Status of PyTimber and PjLSA

V. Baggiolini, R. De Maria, M. Hostettler, T. Levens

Accelerator Software stack

Analysis

Python

pjlsa, pytimber, pyjapc
cmmnbuild-dep-manager

jpype

Operations

Java

LSA, CALS, JAPC, …

CMW

Hardware control

C++

FESA

Simplified view of the present situation

PyTimber

PyTimber -- A Python wrapper of Cern Accelerator Logging System (CALS) API

Extract machine data store in the Logging Database.

Can be used in the GPN and technical network.

Sources, documentation and issues in https://github.com/rdemaria/pytimber

Examples in http://swan.web.cern.ch/content/accelerator-complex

Installed in LCG software distribution and SWAN.

Authors: R. De Maria, T. Levens, C. Hernalsteens, M. Betz, M. Fitterer,R. Castellotti.

PyTimber example

In [1]:
```
import pytimber #import package
db= pytimber.LoggingDB() #start JVM if not already started, connect to CALS
```

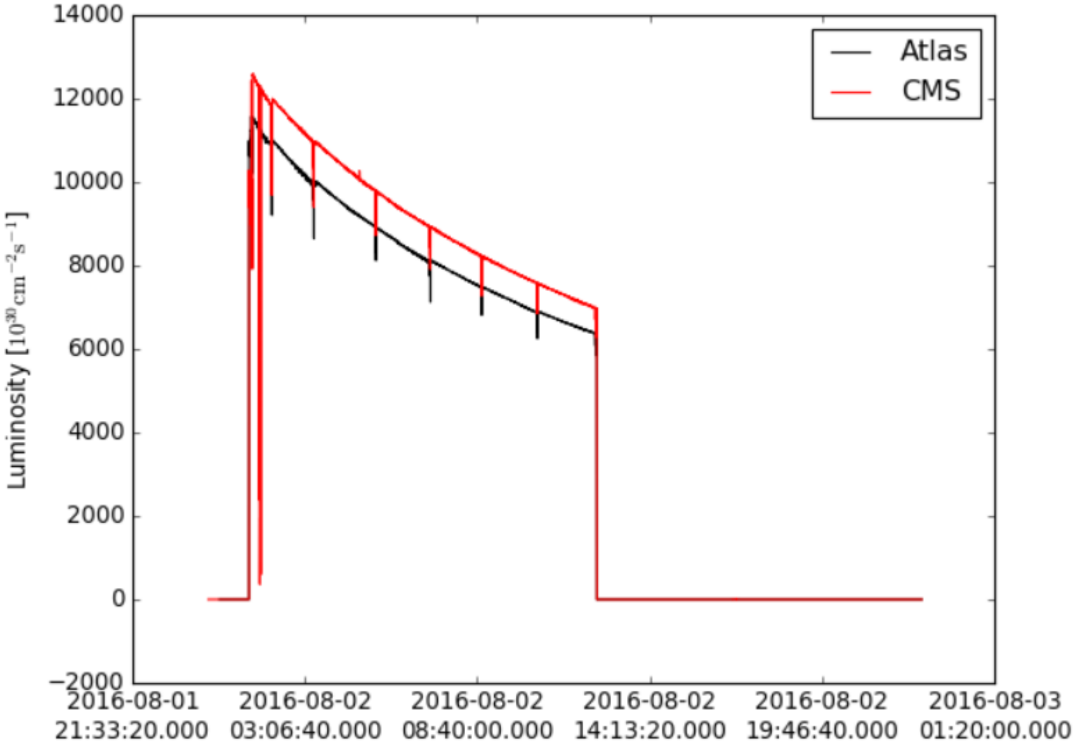
In [2]:
```
db.search("%:LUMI_TOT_INST") # search for variables
```

Out[2]:
```
[u'ALICE:LUMI_TOT_INST',
 u'ATLAS:LUMI_TOT_INST',
 u'CMS:LUMI_TOT_INST',
 u'LHCB:LUMI_TOT_INST',
 u'LHCF:LUMI_TOT_INST',
 u'TOTEM:LUMI_TOT_INST']
```

In [3]:
```
data=db.get(["ATLAS:LUMI_TOT_INST","CMS:LUMI_TOT_INST"],
            "2016-08-02 00:00:00", "2016-08-02 23:00:00") # query
```

In [4]:
```
# if in Jupyter notebook (SWAN) enable interactive plots
%matplotlib notebook
```

In [5]:
```
import matplotlib.pyplot as plt # import generic plotting library
tt,vv=data["ATLAS:LUMI_TOT_INST"]  # extract array of timestamps and records
plt.plot(tt,vv,'-k',label="Atlas") # do plot
tt,vv=data["CMS:LUMI_TOT_INST"]  # extract array of timestamps and records
plt.plot(tt,vv,'-r',label="CMS") # do plot
plt.ylabel(r'Luminosity [$10^{30} \rm cm^{-2}  s^{-1}$]')
plt.legend()
pytimber.set_xaxis_date() # set nice time axis
```



More examples and tutorial… Open in SWAN

Pytimber design goal

Provide a simple interface to CALS for interactive data analysis:

- Get data with a few lines of code
- Work well in the typical scientific Python ecosystem (numpy, Jupyter, etc)
- Use native python objects for input and output: string, list/dict, *np.array*
- Expose a stable and predictable API
- Keep performance under control
- Available from lxplus, TN consoles, SWAN, users machines
- Reduce prerequisites to the minimum

PyTimber developments

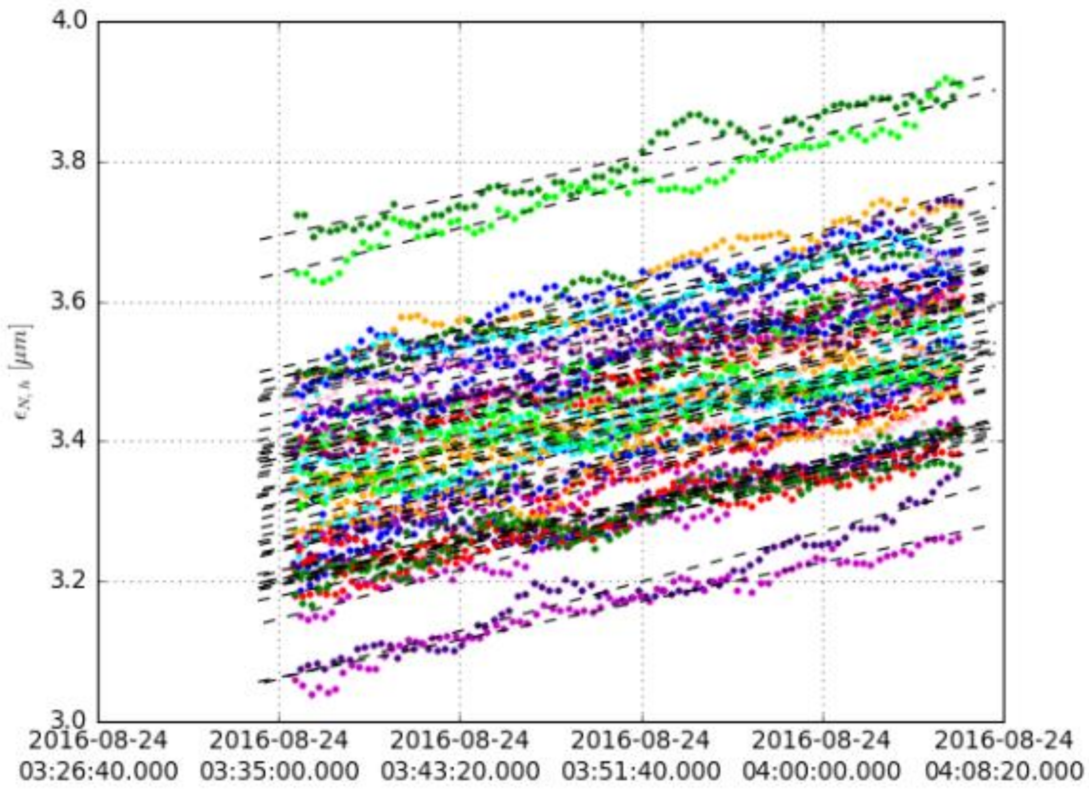
- Developed as an “open source” model… contributions welcome.
- Work started on “helper classes” for commonly required extractions
- Example LHC BSRT:

```
In [4]: import pytimber
        from pytimber import BSRT

In [5]: t1=pytimber.parsedate("2016-08-24 03:34:00.000")
        t2=pytimber.parsedate("2016-08-24 04:08:00.000")
        bsrt = BSRT.fromdb(t1,t2,beam='B1')

In [6]: bsrt = BSRT.fromdb(t1,t2,beam='B1')
```

```
In [8]: bsrt.plot(plane='h')
```




```
Out[8]: <pytimber.LHCBSRT.BSRT at 0x7fca1474e750>
```

Pytimber development and outlook

Development in a collaboration ABP, BI. Basic functionalities stable, users are asking for more features and often contribute to them.

CO is focused on NXCals that will solve problems of data retention, access speed and flexibility of CALS using Hadoop technologies and PySpark for big data analysis (https://wikis.cern.ch/display/NXCALS/NXCALS+Home)

NXCals under testing and expected in production during LS2.

Pytimber may or may not continue as a simple layer on top of NXCals and/or leave as centralized repository of equipment classes (e.g. BSRT).

PjLSA

PjLSA -- A Python wrapping of LHC Software Architecture (LSA) API

Extract machine settings. Read-only mode available from GPN.

Source and example in https://github.com/rdemaria/pjlsa

Very early stage of development, no stable API yet.

Authors:

R. De Maria, M. Hostettler, V. Baggiolini

PjLSA example

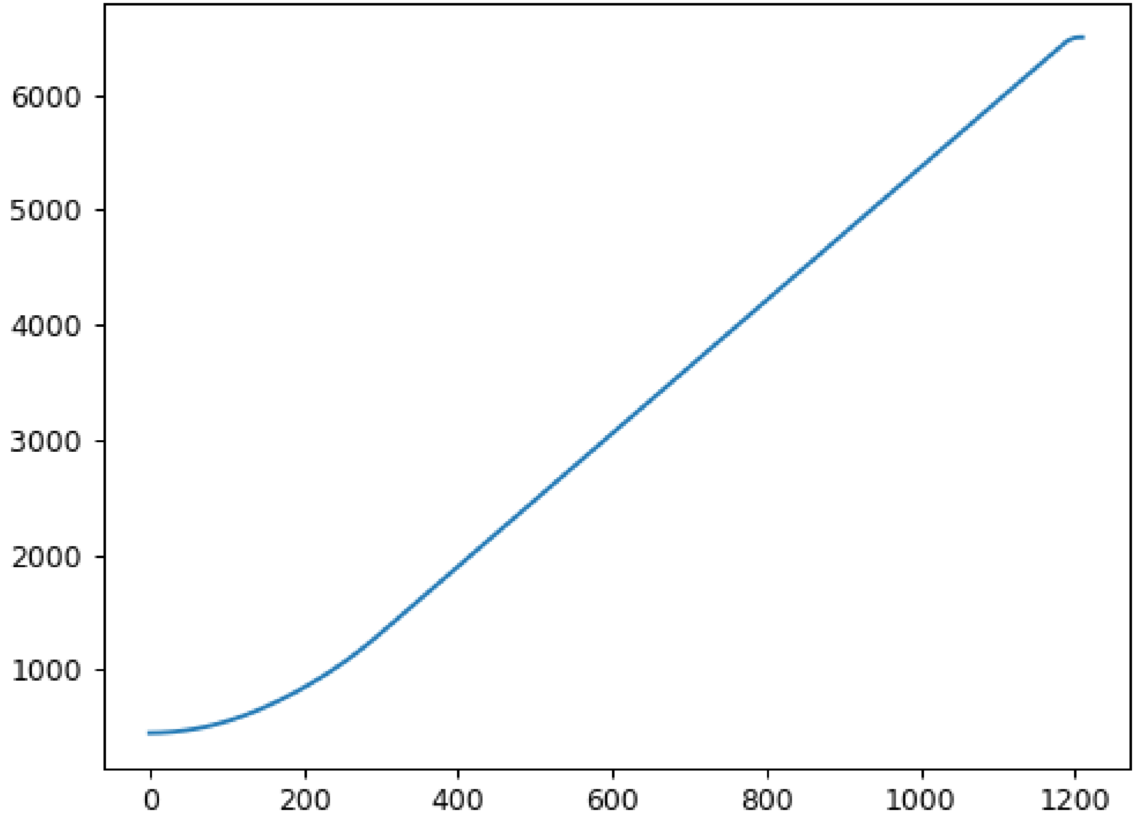
```
In [1]: import pjlsa
```

```
Checking version lsa-domain-cern-10.8.0.jar vs PRO=10.4.2
Checking version lsa-client-common-10.6.0.jar vs PRO=10.0.1
Checking version lsa-domain-10.10.0.jar vs PRO=10.4.1
Checking version lsa-client-10.11.0.jar vs PRO=10.5.2
```

```
In [2]: lsa=pjlsa.LSAClient()
```

```
In [3]: bp='RAMP-SQUEEZE-6.5TeV-ATS-1m-2017_V3_V1'
        ts,(steps,values)=lsa.getLastTrim(bp,'LHCBEAM/MOMENTUM')
```

```
In [5]: %matplotlib notebook
        import matplotlib.pyplot as plt
        plt.plot(steps,values)
```



PjLSA examples

```
In [6]: lsa.getOpticTable('RAMP-SQUEEZE-6.5TeV-ATS-1m-2017_V3_V1')

Out[6]: [OpticTableItem(time=0, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=15, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=30, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=45, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=60, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=90, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=120, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=160, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=200, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=250, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=300, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=350, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=365, id=3789L, name=u'R2017a_A11mC11mA10mL10m'),
 OpticTableItem(time=452, id=3795L, name=u'R2017a_A970C970A10mL970'),
 OpticTableItem(time=491, id=3791L, name=u'R2017a_A920C920A10mL920'),
 OpticTableItem(time=536, id=3802L, name=u'R2017a_A850C850A10mL850'),
 OpticTableItem(time=595, id=3797L, name=u'R2017a_A740C740A10mL740'),
 OpticTableItem(time=669, id=3794L, name=u'R2017a_A630C630A10mL630'),
 OpticTableItem(time=768, id=3792L, name=u'R2017a_A530C530A10mL530'),
 OpticTableItem(time=842, id=3787L, name=u'R2017a_A440C440A10mL440'),
 OpticTableItem(time=882, id=3799L, name=u'R2017a_A360C360A10mL360'),
 OpticTableItem(time=931, id=3796L, name=u'R2017a_A310C310A10mL310'),
 OpticTableItem(time=971, id=3803L, name=u'R2017a_A230C230A10mL300'),
 OpticTableItem(time=1002, id=3807L, name=u'R2017a_A180C180A10mL300'),
 OpticTableItem(time=1047, id=3804L, name=u'R2017a_A135C135A10mL300'),
 OpticTableItem(time=1121, id=3788L, name=u'R2017a_A100C100A10mL300'),
 OpticTableItem(time=1210, id=3788L, name=u'R2017a_A100C100A10mL300')]
```

```
In [8]: ot = lsa.getOpticTable('PHYSICS-2.51TeV-4m-2015_V1@90_[END]')
        lsa.getKnobFactors('LHCBEAM2/IP1_SEPSCAN_Y_MM', ot[0])

Out[8]: {u'RCBCV5.R1B2/KICK': -8.138696574e-05,
 u'RCBCV6.L1B2/KICK': -3.353022415e-05,
 u'RCBCV7.R1B2/KICK': 0.0,
 u'RCBYV4.L1B2/KICK': 0.0,
 u'RCBYVS4.L1B2/KICK': 5.584873181e-05,
 u'RCBYVS4.R1B2/KICK': 0.0001292294752}
```

```
In [9]: lsa.getParameterHierarchy('LHCBEAM1/IP1_SEPSCAN_X_MM')

Out[9]: {'I': ['RCBCH6.L1B1/I',
  'RCBYHS4.R1B1/I',
  'RCBYH4.L1B1/I',
  'RCBYHS4.L1B1/I',
  'RCBCH5.R1B1/I'],
 'IREF': ['RPLB.RR13.RCBCH6.L1B1/IREF',
  'RPLB.RR17.RCBYHS4.R1B1/IREF',
  'RPLB.RR13.RCBYH4.L1B1/IREF',
  'RPLB.RR13.RCBYHS4.L1B1/IREF',
  'RPLB.RR17.RCBCH5.R1B1/IREF'],
 'K': ['RCBCH6.L1B1/KICK',
  'RCBYHS4.R1B1/KICK',
  'RCBYH4.L1B1/KICK',
  'RCBYHS4.L1B1/KICK',
  'RCBCH5.R1B1/KICK'],
 'KNOB': ['LHCBEAM1/IP1_SEPSCAN_X_MM'],
 'K_SMOOTH': ['RCBCH6.L1B1/K_SMOOTH',
  'RCBYHS4.R1B1/K_SMOOTH',
  'RCBYH4.L1B1/K_SMOOTH',
  'RCBYHS4.L1B1/K_SMOOTH',
  'RCBCH5.R1B1/K_SMOOTH']}
```

PjLSA examples

```
In [17]: t1='2015-11-22 00:00:00'
         t2='2015-11-23 00:00:00'
         lsa.getTrims(beamprocess="PHYSICS-2.51TeV-4m-2015_V1@90_[END]",
                      parameter="LHCBEAM2/IP1_SEPSCAN_Y_MM",
                      start=t1, end=t2)

Out[17]: {u'LHCBEAM2/IP1_SEPSCAN_Y_MM': TrimTuple(time=[1448173581L, 1448173597L, 1448173615L], data=[0.25984, 0.29728
         , 0.27937])}
```

```
In [14]: dev=lsa._deviceService.findPowerConverterInfo('RPHGA.RR57.RQ7.R5B1')
         print(dev.getINom())
         print(dev.getDidtMin())
         print(dev.getAccelerationLimit())

         5390.0
         -16.167
         2.0
```

PjLSA Design Goal

As Pytimber:

- Get data with a few lines of code
- Work well in the typical scientific Python ecosystem (numpy, Jupyter)
- Use native python objects for input and output: string, list/dict, *np.array*
- Expose a stable and predictable API (not finished yet)
- Available from lxplus, TN consoles, SWAN (not done yet), users machines
- Reduce prerequisites to the minimum

PjLSA development

Development in a collaboration between ABP, CO, OP.

Next steps:

- Continue to explore needs to define a stable API
- Isolate Python data layer from Java layer (full Java API is available through the underscored services but discouraged)
- Develop a robust language agnostic backend and separate Java interface