If you use revpimodio2 module in Python3, it uses the piCtory configuration to create all the inputs and outputs with their symbolic names as objects. The programmer can address these directly via the symbolic names and access the values of the inputs and outputs – both reading and writing!

import revpimodio2

rpi = revpimodio2.RevPiModIO(autorefresh=True)

# If input t\_on is high, set output h\_on high

if rpi.io.t\_on.value:

rpi.io.h\_on.value = True

# Clean up and sync process image

rpi.exit()

In addition, it provides the developer with many useful functions that can be used to develop cyclic or event-based programs.

If you know the .add\_event\_detect(...) function of the GPIO module from the Raspberry Pi, you can also achieve this behavior with the Revolution Pi:

import revpimodio2

rpi = revpimodio2.RevPiModIO(autorefresh=True)

def event\_detect(ioname, iovalue):

"""Event function."""

# Set actual input value to output 'h\_on'

rpi.io.h\_on.value = iovalue

print(ioname, iovalue)

# Bind event function to input 't\_on'

rpi.io.t\_on.reg\_event(event\_detect)

rpi.mainloop()

Even with hardware changes, but constant names of the inputs and outputs, the actual Python3 source code does not need to be changed!

#### How it works:

|-----------------------------------------------------|

| |

| Python program |

| |

-------------------- | ---------------- -------------------- |

| | | | | | | |

| RevPi hardware | <-----> | RevPiModIO | <----> | Your source code | |

| | | | | | | |

-------------------- | ---------------- -------------------- |

| |

|-----------------------------------------------------|

#### Summary

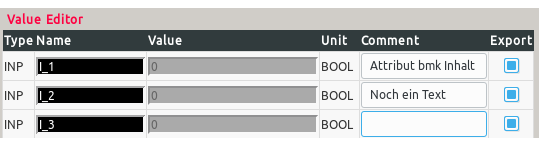
With this module we want to spare all Python developers a lot of work. All communication with the process image is optimally performed inside the module. Changes to the inputs and outputs are also evaluated along with the additional functions of the module give the developer many tools along the way.

**OBJECT .IO**

revpimodio2.io.IOList()

##### CLASSOBJECTS

This class offers all IOs as attributes. The names of the attributes are taken out of the  piCtory configuration.



If an Input with the name “I\_1” is configured in piCtory, then you can use python to access it’s values:

.io.I\_1.value

Outputs whose values can be changed via an assignment are used in the same way. For example, for an Output named “O\_1”:

.io.O\_1.value = **True**

You also can acces the IO object via String:

.io["I\_1"].value

**Note:** For piCtory naming, the [Python rules](https://docs.python.org/3.3/reference/lexical_analysis.html#identifiers) must be respected for name assignments. If this is not possible, access is only possible via string (

.io["§ad#Name"].value

) .

**Getter and setter function**

.get\_defaultvalue()

Always returns the default value from IO as <class ‘bytes’>.

.get\_intdefaultvalue()

Returns the default value from IO as <class ‘int’>.  
*Note*: Only available for IO types from the class <class ‘revpimodio.io.IntIO’> .

.get\_structdefaultvalue()

Returns the default value as <class ‘int’>, taking into account the struct format specification  .  
*Note*: Only available for IO types of the class <class ‘revpimodio.io.StructIO’> .

.get\_value()

Always returns the IO value as <class ‘bytes’> .

.get\_intvalue()

Returns the IO value as<class ‘int’>.  
*Note*: Only available for IO types of the class <class ‘revpimodio.io.IntIO’> .

.get\_structvalue()

Returns the default value as <class ‘int’>, taking into account the struct format specification  .  
*Note*: Only available for IO types of the class <class ‘revpimodio.io.StructIO’> .

.set\_value(value)

Sets the IO value to passed value of type <class ‘bytes’>

.set\_intvalue(value)

Sets the IO value to passed value of type <class ‘int’>  
*Note*: Only available for IO types of the class <class ‘revpimodio.io.IntIO’> .

.set\_structvalue(value)

Returns the default value as <class ‘int’>, taking into account the struct format specification .

**CLASS EVENTCALLBACK**

An instance of this class is passed to any event function, which is registered with [.reg\_event(…, as\_thread=True)](https://revpimodio.org/doku2/io/#reg_event) or [.reg\_timerevent(…, as\_thread=True)](https://revpimodio.org/doku2/io/#reg_timerevent) for IOs.

revpimodio2.helper.EventCallback()

If event functions are created for event monitoring, they must accept a transfer parameter! In this example, it is called  eventcallback, but it can also have ecb  or other short names.

**def** my\_function(eventcallback):

# Name of the IO, which triggered this event

eventcallback.ioname

# Value of the IO, at the Trigger point

eventcallback.iovalue

# Simple 3 second wait time, without program blocking

eventcallback.exit.wait(3)