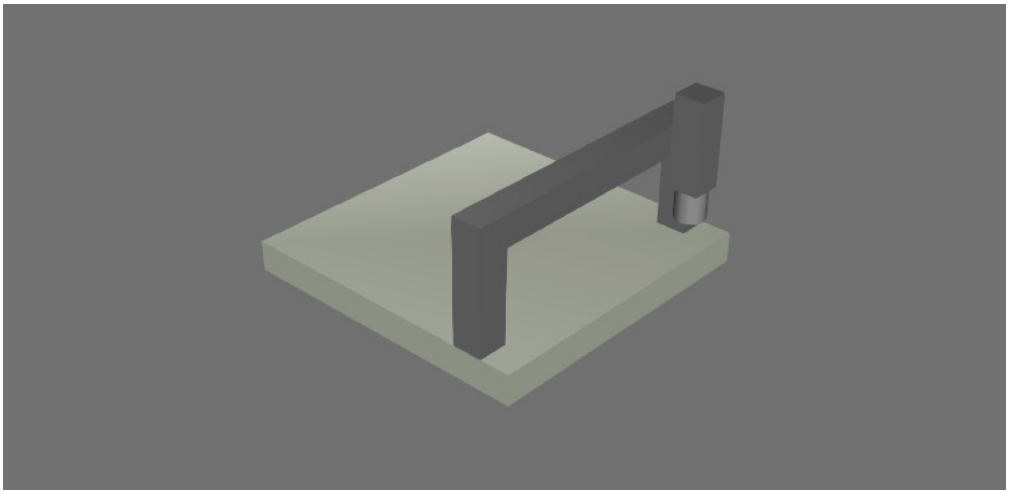
Event Handlers

Visual Components 4.0 | Version: February 24, 2017



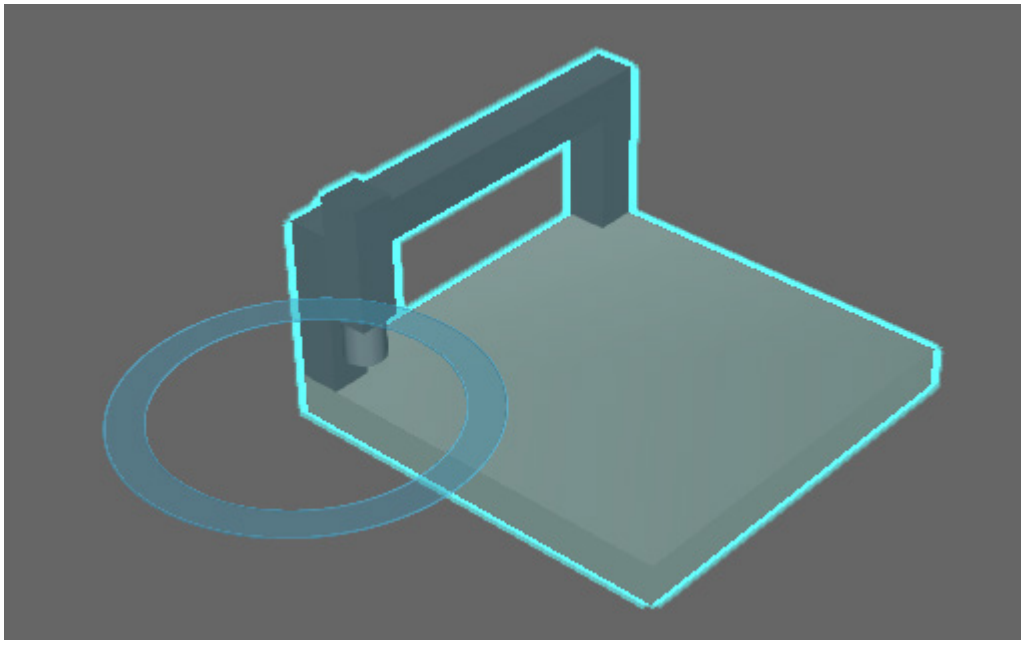
When using Python API to write application and component scripts, you may need to call functions in response to events. These are known as event handlers: functions in a script that are called in response to an object event, for example a change in a property value.

In this tutorial you learn how to:

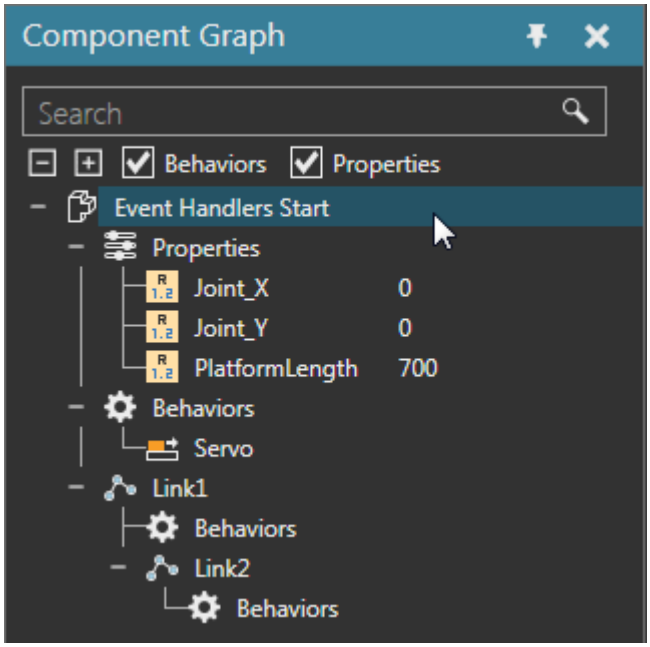
- Execute event handlers when a simulation is or is not running.
- Execute event handlers when property and signal values change.
- Execute event handlers using object event types, for example a servo controller heartbeat/pulse.

Getting Started

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



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



The component has two joints that can be driven by a servo controller. A component script can be used to handle events related to component properties, signals and the pulse of the servo controller.

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

Property Values

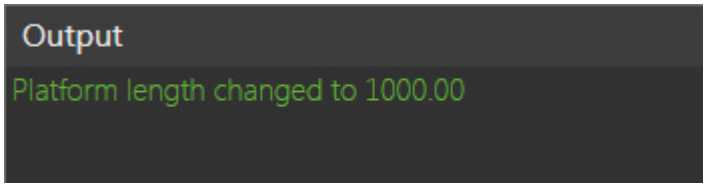
You can create a function and assign it to the OnChanged event of a vcProperty object. Whenever the value of that property changes, the function is called to handle the event.

1. In the script editor, define a function that prints the value of the PlatformLength component property, and then assign that function to the OnChanged event of that property.

```
from vcScript import *


def LengthChange(property):
  print "Platform length changed to %.2f" %property.Value


comp = getComponent()
platform_length = comp.getProperty("PlatformLength")
platform_length.OnChanged = LengthChange
```

2. Compile the code, and then in the Component Graph panel, select the **root node**.

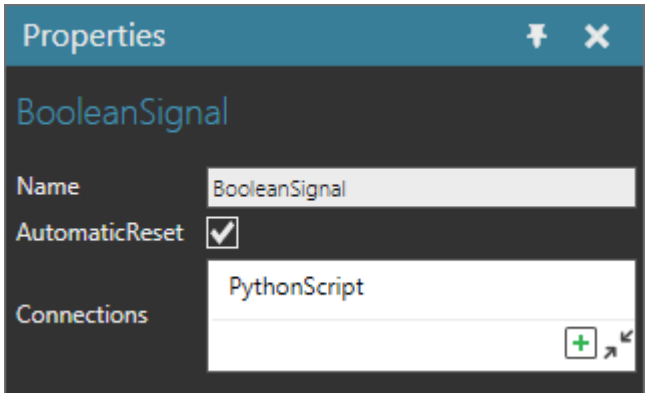
3. In the Component Properties panel, set PlatformLength to **1000**, and then verify feedback was printed in the Output panel about the property's new value.



Signal Values

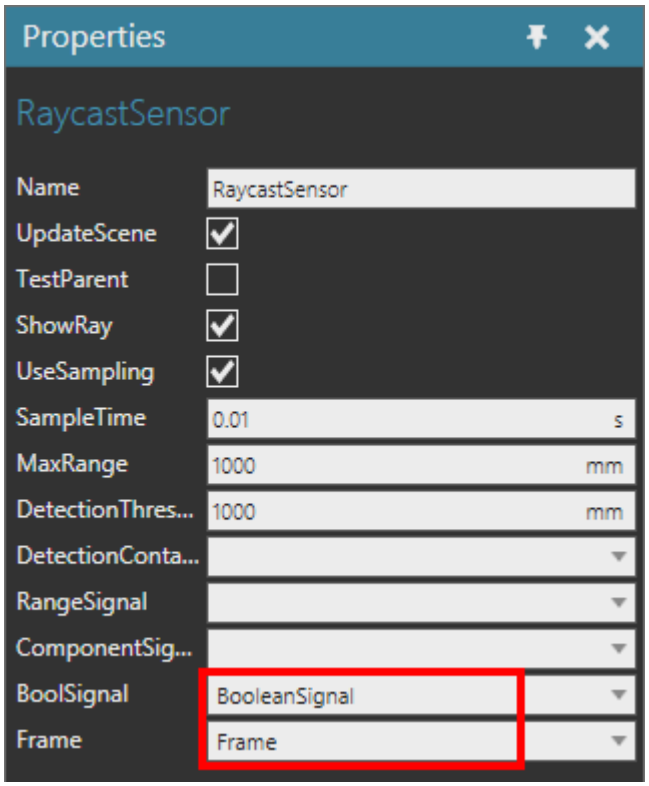
You can create a function and assign it to the OnValueChange event of a vcSignal object. Whenever the value of that signal changes, the function is called to handle the event.

1. Add a **Boolean Signal** behavior, and then connect it to the **Python Script**.



2. Add a **Frame** feature, and then snap the frame to the top face center of the support base. In the Parent coordinate system, the XYZ values are {500,500,100}.

3. Add a **Raycast Sensor** behavior, and then set its Frame and BoolSignal properties to the Frame feature and Boolean Signal behavior created in steps 1 and 2.



A sensor is now located at the top face center of the support base and connected to a signal. The signal value is changed every time the sensor is triggered during a simulation.

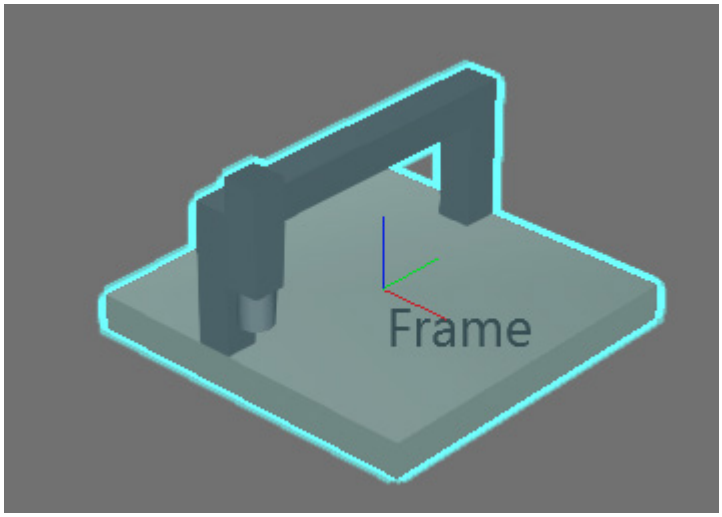
4. In the script editor, OnRun event, create a while loop that moves the first joint assigned to the servo controller to 800mm and then to 0mm with a five second delay in between each movement.

```
def OnRun():
  servo = comp.findBehaviour("Servo")
  while True:
    servo.moveJoint(0,800.0)
    delay(5)
    servo.moveJoint(0,0.0)
```

5. Define a function that prints the joint values of the component when its sensor is triggered, and then assign that function to the OnValueChange event of the sensor signal.

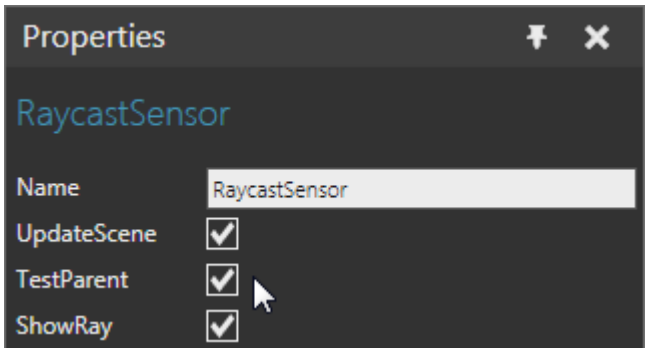
```
def JointValues(signal):
  jointX = comp.Joint_X  #this returns the property value not the vcProperty
  jointY = comp.Joint_Y
  print "Joint X is %s and Joint Y is %s" %(jointX,jointY)


comp = getComponent()
...
signal = comp.findBehaviour("BooleanSignal")
signal.OnValueChange = JointValues
```

6. Compile the code, and then run the simulation. The sensor will most likely not detect the geometry of the links because it is not set to test internal collisions. It may also be necessary to elevate the sensor above the platform base.

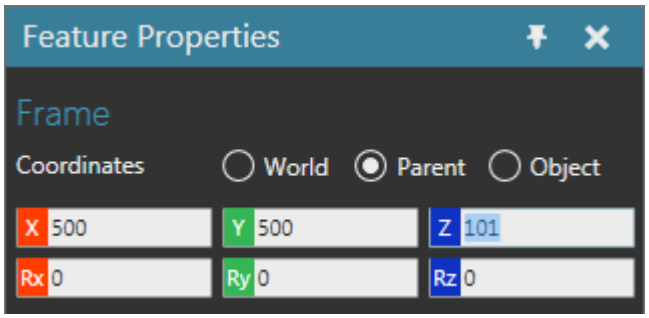


7. Reset the simulation.

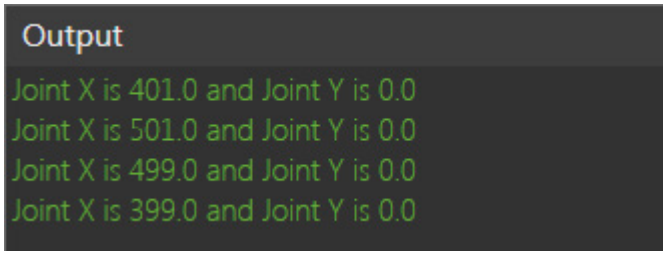
8. In the Component Graph panel, select the **Raycast Sensor**, and then in the Properties panel, select the **TestParent** check box.



9. In the Component Graph panel, select the **Frame** feature, and then in the Feature Properties panel, add **1** to the Z-axis value so that the sensor is technically not touching the platform base, which could prematurely trigger the sensor.



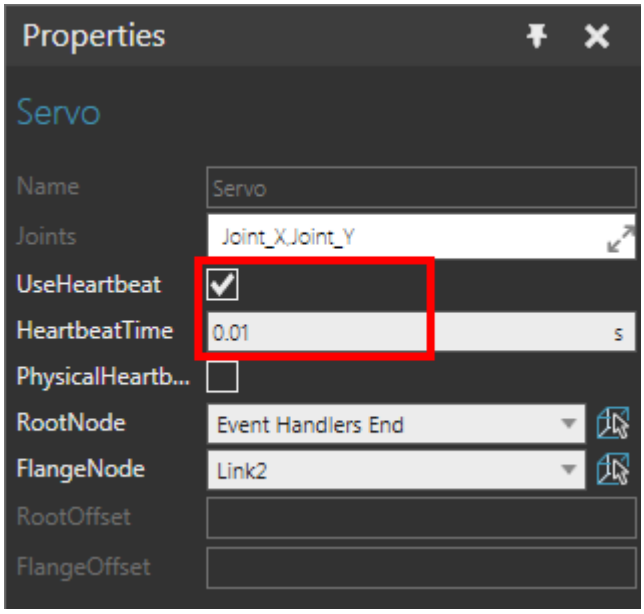
10. Run the simulation, verify the sensor detects the geometry of the links during each pass and signals the joint values, and then reset the simulation.



Heartbeats and States

You can create a function and assign it to the OnHeartbeat event of a vcServoController object. A servo controller can operate in pulse mode, which is also known as heartbeat. In this mode, the servo controller can indicate its current state for every pulse or beat. This is also true for robot controllers.

1. In the Component Graph panel, select the **Servo**, and then in the Properties panel, select the **UseHeartbeat** check box and set HeartbeatTime to **0.01** second.



2. In the script editor, create a dictionary in which a key-value pair is a valid state of the controller and its descriptive label.

```
typemap = {
VC_CONTROLLER_START:"Start",
VC_CONTROLLER_RUNNING:"Running",
VC_CONTROLLER_END:"End",
VC_CONTROLLER_STOP:"Stop"
}
```

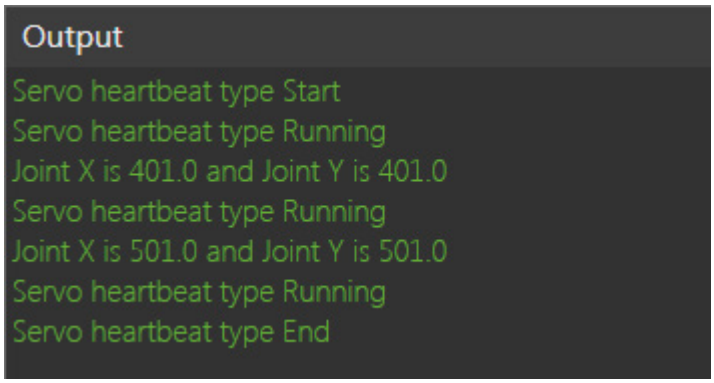
NOTE! You do not need to use all states of the controller.

3. Define a function that prints the state of the servo controller at each heartbeat, and then assign that function to the OnHeartbeat event of the servo controller.

4. In the OnRun event, edit the while loop to move both joints from 0mm to 800mm and back to 0mm, and then compile the code.

```
def HeartbeatState(servo,type):
  print "Servo heartbeat type %s" %typemap[type]
...
def OnRun():
  servo = comp.findBehaviour("Servo")
  servo.OnHeartbeat = HeartbeatState
  while True:
    servo.move(800.0,800.0)
    delay(5)
    servo.move(0.0,0.0)
```

5. Run the simulation, verify feedback is printed in the Output panel about the states of the servo controller, and then reset the simulation.



Review

In this tutorial you learned how to create event handlers for properties, signals and controllers in scripts. An event handler is a function that is executed in response to its event. An event can be triggered during a simulation, for example when a robot sends signals to other components. An event can also be triggered when a simulation is in its initial state, for example when a user changes the properties of a component. In either case, you know how to define functions and assign them to event listeners of objects. In a broader sense, you know how to use Python API to implement event-driven programming in application and component scripts.