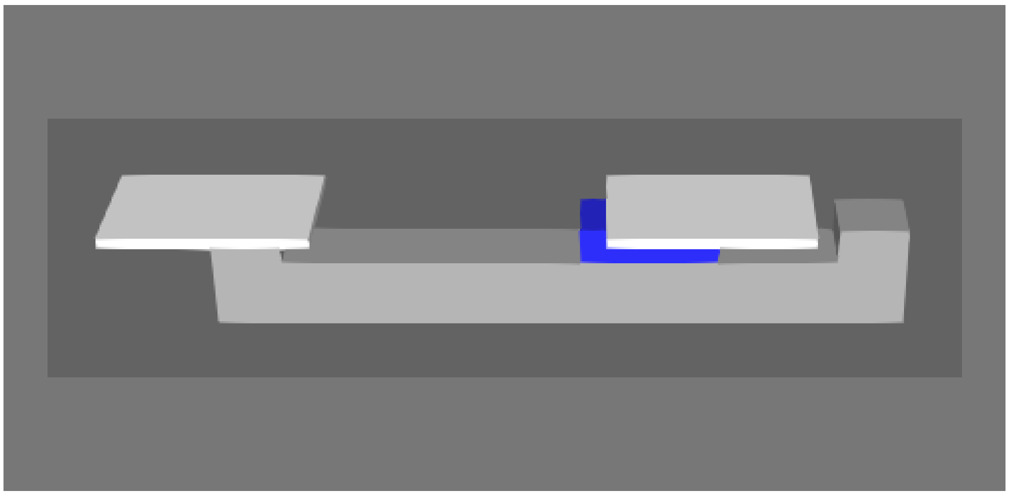
Conditions and Signals

Visual Components 4.0 | Version: February 24, 2017



When using Python API to write application and component scripts, you may need to define conditions. A condition allows you to make a decision and control the flow of execution in a script. For example, you may want a robot to execute a process if a condition is exists, otherwise the robot would either wait or perform a different process.

Conditions can be tested immediately or triggered by an object or event. For example, signals can be used to evaluate a condition and stop or resume the main function of a script.

In this tutorial you learn how use conditions, events and triggers to control the execution of scripts.

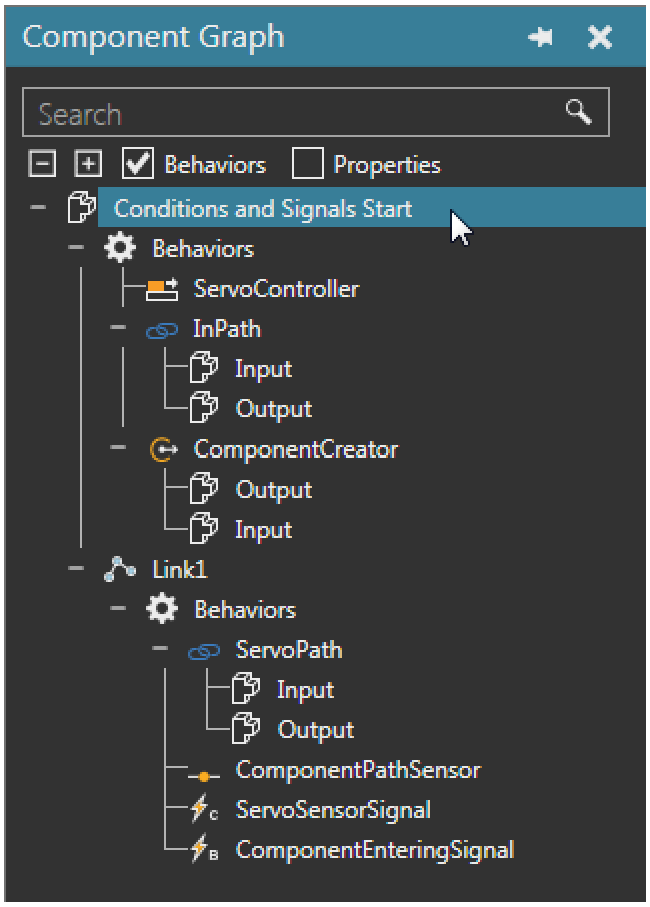
Support
support@visualcomponents.com

Community
community.visualcomponents.net

Getting Started

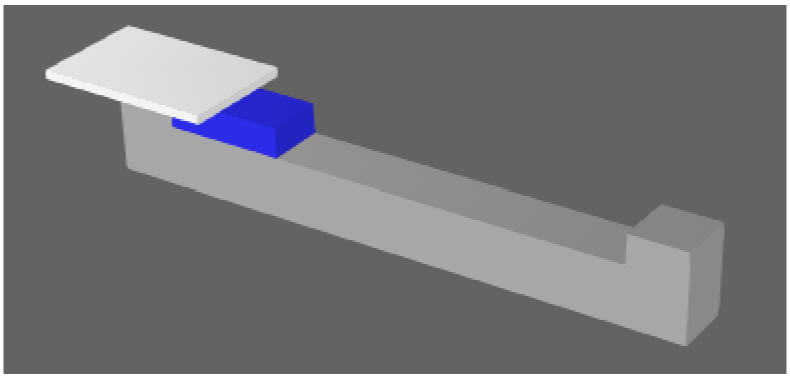
1. Open the **ConditionsAndSignalsStart.vcmx** file for this tutorial.

2. Click the **Modeling** tab, and then in the Component Graph panel, select the **Behaviors** check box, and then expand the component node tree.



```
Component Graph
Search
[–] [+] [✓] Behaviors [ ] Properties
– Conditions and Signals Start
  – Behaviors
      ServoController
    – InPath
        Input
        Output
    – ComponentCreator
        Output
        Input
  – Link1
    – Behaviors
      – ServoPath
          Input
          Output
        ComponentPathSensor
        ServoSensorSignal
        ComponentEnteringSignal
```

3. Run the simulation, verify the component creates parts that move along a path, and then reset the simulation.



The component has a link to support the moving of parts from one end to the other via a platform. All you need to do is add the logic for moving the platform when parts reach a sensor.

4. In the Component Graph panel, select the **root node**, and then add a **Python Script** behavior. The script editor will automatically open when you add the behavior.

5. In the script editor, create variables for the servo controller in the root node, and the path and two signals in the Link1 node.

```
from vcScript import *


comp = getComponent()

servo = comp.findBehaviour("ServoController")

servo_path = comp.findBehaviour("ServoPath")

servo_sensor_signal = comp.findBehaviour("ServoSensorSignal")

part_entering_signal = comp.findBehaviour("ComponentEnteringSignal")
```

NOTE! The ServoSensorSignal is a Component Signal behavior used to identify what component triggers a sensor in ServoPath. The ComponentEnteringSignal is a Boolean Signal behavior used as a transition signal to identify when a component enters and leaves ServoPath.

6. In the OnRun event, use a conditional if statement to make a decision: if the value of the sensor signal is not None, move the first joint assigned to the servo controller to 600mm, else print a message that there is no component at the sensor.

```
def OnRun():
  if servo_sensor_signal.Value != None:
    servo.moveJoint(0,600.0)
  else:
    print "Component not at sensor"
```

7. Compile the code, and then run the simulation. The platform will not move because the script evaluates the condition of the if statement immediately and before a part triggers the sensor. As a result, the condition of the if statement is False. Another reason why the platform does not move is because the script does not know if the value of the signal has changed from None to a vcComponent object.

8. Reset the simulation.

Conditions

In most cases, the first line of code in a new Python Script behavior imports all methods from the vcScript module. Two methods in vcScript that can be used to control the execution of a script are condition() and triggerCondition().

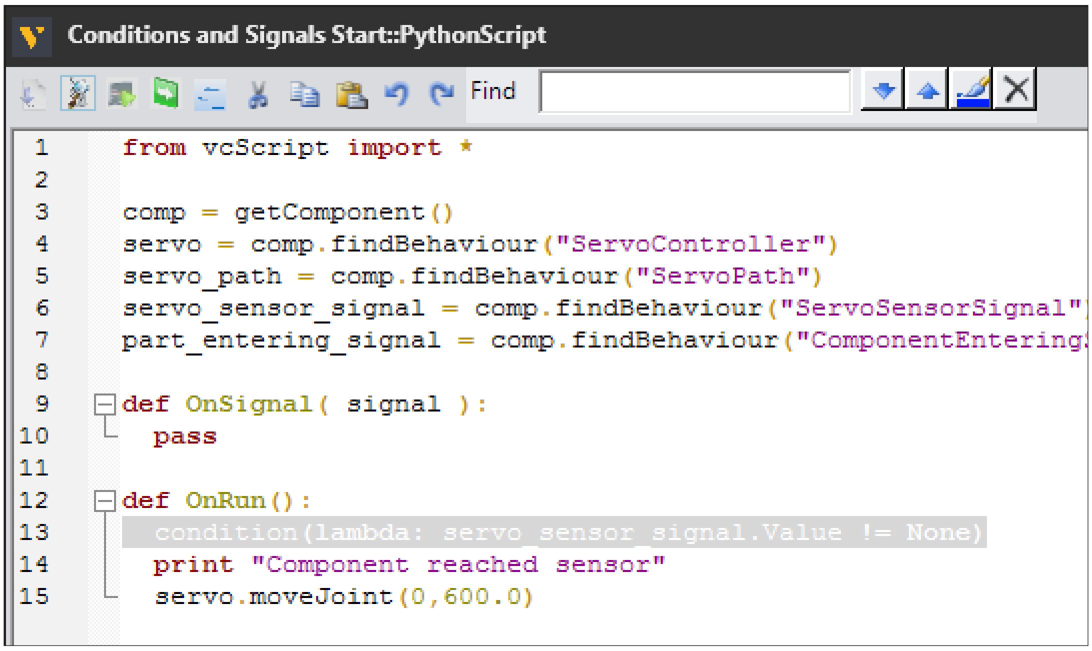
The condition() method allows you to delay the execution of a script until a called function returns a True value. The evaluation of that condition is immediate and continuous. That is, the method will continue to call the function until it returns a True value. The function can be defined in the same script or inlined by using a lambda operator.

1. In the script editor, OnRun event, replace the if statement with a condition() method.

```
def OnRun():
  condition(lambda: servo_sensor_signal.Value != None)
  print "Component reached sensor"
  servo.moveJoint(0,600.0)
```

2. Compile the code, and then enabled **Trace Execution**. This will allow you to know what line of code is being executed in the script.

3. Run the simulation. The platform will not move because the function of the condition never returns a True value. The script has no way of knowing if the value of the sensor signal has changed during the simulation. One way for a signal to communicate its value to other behaviors is to call its signal() method.



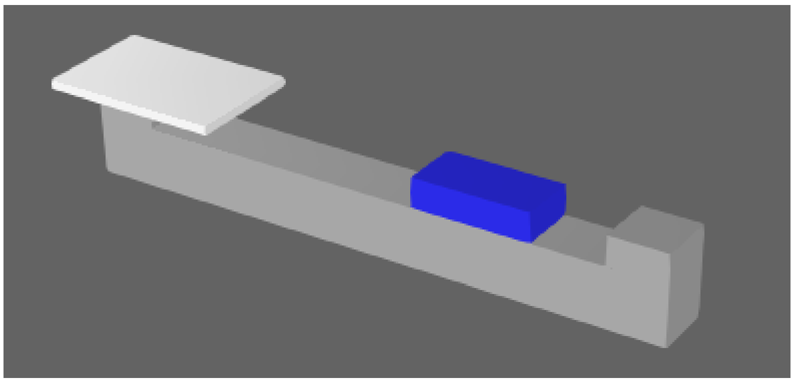
```
from vcScript import *

comp = getComponent()
servo = comp.findBehaviour("ServoController")
servo_path = comp.findBehaviour("ServoPath")
servo_sensor_signal = comp.findBehaviour("ServoSensorSignal"
part_entering_signal = comp.findBehaviour("ComponentEntering

def OnSignal( signal ):
  pass

def OnRun():
  condition(lambda: servo_sensor_signal.Value != None)
  print "Component reached sensor"
  servo.moveJoint(0,600.0)
```

4. Reset the simulation.

5. In the script editor, OnRun event, signal the value of the sensor signal. The value of the signal would still be None, so pass the component of the script as an optional argument for testing purposes.

```
def OnRun():
  servo_sensor_signal.signal(comp)
  condition(lambda: servo_sensor_signal.Value != None)
  print "Component reached sensor"
  servo.moveJoint(0,600.0)
```

6. Compile the code, and then run the simulation. The platform will move but not wait for a part to trigger the sensor.

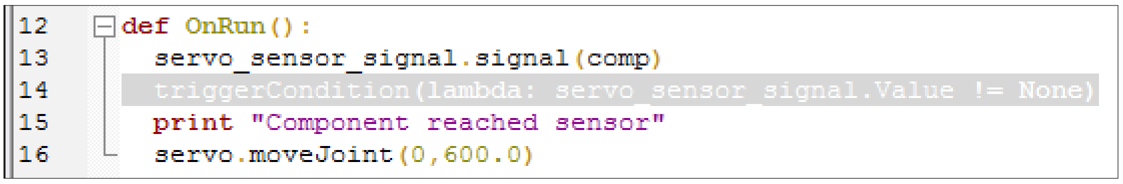


7. Reset the simulation.

The **triggerCondition()** method is like the condition() method, but evaluates its condition when triggered by a signal or other behavior connected to the script.

8. In the script editor, OnRun event, replace condition with triggerCondition, and then compile the code.

9. Run the simulation. The platform will not move because the condition is never evaluated because the script currently has no triggers.



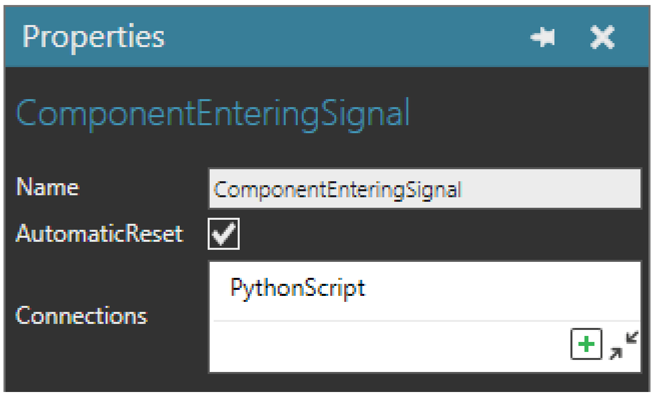
```
def OnRun():
  servo_sensor_signal.signal(comp)
  triggerCondition(lambda: servo_sensor_signal.Value != None)
  print "Component reached sensor"
  servo.moveJoint(0,600.0)
```

10. Reset the simulation.

Triggers

A trigger can be any signal or behavior connected to a Python Script behavior.

1. In the Component Graph panel, select the **ComponentEnteringSignal** behavior, and then add the **Python Script** behavior as a connection.



Properties

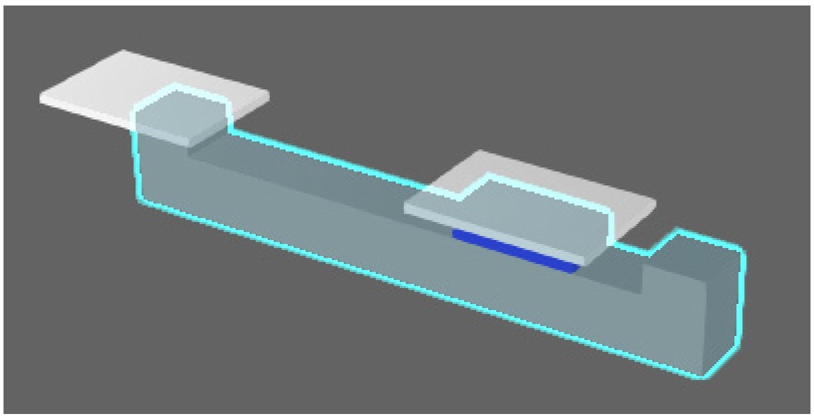
ComponentEnteringSignal

Name: ComponentEnteringSignal

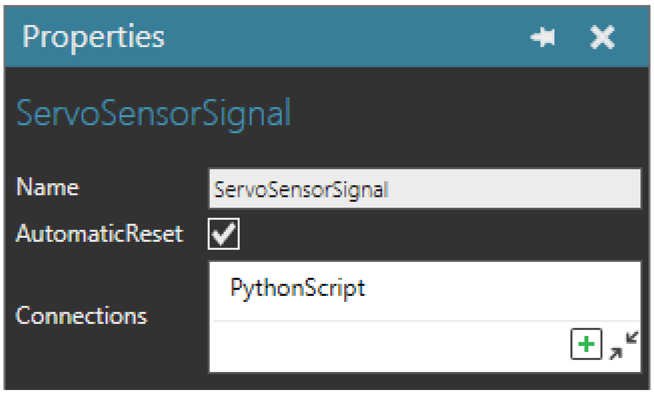
AutomaticReset: ✓

Connections: PythonScript

2. Run the simulation, verify the platform moves, and then reset the simulation. The platform did not move until a part transitioned into the servo path, thereby the path signal triggered the evaluation of the condition in the script. The platform, however, moved before the part reached the sensor.



3. Reset the simulation, and then connect the **Python Script** to the **ServoSensorSignal** behavior.



Properties

ServoSensorSignal

Name: ServoSensorSignal

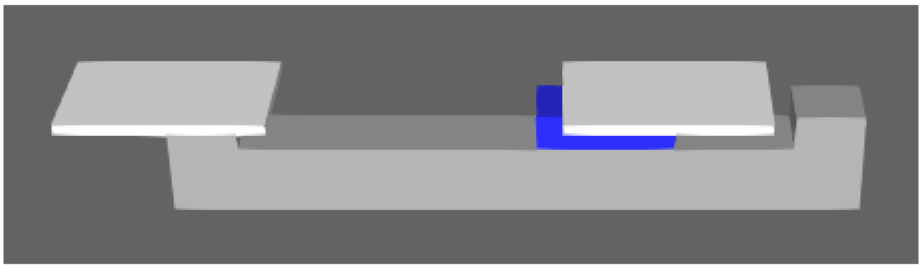
AutomaticReset: ✓

Connections: PythonScript

4. In the script editor, OnRun event, comment the line that signals the value of the sensor signal, and then compile the code. That line of code is now redundant since the script is connected to the signal and receives its signal events.

```
def OnRun():
  #servo_sensor_signal.signal(comp)
  triggerCondition(lambda: servo_sensor_signal.Value != None)
  print "Component reached sensor"
  servo.moveJoint(0,600.0)
```

5. Run the simulation, verify the platform starts to move when a part triggers the sensor, and then reset the simulation.



You can use the **getTrigger()** method in vcScript to evaluate the last recorded trigger of a script. This is helpful if you have a script that is connected to many signals and you need to specify which trigger is required for a condition to be true.

6. In the script editor, replace triggerCondition with condition and use the getTrigger() method to verify the connected sensor signal updates the script, and then compile the code.

```
def OnRun():
  #servo_sensor_signal.signal(comp)
  condition(lambda: getTrigger() == servo_sensor_signal)
  print "Component reached sensor"
  servo.moveJoint(0,600.0)
```

7. Run the simulation, verify the platform starts to move when a part triggers the sensor, and then reset the simulation.

Signal Events

Signals connected to a script can be used to control the execution of a script. For example, you can use the OnSignal event to listen for signal value changes during a simulation.

1. In the script editor, clear the OnRun event and add a pass statement, and then compile the code. This clears the history of OnRun event in the script.

```
def OnRun():
  pass
```

2. In the OnRun event, create a while loop that performs the following process:

   - Wait for a part to trigger the sensor, and then stop the path.
   - Move the platform to the other end of the component, and then restart the path.
   - Wait for the part to leave the platform, and then stop the path.
   - Move the platform back to the other end of the component, and then restart the path.

```
def OnRun():
  while True:
    triggerCondition(lambda: getTrigger()==servo_sensor_signal)
    servo_path.Enabled = False
    servo.moveJoint(0,600.0)
    servo_path.Enabled = True
    condition(lambda: part_entering_signal.Value == False)
    servo_path.Enabled = False
    servo.moveJoint(0,0.0)
    servo_path.Enabled = True
```

3. Compile the code, and then run the simulation to verify the platform moves parts back and forth, and then reset the simulation.

Instead of using the condition() and triggerCondition() methods, you can redefine the process to use signal events. For example, you can use signal events to suspend and resume the OnRun event, which is the main function of a Python Script.

4. In the script editor, modify the code so that the OnSignal and OnRun events work together to move parts, and then compile the code. One solution is to create two subprocesses for moving the platform and controlling its path.

```
def OnSignal( signal ):
  if signal == servo_sensor_signal:
    if signal.Value != None:
      resumeRun()
  elif signal == part_entering_signal:
    if signal.Value == False:
      resumeRun()


def OnRun():
  while True:
    suspendRun()
    servo_path.Enabled = False
    servo.moveJoint(0,600.0)
    servo_path.Enabled = True
    suspendRun()
    servo_path.Enabled = False
    servo.moveJoint(0,0.0)
    servo_path.Enabled = True
```

5. Run the simulation, verify the platform moves parts back and forth, and then reset the simulation.

Review

In this tutorial you learned how to create conditions in a component script for executing processes. You know how to connect and use signals as triggers for evaluating conditions and how signal events can control the main function of a script. You also know how to use a lambda operator to inline expressions.