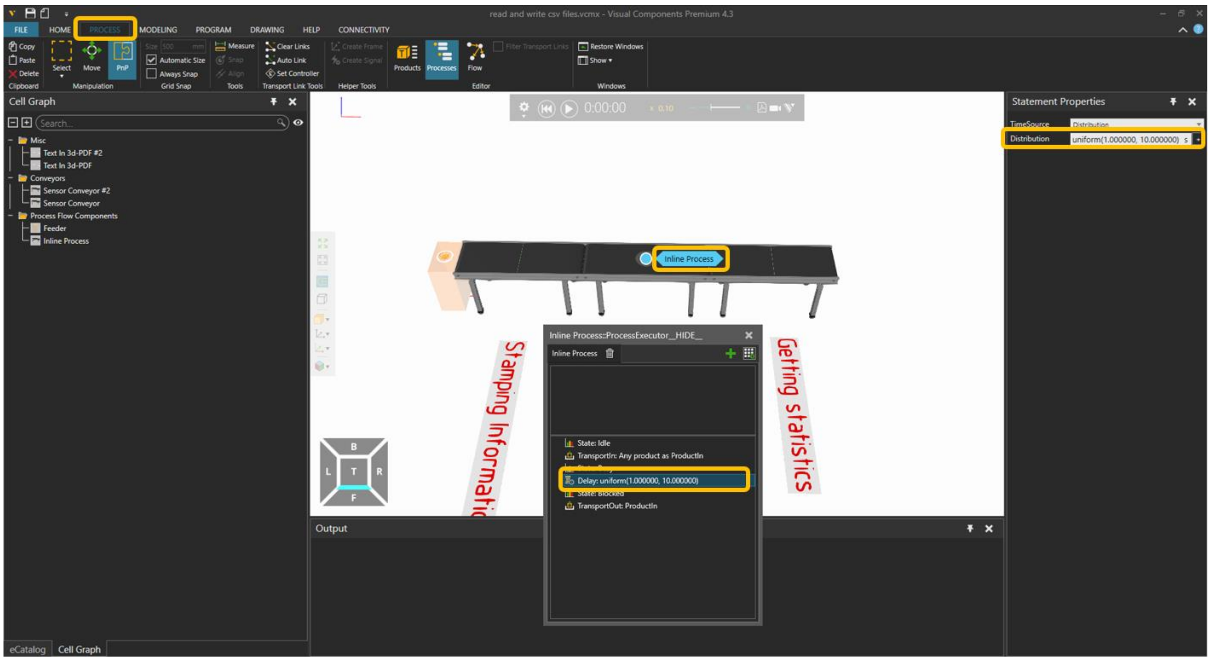
Some use cases for reading and writing CSV files

Stamping information from a CSV file to products

Consider the following example where there is a feeder, two sensor conveyors, and an inline process (get the components from the eCat and connect them as shown in the image below):



In the Inline Process, delete the existing delay statement and create a new one with a Uniform Distribution between 1 and 10:



Statement Properties

TimeSource: Distribution

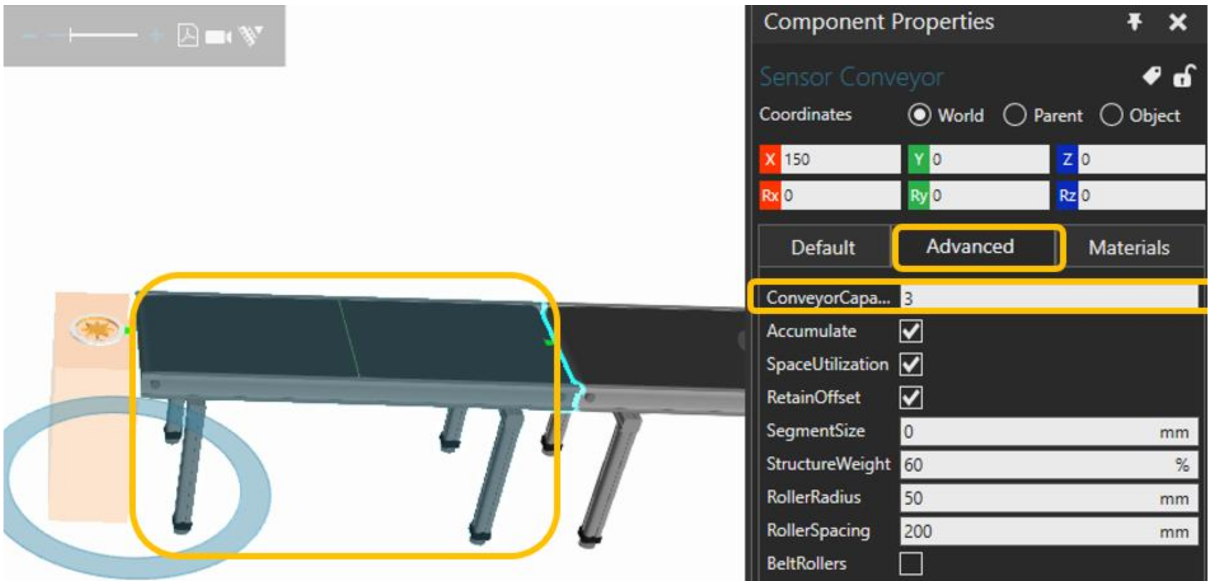
Distribution: uniform(1.000000, 10.000000) s

Inline Process::ProcessExecutor__HIDE__

Inline Process

State: Idle
TransportIn: Any product as ProductIn
State: Busy
Delay: uniform(1.000000, 10.000000)
State: Blocked
TransportOut: ProductIn

Now, put the ConveyorCapacity for the first Sensor Conveyor to 3 for better visual purposes:

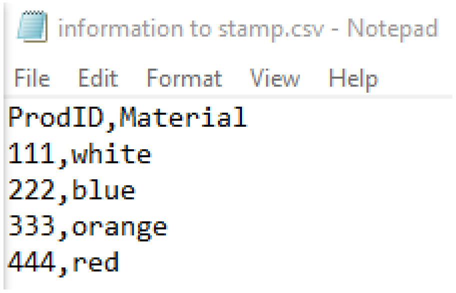


Coordinates (● World ○ Parent ○ Object)

X 150	Y 0	Z 0
Rx 0	Ry 0	Rz 0

ConveyorCapa...	3	
Accumulate	✓	
SpaceUtilization	✓	
RetainOffset	✓	
SegmentSize	0	mm
StructureWeight	60	%
RollerRadius	50	mm
RollerSpacing	200	mm
BeltRollers	☐	

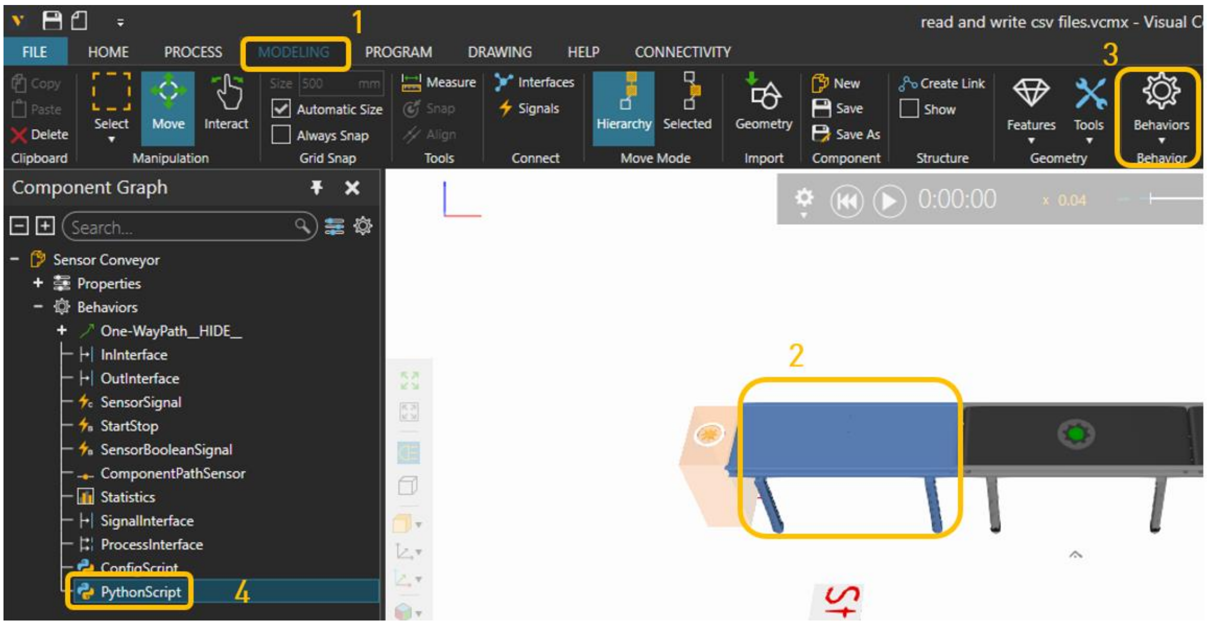
When the products are coming in, we want to stamp some information on to them; we want to give them product IDs and assign materials to them. We also want to stamp the simulation time to the products when they trigger the sensor. So, let's prepare the ProdID and Material data in a CSV format as following:


information to stamp.csv - Notepad
File Edit Format View Help

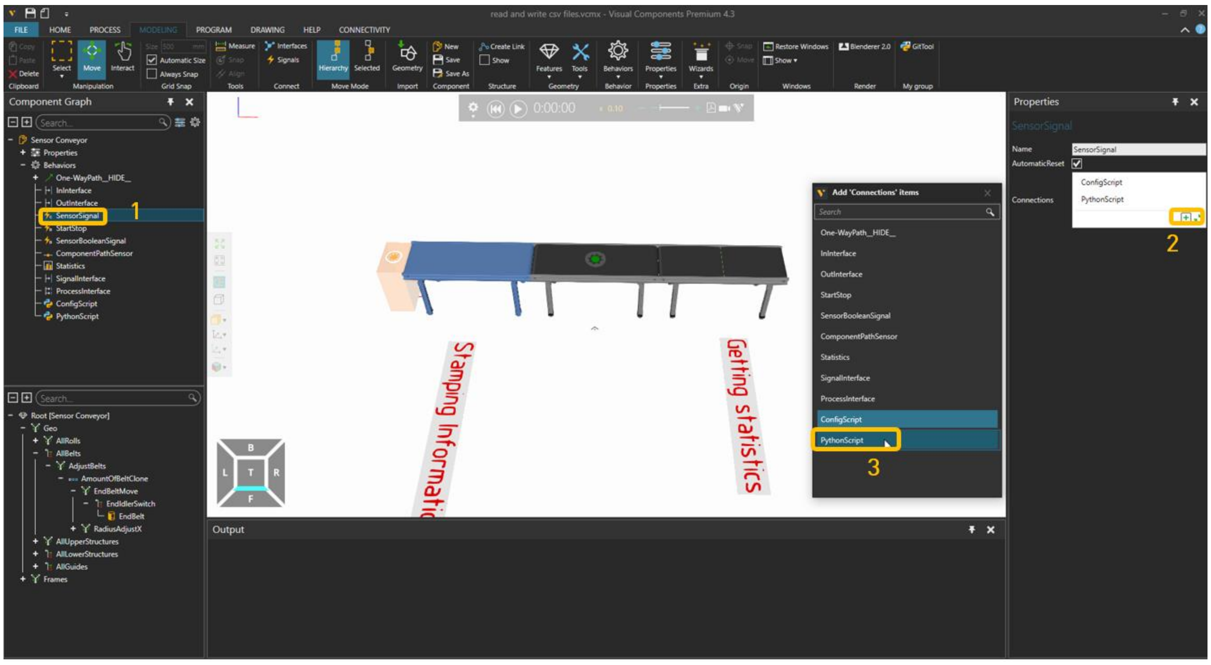
```
ProdID,Material
111,white
222,blue
333,orange
444,red
```

Remember what you name the CSV file and where you save it because you need to use its path in the python script.

Now we need to tell the first sensor conveyor how the information is going to be stamped; we do that with a python script. In our python script, we want to react when the sensors are triggered. Therefore we need to connect our signals with our python scripts. Switch to the Modeling tab, select the Sensor Conveyor, and then create a Python behavior:



To connect SensorSignal to the PythonScript, select the SensorSignal behavior, and add the PythonScript to the Connections:



Once you have done that, delete all the content inside of the PythonScript and copy/paste the following code in there; **remember to modify the uri according to your file name and location:**

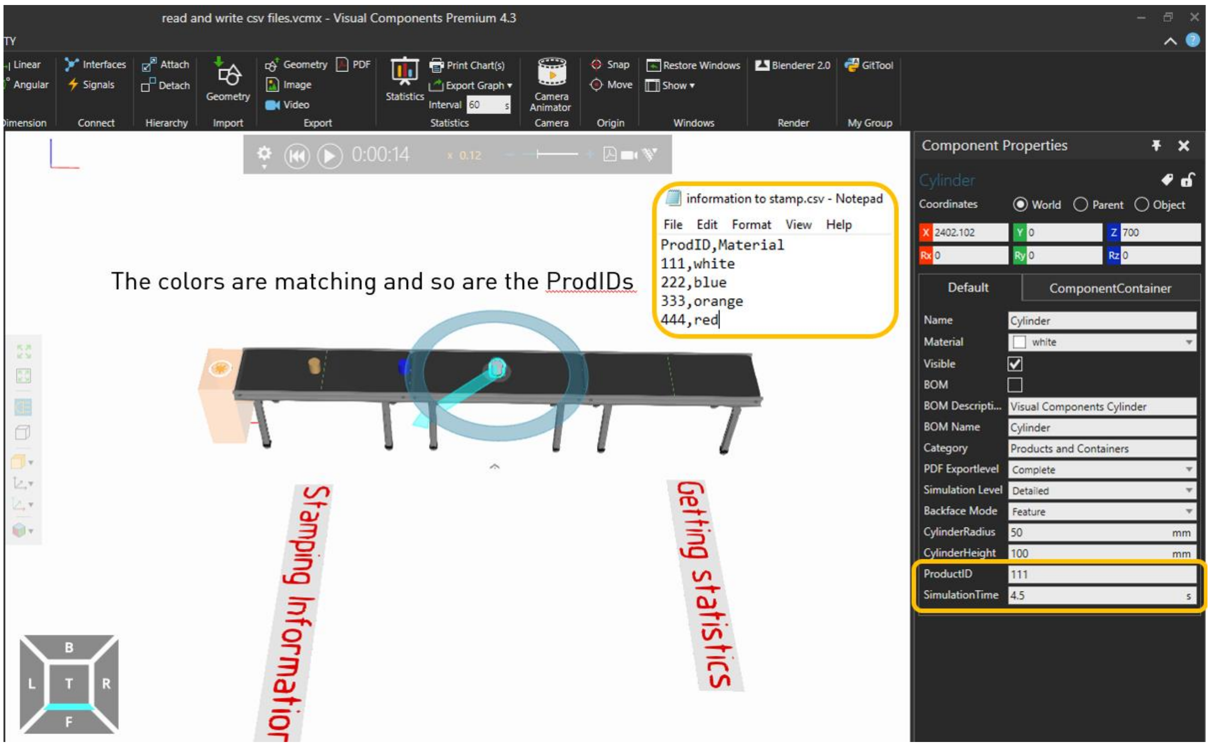
```
from vcScript import *
import csv

app = getApplication()
comp = getComponent()
sensor_signal = comp.findBehaviour('SensorSignal')

def OnStart():
  global product_id_csv,material_csv, parts_stamped
  parts_stamped = 0
  product_id_csv, material_csv = [],[]
  uri = r'…'
  with open(uri) as csv_file: # reading the csv content and storing them 
into arrays
    csv_reader = csv.reader(csv_file, delimiter = ',')
    next(csv_reader) # remove the first row with no actual data
    for row in csv_reader:
      product_id_csv.append(row[0])
      material_csv.append(row[1])

def OnSignal( signal ):
  global parts_stamped
  if signal.Name == "SensorSignal" and sensor_signal.Value:
    part = sensor_signal.Value
    product_id = part.getProperty("ProductID")
    material = part.getProperty("Material")
    sim_time = part.getProperty("SimulationTime")
    material = part.createProperty(VC_MATERIAL,"Material")
    if not product_id:
      product_id = part.createProperty(VC_STRING,"ProductID")
    if not sim_time:
      sim_time = part.createProperty(VC_REAL,"SimulationTime")
    if parts_stamped < len(product_id_csv):
      sim_time.Value = app.Simulation.SimTime
      product_id.Value = product_id_csv[parts_stamped]
      material.Value = material_csv[parts_stamped]
      parts_stamped += 1
```

Now if you compile the script and play the simulation, you should see after some time that the first four parts have different Materials, different product IDs, and also a unique simulation time. You can check those by pausing the simulation and selecting each component:


read and write csv files.vcmx - Visual Components Premium 4.3
0:00:14
Stamping Information
Getting statistics

The colors are matching and so are the ProdIDs

information to stamp.csv - Notepad

```
ProdID,Material
111,white
222,blue
333,orange
444,red
```

Component Properties

Cylinder

Coordinates: (● World) Parent Object

X 2402.102	Y 0	Z 700
Rx 0	Ry 0	Rz 0

Default | ComponentContainer

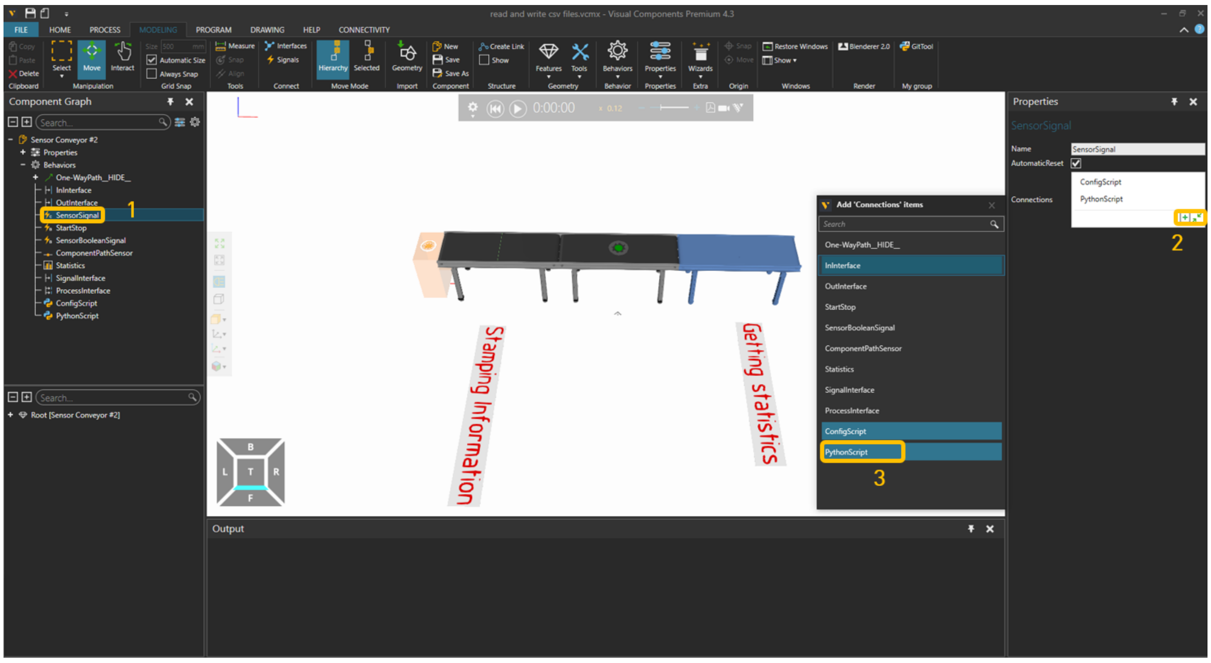
Name	Cylinder
Material	white
Visible	✓
BOM	☐
BOM Descripti...	Visual Components Cylinder
BOM Name	Cylinder
Category	Products and Containers
PDF Exportlevel	Complete
Simulation Level	Detailed
Backface Mode	Feature
CylinderRadius	50 mm
CylinderHeight	100 mm
ProductID	111
SimulationTime	4.5 s

Writing statistics to a CSV file

Now that we have successfully read the products' information from our CSV file and stamped them, let us extract the same information along with *pass through time and write them into a CSV file.

*Pass-through could be described as the difference between the time that one product triggers the first sensor and the time the same part triggers the other sensor.

To extract the statistics from the simulation and write them to a CSV file, create a python behavior for the second Sensor Conveyor. Let's also connect this PythonScript to the SensorSignal in the second Sensor Conveyor:



Let's now copy/paste the following code in PythonScript in a similar way as before.
Remember to modify the uri according to your file name and its location:

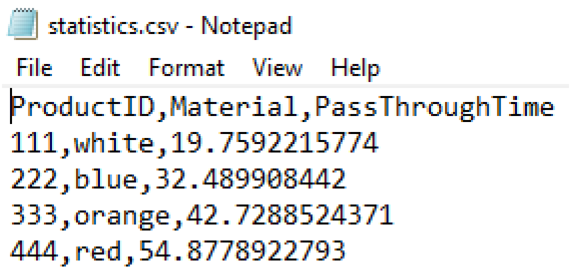
```
from vcScript import *
import csv

app = getApplication()
comp = getComponent()
sensor_signal = comp.findBehaviour('SensorSignal')
product_id, material,pass_through_time = None, None, None
uri = r'C:\Users\MoradSh1\Documents\Visual Components\4.3\Academy
tutorial\read and write csv files\statistics.csv'

def OnStart():
  with open(uri, 'wb') as csv_file:
    csv_writer = csv.writer(csv_file, delimiter = ',')
    csv_writer.writerow(['ProductID', 'Material',
'PassThroughTime'])

def OnSignal( signal ):
  global product_id,material, pass_through_time
  if signal.Name == "SensorSignal" and sensor_signal.Value:
    part = sensor_signal.Value
    if part.getProperty("SimulationTime") and
part.getProperty("ProductID"):
      sim_time1 = part.getProperty("SimulationTime").Value
      sim_time2 = app.Simulation.SimTime
      pass_through_time = sim_time2 - sim_time1
      product_id = part.getProperty("ProductID").Value
      material = part.getProperty("Material").Value.Name
      with open(uri, 'a') as csv_file:
        csv_writer = csv.writer(csv_file, delimiter =
',',lineterminator='\n')
        csv_writer.writerow([product_id, material,
pass_through_time])
```

Now if you compile the script and play the simulation and wait until the first four products go past the Inline Process and stop the simulation, there should be a CSV file generated based on your given name and path:



```
ProductID,Material,PassThroughTime
111,white,19.7592215774
222,blue,32.489908442
333,orange,42.7288524371
444,red,54.8778922793
```

If you let the simulation play for longer, there will be more lines written in the CSV file. Checking the information in the generated CSV file and comparing it with the given inputs and simulation visuals, we have now successfully extracted information from our simulation and saved them into a CSV file.

You can find the complete layout for this tutorial attached to this document. Remember when you open the layout, you get an error about the uri of the CSV file. So remember to modify it.