Laboratório Nacional de Luz Síncrotron

CNPEM Centro Nacional de Pesquisa em Energia e Materiais

Guidelines for Py4Syn Tutorial

PCaPAC 2016

CNPEM/LNLS – Brazil

Instructor: Márcio Paduan Donadio

1. Instructions

To access your VMware virtual machine (VM), use the user guest with password=guest. If needed, the root password is root. All the exercises will be run in this VM. So, at this point on, open VMware and start your VM.

```
> this symbol shows something that needs to be typed
in a Linux terminal. Don't type the symbol itself.
```

For example, the following instruction tells you to type ls in the Linux terminal:
> ls

```
>>> this symbol shows something that needs to be typed
in a Python terminal. Don't type the symbol itself.
```

For example, the following instruction tells you to type print('test') in the Python terminal:

```
>>> print('test')
```

2. LNLS script

The LNLS script was created to make IOC interaction simpler. With it, one does not need to call st.cmd, or to run procServ manually, remembering to list telnet ports used and to configure procServ parameters.

- **lnls status**: list the IOCs currently running on the computer.

Try it! Open a Linux terminal and type:

```
> lnls status
motorSim (pid 2403) is running...
```

- **lnls dbl**: list all the records inside an IOC. It is possible to type only the initial characters of the IOC name in the command parameter, like `lnls dbl <initial characters of the IOC name>`

Try it!

```
> lnls dbl motor

MODIFY:SIM:m1
MODIFY:SIM:m2
MODIFY:SIM:m3
MODIFY:SIM:m4
MODIFY:SIM:m5
MODIFY:SIM:m6
MODIFY:SIM:m7
MODIFY:SIM:m8
```

You can see that record names start with MODIFY. All people in the training room have the same record names and this can cause conflicts. So, the first thing to do is to change the IOC and create a name of our own:

1. Open the file motor.substitutions that is in directory /usr/local/epics/apps/config/CENTOS.PY4SYN/
2. Change every word MODIFY to something you think it is appropriate.
3. Save the file.

The IOC needs to be restarted as changes are loaded. For this, we will use the lnls rioc command.

- **lnls rioc <IOC name>**: restarts the IOC.

Try it with sudo!

```
> sudo lnls rioc motorSim
Restarting IOC motorSim:
          First killing child process: 3545
                                                          [   OK   ]

          Starting: motorSim
```

Use lnls dbl to check that all record names were changed.

- **lnls restart**: restart all IOCs. Take care when using this command, as you can stop processes that are not meant to be stopped.

Try it with sudo!

```
> sudo lnls restart
Stopping IOCs :

          motorSim
          First killing child process: 67935
          Now, Killing parent process: 67934


Waiting the TCP/UDP sockets to close...OK!
Starting LNLS Services:
          Starting: motorSim
```

- **lnls stop:** stop all IOCs. Take care when using this command, as you can stop processes that are not meant to be stopped.

Try it with sudo!

```
> sudo lnls stop

Stopping IOCs :

          motorSim
          First killing child process: 74246
          Now, Killing parent process: 74245
```

Check that the motor IOC really stopped, trying (remember to substitute YOUR_MOTOR by the record name you invented):

```
> caget YOUR_MOTOR:SIM:m1.RBV
```

- **lnls start:** start all IOCs. If there are IOCs already running they will be firstly stopped. Take care when using this command, as you can stop processes that are not meant to be stopped.

Try it with sudo!

```
> sudo lnls start

Starting LNLS Services:
         Starting: motorSim
```

Check that the motor IOC really started, trying (remember to substitute YOUR_MOTOR by the record name you invented):

```
> caget YOUR_MOTOR:SIM:m1.RBV
```

• **lnls stopioc and startioc:** stop/start a given IOC. This commands work the same way as for stop/start but act only on the IOC called as a parameter.

Try it with sudo!

```
> sudo lnls stopioc motorSim

Stopping IOC motorSim:
         First killing child process: 74408
                                                [  OK  ]
> sudo lnls startioc motorSim

         Starting: motorSim
```

• **lnls tn:** does a telnet to access the IOC shell.

Try it! For this command, there is no need to use sudo.

```
> sudo lnls tn motorSim
```

Type ENTER and you will see the epics> prompt appearing. You can try the command `help` to see the IOC commands available. You can also try `dbl` inside the IOC shell to see the name of your records.

To escape again to the Linux terminal, type CTRL+] and `quit`.

To see all options provided by LNLS script, type `lnls` in the Linux terminal and see a summary of them.

But how the magic happens?

LNLS script searches inside a predefined path: /usr/local/epics/apps/config/$(EPICS_HOSTNAME). The environment variable EPICS_HOSTNAME must be defined previously in the system.

When a start/restart/startioc/rioc command is issued by the script, it searches for all *.cmd files inside the predefined path. During the computer boot, the process is the same. For each *.cmd file found, a procServ service is started running the EPICS script inside the cmd file. The map between IOC filename and telnet port number is registered, in a way that nobody needs to know which port relates to which IOC.

If you go to /usr/local/epics/apps/config/CENTOS.PY4SYN you will see that the only cmd file corresponds to motorSim.cmd.

```
> cd /usr/local/epics/apps/config/CENTOS.PY4SYN
> ls
motorSim.cmd  motor.substitutions  pid.txt
```

3. Py4Syn

All Py4Syn exercises will be made inside the Python shell. To start it, open a Linux terminal and type:

```
> python3.5
```

Don't close the Python shell until the end of the tutorial. We are going to use all imports and variables until the end.

3.1. Motor

In this section we will see how to operate motors inside the Python shell, using Py4Syn. In the Python terminal, type the following:

```
>>> from py4syn.epics.MotorClass import Motor
>>>
>>> # Substitute YOUR_MOTOR by the record name you
>>> # invented
>>> mtop = Motor("YOUR_MOTOR:SIM:m1", "mtop")
```

To see the EPICS motor record updating info, open another Linux terminal and type:

```
> camonitor YOUR_MOTOR:SIM:m1.RBV
```

RBV is the field responsible for showing the motor position.

Back to Python shell, let's try the following exercises:

```
>>> # To move the motor to an absolute position:
>>> mtop.setAbsolutePosition(10)
>>> # See the RBV field changing in camonitor on the
>>> # other terminal.

>>> # To move the motor by 5 units back:
>>> mtop.setRelativePosition(-5)
>>> # See the RBV field changing in camonitor on the
>>> # other terminal.

>>> # To check the velocity in units per second:
>>> mtop.getVelocity()
1.0

>>> # Too slow? Let's increase it!
>>> mtop.setVelocity(5)
```

```
>>> # Move to position 10 and 20 and see the result in
>>> # camonitor
>>> mtop.setAbsolutePosition(10)
>>> mtop.setAbsolutePosition(20)

>>> # Retrieval of motor soft limits:
>>> mtop.getHighLimitValue()
100.0
>>> mtop.getLowLimitValue()
-100.0

>>> # Try to go to a position beyond the soft limit:
>>> mtop.setAbsolutePosition(101)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/local/lib/python3.5/site-packages/Py4Syn-
0.3.0+36.gccd5471-
py3.5.egg/py4syn/epics/MotorClass.py", line 550, in
setAbsolutePosition
Exception: Can't move motor Eixo 1 (SOL1:DMC1:m1) to
desired position: 101, Target beyond value for
software high limit.
```

An error message is shown and the motor does not move.

To change soft limits:

```
>>> mtop.setHighLimitValue(105)
>>> mtop.getHighLimitValue()
105.0
>>> # Now it is possible to move to position 101
>>> mtop.setAbsolutePosition(101)
```

There is another way to work with motors in Py4Syn, by the use of motor utils functions. Try the following in a Python shell and note the difference between the previous exercises.

```
>>> from py4syn.utils.motor import *
>>>
>>> mtop = 'mtop'
>>> # Substitute YOUR_MOTOR by the record name you
>>> # invented
>>> createMotor(mtop, 'YOUR_MOTOR:SIM:m1')
>>>
>>> # To move the motor to an absolute position:
>>> # See the RBV field changing in camonitor on the
>>> # other terminal.
>>> umv(mtop, 10)

        Moving  mtop
        10.0000
>>>
>>> # To move the motor by 5 units from current
>>> # position:
>>> umvr(mtop, 5)

        Moving 5
        15.0000
>>>
>>> # To see the motor position:
>>> wmr(mtop)
15.0
>>>
>>> # To see the position of all motors:
>>> wa()

          Motor:                User:                Dial:
            mtop             15.0000              15.0000
```

```
>>>
>>> # Creating a new motor (remember to substitute
>>> # YOUR_MOTOR by the record name you invented):
>>> mbot = 'mbot'
>>> createMotor(mbot, 'YOUR_MOTOR:SIM:m2')
>>>
>>> # Observing the position of all motors again:
>>> wa()

          Motor:                User:                 Dial:
            mbot              0.0000                0.0000
            mtop             15.0000               15.0000
```

Now, with 2 motors created, open a third Linux terminal and type:

```
> camonitor YOUR_MOTOR:SIM:m2.RBV
```

as you can see both motor records updating.

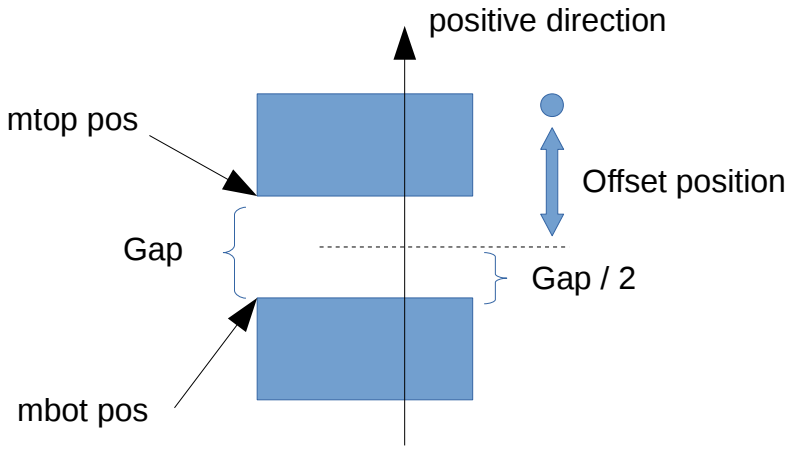
```
>>> # Moving 2 motors to an absolute position at the
>>> # same time:
>>> ummv(mtop=10,mbot=10)
>>> wa()

          Motor:                User:                 Dial:
            mbot             10.0000               10.0000
            mtop             10.0000               10.0000
>>>
>>> # Moving 2 motors from a position relative to the
>>> # current one at the same time:
>>> ummvr(mtop=5,mbot=5)
>>> wa()

          Motor:                User:                 Dial:
```

```
mbot                15.0000              15.0000
mtop                15.0000              15.0000
```

Remember to don't close the Python shell.

3.2. Pseudo-motors

Suppose the control of a slit. You want to control the gap size and the offset of the center of the gap. These two parameters can be manipulated as if they were two motor axis. Using Py4Syn you can define pseudo-motors using equations and apply them to real motors. Pseudo-motors are used by Py4Syn scripts the same as real motors.



For the following exercises, the mtop position will be related to the lower side of the upper part. mbot position will be related to the upper side of the lower part. A Gap is defined as the distance between mtop and mbot. An offset position is defined as the distance of the Gap center from a previously defined reference. We are considering the positive direction as the one that makes both motors go up.

We will use the two terminals with camonitor previously opened. m1 corresponds to mtop and m2 to mbot.

In order to define a pseudo-motor we need to:

- Write one equation to describe the pseudo-motor position based on the position of all the real motors that build the pseudo-motor.
- Write one equation for each one of the motors that build the pseudo-motor to change their targets when the user needs to change the pseudo-motor position.

```
>>> # Equations for gap
>>> gapRBV = 'A[mtop]-A[mbot]' # Equation to describe
>>>                             # gap size
>>> gapTopTarget = 'A[offset]+T[gap]/2.0' # Equation
>>>                       # to define gap mtop target
>>> gapBotTarget = 'A[offset]-T[gap]/2.0' # Equation
>>>                       # to define gap mbot target
>>>
>>> # Equations for offset
>>> offsetRBV = 'A[mtop]-A[gap]/2.0' # Equation to
>>>                           # describe offset position
>>> offsetTopTarget = 'T[offset]+A[gap]/2.0' #
>>>           # Equation to define offset mtop target
>>> offsetBotTarget = 'T[offset]-A[gap]/2.0' #
>>>          # Equation to define offset mbot target
>>>
>>> gap = 'gap'
>>> createPseudoMotor(gap, 'gap slit 1', gapRBV,
{mtop:gapTopTarget, mbot: gapBotTarget})
>>> offset = 'offset'
>>> createPseudoMotor(offset, 'off slit 1', offsetRBV,
{mtop: offsetTopTarget, mbot: offsetBotTarget})
>>> # Move mtop and mbot
>>> umv(mbot, 14)

        Moving  mbot
        14.0000
>>> umv(mtop, 16)
```

```
        Moving  mtop
        16.0000
>>> wa()

          Motor:                User:                Dial:
            mbot             14.0000              14.0000
            mtop             16.0000              16.0000
             gap              2.0000               2.0000
          offset              15.0000              15.0000
>>> umvr(offset, -5)

        Moving -5
        10.0000
>>> wa()

          Motor:                User:                Dial:
            mbot              9.0000               9.0000
            mtop             11.0000              11.0000
             gap              2.0000               2.0000
          offset              10.0000              10.0000
>>> umv(gap, 4)

        Moving  gap
        4.0000
>>> wa()

          Motor:                User:                Dial:
            mbot              8.0000               8.0000
            mtop             12.0000              12.0000
             gap              4.0000               4.0000
          offset              10.0000              10.0000
```

```
>>> umv(mbot, 3)

        Moving  mbot
        3.00000
>>> wa()

          Motor:                User:                Dial:
            mbot              3.0000               3.0000
            mtop             12.0000              12.0000
             gap              9.0000               9.0000
          offset              7.5000               7.5000
```

3.3. Counters

Counters are equipment that can count pulses. In EPICS, the record type most used to interact with a counter is the SCALER one. Py4Syn can understand a SCALER record and any other type of record that can be interpreted as a counter. For example, the temperature read from a termopar can be programmed as a counter.

For the exercises in this module, we will use simulated counters, not connected to an IOC.

```
>>> from py4syn.epics.SimCountableClass import
SimCountable
>>> from py4syn.utils.counter import *
>>> # Create a fake SCALER record
>>> fakeScaler = SimCountable('FAKEPV', 'sim')
>>> # Create 2 simulation counters for the fake scaler
>>> createCounter("Sim1", fakeScaler)
>>> createCounter("Sim2", fakeScaler)
>>>
>>> # ct() is used to shot a 1 second count and show
>>> # results in screen. If you want to count for a
>>> # time different from 1 second, just put the
```

```
>>> # desired time inside the parenthesis.
>>> ct()
--------------------------------
Counts:
--------------------------------
      Sim1                      12
      Sim2                      12
--------------------------------
>>>
>>> # ctr is used to shot a count and to retrieve
>>> # a Python dictionary containing its values.
>>> # If you want to count for a
>>> # time different from 1 second, just put the
>>> # desired time inside the parenthesis.
>>> counts = ctr(2)
>>> print(counts)
OrderedDict([('Sim1', 43.0), ('Sim2', 43.0)])
>>> counts['Sim1']
43.0
```

3.4. Scan

The built-in Scan resource of Py4Syn moves the motors to the desired positions and measure any configured counter in each position.

```
>>> from py4syn.utils.scan import *
>>> # Make a scan using mtop motor from position
>>> # 0 until position 10, with 10 intervals between
>>> # points and 0.1 seconds to the counters.
>>> scan(mtop, 0, 10, 10, 0.1)
points  mtop  Sim1  Sim2
0.0000 0.0000 6.0000 6.0000
1.0000 1.0000 0.0000 0.0000
2.0000 2.0000 1.0000 1.0000
```

```
3.0000 3.0000 4.0000 4.0000
4.0000 4.0000 2.0000 2.0000
5.0000 5.0000 1.0000 1.0000
6.0000 6.0000 8.0000 8.0000
7.0000 7.0000 3.0000 3.0000
8.0000 8.0000 1.0000 1.0000
9.0000 9.0000 8.0000 8.0000
10.0000 10.0000 9.0000 9.0000
Peak =  9.0  at  10.0
Fwhm =  125.02713844525543  at  -35.565164452884545
COM =  6.18604651163
>>>
>>>
>>> # Make a scan using mtop and mbot motors from
>>> # position 0 until position 10 for mtop and
>>> # position 0 until position 10 for mbot,
>>> # with 10 intervals between
>>> # points and 0.1 seconds to the counters.
>>> scan(mtop, 0, 10, mbot, 0, 10, 10, 0.1)
points  mtop  mbot  Sim1  Sim2
0.0000 0.0000 0.0000 7.0000 7.0000
1.0000 1.0000 1.0000 7.0000 7.0000
2.0000 2.0000 2.0000 0.0000 0.0000
3.0000 3.0000 3.0000 1.0000 1.0000
4.0000 4.0000 4.0000 6.0000 6.0000
5.0000 5.0000 5.0000 2.0000 2.0000
6.0000 6.0000 6.0000 8.0000 8.0000
7.0000 7.0000 7.0000 0.0000 0.0000
8.0000 8.0000 8.0000 6.0000 6.0000
9.0000 9.0000 9.0000 8.0000 8.0000
10.0000 10.0000 10.0000 8.0000 8.0000
Peak =  8.0  at  6.0
Fwhm =  47.47852766703641  at  -12.447665049216948
COM =  5.50943396226
```

```
>>>
>>>
>>> # How to change the chart to show mbot positions
>>> # rather than mtop ones?
>>> setX(mbot)
>>> # Now, the scan is setting mbot positions from 20
>>> # to 39. See the result in the chart.
>>> scan(mtop, 0, 10, mbot, 20, 39, 10, 0.1)
points  mtop  mbot  Sim1  Sim2
0.0000 0.0000 20.0000 4.0000 4.0000
1.0000 1.0000 21.9000 3.0000 3.0000
2.0000 2.0000 23.8000 4.0000 4.0000
3.0000 3.0000 25.7000 5.0000 5.0000
4.0000 4.0000 27.6000 3.0000 3.0000
5.0000 5.0000 29.5000 7.0000 7.0000
6.0000 6.0000 31.4000 8.0000 8.0000
7.0000 7.0000 33.3000 8.0000 8.0000
8.0000 8.0000 35.2000 5.0000 5.0000
9.0000 9.0000 37.1000 2.0000 2.0000
10.0000 10.0000 39.0000 3.0000 3.0000
Peak =  8.0  at  31.4
Fwhm =  5.183328106123741  at  32.02123618689432
COM =  29.6826923077
>>>
>>>
>>> # How to change the chart to show countings for
>>> # Sim2 rather than Sim1 ones?
>>> setY('Sim2')
>>> scan(mtop, 0, 10, mbot, 20, 30, 10, 0.1)
points  mtop  mbot  Sim1  Sim2
0.0000 0.0000 20.0000 7.0000 7.0000
1.0000 1.0000 21.0000 4.0000 4.0000
2.0000 2.0000 22.0000 9.0000 9.0000
3.0000 3.0000 23.0000 5.0000 5.0000
```

```
4.0000 4.0000 24.0000 6.0000 6.0000
5.0000 5.0000 25.0000 3.0000 3.0000
6.0000 6.0000 26.0000 8.0000 8.0000
7.0000 7.0000 27.0000 2.0000 2.0000
8.0000 8.0000 28.0000 4.0000 4.0000
9.0000 9.0000 29.0000 7.0000 7.0000
10.0000 10.0000 30.0000 7.0000 7.0000
Peak =  9.0  at  22.0
Fwhm =  5.8116025898284045  at  27.29721333390759
COM =  24.8870967742
>>>
>>>

>>> # Listing the last scan command that was used
>>> getScanCommand()
'scan(mtop, 0, 10, mbot, 20, 30, 10, 0.1)'
>>>

>>> # Scan can be used with pseudo-motors.
>>> # Let's scan the offset, given a fixed gap.
>>> umv (gap, 5)

        Moving  gap
        5.0000
>>> scan(offset, 0, 15, 10, 0.1)
points  offset  Sim1  Sim2
0.0000 0.0000 3.0000 3.0000
1.0000 1.5000 2.0000 2.0000
2.0000 3.0000 5.0000 5.0000
3.0000 4.5000 9.0000 9.0000
4.0000 6.0000 9.0000 9.0000
5.0000 7.5000 0.0000 0.0000
6.0000 9.0000 1.0000 1.0000
```

```
7.0000 10.5000 5.0000 5.0000
8.0000 12.0000 2.0000 2.0000
9.0000 13.5000 6.0000 6.0000
10.0000 15.0000 4.0000 4.0000
Peak =  9.0  at  4.5
Fwhm =  2.460178926561582  at  5.030806469492366
COM =  7.36956521739
>>> wa()

          Motor:                User:                Dial:
            gap              5.0000               5.0000
           mtop             17.5000              17.5000
         offset             15.0000              15.0000
           mbot             12.5000              12.5000
>>>
```

3.5. Mesh

Mesh moves the motors to all position combinations.

```
>>> mesh(mtop, 0, 10, 5, mbot, 20, 30, 5, 0.1)
points  mtop  mbot  Sim1  Sim2
0.0000 0.0000 20.0000 8.0000 8.0000
1.0000 0.0000 22.0000 7.0000 7.0000
2.0000 0.0000 24.0000 5.0000 5.0000
3.0000 0.0000 26.0000 4.0000 4.0000
4.0000 0.0000 28.0000 8.0000 8.0000
5.0000 0.0000 30.0000 1.0000 1.0000
6.0000 2.0000 20.0000 6.0000 6.0000
7.0000 2.0000 22.0000 9.0000 9.0000
8.0000 2.0000 24.0000 0.0000 0.0000
9.0000 2.0000 26.0000 6.0000 6.0000
10.0000 2.0000 28.0000 6.0000 6.0000
```

```
11.0000 2.0000 30.0000 6.0000 6.0000
12.0000 4.0000 20.0000 9.0000 9.0000
13.0000 4.0000 22.0000 4.0000 4.0000
14.0000 4.0000 24.0000 1.0000 1.0000
15.0000 4.0000 26.0000 6.0000 6.0000
16.0000 4.0000 28.0000 9.0000 9.0000
17.0000 4.0000 30.0000 1.0000 1.0000
18.0000 6.0000 20.0000 2.0000 2.0000
19.0000 6.0000 22.0000 1.0000 1.0000
20.0000 6.0000 24.0000 9.0000 9.0000
21.0000 6.0000 26.0000 8.0000 8.0000
22.0000 6.0000 28.0000 9.0000 9.0000
23.0000 6.0000 30.0000 2.0000 2.0000
24.0000 8.0000 20.0000 0.0000 0.0000
25.0000 8.0000 22.0000 5.0000 5.0000
26.0000 8.0000 24.0000 9.0000 9.0000
27.0000 8.0000 26.0000 8.0000 8.0000
28.0000 8.0000 28.0000 4.0000 4.0000
29.0000 8.0000 30.0000 1.0000 1.0000
30.0000 10.0000 20.0000 6.0000 6.0000
31.0000 10.0000 22.0000 5.0000 5.0000
32.0000 10.0000 24.0000 5.0000 5.0000
33.0000 10.0000 26.0000 9.0000 9.0000
34.0000 10.0000 28.0000 9.0000 9.0000
35.0000 10.0000 30.0000 1.0000 1.0000
Peak =  9.0  at  2.0
Fwhm =  63.51054843454686  at  -23.016573306087462
COM =  4.96296296296
```

3.6. Timescan

Timescan moves no motor, only measure in time.

```
>>> # Let's measure the counters 11 times using
>>> # 2 seconds between countings.
>>> timescan(2, repeat=10)
points  Sim1  Sim2
0.0000 112.0000 112.0000
1.0000 54.0000 54.0000
2.0000 4.0000 4.0000
3.0000 157.0000 157.0000
4.0000 33.0000 33.0000
5.0000 93.0000 93.0000
6.0000 162.0000 162.0000
7.0000 124.0000 124.0000
8.0000 76.0000 76.0000
9.0000 162.0000 162.0000
10.0000 133.0000 133.0000
Peak =  162.0  at  6
Fwhm =  104.75771992701688  at  49.93156996554649
COM =  5.73513513514
```