MediaMatrix®

A Division of Peavey Electronics Corporation

python™

Python Developer's Guide

Version 2.0.0.0

July 7, 2017

Copyright notice

Scope

This guide describes how to use the Python programming language to customize the control interfaces of NWare projects.

This guide has been written for users who already understand the basic concepts of a programming language, but want to know more about using Python with NWare and MediaMatrix hardware devices.

This guide does not include a full Python language reference. To view a full language reference, refer to the *language reference for Python 2.4 on the Python website* (*http://docs.python.org/release/2.4/*).

Contents

Chapter 1
Introduction

In This Chapter

What is Python?

Python is a widely used, high-level programming language. It is designed to be clear to read, and its syntax allows programmers to express concepts in fewer lines than in other languages, such as C++ or Java. Python is used by many well-known organizations, including Google and Yahoo, as part of their search engine technology, and Industrial Light and Magic, in the production of special effects.

Here are some useful Python resources:

- *http://www.python.org/* (*http://www.python.org/*)
- *https://www.youtube.com/watch?v=dYRJJ_w3lNY* (*https://www.youtube.com/watch?v=dYRJJ_w3lNY*)
- *https://www.youtube.com/results?search_query=python+tutorial* (*https://www.youtube.com/results?search_query=python+tutorial*)
- *http://www.learnpython.org/* (*http://www.learnpython.org/*)
- *https://www.codecademy.com/* (*https://www.codecademy.com/*).

Installing Python on your PC

In order to try out some basic examples to introduce yourself to Python, we recommend that you install Python on your PC. That way, you can try out some code before you switch to NWare and start using Python in a project.

Note:

- We recommend that you use Python 2.4, as this is the version supported by MediaMatrix. If you install a different version, the list of supported commands and syntax may be different.
- You do not need to install Python on your PC to use Python in an NWare project. However, it is useful for test purposes.

⏩ *To install Python*

1. Open a browser and *download Python 2.4* (*https://www.python.org/ftp/python/2.4/python-2.4.msi*).

   Information on the release can be found here: *https://www.python.org/download/releases/2.4/* (*https://www.python.org/download/releases/2.4/*).
2. Run the msi installer and follow the instructions to install Python 2.4.

   A *Python 2.4* folder will be added to the Start menu.

Basic Python examples

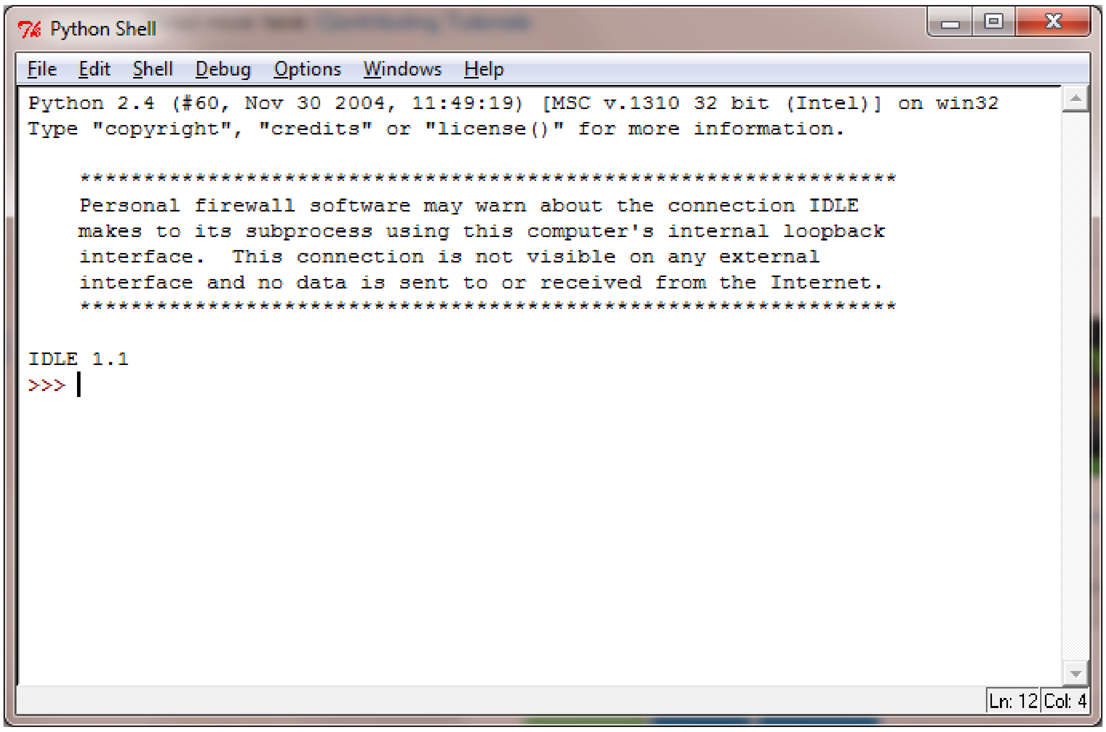
Before you start using Python in NWare, try out some basic examples in the Python shell. When you are familiar with how these examples work, we recommend looking online for others, so you are comfortable with the syntax of a broader range of language concepts.

This section assumes you have installed Python 2.4.

Starting the Python shell

- On the Start menu, point to **All Programs/Python 2.4**, and then click **IDLE (Python GUI)**.

  The Python Shell is displayed.



```
Python 2.4 (#60, Nov 30 2004, 11:49:19) [MSC v.1310 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.

    ****************************************************************
    Personal firewall software may warn about the connection IDLE
    makes to its subprocess using this computer's internal loopback
    interface.  This connection is not visible on any external
    interface and no data is sent to or received from the Internet.
    ****************************************************************

IDLE 1.1
>>> 
```

Printing hello world

- At the >>> prompt, type:

  ```
  print "hello world!"
  ```

  The Python Shell will display the following message.

  ```
  hello world!
  ```

Using basic variables

1. Assign values to some variables:

   ```
   >>> one = 1
   >>> two = 2
   >>> three = one + two
   ```

```
>>> hello = "hello"
>>> world = "world"
>>> helloworld = hello + " " + world
```

2. Try printing the values of the variables:

```
>>> three
3

>>> helloworld
'hello world'
```

Tip: Assigning a value to a variable when it has not been referenced before will also instantiate it. You do not need to define the variable first.

Creating an array and a dictionary and printing values

1. Define a small array:

```
>>> phonelist = [938477566, 938377264,947662781]
```

2. Print out the contents of the array:

```
>>> print phonelist
[938477566, 938377264, 947662781]
```

3. Print out the value of one of the records in the array:

```
>>> print phonelist[0]
938477566
```

4. Define a small dictionary:

```
>>> phonebook = {"John" : 938477566, "Jack" : 938377264, "Jill" : 947662781}
```

5. Print out the contents of the dictionary:

```
>>> print phonebook
{'Jill': 947662781, 'John': 938477566, 'Jack': 938377264}
```

6. Print out the value of one of the records in the dictionary:

```
>>> print phonebook["John"]
938477566
```

Looping through a predefined list

1. Define a list containing numbers:

```
>>> primes = [2, 3, 5, 7]
```

2. Specify a loop to iterate over the list and display each member:

```
>>> for prime in primes:
        print prime
2
```

```
3

5

7
```

Note: Line indentation is very important. It specifies which statements are part of a block. In the example above, the *print* statement must be indented to be part of the *for* loop block. And if you want to include further statements in the loop block, they need to be given the *exact same level of indentation*. Without the correct level of indentation, statements will not be included in blocks in your code.

Looping using the xrange function

- Use iteration to print out the numbers 0 to 4:

```
>>> for x in xrange(5):

        print x

0

1

2

3

4
```

Tip: You can substitute the *range* statement for the *xrange* statement. However, xrange will normally be more efficient than range, especially if you are iterating over a large range of numbers.

Chapter 2

Getting started with Python in NWare

In This Chapter

Introduction

In MediaMatrix, Python can be used to add programmable logic to an NWare project. The logic can be triggered manually – by clicking a button, for example – or automatically,as soon as a project is deployed or emulated.

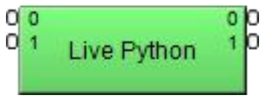
MediaMatrix supports Python version 2.4. Scripts can be run on NIONs, nControls and nTouch 180s. They can also be run in NWare in emulation mode, when you are developing and testing your project. The examples in this section will use NIONs, but you use one of the other devices if you prefer.

Python script is written and linked to the control flow of a project using Live Python blocks or Python Scripting blocks.

The NWare Live Python block

Live Python blocks allow scripts to be edited and updated at any time, even when a project is running. You can also use them to copy and paste scripts between projects, making them very flexible. We recommend that you use Live Python blocks in favor of Python Scripting blocks.

The Live Python block can be configured with a set number of inputs and outputs, allowing you to wire it to other devices in your design.

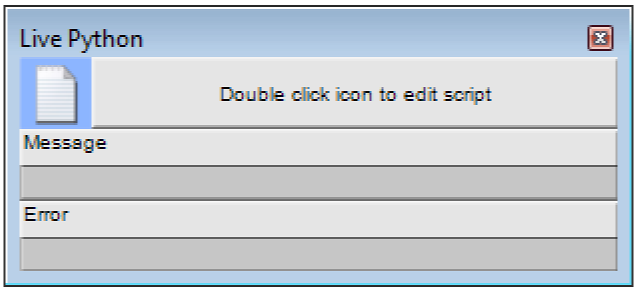


The inputs and outputs on the block can be referenced from Python code. You can work with values from other devices in the design via the inputs, and assign values to other devices using both the inputs and the outputs.

In Python code, the inputs and outputs are referenced using the arrays *inputs[]* and *outputs[]*. For example, the statement below will retrieve the value received via the first input on the block, and assign it to the variable *x*.

```
x = inputs[0].value_get()
```

When you open the block, it displays an icon for opening the code editor, and two fields for displaying status messages and error messages. You can wire these fields to other devices in your project to perform different operations when their values change.



Note: Before you can type in or paste in script to the device, you must emulate or deploy the project.

Tip: To use a different editor program, on the **Tools** menu, click **User Preferences**. Then, under **External String Editor**, click **Browse** to choose a new program. You can specify any editor program you want to use, but we recommend that it is *language aware*. This will make it easier to read the code that you write, and minimize the number of errors.

As soon as the project is emulated or deployed, you can double-click the edit icon to update the Python script for the block.

By default, the Python script is run when the value received on one of the inputs changes, but you can customize the behavior to suit your project. For more information, see *Controlling the execution of Python code* (on page 18).

If there is an error in the Python script, an error message will be displayed in the Error box on the control surface. For more information see *Debugging Python scripts* (on page 11).

PySNMP support

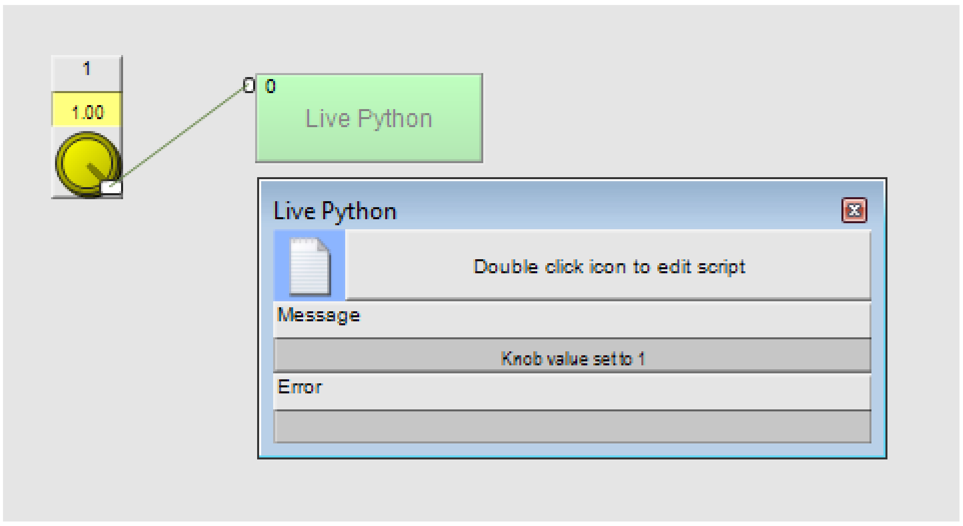
Python provides native support for SNMP, using a technology called PySNMP.

Note: PySNMP is supported on nControl and nTouch 180 units only.

For more information on PySNMP, see *http://pysnmp.sourceforge.net/* (*http://pysnmp.sourceforge.net/*).

Building a simple Python example in NWare

In this example, we are going to wire a knob control to a Live Python block. The Python code will display a message when the value of the knob reaches 1.



Knob value set to 1

To add the devices and wire them together

1. Add a NION to your project.
2. Expand the device tree to display **Control / Knobs, Faders Etc. / Generic Controls**.
3. Drag a **Generic Controls** device over to the design page.
4. Click **OK** to create a single knob control.
5. Open the **Controls** block, and copy the knob control to the design page.

6. Click the knob control to select it, and then press SHIFT+CTRL+M to add a master wiring node.
7. Expand the device tree to display **Control / Live Python**.
8. Drag the **Live Python** folder over to the design page, and then click **1 input - 0 output**.
9. On the toolbar, click the **Wire Mode** button to switch to wiring mode.
10. Drag the master wiring node on the knob control over to the input wiring node on the Live Python block to wire the devices together.

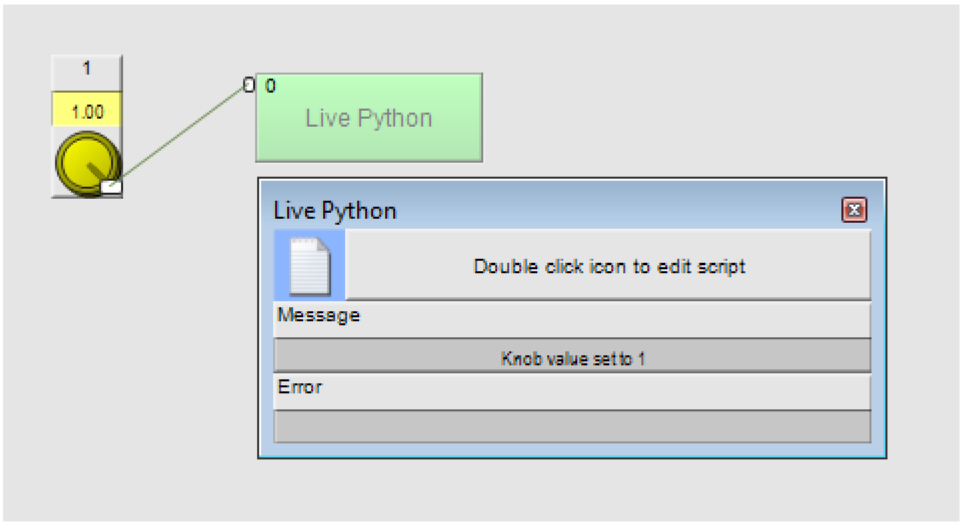
⏩ *To add the Python code and test out the project*

1. Emulate the project.
2. Double-click the **Live Python** block to open it.
3. Double-click the **Edit** button to open the Python source code editor.
4. Type the code below into the editor and then save the file.

```
message.string_set( "" )
if (inputs[0].value_get() == 1):
  message.string_set( "Knob value set to 1" )
```

   The code looks at the value received via the input on the Live Python block. If it is equal to 1, a message is displayed in the *Message* field on the control surface of the device.

5. Gesture the knob and watch the control surface on the Live Python device.



   While the value stays at 1, the message is displayed. However, if the value of the knob reduces, the message will be hidden.

   The link between the devices allows Python code in the Live Python block to monitor the value of the knob control as it is gestured by the user.

Changing Python code while a project is running

One of the benefits of using the Live Python block is that you can make changes to Python code while the project is running, without the need to recompile or redeploy.

Notes:

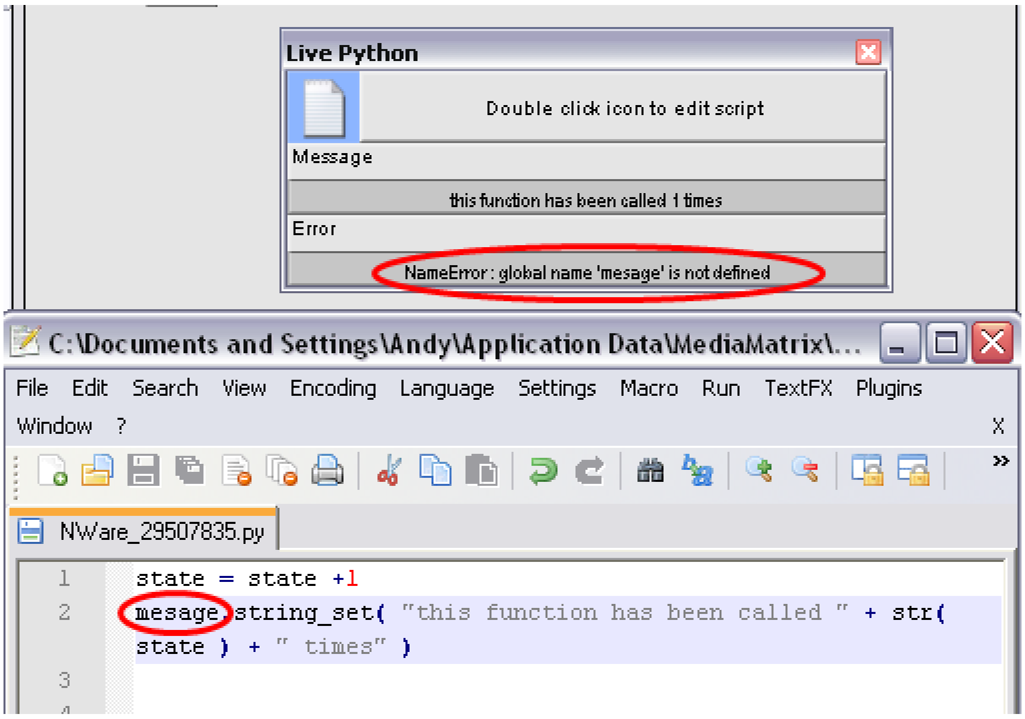
- When you save the Python script in the editor, changes you have made will take effect immediately. You must be sure the code will function correctly before you save.
- Saving the code in the editor does not save the code permanently in the project file. When you stop emulating or disconnect from a deployed project, the default settings must be saved, otherwise the changes to the Python code will be lost. If you are prompted to save the settings, select Yes. If you are not prompted, the settings will be saved automatically. The behavior is controlled by the *save default settings* options on the *Advanced* tab of the *User Preferences* dialog box.
- Python Scripting blocks do not support live code updates. This feature is only available for Live Python blocks.

To change Python code while a project is running

1. Double-click the **Live Python** block to open it.
2. Double-click the **Edit** button to open the Python source code editor.
3. Edit the code in the editor window, and then save the file.

Debugging Python scripts

The Live Python device does not include a debugger, but will display an error message when the code generates errors. In the example below, the typo on line 2 has produced a *NameError*.


Live Python
Double click icon to edit script
Message
Error
C:\Documents and Settings\Andy\Application Data\MediaMatrix\...
File Edit Search View Encoding Language Settings Macro Run TextFX Plugins
Window ?
NWare_29507835.py

this function has been called 1 times

NameError: global name 'mesage' is not defined

```
state = state +1
mesage.string_set( "this function has been called " + str(
state ) + " times" )
```

We recommend that you develop Python code using a development tool that includes a debugger, such as *Idle* (*http://docs.python.org/library/idle.html*). Once the code is close to completion, you can paste it into a Live Python device, where you will then be able to test it out with MediaMatrix devices.

An example with multiple input devices

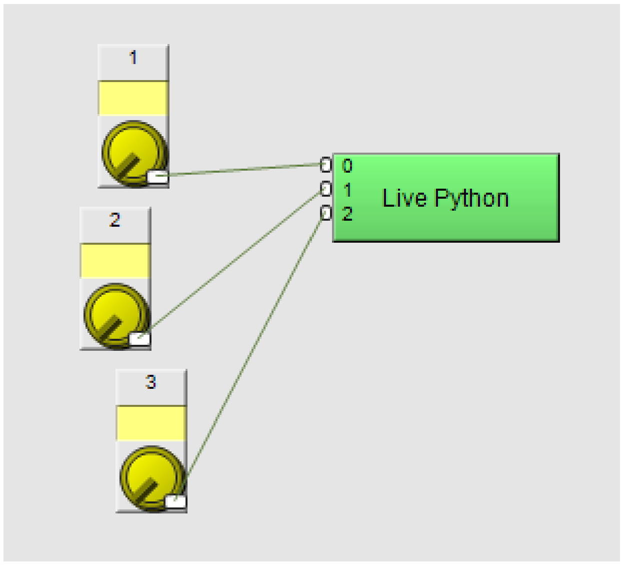
In this example, we will see see how to work with the values of multiple controls wired to the inputs on the Live Python block.

By default, a Live Python block is configured to monitor all the inputs, and run the Python code inside the block as soon as an input value changes. This is the behavior that we want for this example, so we will leave the property (*watch input*) set to *any* on the device properties.

To set up a Live Python block with multiple input devices

1. Add a NION to your project.

1. Expand the device tree to display **Control / Knobs, Faders Etc. / Generic Controls**.
2. Drag a **Generic Controls** device over to the design page.
3. On the **Controls Properties** dialogs box, in the **Number of controls** box, click 3.
4. Click **OK**.
5. Open the **Controls** block, and copy the knob controls to the design page.
6. Drag a selection box around the knob controls, and then press SHIFT+CTRL+M to add master wiring nodes.
7. Drag the Live Python folder over to the design page, and then click **3 input - 0 output**.
8. On the toolbar, click the **Wire Mode** button to switch to wiring mode.
9. Wire all three knob controls to the inputs of the Live Python block.

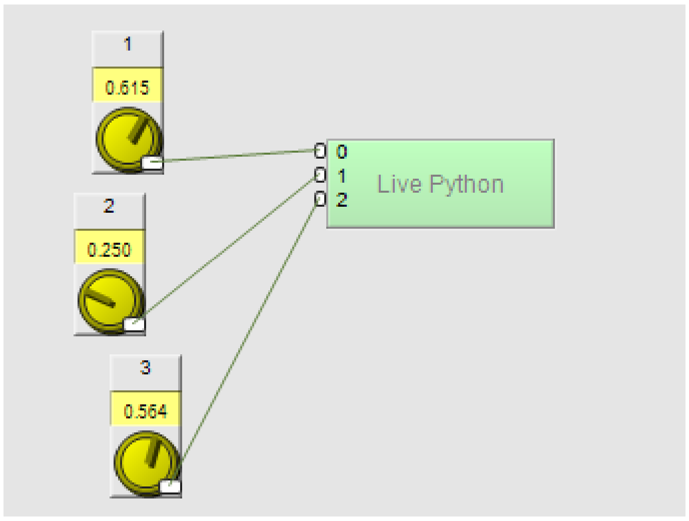


10. Emulate the project.
11. Double-click the **Edit** button to open the Python source code editor.
12. Type the following Python code in the editor:

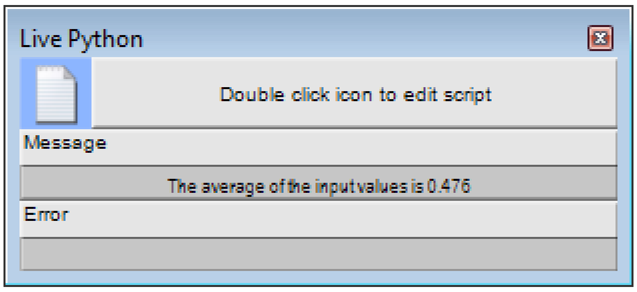
```
total=0

average=0
```

```
for x in range(len(inputs)):
    total += inputs[x].value_get()
average=round(total/len(inputs),3)
message.string_set("The average of the input values is " + str( average ))
```

13. Save the Python code.
14. Gesture the knobs.



The message on the control surface of the Live Python block will show the average of the knob values.



The average of the input values is 0.476

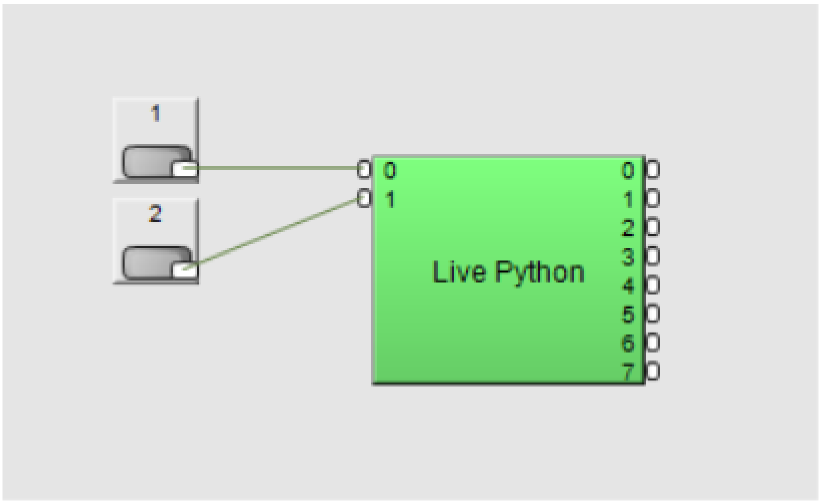
Building an example that processes both inputs and outputs

In this example, we will see how to check whether input values have changed, and then set output values accordingly.

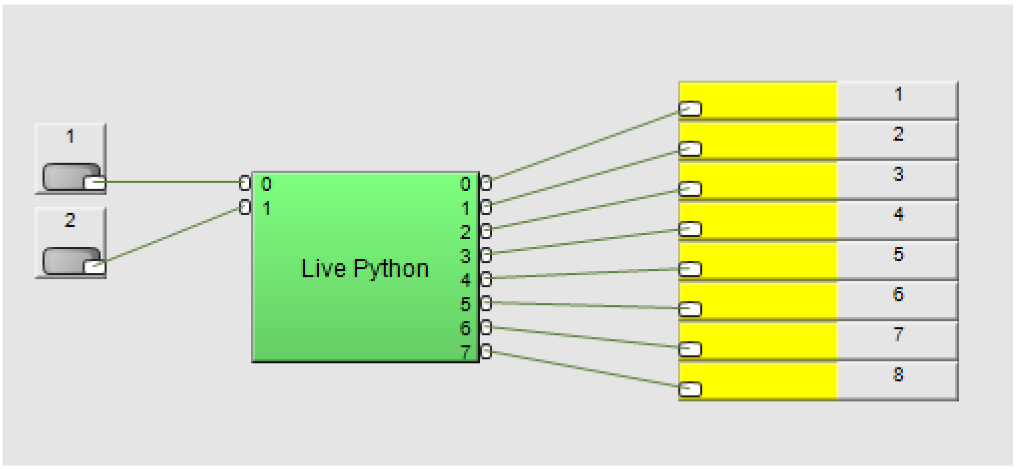
To build an example that processes both inputs and outputs

1. Add a NION to the design page.
2. Add a Live Python block to the design page with 2 inputs and 8 outputs.

3. Add two buttons to the design page, then wire them to the inputs on the Live Python block.



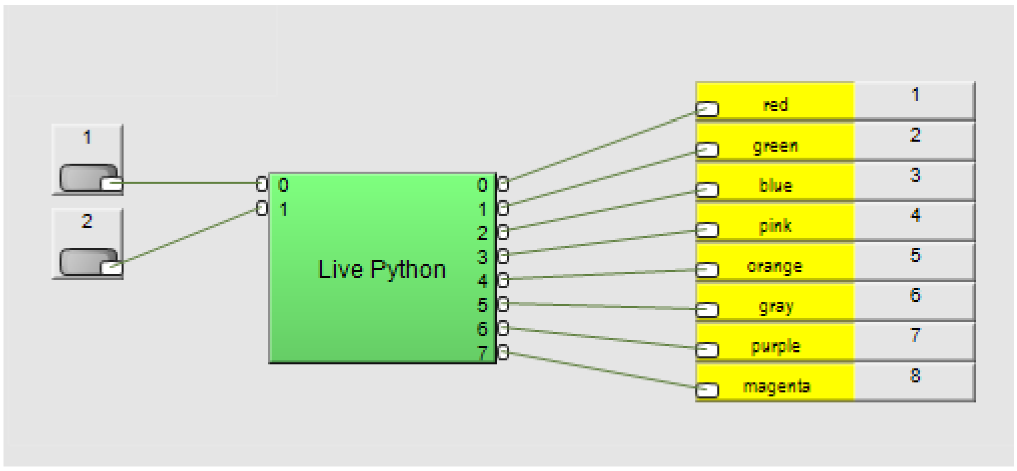
4. Add 8 text box generic controls to the design page, then wire them to the outputs on the Live Python block.



5. Emulate the project.
6. Double-click the **Edit** button to open the Python source code editor.
7. Type the following Python code in the editor:

```
colors = ['red','green', 'blue','pink','orange','gray','purple','magenta']

if inputs[0].changed_get():
    for i in range(len(outputs)):
        outputs[i].string_set(colors[i])
else:
    for i in range(len(outputs)):
        outputs[i].string_set(" ")
```

8. Click the first button to see the colors from the array displayed in the yellow text boxes.



9. Click the second button to clear the values in the text boxes.

Using alternative Python statements in the example

Different Python statements can be substituted for the ones used in the previous example. Experimenting in this way can help you to get used to using different approaches when structuring your Python code.

For example, the *for loops* could be swapped for *while loops*.

Original code with for loops

```
colors = ['red','green',
'blue','pink','orange','gray','purple','magenta']


if inputs[0].changed_get():
    for i in range(len(outputs)):
        outputs[i].string_set(colors[i])
else:
    for i in range(len(outputs)):
        outputs[i].string_set(" ")
```

Replacement code with while loops

```
colors = ['red','green',
'blue','pink','orange','gray','purple','magenta']


if inputs[0].changed_get():
    i = 0
    while i < len(outputs):
        outputs[i].string_set(colors[i])
        i = i + 1
else:
    i= 0
    while i < len(outputs):
        outputs[i].string_set(" ")
        i = i + 1
```

How different button types affect code execution

When you use buttons wired to the Live Python block, it is important to understand how the different button operating modes (toggle/momentary/string) affect the triggering of the Python script.

A toggle button will execute the Python code once when clicked. However, the state of a momentary button will change twice when it is clicked (from *on* and then back to *off*). This means the Python code inside the wired block will be executed twice. You can test out this behavior using an example that displays the value of the state variable, when a button wired to the Live Python block is clicked:

```
if inputs[0].changed_get():
  state+=1
  message.string_set( "This code has been executed " + str( state ) +
" times" )
```

Clicking a momentary button will display a message showing the Python code has been executed twice.

If you want to use a momentary button, and ensure that the function is only called once, you need to add an extra *if* statement to the code:

```
if inputs[0].changed_get():
  if inputs[0].value_get()==1:
    state+=1
    message.string_set( "This code has been executed " + str( state )
+ " times" )
```

Chapter 3

Using more advanced features

In This Chapter

Controlling the execution of Python code

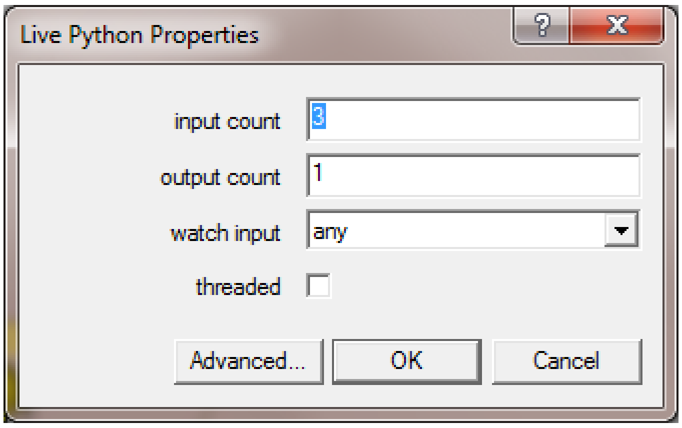
By default, Python code inside a Live Python block runs from start to finish whenever a value is received via one of the inputs on the block. The value could be from a button that has been clicked, or from a knob control that the user has gestured, for example. You can configure the block to monitor all of the inputs (several may be wired to different devices) or just the first or last input.

Note: User-defined Python variables do not retain the values assigned to them from the last time the code was executed. Only the built-in *state* variable does this. For more information, see *Using the persistent state variable* (on page 19).

Another way to run Python code is automatically using a thread. Threaded code will run as soon as the project has been deployed or emulated, it does not wait for a value to be received on one of the inputs. For more information, see *Using threading to run scripts automatically* (on page 24). The remainder of this section assumes you want to control execution of the script using the inputs on the Live Python block.

To specify when to execute the Python code

1. Right-click on the Live Python block, and then click **Properties**.


Live Python Properties

input count: 3
output count: 1
watch input: any
threaded: ☐

2. In the **watch input** list, click an option to specify which input value must change in order to trigger the execution of the Python code.

any	A change to the value of any input will trigger the execution of the code.
first	Only a change to the value of the first input will trigger the execution of the code.
last	Only a change to the value of the last input will trigger the execution of the code.

3. Click **OK**.

Using the state variable to store a value persistently

Each Live Python block has a built in variable called *state*, which is initialized to zero when the project is first emulated or deployed, but retains its value while the project is running. Unlike variables initialized inside the Live Python block, the value of *state* will be retained even when the input values change, and the Python code is executed repeatedly.

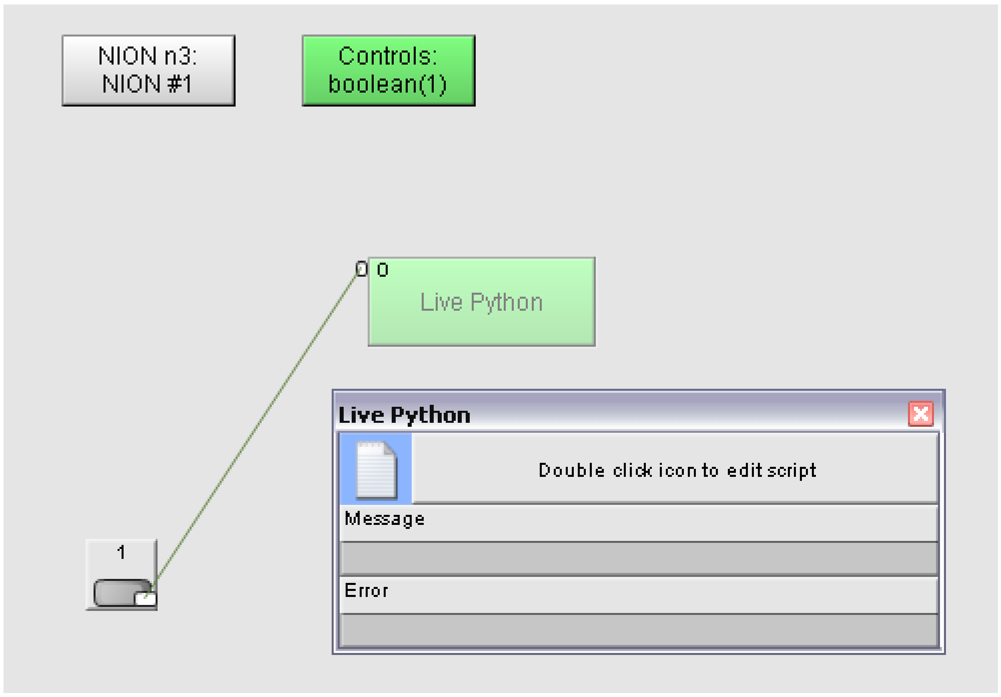
The example in this section will show how the value of *state* can be incremented by successive calls to the Python code. The calls are initiated by a button wired to an input on the Live Python block. Each time the button is clicked, the Python code will run. The changing value of *state* is displayed on the control surface of the Live Python block.

Tip: If you need to save more variable values than just that of the state variable, you can store the values using generic controls. For more information, see *Storing and reusing values from Python code* (on page 21).

To use the persistent state variable

1. Add a NION to your project.
2. Expand the device tree to display **Control / Knobs, Faders Etc. / Generic Controls**.
3. Drag a **Generic Controls** device over to the design page.
4. In the **Type of control(s)** list, click **boolean**.
5. Click **OK.**
6. Double-click the control block to open it, then copy the button control out to the design page.
7. Click the button to select it, and then press SHIFT+CTRL+M to add a master wiring node.
8. Expand the device tree to display **Control / Live Python**.
9. Drag the **Live Python** folder over to the design page, and then click **1 input - 0 output**.
10. On the toolbar, click the **Wire Mode** button to switch to wiring mode.

11. Drag the master wiring node on the button over to the input wiring node on the Live Python block to wire the devices together.

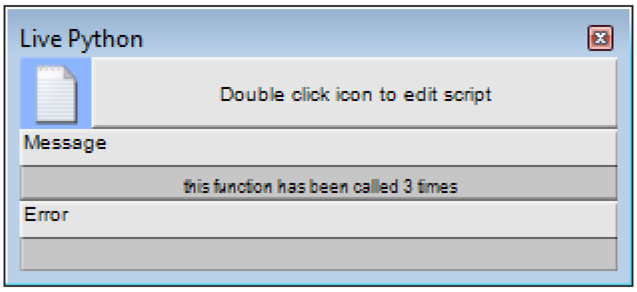


To add the Python code and test out the project

1. Emulate the project.
2. Double-click the **Live Python** block to open it.
3. Double-click the **Edit** button to open the Python source code editor.
4. Type the code below into the editor and then save the file.

```
message.string_set("this function has been called " + str( state ) + " times"
)
state = state + 1
```

Each time you click the button, the message on the control surface of the Python block will be updated.



this function has been called 3 times

Storing and reusing values from Python code

The state variable can store and retain a single value while the project is running, but if you want to store multiple values persistently, you will need to pass them from the Python code inside the Live Python block to generic controls. The values can then be passed back to the Python code when they are needed.

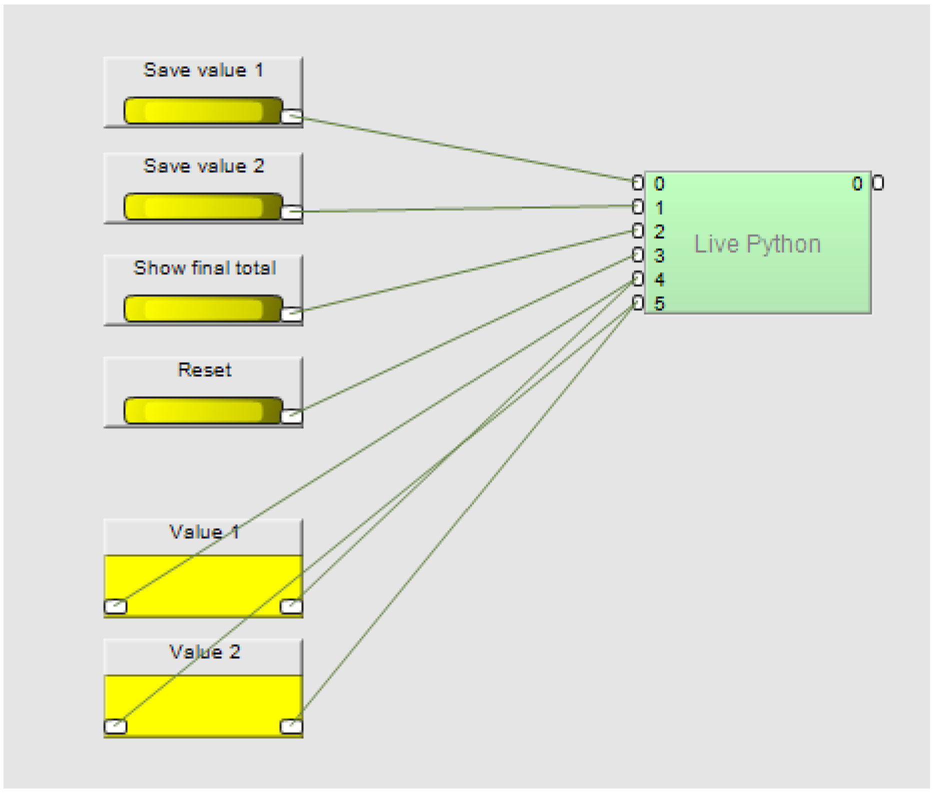
Notes:

- We recommend using a string control to store values, as shown in the example below. If you use a different type of control, such as a float, the values you expect to be passed to the control must be within the range specified on the device properties of the control.
- If you use a string control to store numeric values, you will need to convert the values back to the numeric format before you can use them in calculations, or assign them to numeric variables.

⏩ *To store and reuse values from Python code*

1. Add a NION to the design page.
2. Add a Live Python block to the design page with six inputs.
3. Add four buttons to the design page, then wire them to the first four inputs on the Live Python block.
4. Label the buttons as follows:
   - Save value 1
   - Save value 2
   - Show final total
   - Reset
5. Add two string controls (*Value 1* and *Value 2*) to the design page, then add both master and slave wiring nodes to them.
6. Wire both the master and slave nodes on the first string control to the fifth input wiring node on the Live Python block.

7. Wire both the master and slave nodes on the second string control to the last input wiring node on the Live Python block.



8. Emulate the project.
9. Double-click the **Live Python** block to open it.
10. Double-click the **Edit** button to open the Python source code editor.
11. Type the code below into the editor and then save the file.

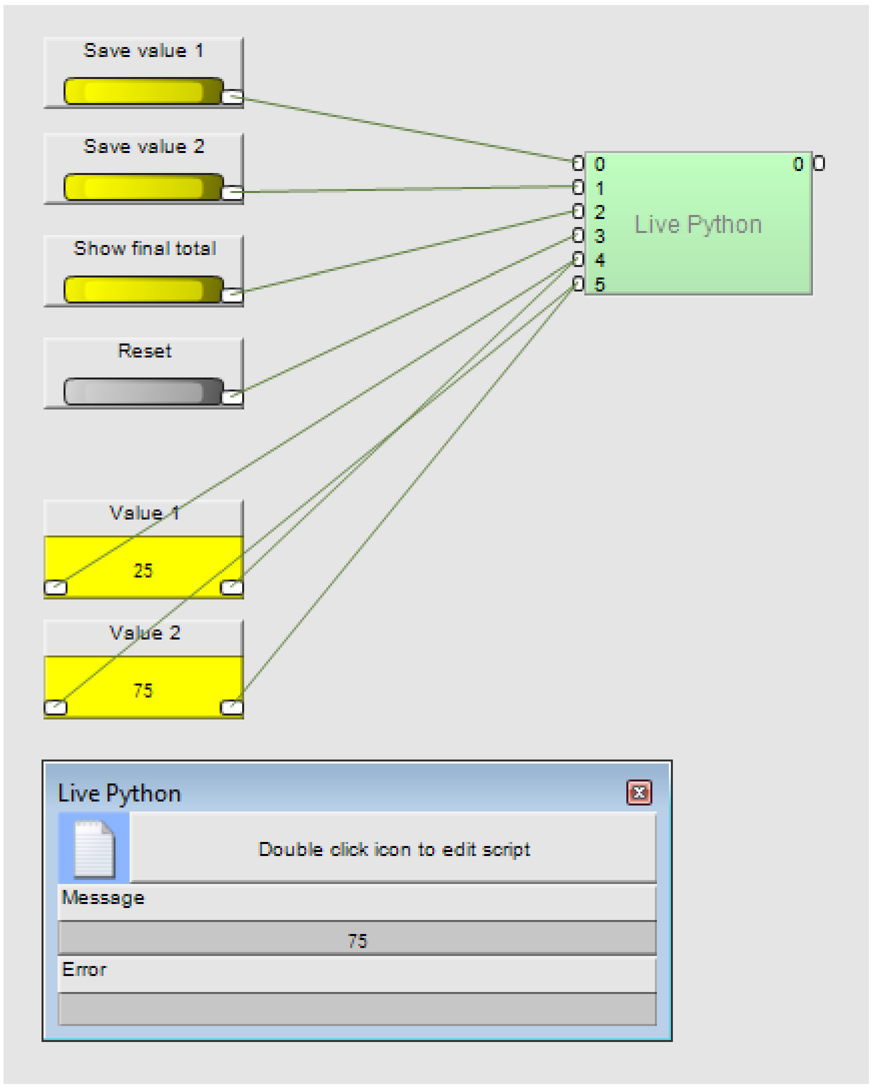
```
total=0

# Save value 1 button as a string
if inputs[0].changed_get():
  inputs[4].string_set("25")

# Save value 2 button as a string
if inputs[1].changed_get():
  inputs[5].string_set("50")

# Show add integer values of saved strings and display
if inputs[2].changed_get():
  total = int(inputs[4].string_get()) + int(inputs[5].string_get())
  message.string_set(total)

# Reset displayed values
if inputs[3].changed_get():
```

```
inputs[4].string_set("")
inputs[5].string_set("")
message.string_set("")
```

12. Click the **Save value 1** button.
13. The value 25 is saved in the first string control.
14. Click the **Save value 2** button.

    The value 50 is saved in the second string control.
15. Click the **Show final total** button.

    The final value 75 is calculated using the saved values, then it is displayed on the control surface of the Live Python block.



Message

75

Error

If you want to run the sequence again, click the **Reset** button, and refer to step 12.

Using threading to run scripts autonomously

Threading allows your Python code to run autonomously in the background, without requiring an input state change to run it. Code inside a thread will run independently of all other Python code in the project – this includes code inside the same Live Python block, or inside other Live Python blocks.

Threaded Python scripts are not triggered by values received on the inputs of the Live Python device. They start running automatically, as soon as the NWare project is deployed.

Example applications for threaded scripts include:

- *Automatic muting of the audio outputs until the power up sequence has completed* (on page 25).
- Watching multiple controls independently, and performing different actions depending on the values.

Notes:

- Threading is an advanced feature that must be used carefully. If not structured correctly, threaded code can consume all the available resources on the node hosting the project, and prevent the project from running.
- Threading must be enabled on the Device Properties of the Live Python block. It is not supported by default.

Syntax and structure of a threaded script

A threaded script can use the following basic structure:

```
while( 1 ):

    # do something here

    # wait for a short time and then start the loop again

    event.wait( 400 )
```

The script performs some actions, and then waits for a short time (400 milliseconds) before returning to the start of the loop. If the user redeploys the project, shuts down the node hosting the project, or changes the script via the Live Python device, it is during the 400ms time period that execution of the script pauses, and other processes running on the node can perform checks to see whether to halt the script altogether. If there is no reason to halt the script, the loop will continue running indefinitely.

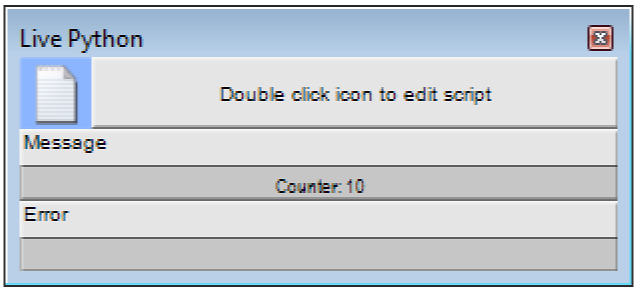
Important Note: The *event.wait* statement relinquishes control to other processes to allow them to run. Without this statement, the Python code would consume *all available CPU resources*, until the node hosting the project stopped functioning correctly. At this point, the role would need to be stopped and then erased via the web interface (or front panel).

Example threaded script - counter

This example threaded script will display a message on the control surface of the Live Python block that updates every second.

```
while (1):
    message.string_set("Counter: "+str(state))
    state = state +1
    event.wait(1000)
```

The counter will start as soon as the project is emulated or deployed, it does not require any values to be received on the inputs.



Counter: 10

Note: Threading must be enabled on the Device Properties of the Live Python block. It is not supported by default. If it is not enabled, when the project is run, the message *RuntimeError : exit thread...* will be displayed in the *Message* field.

Example threaded script - power up mute delay

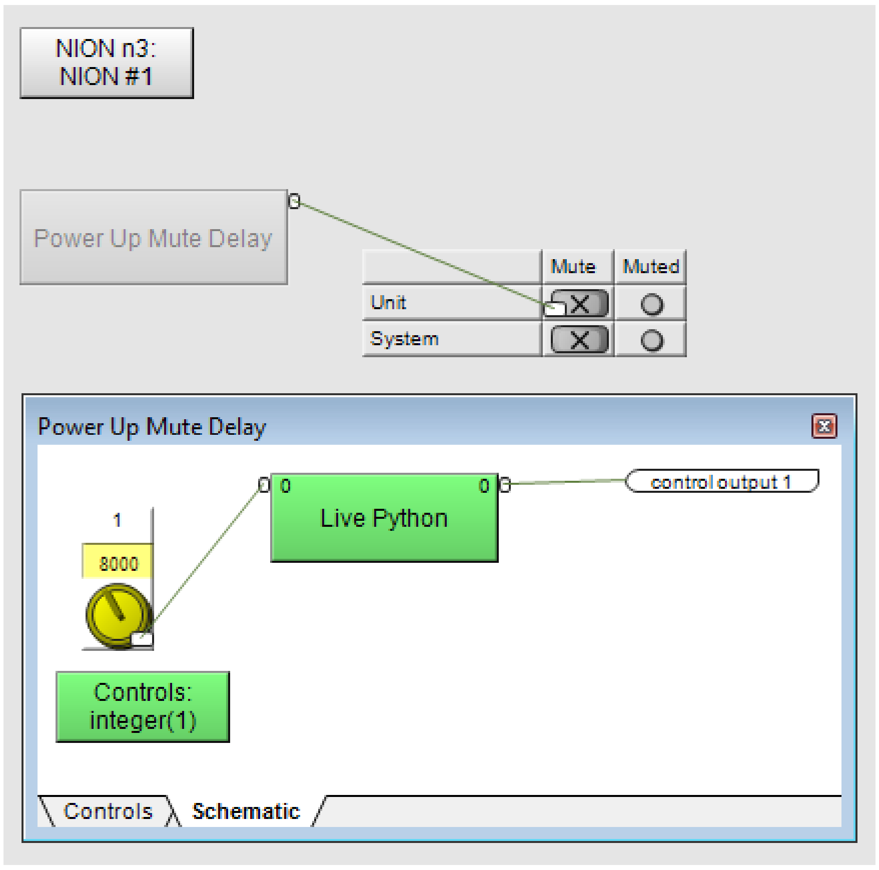
This example uses a Live Python threaded script to mute the audio outputs of a NION as soon as the project starts running. The unit is then kept muted until a set time has elapsed, allowing other devices in the system, like amps, for example, to finish their own boot up sequence, which could take longer.

Using a Live Python block allows the code to run immediately – as soon as the NION powers up – and mute the outputs before performing any other functions, and without user interaction.

The project contains a NioNode block with the controls on the Mute tab copied out to the design page. The Unit mute (for the individual NION) is wired to a control block, which will control when the NION is muted.

Note: The unit mute is used, rather than the system mute. This is because different NIONs in a project may take different lengths of time to start up. Attempting to mute all the NIONs in the project at the same time using the system mute may result in only some of the units being muted.

Inside the *Power Up Mute Delay* block, a control with a delay value (8000 milliseconds) is wired to the input on the Live Python block. This is the length of time the control output on the Live Python block will remain set to 1 (muting the NION) before being set to 0 (unmuting the NION).



The Live Python code sets the output on the block to 1 to action the mute immediately. It then waits for the length of time specified by the wired input control. Finally, it sets the output on the block to 0 to switch off the mute.

```
outputs[0].value_set(1)

event.wait(inputs[0].value_get())

outputs[0].value_set(0)
```

Note: Most threaded scripts are configured to run continuously in the background, using a loop and the *event.wait()* statement. This example, however, is designed to run only once, at start up.

Chapter 4

Python reference

In This Chapter

Predefined Variables

Predefined variables	
inputs	Object referencing all the controls wired to the inputs of the Python device. For example, `inputs[0]` returns a reference to the control wired to the first input node, `inputs[1]` returns a reference to the control wired to the second input node, and so on.
outputs	Object referencing all the controls wired to the outputs of the Python device. For example, `outputs[0]` returns a reference to the control wired to the first output node, `outputs[1]` returns a reference to the control wired to the second output node, and so on.
message	Text control displayed on the control surface of the Python block. Its value can be set using `message.string_set(value)`.
state	A persistent integer variable. The default value is 0 (zero). For an example showing how you can work with the value of the state variable, see *Using the persistent state variable* (on page 19).

Input and output control functions

Input controls (wired to the inputs of the Python or Live Python block)	
position_get()	Returns the position of a control in floating point format.
position_set()	Sets the position of a control using a floating point format value.
value_get()	Returns the numeric value of a control.
value_set (value)	Sets the numeric value of a control.
string_get()	Returns the value of a control as a string.
string_set (value)	Sets the value of a control in string format.
changed_get()	Returns true if the input control has changed since the last time the script function was run.

Output controls (wired to the outputs of the Python or Live Python block)	
position_set (value)	Sets the position of a control. The value must be a floating point or integer.
value_set (value)	Sets the value of a control. The value must be a floating point or integer.
string_set (value)	Sets the value of a control in string format. The value must be a string.

Threaded script functions

event.wait(timeout)

Pause the script for a specified period of time before continuing.

Note: It is not possible to pause the script indefinitely.

timeout	Time in milliseconds for the routine to wait before continuing. If a project role is stopped while waiting in this routine, the script will exit immediately at this line and no further processing will occur. This argument is required. If you specify zero, the script will continue immediately.

event.wait_input_change ([timeout])

Pause the script until one of the inputs of a device changes (or a specified period of time elapses).

Returns true if an input has changed, or false if no input has changed.

[timeout]	Time in milliseconds for the routine to wait before continuing. If a project role is stopped while waiting in this routine, the script will exit immediately at this line and no further processing will occur. This argument is optional. **Note:** If this argument is omitted, the routine will wait indefinitely for an input to change.

To determine if any inputs have changed after the script returns from running *wait_input_change([timeout])*, query the input objects to obtain their current state. For example:

```
for input in inputs:
  if input.change_get():
    #do something
```

event.wait_input_should_exit([timeout])

Pause the script until one of the inputs of a device changes (or a specified period of time elapses).

[timeout]	Time in milliseconds for the routine to wait before continuing. If a project role is stopped while waiting in this routine, a non-zero value will be returned to allow the script to perform any cleanup that might be required before exiting. If zero is specified, the script will continue immediately. This argument is optional. **Note:** If this argument is omitted, the routine will wait indefinitely for an input to change.

event.wait_should_exit(timeout)

Pause the script for a specified period of time before continuing.

timeout	Time in milliseconds for the routine to wait before continuing. If a project role is stopped while waiting in this routine, a non-zero value will be returned to allow the script to perform any cleanup that might be required before exiting. If zero is specified, the script will continue immediately. This argument is optional. **Note:** If this argument is omitted, the routine will wait indefinitely for an input to change.

Serial port control functions

serial_port.create

Opens communication to a serial port on the device.

By default, the *Duplex* argument is set to zero, which means the RS-422 protocol will be used. If you want to use RS-485, set the *Duplex* argument to 1.

Name	The name of the port. Data type: string. On nControl units, the format is `COM<number>`. For example: `COM1`. On NIONs, this is the path to the port. For example: `/dev/ttyS1`. This argument is required.	
Baud rate	Baud rate for data transmission and reception in bits per second. Data type: Integer. This argument is required.	
Data bits	Data type: Integer. Default is 8.	
Parity	Data type: String. Default is none.	
	none	No parity bit is sent. Error detection is handled by the communication protocol.
	even	The parity bit is set to 1 if the number of ones in the set of bits is odd, therefore making the number of ones even.
	odd	The parity bit is set to 1 if the number of ones in the set of bits is even, therefore making the number of ones odd.
	mark	Parity bit is always set to the mark signal condition (logical 1).
	space	Parity bit is always set to the space signal condition.
Stop bits	Bits sent at the end of every character to signify the end of the	

	character in the data transmission. Normally, 1 stop bit is used. Data type: Float. Can be set to 1, 1.5, 2. Default is 1.	
Duplex	Integer. Default is 0.	
	0	Full duplex.
	1	Half duplex.

Support for Python modules

Python scripting is supported by NION, nControl and nTouch 180. It is also supported by NWare in emulation mode. However, these different platforms do not all support the same set of Python modules. It is important to check the sections below to make sure the modules you are intending to use in your project are available for the platforms you are using and whether they are built-in or need to be included manually with an *import* statement (indicated by the term *module*).

Modules (underscore _)

Module name	Emulation	NION	nControl	nTouch 180
__built in__	Built-in	Built-in	Built-in	Built-in
__future__	Module	N/A	Module	Module
__phello__.foo	N/A	N/A	N/A	N/A
_bisect	Built-in	N/A	Built-in	Built-in
_bsddb	N/A	N/A	N/A	Built-in
_codecs	Built-in	Built-in	Built-in	Built-in
_codecs_cn	Built-in	N/A	Built-in	Built-in
_codecs_hk	Built-in	N/A	Built-in	Built-in
_codecs_iso2022	Built-in	N/A	Built-in	Built-in
_codecs_jp	Built-in	N/A	Built-in	Built-in
_codecs_kr	Built-in	N/A	Built-in	Built-in
_codecs_tw	Built-in	N/A	Built-in	Built-in
_csv	Built-in	N/A	Built-in	Built-in

Module name	Emulation	NION	nControl	nTouch 180
_curses	N/A	N/A	N/A	N/A
_curses_panel	Built-in	N/A	N/A	N/A
_heapq	Built-in	N/A	Built-in	Built-in
_hotshot	Built-in	N/A	Built-in	Built-in
_locale	Built-in	N/A	Built-in	Built-in
_LWPCookieJar	Module	N/A	Module	Module
_MozillaCookieJar	Module	N/A	Module	Module
_multibytecodec	Built-in	N/A	Built-in	Built-in
_mysql	N/A	Module	N/A	N/A
_mysql_exceptions	N/A	Module	N/A	N/A
_random	Built-in	Module	Built-in	Built-in
_socket	Module	Module	Module	Module
_sre	Built-in	Built-in	Built-in	Built-in
_ssl	N/A	N/A	N/A	Built-in
_strptime	Module	N/A	Module	Module
_symtable	Built-in	Built-in	Built-in	Built-in
_testcapi	N/A	N/A	N/A	N/A
_threading_local	Module	N/A	Module	Module
_weakref	Built-in	N/A	Built-in	Built-in

Modules (a-d)

Module name	Emulation	NION	nControl	nTouch 180
aifc	Module	N/A	Module	Module
anydbm	Module	N/A	Module	Module
array	Built-in	N/A	Built-in	Built-in
asynchat	Module	N/A	Module	Module
asyncore	Module	N/A	Module	Module
atexit	Module	N/A	Module	Module
audiodev	Module	N/A	Module	Module
audioop	Built-in	N/A	Built-in	Built-in
base64	Module	Module	Module	Module
BaseHTTPServer	Module	N/A	Module	Module
Bastion	Module	N/A	Module	Module
bdb	Module	N/A	Module	Module
binascii	Built-in	Module	Built-in	Built-in
binhex	Module	N/A	Module	Module
Bio	N/A	Module	N/A	N/A
BioSQL	N/A	Module	N/A	N/A
bisect	Module	N/A	Module	Module
bsddb	N/A	N/A	N/A	N/A
bz2	N/A	Module	N/A	N/A
calendar	Module	N/A	Module	Module
Canvas	N/A	N/A	N/A	N/A
CDROM	N/A	N/A	N/A	N/A
CGIHTTPServer	Module	N/A	Module	Module

Module name	Emulation	NION	nControl	nTouch 180
cgitb	Module	N/A	Module	Module
chunk	Module	N/A	Module	Module
cmath	Built-in	Module	Built-in	Built-in
cmd	Module	N/A	Module	Module
code	Module	N/A	Module	Module
codecs	Module	Module	Module	Module
codeop	Module	N/A	Module	Module
collections	Built-in	N/A	Built-in	Built-in
colorsys	Module	N/A	Module	Module
commands	Module	N/A	Module	Module
CompatMysqldb	N/A	Module	N/A	N/A
compileall	Module	N/A	Module	Module
compiler	Module	N/A	Module	Module
ConfigParser	Module	N/A	Module	Module
Cookie	Module	N/A	Module	Module
cookielib	Module	N/A	Module	Module
copy	Module	N/A	Module	Module
copy_reg	Module	Module	Module	Module
cPickle	Built-in	Module	Built-in	Built-in
crypt	N/A	Module	N/A	N/A
cStringIO	Built-in	N/A	Built-in	Built-in
csv	Module	N/A	Module	Module
curses	N/A	N/A	Module	N/A
datetime	Built-in	Module	Built-in	Built-in
dbhash	N/A	N/A	Module	N/A

Module name	Emulation	NION	nControl	nTouch 180
dbm	N/A	N/A	N/A	N/A
decimal	Module	N/A	Module	Module
Dialog	N/A	N/A	N/A	N/A
difflib	Module	N/A	Module	Module
dircache	Module	N/A	Module	Module
dis	Module	N/A	Module	Module
distutils	N/A	N/A	N/A	N/A
dl	N/A	N/A	N/A	N/A
DLFCN	N/A	N/A	N/A	N/A
doctest	Module	N/A	Module	Module
DocXMLRPCServer	Module	N/A	Module	Module
dumbdbm	Module	N/A	Module	Module
dummy_thread	Module	N/A	Module	Module
dummy_threading	Module	N/A	Module	Module

Modules (e-h)

Module name	Emulation	NION	nControl	nTouch 180
email	Module	Module	Module	Module
encodings	Module	N/A	Module	Module
errno	Built-in	Built-in	Built-in	Built-in
exceptions	Built-in	Built-in	Built-in	Built-in
fcntl	N/A	N/A	N/A	N/A
filecmp	Module	N/A	Module	Module
FileDialog	N/A	N/A	N/A	N/A

Module name	Emulation	NION	nControl	nTouch 180
fileinput	Module	N/A	Module	Module
FixTk	N/A	N/A	N/A	N/A
fnmatch	Module	N/A	Module	Module
formatter	Module	N/A	Module	Module
fpformat	Module	N/A	Module	Module
ftplib	Module	N/A	Module	Module
gc	Built-in	Built-in	Built-in	Built-in
gdbm	N/A	N/A	N/A	N/A
getopt	Module	N/A	Module	Module
getpass	Module	N/A	Module	Module
gettext	Module	N/A	Module	Module
glob	Module	N/A	Module	Module
gopherlib	Module	N/A	Module	Module
grp	N/A	N/A	N/A	N/A
gzip	N/A	N/A	N/A	N/A
heapq	Module	N/A	Module	Module
hmac	Module	Module	Module	Module
hotshot	Module	N/A	Module	Module
htmlentitydefs	Module	N/A	Module	Module
htmllib	Module	N/A	Module	Module
HTMLParser	Module	N/A	Module	Module
httplib	Module	N/A	Module	Module

Modules (i-n)

Module name	Emulation	NION	nControl	nTouch 180
idlelib	N/A	N/A	N/A	N/A
ihooks	Module	N/A	Module	Module
imageop	Built-in	N/A	Built-in	Built-in
imaplib	Module	N/A	Module	Module
imghdr	Module	N/A	Module	Module
Imp	Built-in	Built-in	Built-in	Built-in
imputil	Module	N/A	Module	Module
IN	N/A	N/A	N/A	N/A
inspect	Module	N/A	Module	Module
itertools	Built-in	N/A	Built-in	Built-in
keyword	Module	N/A	Module	Module
linecache	Module	Module	Module	Module
linuxaudiodev	N/A	N/A	N/A	N/A
locale	Module	Module	Module	Module
logging	Module	N/A	Module	Module
macpath	Module	N/A	Module	Module
macurl2path	Module	N/A	Module	Module
mailbox	Module	N/A	Module	Module
mailcap	Module	N/A	Module	Module
markupbase	Module	N/A	Module	Module
marshal	Built-in	Built-in	Built-in	Built-in
Martel	N/A	Module	N/A	N/A
math	Built-in	Module	Built-in	Built-in

Module name	Emulation	NION	nControl	nTouch 180
md5	Built-in	Module	Built-in	Built-in
mhlib	Module	N/A	Module	Module
mimetools	Module	N/A	Module	Module
mimetypes	Module	N/A	Module	Module
MimeWriter	Module	N/A	Module	Module
mmap	Built-in	N/A	Built-in	Built-in
modulefinder	Module	N/A	Module	Module
multifile	Module	N/A	Module	Module
mutex	Module	N/A	Module	Module
mx	N/A	Module	N/A	N/A
MySQLdb	N/A	Module	N/A	N/A
netrc	Module	N/A	Module	Module
new	Module	N/A	Module	Module
nis	N/A	N/A	N/A	N/A
nntplib	Module	N/A	Module	Module
ntpath	Module	Module	Module	Module
nturl2path	Module	N/A	Module	Module

Modules (o-r)

Module name	Emulation	NION	nControl	nTouch 180
opcode	Module	N/A	Module	Module
operator	Built-in	N/A	Built-in	Built-in
optparse	Module	N/A	Module	Module
os	Module	Module	Module	Module

Module name	Emulation	NION	nControl	nTouch 180
os2emxpath	Module	N/A	Module	Module
ossaudiodev	N/A	N/A	N/A	N/A
parserpwd	N/A	N/A	N/A	N/A
pdb	Module	N/A	Module	Module
pickle	Module	N/A	Module	Module
pickletools	Module	N/A	Module	Module
pipespkgutil	N/A	N/A	N/A	N/A
platform	Module	N/A	Module	Module
popen2	Module	N/A	Module	Module
poplib	Module	N/A	Module	Module
posix	N/A	Built-in	N/A	N/A
posixfile	Module	N/A	Module	Module
posixpath	Module	Module	Module	Module
pprint	Module	N/A	Module	Module
profile	Module	N/A	Module	Module
pstats	Module	N/A	Module	Module
pty	N/A	N/A	N/A	N/A
py_compile	Module	N/A	Module	Module
pyclbr	Module	N/A	Module	Module
pydoc	Module	N/A	Module	Module
pyexpat	Module	Module	Module	Module
Queue	Module	N/A	Module	Module
quopri	Module	Module	Module	Module
random	Module	Module	Module	Module
re	Module	Module	Module	Module

Module name	Emulation	NION	nControl	nTouch 180
readline	N/A	N/A	N/A	N/A
reconvert	Module	N/A	Module	Module
regex	Built-in	Module	Built-in	Built-in
regex_syntax	Module	N/A	Module	Module
regsub	Module	N/A	Module	Module
repr	Module	N/A	Module	Module
resource	N/A	N/A	N/A	N/A
rexec	Module	N/A	Module	Module
rfc822	Module	Module	Module	Module
rgbimg	Module	N/A	Built-in	Built-in
rlcompleter	N/A	N/A	N/A	N/A
robotparser	Module	N/A	Module	Module

Modules (s-t)

Module name	Emulation	NION	nControl	nTouch 180
sched	Module	N/A	Module	Module
ScrolledText	N/A	N/A	N/A	N/A
select	Module	Module	Module	Module
sets	Module	N/A	Module	Module
sgmllib	Module	N/A	Module	Module
sha	Built-in	Module	Built-in	Built-in
shelve	N/A	N/A	N/A	N/A
shlex	Module	N/A	Module	Module
shutil	Module	N/A	Module	Module

Module name	Emulation	NION	nControl	nTouch 180
signal	Built-in	Built-in	Built-in	Built-in
SimpleDialog	N/A	N/A	N/A	N/A
SimpleHTTPServer	Module	N/A	Module	Module
SimpleXMLRPCServer	Module	N/A	Module	Module
site	Module	Module	Module	Module
smtplib	Module	N/A	Module	Module
sndhdr	Module	N/A	Module	Module
socket	Module	Module	Module	Module
SocketServer	Module	N/A	Module	Module
sre	Module	Module	Module	Module
sre_compile	Module	Module	Module	Module
sre_constants	Module	Module	Module	Module
sre_parse	Module	Module	Module	Module
stat	Module	Module	Module	Module
statcache	Module	N/A	Module	Module
statvfs	Module	N/A	Module	Module
string	Module	Module	Module	Module
StringIO	Module	N/A	Module	Module
stringold	Module	N/A	Module	Module
stringprep	Module	N/A	Module	Module
strop	Built-in	N/A	Built-in	Built-in
struct	Built-in	Module	Built-in	Built-in
subprocess	Module	N/A	Module	Module
sunau	N/A	N/A	N/A	N/A
sunaudiosymbol	N/A	N/A	N/A	N/A

Module name	Emulation	NION	nControl	nTouch 180
symtable	Module	N/A	Module	Module
sys	Built-in	Built-in	Built-in	Built-in
syslog	N/A	N/A	N/A	N/A
tabnanny	Module	N/A	Module	Module
tarfile	Module	N/A	Module	Module
telnetlib	Module	Module	Module	Module
tempfile	Module	N/A	Module	Module
termios	N/A	N/A	N/A	N/A
test	N/A	N/A	N/A	N/A
textwrap	Module	N/A	Module	Module
this	Module	N/A	Module	Module
thread	Built-in	Built-in	Built-in	Built-in
threading	Module	N/A	Module	Module
time	Built-in	Module	Built-in	Built-in
timeit	Module	N/A	Module	Module
timing	N/A	N/A	N/A	N/A
Tix	N/A	N/A	N/A	N/A
tkColorChooser	N/A	N/A	N/A	N/A
tkCommonDialog	N/A	N/A	N/A	N/A
Tkconstants	N/A	N/A	N/A	N/A
Tkdnd	N/A	N/A	N/A	N/A
tkFileDialog	N/A	N/A	N/A	N/A
tkFont	N/A	N/A	N/A	N/A
Tkinter	N/A	N/A	N/A	N/A
tkMessageBox	N/A	N/A	N/A	N/A

Module name	Emulation	NION	nControl	nTouch 180
tkSimpleDialog	N/A	N/A	N/A	N/A
toaiff	Module	N/A	Module	Module
token	Module	N/A	Module	Module
tokenize	Module	N/A	Module	Module
trace	Module	N/A	Module	Module
traceback	Module	N/A	Module	Module
tty	N/A	N/A	N/A	N/A
turtle	N/A	N/A	N/A	N/A
types	Module	Module	Module	Module
TYPES	N/A	N/A	N/A	N/A

Modules (u-z)

Module name	Emulation	NION	nControl	nTouch 180
unicodedata	Module	N/A	Module	Module
unittest	Module	N/A	Module	Module
urllib	Module	N/A	Module	Module
urllib2	Module	N/A	Module	Module
urlparse	Module	N/A	Module	Module
user	Module	N/A	Module	Module
UserDict	Module	Module	Module	Module
UserList	Module	N/A	Module	Module
UserString	Module	N/A	Module	Module
uu	Module	N/A	Module	Module
warnings	Module	Module	Module	Module

Module name	Emulation	NION	nControl	nTouch 180
wave	Module	N/A	Module	Module
weakref	Module	N/A	Module	Module
webbrowser	Module	N/A	Module	Module
whichdb	Module	N/A	Module	Module
whrandom	Module	Module	Module	Module
xdrlib	Module	N/A	Module	Module
xml	Module	N/A	Module	Module
xmllib	Module	N/A	Module	Module
xmlrpclib	Module	N/A	Module	Module
xxsubtype	Built-in	Built-in	Built-in	Built-in
zipfile	N/A	N/A	N/A	N/A
zipimport	Built-in	Built-in	Built-in	Built-in
zlib	Built-in	Module	N/A	N/A