LNLS script and Py4Syn

LNLS Script General View

What is the LNLS script?

- An easy way to start / stop IOCs
- An easy way to access IOCs EPICS shell
- An easy way to know the status of all IOCs running in the machine
- Based on procServ
  - No need to know the mapping between IOCs and telnet ports
  - LNLS script takes care of port numbers transparently

LNLS Script Exercises

- Start VMware
- Start the VM for the tutorial
- If VMware asks, tell it you did a copy of the VM
- User = guest / password = guest
- If needed, user = root / password = root

Py4Syn - General View

What is Py4Syn?

Py4Syn *(Python For Synchrotron)* is:

- A Python package
- Used as a high level tier
- A Device manipulation tool
- Used for scans
- Used to see charts in real-time
- Simple

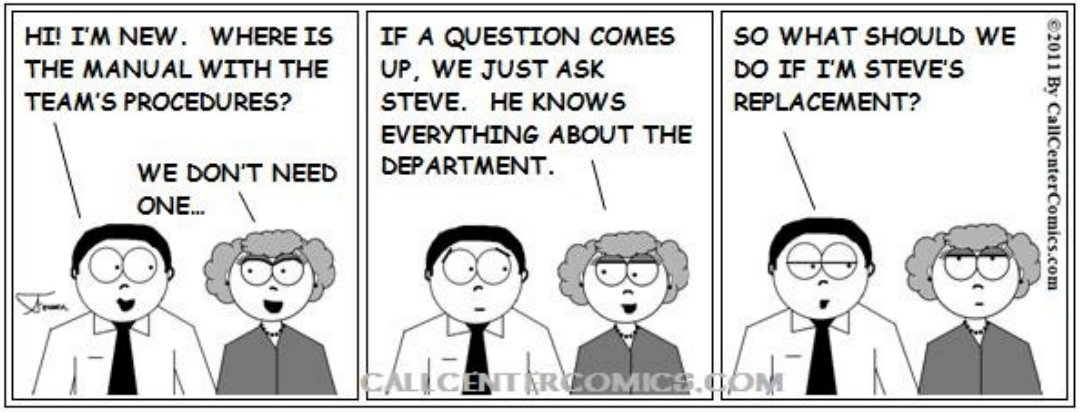
Motivation

Requirements for ideal scenario:

- Free.
- Open source.
- Easy to maintain and improve.
- Standardized but scalable to attend the most different needs.
- Support to beamline's devices.
- Motion control, counters, scans, pseudo-motors.
- Realtime charts.
- Simple syntax, clear and most closer to what the users are used to as possible.

Features

- **Devices**
  - Motors, Counters, CCDs, etc…
- **Utility functions**
  - Motor movement
  - Counting
  - Scans
  - Charts
  - Statistics
- **Simple, direct and user focused documentation.**



Motor Movement

```
from py4syn.utils.motor import *

createMotor('mtop','SOL:DMC1:m3') # Create a motor named mtop

umv('mtop', 10)  # Move mtop to absolute position 10
                 # Wait for arrival
                 # Show position while moving

wa() # Show the position of all motors created

currentPosition = wmr('mtop')# Read the current position

mv('mtop', 20, wait=False) # Move mtop to absolute position 20
                           # Do not wait for arrival
                           # Do not show position while moving

# Here code can run while motor is moving
```

Complete list in:
http://py4syn.readthedocs.org/en/latest/utils/utils_motor.html

Pseudo-Motors

Virtual motors that are described by equations relating one or more motors.

Note: There is no automatic velocity and acceleration settings to make all motors arrive in the target at the same time. The total movement time is the time given by the motor that arrives last.

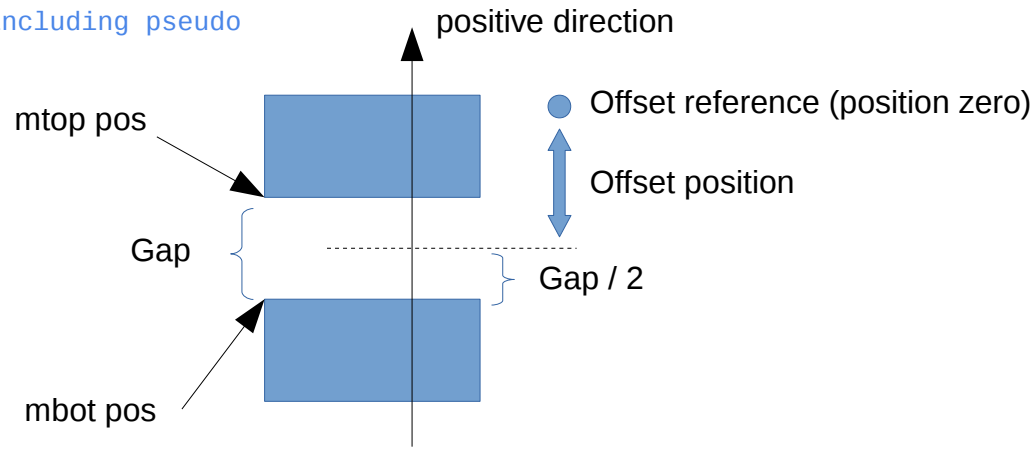
Pseudo-Motor structure:

- Name
- Description
- Readback equation (how the pseudo position is calculated)
- Target equation dictionary (how to define the target of each axis that is part of the pseudo-motor)

Pseudo-Motors

Example: create a pseudo-motor to control gap and offset of a slit

```
from py4syn.utils.motor import *

createMotor('mtop','SOL:DMC1:m3') # Create motor mtop
createMotor('mbot','SOL:DMC1:m4') # Create motor mbot

gapRBV = 'A[mtop]-A[mbot]' # Equation to describe gap size
gapTopTarget = 'A[offset]+T[gap]/2.0' # Equation to define mtop target
gapBotTarget = 'A[offset]-T[gap]/2.0' # Equation to define mbot target

offsetRBV = 'A[mtop]-A[gap]/2.0' # Equation to describe offset position
offsetTopTarget = 'T[offset]+A[gap]/2.0' # Equation to define mtop target
offsetBotTarget = 'T[offset]-A[gap]/2.0' # Equation to define mbot target

createPseudoMotor('offset', 'off slit 1', offsetRBV, {mtop: offsetTopTarget, mbot: offsetBotTarget})
createPseudoMotor('gap', 'gap slit 1', gapRBV, {mtop:gapTopTarget, mbot: gapBotTarget})

wa() # Show motor position for all motors, including pseudo
umv('gap', 5) # Move gap to 5 and wait
wa() # Show motor position for all motors, including pseudo
```



Counters

```
from py4syn.epics.ScalerClass import *
from py4syn.utils.counter import *

scaler = Scaler('SOL:SCALER', 10, 'scaler1') # Create a scaler

createCounter('det',scaler, 3) # Create a counter named det
createCounter('mon',scaler, 10) # Create a counter named mon

ct(1)# Run a 1 preset time count in all counters
                        # Show results at the end

counting = ctr(1000, use_monitor=True)# Run a counting and return data

disableCounter('det') # Disable det counter
```

Complete list: http://py4syn.readthedocs.org/en/latest/utils/utils_counter.html

Scans

There are 3 scanning types available:

- **Scan**
  - In a Scan, the number of points is the same for all used devices.
  - Devices will search for its targets at the same time.
  - There are no restrictions to the number of devices or points used.
  - There are no restrictions to devices types being used as long as the Iscannable interface is implemented.

- **Mesh**
  - In a mesh the number of points is independent for each used device.
  - For each point of a device, the next device will do a complete scan before the former device goes to the next point.
  - There are no restrictions to the number of devices or points used.
  - There are no restrictions to devices types being used as long as the IScannable interface is implemented.

- **Timescan**
  - In a timescan no devices but counters are used. This scan runs in time defined intervals.
  - It is possible to define a delay time between measurements.

scan function

Standard format of scan function:

scan(dev1, start1, end1, dev2, start2, end2,... ,devn, startn, endn, points, time)

Where:

devN is any IScannable device.

startN is the initial value.

endN is the final value.

points is the number of intervals between points to be done (Py4Syn includes initial and final values).

time is the integration time to be configured in used counters.

Optionally, the pair **startN, endN** can be changed by an array, as long as all devices use the same number of points and each array size equals the parameter “points”.

The same is applied to **time**, where an array of integration time can be used, as long as the array size equals the parameter “points”.

In a scan, the total number of points is given by “points” + 1.

scan function

Example:

- scan('m1', 0, 2, 'm2', 2, 6, 2, 0.1)

Where:

- m1 will go from 0 to 2 with steps of value 1 ((2-0)/2)
- m2 will go from 2 to 6 with steps of value 2 ((6-2)/2)
- 3 points in total (2 points + 1 final point)
- 0.1 seconds of integration time

Results in:

Point	m1	m2
0	0.0	2.0
1	1.0	4.0
2	2.0	6.0

mesh function

Standard format of mesh function:

mesh(dev1, start1, end1, points1, dev2, start2, end2, points2,... ,devN, startN, endN, pointsN, time)

Where:

devN is any IScannable device.

startN is the initial value.

endN is the final value.

pointsN is the number of intervals between points to be done (Py4Syn includes initial and final values).

time is the integration time to be configured in used counters.

Optionally, the pair **startN, endN** can be changed by an array and, in this case, the parameter pointsN is omitted.

In mesh, the total number of points is given by (points1+1)*(points2+1)*(...)*(pointsN+1).

mesh function

Example:

- mesh('m1', 0, 2, 2, 'm2', 2, 6, 2, 0.1)

Where:

- m1 will go from 0 to 2 with steps of value 1 ((2-0)/2)
- m2 will go from 2 to 6 with steps of value 2 ((6-2)/2)
- 9 points in total (2 points + 1 final point) to m1 and m2, and for mesh, total = pm1*pm2 = 3*3
- 0.1 seconds of integration time

Results in:

Point	m1	m2
0	0.0	2.0
1	0.0	4.0
2	0.0	6.0
3	1.0	2.0
4	1.0	4.0
5	1.0	6.0
6	2.0	2.0
7	2.0	4.0
8	2.0	6.0

timescan function

Standard format of timescan function:

timescan(time, delay, repeat)

Where:

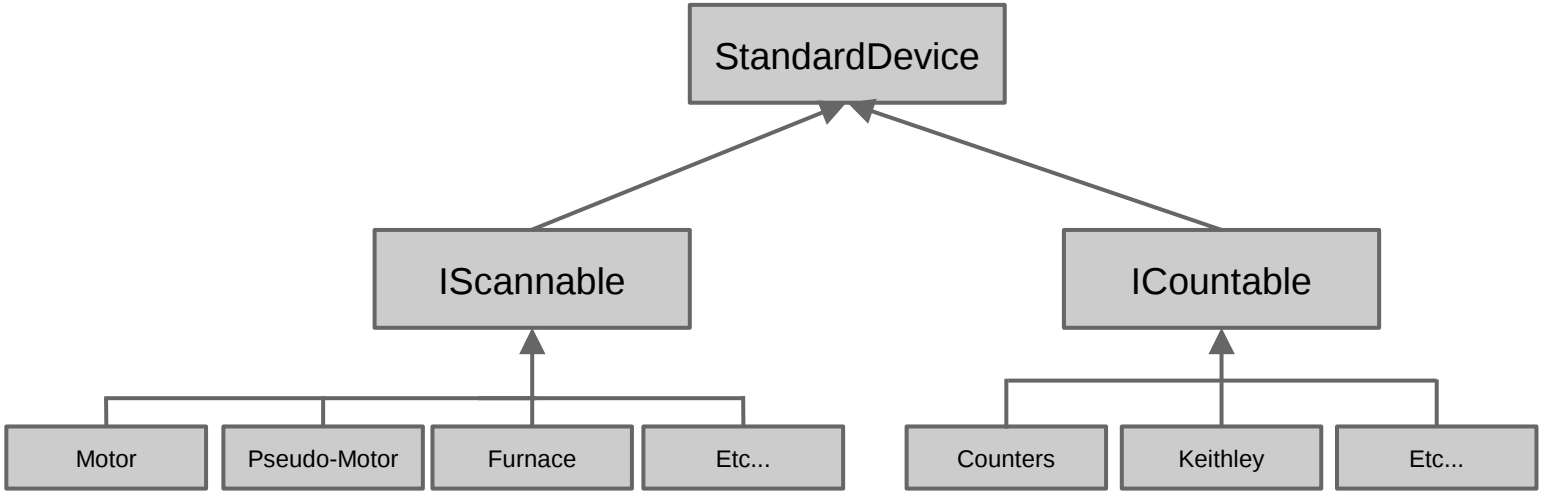
time is the integration time to be configured in used counters. Default value is 1 second.

delay is the time to wait until the next counting. Default value is 1 second.

repeat is the number of counting to be done. Default value is infinite.

Devices

Devices classification



Complete list: http://py4syn.readthedocs.org/en/latest/epics.html

Scans

In Py4Syn library, devices are divided into two groups:

IScannable

The IScannable interface defines some standard operations that all devices intended to be used during a scan must implement. For example:

- **setValue** - Define the target value
- **getValue** - Read the current value
- **wait** - Wait until the target is hit
- **getLowLimit -** Read low limit
- **getHighLimit** - Read high limit

ICountable

The ICountable interface defines some standard operations that all devices intended to be used as counters must implement:

- **getValue** – Read the current value
- **setCountTime** - Define integration time
- **setPresetValue** - Define preset value
- **startCount –** Start a counting
- **stopCount**- Stop a counting
- **canMonitor** – Define if it can be used as a monitor
- **canStopCount** - Define if counting can be interrupted
- **isCounting –** Verify if counting is being done
- **wait –** Wait until counting ends

These special interfaces allow a generalization of devices for scanning functions independently of its type: a motor, a furnace, a MCA, a CCD, etc.

Callbacks

Independently of the scan type, many customizations are possible to the actions by means of user defined functions (*Callbacks*).

There are 7 available callbacks for all scan types:

- **Pre Scan Callback**
  - Called before the start of a scan.
- **Pre Point Callback**
  - Called before any device is commanded to its target.
- **Pre Operation Callback**
  - Called after all devices arrive in the target and before the start of the counters.
- **Operation Callback**
  - Called while the counters are counting.
- **Post Operation Callback**
  - Called after the counting and before the chart updates.
- **Post Point Callback**
  - Called after all operations and before the start of a new point.
- **Post Scan Callback**
  - Called after the end of the scan.

Callbacks

Example: Defining a callback to be called during the counting

```
from py4syn.utils.scan import *

def myCallback(**kwargs): # Function that will be called during the operation.
   scanObject = kwargs['scan'] # Reference to scan object.
   indexArray = kwargs['idx'] # Reference to the point index.
   positionArray = kwargs['pos'] # Reference to the devices position.
   print('Callback message')

setPreScanCallback(defaultPreScanCallback)  # Configuring PreScanCallback to default.

setOperationCallback(mycallback)  # Configuring OperationCallback to our function.

scan('m1', 0, 180, 10, 1) # Running the scan with the configuration.
```

Charts

There is a need to show data in real-time and the need that data rendering would not generate lateness in running scans.

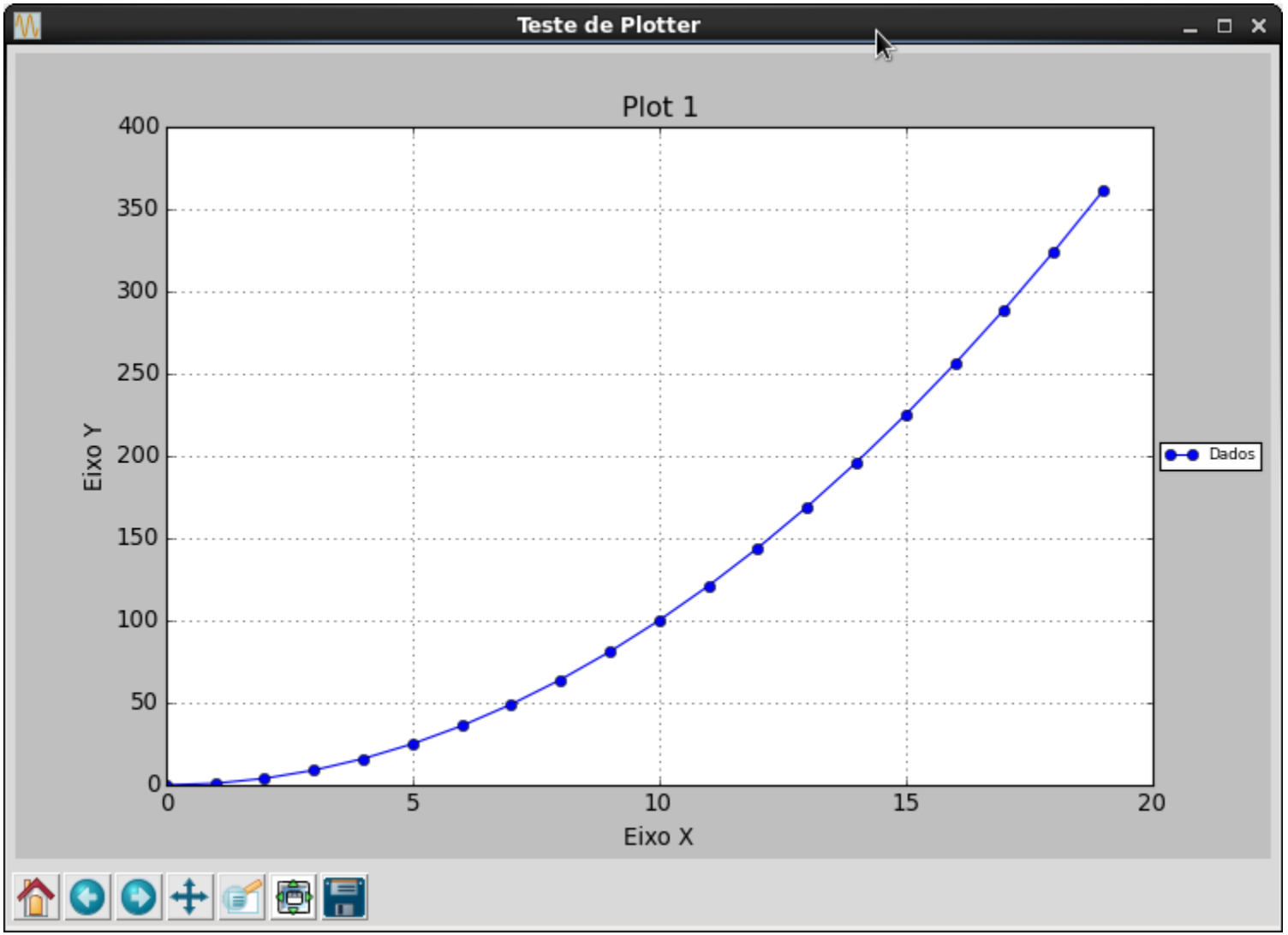
- Real-time rendering.
- Many charts in the same windows.
- Data overlap in the same chart.

Example 1: Simple chart

```
import time
from py4syn.utils.plotter import *

pl = Plotter("Teste de Plotter", daemon=False) # Create a Plotter (Window)
pl.createAxis("Plot 1", xlabel="Eixo X", ylabel="Eixo Y", grid=True,
line_style="-", line_marker="o", line_color="blue", label="Dados")  # Create a
new axis

for i in range(0, 20):
    pl.plot(i, i*i, axis=1) # Put a new point in chart
    time.sleep(0.1)
```

Complete list: http://py4syn.readthedocs.org/en/latest/utils/utils_plotter.html

Charts


Teste de Plotter
Plot 1
Eixo Y
Eixo X
Dados
0
50
100
150
200
250
300
350
400
0
5
10
15
20

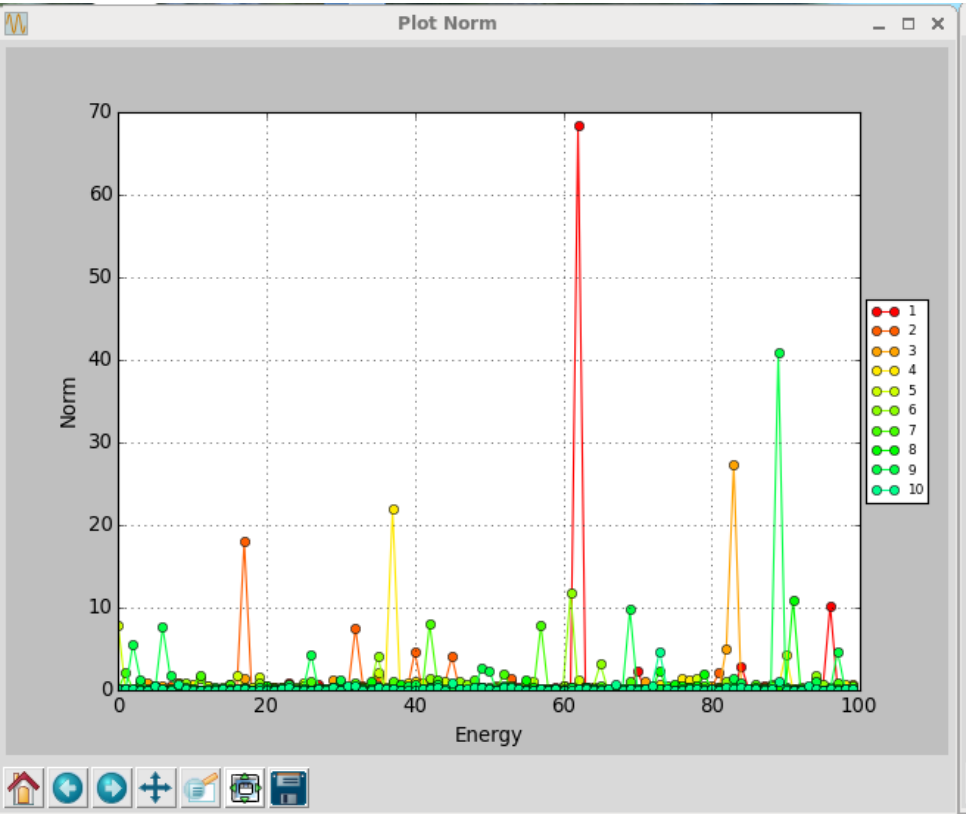
Charts

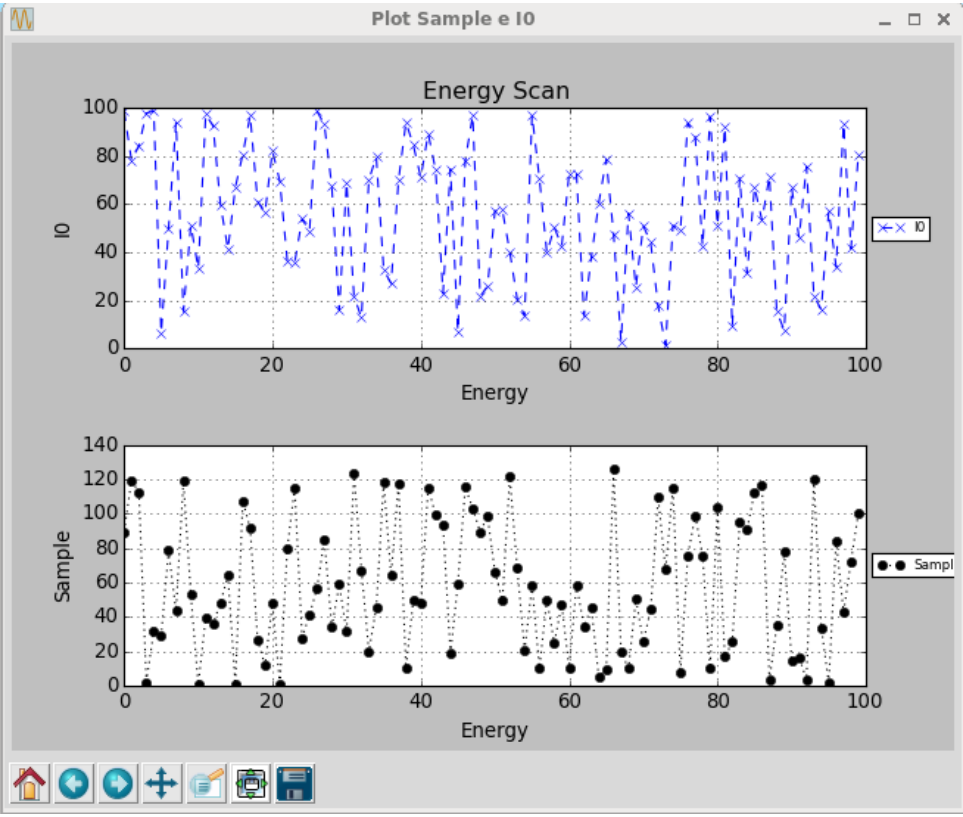
Example 2: Advanced chart

```
import time
from py4syn.utils.plotter import *

N = 100 # Number of points in chart
pl = Plotter('Plot Sample e I0', daemon=True) # Create a Plotter for Sample and I0
pl.createAxis(title="Energy Scan", xlabel="Energy", ylabel="I0", grid=True, line_style="--",
line_marker="x", line_color="blue", label="I0") # Create an axis for I0
pl.createAxis(title="", xlabel="Energy", ylabel="Sample", grid=True, line_style=":", line_marker="o",
line_color="black", label="Sample") # Create an axis for Sample

pl1 = Plotter('Plot Norm', daemon=True) # Create a Plotter for normalized data
pl1.createAxis(title="", xlabel="Energy", ylabel="Norm", grid=True, label="1")# Create an axis for
Norm

idx = 1
for i in range(0, 10): # Number of scans we will run
    if(i >= 1):
        idx += 1
        pl1.createAxis(title="", parent=1, label=str(i+1)) # For each scan, create a new axis for Norm

    pl.clear(1) # For each scan, clean I0 chart
    pl.clear(2) # For each scan, clean Sample chart

    for ii in range(N): # For each point
        i0 = random.random()*100 # Generate a value for I0
        sample = random.random()*127 # Generate a value for Sample
        norm = (sample/i0)/10 # Calculate normalization
        pl.plot(ii, i0, axis=1) # Update I0 chart
        pl.plot(ii, sample, axis=2)  # Update Sample chart
        pl1.plot(ii, norm, axis=idx) # Update Norm chart in the correct axis
        time.sleep(0.01)
```

Complete list: http://py4syn.readthedocs.org/en/latest/utils/utils_plotter.html

Charts


Plot Norm
Norm
Energy
0
10
20
30
40
50
60
70
0
20
40
60
80
100
1
2
3
4
5
6
7
8
9
10


Plot Sample e I0
Energy Scan
I0
Energy
Sample
Energy
0
20
40
60
80
100
120
140
I0
Sampl

Special Variables

In scan module it is possible to access special parameters by its read and write methods. For example:

- SCAN_DATA – Dictionary that saves all scan data.
- SCAN_CMD – Last scan command run.
- XFIELD / YFIELD – X/Y axis of the automatic chart.
- PEAK / PEAK_AT – Value and position of the peaks among the data.
- FIT_SCAN – Define if data might or might not be automatically fit at the scan's end.
- PLOT_GRAPH – Define if data might or might not be automatically shown in chart.
- among others…

Complete list: http://py4syn.readthedocs.org/en/latest/utils/utils_scan.html#special-variables-in-the-scan-module

Data

Focusing in better performance and the lowest possible dead-time, data is kept in memory during data registration. Only at the end of a scanning (by error or success) data is transferred to disk. To change this default behavior, one must use the function setPartialWrite(True). Doing so, data is saved at each iteration.

The standard file format is the same used by SPEC and PyMCA because beamlines in LNLS are historically used to it. This format is supported by many analysis programs, too.

Example:

```
#E 1413896783
#D Tue Oct 21 11:06:23 2014
#C py4syn User = user.name
#C This is a comment, ignore it
#S 1 scan(gap, 10, 11, 10, 0.01)
#D Tue Oct 21 11:06:15 2014
#T 0.01  (Seconds)
#N 7
#L points  gap  seconds  I0  cyber  testeField  timestamp
0 10.0 7.495 0.50 0.733 N/A 2014-10-21 11:06:17.423802
1 10.1 5.973 0.40 0.496 N/A 2014-10-21 11:06:17.988520
…
…
```

Data

Saving data using a different format:

- Create a class that implements FileWriter interface
- Methods to be implemented:
  - writeHeader(self) – Write the header to a file
  - writeData(self, partial=False, idx=-1) – Write data to file
  - close(self) – Close the file
- Pass the new class to scan object:
  scan.setFileWriter(writer)

writeData must use data information provided by SCAN_DATA dictionary. Its structure is described in http://py4syn.readthedocs.org/en/latest/utils/utils_scan.html#handling-the-data

Statistics

Currently the interpolation module has only a Gaussian one and gives information such:

- Full width at half maximum / Position
- Maximum / Position
- Minimum / Position
- Center of mass / Position

Example:

```
import pylab as pl
from py4syn.utils.fit import *
data = 'testdata.txt'
X = pl.loadtxt(data); # Load a data file
x = X[:,0]; # Read X values
y = X[:,7]; # Read Y values

pkv, pkp, minv, minp, fwhm, fwhmp, com = fitGauss(x, y) # Run fitGauss
print("Peak ", pkv, " at ", pkp) # Print values and its calculated positions
print("Min ", minv, " at ", minp)
print("Fwhm ", fwhm, " at ", fwhmp)
print("COM = ", com)
```

Py4Syn

Py4Syn is open-source. Code can be get at https://github.com/hhslepicka/py4syn

Complete documentation: http://py4syn.lnls.br

Published paper: http://scripts.iucr.org/cgi-bin/paper?S1600577515013715

Laboratório Nacional
de Luz Síncrotron