beamline Documentation

Release 1.3.6

Tong Zhang

Oct 30, 2016

beamline Python package

`beamline`: accelerator online-modeling toolkits.

Install: `pip install beamline`

PDF documentation: `Download`

Author Tong Zhang

E-mail zhangtong@sinap.ac.cn

Date 2016

CHAPTER ONE

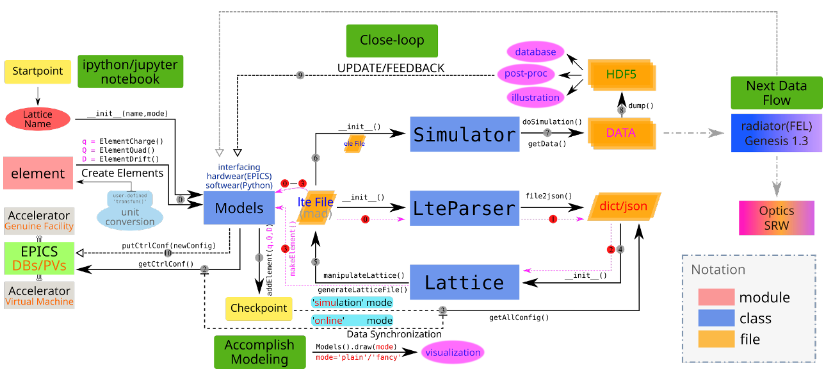
INTRODUCTION

Python package `beamline` is created for accelerator online-modeling requirement, trying to supply both CLI and GUI working environment, upon the already built infrastructure, extensions/tools could be developed to solve specific problems.

Interfaces between EPICS control environment have been designed, manually modeling machine could be achieved following the examples that this manual provides, automatic modeling from lattice file is also supported.

Tracing back to the very beginning of this package, ideas about to build a full-featured OOP high-level development environment by Python has been incubated, as well as the highly flexibility to meet different requirements.

Below is the design picture inside this package, to accomplish various goals:



- Parsing `Elegant` (electron accelerator tracking code) lattice file (`.lte`) to be python dict or json string for further operations.
- Modeling accelerator magnetic elements, such as dipole, quadrupole, drift, etc. to be python objects, from EPICS control environment to OOP level.
- Automatic modeling from `.lte` lattice file definition with postfixed `!epics` anotation to define EPICS control configurations.
- Support unit conversion between EPICS PV raw value and the physical real value of elements.
- Modeling lattice beamline from modeled elements, constructing Lattice instance, dumping `.lte` file for code tracking.
- Feeding defined elements with new configuration, interfacing with EPICS environment, to form the close-loop online system.
- Visualizing the lattice layout by predefined elements' style.
- Friendly native-look GUI application to facilitate these (part of) functionalities.

CHAPTER

TWO

DEPLOYMENT

Deploy `beamline` to different operating systems is quite simple, both online and offline approaches are provided. Before installing this package into system, there may be packages/libraries dependence issues to be resolved first.

Prerequisites

Required Python packages: `pyrpn`, `h5py`, `numpy`, `scipy`, `matplotlib`, `wxPython`, `pyepics`.

Optional Python packages: `sdds`, one compiled wheel package by @smartsammler could be found at sddswhl.zip, or see more information at http://www.aps.anl.gov/Accelerator_Systems_Division/Accelerator_Operations_Physics/software.shtml#PythonBinaries.

Packages with version infomation:

```
numpy      : 1.11.1
h5py       : 2.6.0
matplotlib : 1.5.1
pyepics    : 3.2.6
pyrpn      : 1.0.3
wxPython   : 3.0.2.0
```

Note:

- `wxPython` should be built with unicode support
- Install them by `pip install <package_name>` (recommanded)

Installation

Online approach

```
pip install beamline --prefix=/opt/high-level-apps -i https://pypi.python.org/pypi
```

or precisely define the version number:

```
pip install beamline==1.3.5 --prefix=/opt/high-level-apps -i https://pypi.python.org/pypi
```

Offline approach

Download beamline from PyPI.

```
pip install beamline-1.3.5-py2.py3-none-any.whl --prefix=/opt/high-level-apps
```

Note:

- `root` privilege may be asked.
- System wide environment variables, like `PATH`, `PYTHONPATH` and system menu integration issues are handled when deploying another package — `felapps`, details please see its documentation.

CHAPTER

THREE

EXAMPLES AND DEMONSTRATIONS

Example 1

Use deprecated modules from `beamline` package: `blparser`, `elements` and `pltutils` to do simple lattice visualization task.

Below is the lattice definition file (`ele.list`)to be used:

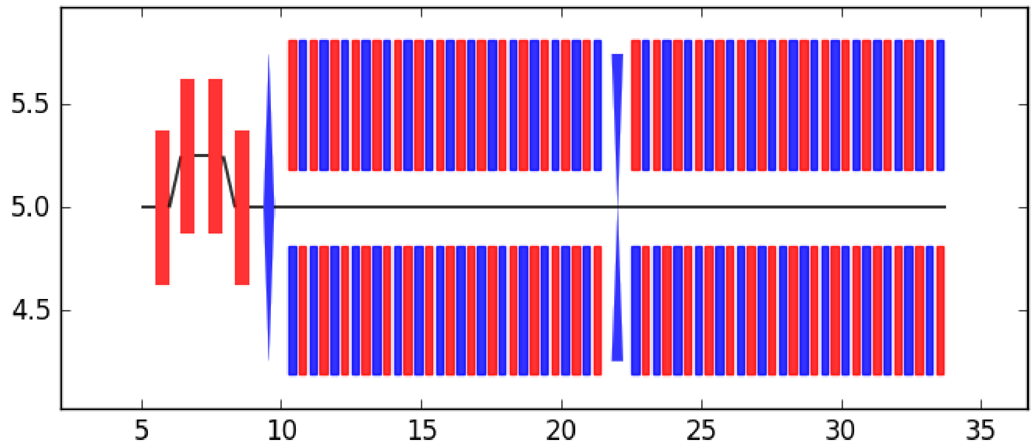
```
# lattice file for demostrations
D1: drift, l = 0.5
B1: rbend, l = 0.5, angle =  30
D2: drift, l = 1
B2: rbend, l = 0.5, angle = -30
D3: drift, l = 1
B3: rbend, l = 0.5, angle = -30
D4: drift, l = 1
B4: rbend, l = 0.5, angle =  30
D5: drift, l = 1
Q1: quad, k1= 20, l = 0.4, angle = 75
Q2: quad, k1=-20, l = 0.4, angle = 75
Q3: quad, k1= 20, l = 0.4, angle = 75
Q4: quad, k1=-20, l = 0.4, angle = 75
U1: undulator, xlamd = 0.5, nwig = 15

BL: line = (D1, B1, D1, B2, D1, B3, D1, B4, D1, Q1, D1, U1, D1, Q2, D1, U1)
BL2: line = 
↪(D1,D2,Q1,D1,B1,D2,B2,D3,U1,Q2,D3,B3,D4,B4,D5,Q1,U1,D1,D2,D3,Q1,D1,B1,D2,B2,D3,Q2,D3,B3,D4,B4,D5,Q1,D2,D2,Q3,D2,D2,Q4,D2,D2,Q2,D3,B3,D4,B4,D5,Q1,D2,D2,Q
BL3: line = (B1,D2,B2,D3,B3,D4,B4,D5,U1)
```

> **Warning:** Note that this module only support beamline definition with all elements on the same line. To be more flexible, use the new parsing modules.

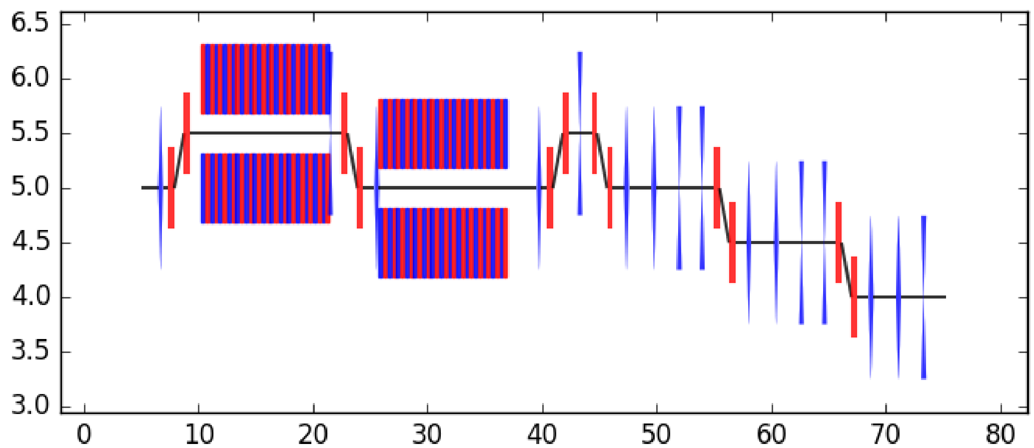
Now visualization beamline named `BL` could be achieved by following steps:

```python
from beamline import blparser
from beamline import pltutils
beamlinelist = blparser.madParser('ele.list', 'BL')
beamlineplot, xlim, ylim = pltutils.makeBeamline(beamlinelist, startpoint=(5, 5))
pltutils.plotLattice(beamlineplot, xranges=xlim, yranges=ylim, zoomfac=1.2, fig_size=8, fig_ratio=0.4)
```

Which should show figure like:



And if choosing `BL2`, should get figure like:



Example 2

Manually modeling accelerator machine step by step, by using `beamline` modules: `element`, `lattice`, `models` and `simulation`, etc.

Warming Up

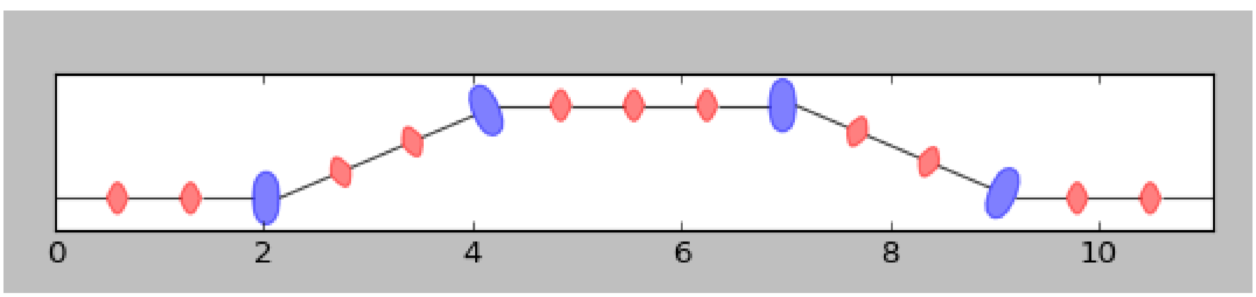
```
#!/usr/bin/python

import beamline
import matplotlib.pyplot as plt


# define elements by ``Element*`` class in ``beamline`` package
q  = beamline.ElementCharge(name='q',)
q1 = beamline.ElementQuad(name='Q1', config='k1=10,l=0.2')
b1 = beamline.ElementCsrcsben(name='B1', config='angle=+0.4,l=0.25')
b2 = beamline.ElementCsrcsben(name='B2', config='angle=-0.4,l=0.25')
b3 = beamline.ElementCsrcsben(name='B3', config='angle=-0.4,l=0.25')
b4 = beamline.ElementCsrcsben(name='B4', config='angle=+0.4,l=0.25')
```

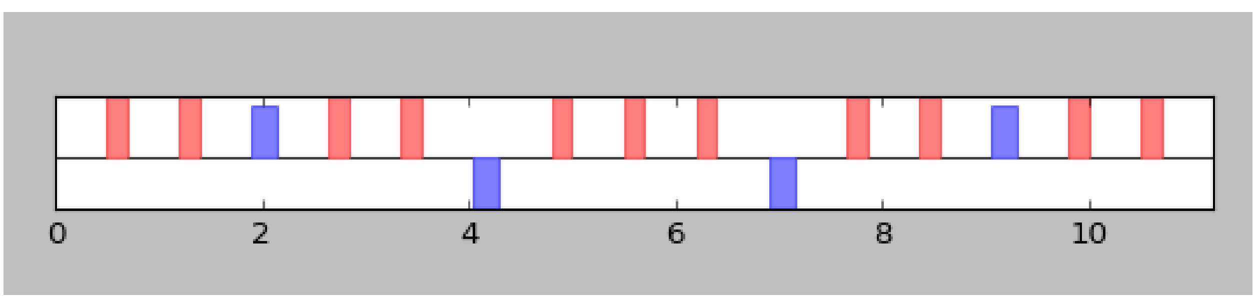
```
d1 = beamline.ElementCsrdrift(name='D1', config='l=0.5')

# create lattice model by ``Models`` class
lattice = beamline.Models(name='BL', mode='simu')
qline = (d1, q1, d1)
chi1   = (q1, d1, b1, d1, q1, d1, q1, d1, b2, d1, q1)
chi2   = (q1, d1, b3, d1, q1, d1, q1, d1, b4, d1, q1)
lattice.addElement(q, qline, chi1, qline, chi2, qline)
#ptches, anotelist, xrange, yrange = lattice.draw(showfig=False, mode='fancy')
ptches, anotelist, xrange, yrange = lattice.draw(showfig=False, mode='plain')

# show lattice in "fancy" mode
fig = plt.figure()
ax = fig.add_subplot(111, aspect='equal')
[ax.add_patch(i) for i in ptches]
ax.set_xlim(xrange)
ax.set_ylim(yrange)
ax.set_yticks([])
#plt.grid()
plt.show()
```

The above snippet shows how to build a simple lattice line from scratch, that is:

- Create elements, one by one, with `Elements*` class[es] that within `beamline` package, always assign the element with a meaningful name, it's better to follow some naming rules;
- Create lattice model instance, by the `Models` class, define the beamline name to be modeled by `name` keyword, and `mode` keyword to indicate which mode should be modelled, could be selected from `simu` and `online` options, which stands for `simulation` mode and `online` mode, if `mode='simu'`, then all parameter values should be assigned with fixed values (usually defined in the element creation step), while `mode='online'` would trig such on-line procedure:
  - The data acquiring process will try to get the value from control fields, e.g. some EPICS PV string named channels;
  - If failed, then rollback to simulation mode, i.e. using the values that defined for simulation mode;
  - The EPICS PV control fields could be defined by `setConf(type='ctrl')` method for all `Element*` classes.

`lattice` is now an instance of class `Models`, the `draw()` method is responsible for showing the lattice layout with defined style, i.e. `fancy` (left) or `plain` (right):





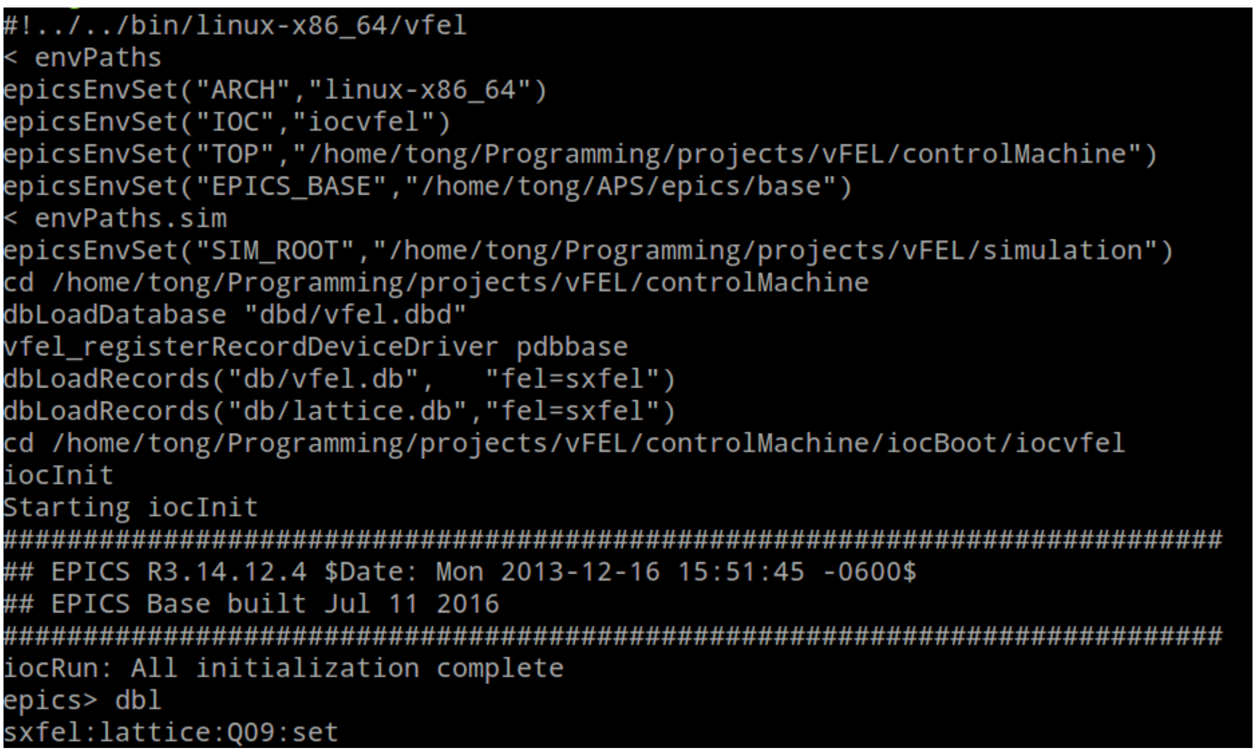
Online-modeling a Chicane

Below is an example to modeling a four-dioples separated chicane by hand, to follow the steps:

1. Start an IOC to provide the EPICS control environment, e.g. to link a few (one or two) valid PVs with some quadrupoles;
2. Create magnetic elements one by one, it is suggested to follow the lattice/beamline element appearance sequence;
3. Instantiate with `Models` class;
4. **Create lattice:**
    - Generate lattice file for simulation;
    - Other manipulations;

Here goes the details:

Start IOC to mimic real control environment



```
#!../../bin/linux-x86_64/vfel
< envPaths
epicsEnvSet("ARCH","linux-x86_64")
epicsEnvSet("IOC","iocvfel")
epicsEnvSet("TOP","/home/tong/Programming/projects/vFEL/controlMachine")
epicsEnvSet("EPICS_BASE","/home/tong/APS/epics/base")
< envPaths.sim
epicsEnvSet("SIM_ROOT","/home/tong/Programming/projects/vFEL/simulation")
cd /home/tong/Programming/projects/vFEL/controlMachine
dbLoadDatabase "dbd/vfel.dbd"
vfel_registerRecordDeviceDriver pdbbase
dbLoadRecords("db/vfel.db",   "fel=sxfel")
dbLoadRecords("db/lattice.db","fel=sxfel")
cd /home/tong/Programming/projects/vFEL/controlMachine/iocBoot/iocvfel
iocInit
Starting iocInit
############################################################################
## EPICS R3.14.12.4 $Date: Mon 2013-12-16 15:51:45 -0600$
## EPICS Base built Jul 11 2016
############################################################################
iocRun: All initialization complete
epics> dbl
sxfel:lattice:Q09:set
```

Fig. 3.1: Start IOC

Create elements

Import proper packages:

```python
#!/usr/bin/python
# coding: utf-8

import beamline
import os
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import copy
```

```
Starting iocInit
############################################################################
## EPICS R3.14.12.4 $Date: Mon 2013-12-16 15:51:45 -0600$
## EPICS Base built Jul 11 2016
############################################################################
iocRun: All initialization complete
epics> dbl
sxfel:lattice:Q09:set
sxfel:lattice:Q10:set
sxfel:trig
sxfel:lattice
sxfel:simulator
sxfel:calcTrig
sxfel:lattice:Q09
sxfel:lattice:Q10
sxfel:randomM
sxfel:elefile
sxfel:ltefile
sxfel:outfilename1
sxfel:simfolderhead
sxfel:lattice:array1
sxfel:prof1
epics>
```

Fig. 3.2: List available records (PVs)

Next, the very first step need to push forward is to correctly model the physical elements (one by one), in the `beamline` package, magnet components classes could be found in `element` module, e.g. quadrupole is abstracted in `ElementQuad` class, charge is in `ElementCharge`, etc., they are all inherited from `MagBlock`.

The common or shared information/configurations for all these elements could be predefined in `MagBlock` class, e.g. we can put information like facility name, time stamp, author/operator, etc., common information is presumed not changed, so please defined in the first step, see STEP 1.

```
# STEP 1: define common information
# commdinfo = {'DATE': '2016-03-22', 'AUTHOR': 'Tong Zhang'}
comminfo = 'DATE = 2016-03-24, AUTHOR = Tong Zhang'
beamline.MagBlock.setCommInfo(comminfo)

# set visualization style
beamline.MagBlock.setStyleConfig(
        config={'quad': {'fc': 'blue', 'ec': 'blue'},
                'bend': {'fc': 'red', 'ec': 'red'}})
```

To set the elements' configurations, method `setConf(config,type)` could be used, in which `config` is either configuration string with the format like `k1=10.0,l=0.1` or python dictionary like `{'k1': 10.0,'l': 0.1}`, and `type` is the configuration type to be confiugred, could be `comm` (common configuration), `ctrl` (control configuration), `simu` (simulation configuration), `misc` (miscellaneous configuration) and `all` (all configurations).

The unit between EPICS PV value and real physical variable is usually required to do conversions, so in the design stage, the method `unitTrans(inval,direction='+',transfun=None)` is created for handling such issue. One can define this conversion function at the class stage, but this approach is limited to the case that all the elements with the same type only could share the same conversion function, which is not proper in the real situation. Thus, `transfun` is created as the input function parameter for `unitTrans` method, which is a user-defined function for each element object.

```
# STEP 2: create elements

# charge, this is visual element for the real accelerator,
# but is a must for elegant tracking
chconf = {'total': 1e-9}
q = beamline.ElementCharge(name='q', config=chconf)

# csrcsben, use elegant element name
# simconf is complementary configurations for elegant tracking,
# should set with setConf(simconf, type='simu') method
simconf = {"edge1_effects": 1,
           "edge2_effects":1,
           "hgap":0.015,
           "csr":0,
           "nonlinear":1,
           "n_kicks":100,
           "integration_order":4,
           "bins":512,
           "sg_halfwidth":1,
           "block_csr":0,
           'l':0.5,}
angle = 0.1 # rad

B1 = beamline.ElementCsrcsben(name='b1', config={'angle':angle, 'e1':0, 'e2':angle})
B1.setConf(simconf, type='simu')
B2 = beamline.ElementCsrcsben(name='b2', config={'angle':-angle, 'e1':-angle, 'e2':0})
B3 = beamline.ElementCsrcsben(name='b3', config={'angle':-angle, 'e1':0, 'e2':-angle})
B4 = beamline.ElementCsrcsben(name='b4', config={'angle': angle, 'e1':angle, 'e2':0})
B2.setConf(simconf, type='simu')
B3.setConf(simconf, type='simu')
B4.setConf(simconf, type='simu')

# drift
D0 = beamline.ElementDrift(name='D0', config="l=1.0")
```

Now the dipole and drift sections are created, here you can also issue some methods to get the defined elements' properties, e.g.: `print(B1.calcTransM(gamma=200))` to get transport matrix and `B1.printConfig(type='all')` to show all the configurations.

Note: Always try to hit <TAB> when you're working under command line interface, especially in IPython shell when you're playing Python.

Next create quadrupoles and incorporate the unit-conversion feature.

Define the conversion function, the `direction` parameter indicates the conversion direction, i.e.:

- `'+'`: conversion rule for EPICS PV (raw) value to physical (real) value;
- `'-'`: conversion rule for physical (real) value to EPICS PV (raw) value.

```
def fUnitTrans(val, direction):
    if direction == '+':
        return val*4.0
    else:
        return val*0.25
```

To use conversion function:

```
# create instance and apply user-defined unit conversion function
Q1 = beamline.ElementQuad(name = 'Q1', config = "k1 = 10, l = 0.5")
simuconf = {'tilt':"pi 4 /"}
Q1.setConf(simuconf, type = 'simu')
# control configurations for Q1
ctrlconf = {"k1":{'pv':"sxfel:lattice:Q09",'val':''}}
Q1.setConf(ctrlconf, type = 'ctrl')
Q1.transfun = fUnitTrans # apply unit conversion function
```

Some testing to show the conversion rule has been applied successfully:

```
>>> Q1.printConfig(type='ctrl')
---------- Configuration START  ----------
Element name: Q1 (ElementQuad)
Control configs:
k1     = sxfel:lattice:Q09, raw:   10.0, real:   40.0
----------  Configuration END   ----------
>>> Q1.printConfig(type='simu')
---------- Configuration START  ----------
Element name: Q1 (ElementQuad)
Simulation configs:
tilt   = pi 4 /
l      = 0.5
k1     = 10
----------  Configuration END   ----------
>>> Q1.getK1(type='ctrl')
40
>>> Q1.getK1(type='simu')
10
```

Create models

Use `Models` class of `models` module, change mode to be `simu` to start simulation mode, `online` mode will trig EPICS get/put processes when control configurations could be found in elements' configuration.

```
latline_online = beamline.Models(name='blchi', mode='online')
qline = (D0, Q1, D0)
chi   = (B1, D0, B2, D0, D0, B3, D0, B4)
latline_online.addElement(q, qline, chi, qline)

# show artist layout
#latline_online.draw(showfig=True,mode='fancy')
```

To access what `latline_online` (which is an instance of `beamline.models.Models`) provides:

```
>>> eleb1, = latline_online.getElementsByName('b1')
>>> print eleb1.name
b1
>>> # change b1 configuration, e.g. angle
>>> eleb1.setConf('angle=0.5', type='simu')
>>> eleb1.printConfig()
```

```
>>> # print out all added elements
>>> latline_online.printAllElements()
----------  Configuration END   ----------
ID : Name         Type       Class Name
001: q            CHARGE     ElementCharge
002: D0           DRIFT      ElementDrift
003: Q1           QUAD       ElementQuad
004: D0           DRIFT      ElementDrift
005: b1           CSRCSBEN   ElementCsrcsben
006: D0           DRIFT      ElementDrift
007: b2           CSRCSBEN   ElementCsrcsben
008: D0           DRIFT      ElementDrift
009: D0           DRIFT      ElementDrift
010: b3           CSRCSBEN   ElementCsrcsben
011: D0           DRIFT      ElementDrift
012: b4           CSRCSBEN   ElementCsrcsben
013: D0           DRIFT      ElementDrift
014: Q1           QUAD       ElementQuad
015: D0           DRIFT      ElementDrift
```

```
>>> # get configuration of 'Q1'
>>> print latline_online.getAllConfig(fmt='dict')['Q1']
{'QUAD': {'tilt': 'pi 4 /', 'k1': 2.5, 'l': '0.5'}}
```

```
>>> # get all 'Q1' and change the drawing style of orange facecolor
>>> eleQ1all = latline_online.getElementsByName('Q1')
>>> map(lambda x: x.setStyle(fc='orange'), eleQ1all)
>>> print eleQ1all
[<beamline.element.ElementQuad object at 0x7f335a40b610>,
 <beamline.element.ElementQuad object at 0x7f335a40b9d0>]
```

Note: `addElement()` method support replica expansion functionality, when same named items are added into the lattice tuple (e.g. `qline` in this example), so when `getElementsByName()` is invoked, a list would be returned, it is the user's liability to discriminate which one should be selected and modified since they share the same literal name, it is better to assign a new name once seleced from the list.

```
>>> # update Q1's EPICS PV value, aim to set its true value
>>> latline_online.putCtrlConf(eleQ1, 'k1', 2.5, type='real')
>>> eleQ1.printConfig(type='all')
---------- Configuration START  ----------
Element name: Q1 (ElementQuad)
Position: s = 1.000 [m]
Common configs:
DATE   = 2016-03-24
AUTHOR = Tong Zhang
Simulation configs:
tilt   = pi 4 /
l      = 0.5
k1     = 10
Control configs:
k1     = sxfel:lattice:Q09, raw:  0.625, real:    2.5
----------  Configuration END   ----------
```

```
>>> # update Q1's EPICS PV value, aim to set its raw value
>>> latline_online.putCtrlConf(eleQ1, 'k1', 2.5, type='raw')
>>> eleQ1.printConfig(type='all')
---------- Configuration START  ----------
Element name: Q1 (ElementQuad)
Position: s = 1.000 [m]
Common configs:
DATE   = 2016-03-24
AUTHOR = Tong Zhang
Simulation configs:
tilt   = pi 4 /
l      = 0.5
k1     = 10
Control configs:
k1     = sxfel:lattice:Q09, raw:    2.5, real:   10.0
----------  Configuration END   ----------
```

Warning: The unit of 'raw' value here should be 'A', and 'T/m' for 'real' value, however it's also the user's liability to define correct unit conversion function at the element creation stage.

Lattice modeling

`Lattice` class in `lattice` module is created for make lattice instance, the initial idea is to bridge the real machine and the numerical simulation world, e.g. generate lattice file that could be used by simulation code like `Elegant`.

```
# e.g. '.lte' for elegant tracking, require all configurations
latins = beamline.Lattice(latline_online.getAllConfig())
latfile = os.path.join(os.getcwd(), '../../tests/tracking/test.lte')
latins.generateLatticeFile(latline_online.name, latfile)
#latins.dumpAllElements()
```

The generated lattice file:

```
!-----------------------------------------------------------------------!
! This file is automatically generated by 'generateLatticeFile()' method, !
!                could be used as elegant lattice file.                  !
!                       ------------------------                         !
!              Author: Tong Zhang (zhangtong@sinap.ac.cn)                !
!                Generated Date: 2016-08-23 16:29:01 CST                 !
!-----------------------------------------------------------------------!

! Element definitions:
D0        :     DRIFT, l = "1.0"
Q1        :      QUAD, tilt = "pi 4 /", k1 = "2.5", l = "0.5"
```

```
B1        :  CSRCSBEN, hgap = "0.015", integration_order = "4", block_csr = "0", angle = "0.1", n_kicks = "100", edge2_effects = "1",
↪edge1_effects = "1", l = "0.5", nonlinear = "1", sg_halfwidth = "1", csr = "0", e1 = "0", bins = "512", e2 = "0.1"
B2        :  CSRCSBEN, hgap = "0.015", integration_order = "4", block_csr = "0", angle = "-0.1", n_kicks = "100", edge2_effects = "1
↪", edge1_effects = "1", l = "0.5", nonlinear = "1", sg_halfwidth = "1", csr = "0", e1 = "-0.1", bins = "512", e2 = "0"
B3        :  CSRCSBEN, hgap = "0.015", integration_order = "4", block_csr = "0", angle = "-0.1", n_kicks = "100", edge2_effects = "1
↪", edge1_effects = "1", l = "0.5", nonlinear = "1", sg_halfwidth = "1", csr = "0", e1 = "0", bins = "512", e2 = "-0.1"
B4        :  CSRCSBEN, hgap = "0.015", integration_order = "4", block_csr = "0", angle = "0.1", n_kicks = "100", edge2_effects = "1",
↪edge1_effects = "1", l = "0.5", nonlinear = "1", sg_halfwidth = "1", csr = "0", e1 = "0.1", bins = "512", e2 = "0"
Q         :    CHARGE, total = "1e-09"

! Beamline definitions:
BLCHI     : line = (q, D0, Q1, D0, b1, D0, b2, D0, D0, b3, D0, b4, D0, Q1, D0)
```

Now, the particle tracking by `Elegant` is possible by using the generated lattice file.

```
simpath = '../../tests/tracking/'
elefile = os.path.join(simpath, 'test.ele')
h5out   = os.path.join(simpath, 'tpout.h5')
elesim = beamline.Simulator()
elesim.setMode('elegant')
elesim.setScript('runElegant.sh')
elesim.setExec('elegant')
elesim.setPath(simpath)
elesim.setInputfiles(ltefile=latfile, elefile=elefile)

elesim.doSimulation()
```

The tracking output files could be found in `simpath`, that is in `../../tests/tracking` folder.

An example of `elefile` is shown like:

```
&run_setup
        lattice = test.lte,
        default_order = 2,
        use_beamline = BLCHI,
        p_central = 70
        sigma = %s.sig,
        centroid = %s.cen,
        output = %s.out,
        magnets = %s.mag
        !final = %s.fin,
        print_statistics = 1
        !parameters = %s.param
&end

&twiss_output
    filename = %s.twi
    matched = 0
    alpha_x = 0
    alpha_y = 0
    beta_x  = 10
    beta_y  = 10
&end

&run_control
        n_steps = 1
&end

&bunched_beam
        n_particles_per_bunch = 2500,
        one_random_bunch=0,
        emit_nx = 2.0e-6,
        emit_ny = 2.0e-6,
        beta_x = 20, alpha_x = 10,
        beta_y = 20, alpha_y = 10,
        sigma_dp = 0.001,
        sigma_s = 300e-6,
        distribution_type[0] = 3*"gaussian",
        distribution_cutoff[0] = 3*3,
        symmetrize = 1,
        enforce_rms_values[0] = 1,1,1,
        !bunch = %s.bun
&end

&track &end
```

Note:

- `simpath`, `elefile` and `h5out` are file/folder locations that should be defined by user, it is suggested that make them share with the same directory root (the value of `simpath`), thus all the output files also could be located in a much more clearer manner.
- Currently, `simulation` module does not automatically handle the configuration of `elefile`, which is left for the user to adapt correctly, expecially the `use_beamline` keyword.
- `Simulator` class is designed to have the ability to interface with different simulation software, e.g. `Elegant`, `MAD`, etc., by `setMode()` method, and the regarding runtime scripts (`setScript()` and `setExec()`), to make the `doSimulation()` method works.

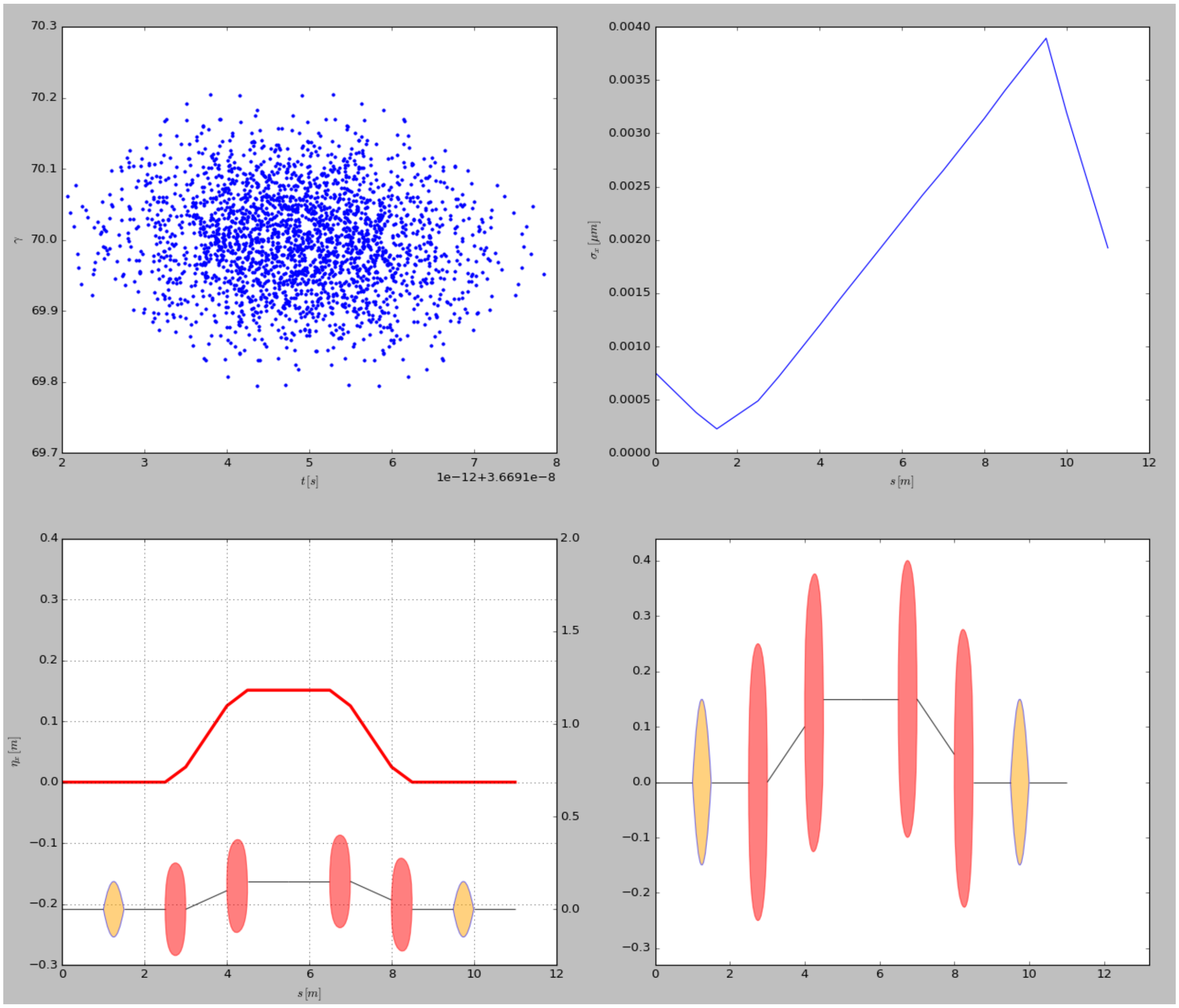
Below is the `runElegant.sh` script, it is a quite simple shell script, to enter right directory and trig right simulation procedure, if other simulation tool is configured by `setMode()`, this script may need to be altered.

```
#!/bin/bash

#
# script made for simulation.py module
# input parameters:
#  elefile: .ele file for elegant tracking, full name
#  simpath: simulation path for data
#  simexec: elegant path
#

elefile=$1
simpath=$2
simexec=$3

cd ${simpath}
${simexec} ${elefile} >& /dev/null
```

The simulated data are transformed into hdf5 format, could be read and plot:

```
# data columns could be extracted from simulation output files,
# to memory or h5 files.
data_tp = elesim.getOutput(file='test.out', data=('t', 'p'))#, dump = h5out)
#data_tp = elesim.getOutput(file='test.out', data=('t', 'p'), dump=h5out)
data_sSx = elesim.getOutput(file='test.sig', data=('s', 'Sx'))
data_setax = elesim.getOutput(file='test.twi', data=('s', 'etax'))

# visualize data

fig = plt.figure(1)
ax1 = fig.add_subplot(221)
ax1.plot(data_tp[:,0],data_tp[:,1],'.')
ax1.set_xlabel('$t\,[s]$')
ax1.set_ylabel('$\gamma$')

ax2 = fig.add_subplot(222)
ax2.plot(data_sSx[:,0],data_sSx[:,1],'-')
ax2.set_ylabel('$\sigma_x\,[\mu m]$')
ax2.set_xlabel('$s\,[m]$')

ax3 = fig.add_subplot(223)
ax3.plot(data_setax[:,0],data_setax[:,1],'r-', lw=3,)
ax3.set_ylabel('$\eta_{x}\,[m]$')
ax3.set_xlabel('$s\,[m]$')
# #### Lattice layout visualization

# generate lattice drawing plotting objects
ptches, anotes, xr, yr = latline_online.draw(mode='fancy', showfig=False)

# show drawing at the
ax3t = ax3.twinx()
[ax3t.add_patch(i) for i in ptches]
xr3 = ax3.get_xlim()
yr3 = ax3.get_ylim()
x0, x1 = min(xr[0],xr3[0]), max(xr[1], xr3[1])
y0, y1 = min(yr[0],yr3[0]), max(yr[1], yr3[1])
ax3t.set_xlim(x0, x1)
```

```
ax3t.set_ylim(y0, y1*5)
ax3.set_xlim(x0, x1)
ax3.set_ylim(y0, y1)
ax3.grid()

# show lattice drawing in a single plot
newptches = beamline.MagBlock.copy_patches(ptches)
#for i,val in enumerate(newptches):
#    print id(newptches[i]), id(ptches[i])
ax4 = fig.add_subplot(224)
[ax4.add_patch(i) for i in newptches]
ax4.set_xlim(x0*1.1, x1*1.1)
ax4.set_ylim(y0*1.1, y1*1.1)

plt.show()
```



Note: Method `getOutput()` of `Simulator` takes key-value parameters, if `dump` keyword is given, then the data would be extracted to the hdf5 file named by `dump`, or simply assigned to a numpy array.

Note:

Tips for data visualization:

- LaTeX formated string, i.e. wrapped by two `$` could make your labels/texts/annotations more beautiful;
- Use `matplotlib` in the OOP approach;
- Method `twinx()` could be used to draw two y-axis, similar as `plotyy()` in MATLAB;
- `Patches` in `matplotlib` could be only used in one place, if reused is needed, remember make a copy by `copy_patches()`.

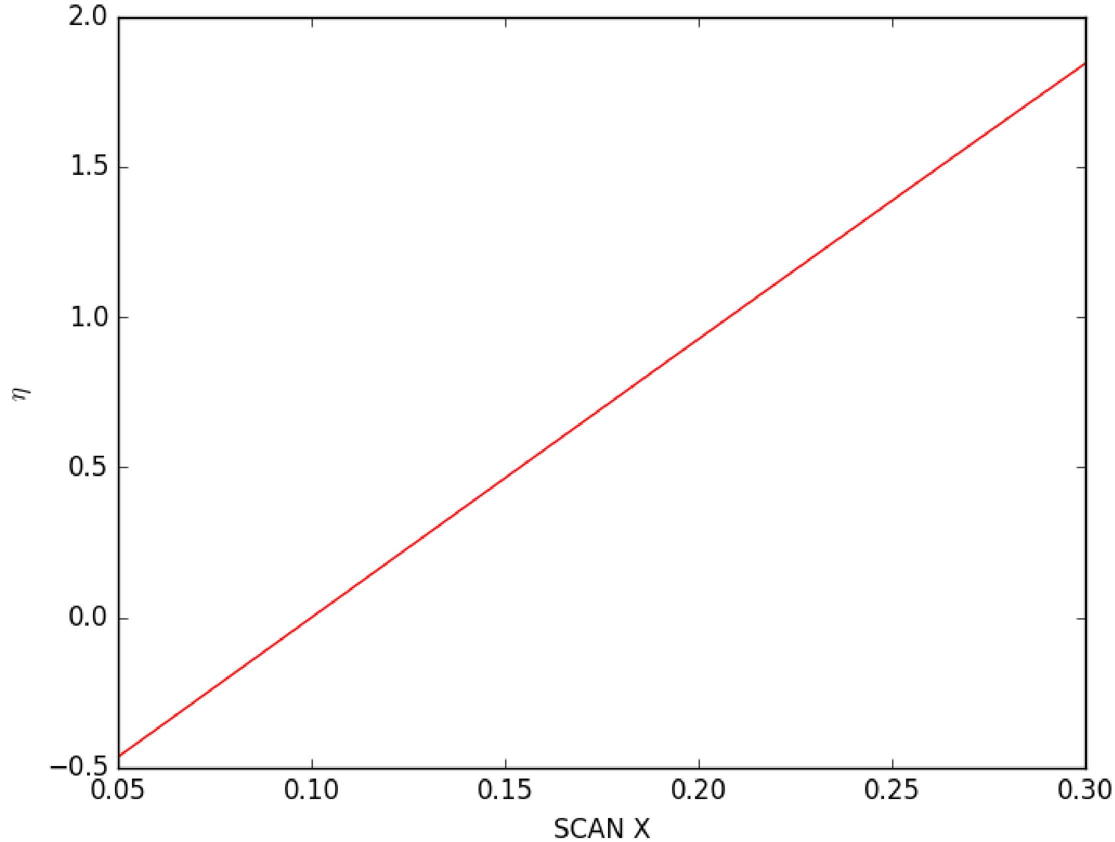
Example of how to make use of online model

As long as the online-model procedure is accomplished, we can manipulate the facility from the software side, i.e. specific algorithms could be designed to manipulate/optimize the variables that have been already modeled, see the following simple example.

Task Tunning one of the dipole strength of chicane to get the response of the dispersive value.

```
# Scan parameter: final Dx v.s. angle of B1
import numpy as np
dx = []
thetaArray = np.linspace(0.05,0.3,20)
for theta in thetaArray:
    eleb1.setConf({'angle':theta}, type='simu')
    latins=beamline.Lattice(latline_online.getAllConfig())
    latins.generateLatticeFile(latline_online.name, latfile)
    elesim.doSimulation()
    data=elesim.getOutput(file='test.twi', data=(['etax']))
    dx.append(data[-1])
dxArray = np.array(dx)

plt.figure()
plt.plot(thetaArray, dxArray, 'r')
plt.xlabel('SCAN X')
plt.ylabel('$\eta$')
plt.show()
```

The scanned figure:



Automatic Modeling

`beamline` provides the solution to automatically online-modeling the machine, by utilizing the `Elegant` lattice file (i.e. `.lte` file). Here is the basic idea and reasons:

- Usually `.lte` file is a well-maintained lattice file that intended to be used by particl tracking code — `Elegant`, so it could be relied to master the machine configuration;
- Add control EPICS directive into `.lte` file to meet additional requirement, but still could be recognized by `Elegant`, i.e. the `.lte` file with EPICS directive produce the same tracking results;
- When there are thousands of machine elements to be modeled, modeling from the `.lte` file should be much more efficient;

Note:

- In order to model the real machine precisely, the `.lte` file should be also precisely and kept pace with the real machine, so maintain the `.lte` file precisely;
- EPICS directive: append `!epics` in the element definition.

How To:

!epics directive

```
Q01L0: QUAD, L=0.1, K1= 3.95 !epics {'k1':{'pv':'sxfel:lattice:Q09','val':0}}
Q02L0: QUAD, L=0.2, K1=-3.75 !epics {'k1':{'pv':'sxfel:lattice:Q10','val':''}}
Q03L0: QUAD, L=0.1, K1= 3.95
```

Automatic modeling

```python
#!/usr/bin/python
# -*- coding: utf-8 -*-
"""
    Demonstration to modeling accelerator with the lte file.
    SXFEL

    Author       : Tong Zhang
    Created      : 2016-04-12 10:11:06 AM CST
    Last updated : 2016-04-12 21:20:16 PM CST
"""

import beamline
import os
import matplotlib.pyplot as plt

### STEP 1: read lattice configurations from .lte file
ltefile = 'sxfel_all.lte'
lpins   = beamline.LteParser(ltefile)

# generate lte file with all the element-definitions regarding to beamline
blname = 'sxfel'
newltefile = 'sxfel_new.lte'
latins = beamline.Lattice(lpins.file2json())
latins.generateLatticeFile(blname, newltefile)

# use the concise version of lte file
newlpins = beamline.LteParser(newltefile)
newlatins = beamline.Lattice(newlpins.file2json())


# demonstrate to create a new element from keyword name,
# there are two approaches to create:
#   1: use element classes from element module
#   2: use makeElement() method from LteParser class or Lattice class

kw_name  = 'Q01L0'

## create element approach 1:
#kw_dict   = newlpins.getKwAsDict(kw_name)
#kw_type   = newlpins.getKwType(kw_name)
#kw_config = newlpins.getKwConfig(kw_name)
#kw_eobj = beamline.ElementQuad(name=kw_name, config=kw_config)

## create element approach 2:
kw_eobj = newlpins.makeElement(kw_name)
#kw_eobj = newlatins.makeElement(kw_name)

kw_eobj.printConfig(type='all')
print newlpins.ctrlconf_dict[kw_name]

## show element drawing:
#kw_eobj.setDraw(mode='fancy') # or mode='plain'
#kw_eobj.showDraw()

### STEP 2: initialise all element objects for beamline model
#for ele in beamline.Models.flatten(newlatins.getAllKws()):
latmodel = beamline.Models(name=blname, mode='simu')
ele_name_list = newlatins.getElementList(blname)
ele_eobj_list = []
for ele in ele_name_list:
    eobj = newlatins.makeElement(ele)
    ele_eobj_list.append(eobj)

latmodel.addElement(*ele_eobj_list)
# show all configurations | pjson
#print latmodel.getAllConfig()

# find element by name
Q_list = latmodel.getElementsByName(kw_name.lower())

# add other configurations, e.g. control configurations, etc.
Q_list[0].printConfig(type='all')
```

```python
# csrcsben example:
B1LH_list = latmodel.getElementsByName('B1LH'.lower())
B1LH_list[0].printConfig(type='all')
```

Note:

- `LteParser()` is a class that could parse `.lte` file;
- `Lattice()` is a class that could model the lattice, with the input of JSON string that generared by `LteParser()`;
- Do `LteParser()` — `Lattice()` twice is to generate clean `.lte` file that only contain necessary elements;
- Pay attention to the `makeElement()` method, which should be invoked to model all the elements automatically;
- `getElementList()` to get the element list that comprising the seleced beamline keyword, and build lattice model by `addElement()` method.

The reformated `.lte` file:

```
!------------------------------------------------------------------------!
! This file is automatically generated by 'generateLatticeFile()' method, !
!                could be used as elegant lattice file.                  !
!                ------------------------                                !
!              Author: Tong Zhang (zhangtong@sinap.ac.cn)                !
!                Generated Date: 2016-08-24 10:35:47 CST                 !
!------------------------------------------------------------------------!

! EPICS control definitions:
!!epics Q01L0     :     {"k1": {"pv": "sxfel:lattice:Q09", "val": 0}}
!!epics Q02L0     :    {"k1": {"pv": "sxfel:lattice:Q10", "val": ""}}

! Element definitions:
AC1O1L2_1 :      RFCW, lsc = "1.0", cell_length = "0.020994", volt = "57638600.0", trwakefile = "cband_T_4pi5_1mm.sdds", lsc_high_
↪frequency_cutoff0 = "0.25", lsc_high_frequency_cutoff1 = "0.3", smoothing = "1.0", zwakefile = "cband_L_4pi5_1mm.sdds", END2_FOCUS␣
↪= "1.0", wzcolumn = "W", END1_FOCUS = "1.0", n_kicks = "20.0", interpolate = "1.0", lsc_interpolate = "1.0", phase = "104.0", freq␣
↪= "5712000000.0", wxcolumn = "W", change_p0 = "1.0", wycolumn = "W", l = "1.68568", lsc_bins = "512.0", tcolumn = "t"
AC1O1L2_2 :      RFCW, lsc = "1.0", cell_length = "0.020994", volt = "57638600.0", trwakefile = "cband_T_4pi5_1mm.sdds", lsc_high_
↪frequency_cutoff0 = "0.25", lsc_high_frequency_cutoff1 = "0.3", smoothing = "1.0", zwakefile = "cband_L_4pi5_1mm.sdds", END2_FOCUS␣
↪= "1.0", wzcolumn = "W", END1_FOCUS = "1.0", n_kicks = "20.0", interpolate = "1.0", lsc_interpolate = "1.0", phase = "104.0", freq␣
↪= "5712000000.0", wxcolumn = "W", change_p0 = "1.0", wycolumn = "W", l = "1.68568", lsc_bins = "512.0", tcolumn = "t"
AC1O1L3_1 :      RFCW, lsc = "1.0", cell_length = "0.020994", volt = "57638600.0", trwakefile = "cband_T_4pi5_1mm.sdds", lsc_high_
↪frequency_cutoff0 = "0.25", lsc_high_frequency_cutoff1 = "0.3", smoothing = "1.0", zwakefile = "cband_L_4pi5_1mm.sdds", END2_FOCUS␣
↪= "1.0", wzcolumn = "W", END1_FOCUS = "1.0", n_kicks = "20.0", interpolate = "1.0", lsc_interpolate = "1.0", phase = "108.0", freq␣
↪= "5712000000.0", wxcolumn = "W", change_p0 = "1.0", wycolumn = "W", l = "1.68568", lsc_bins = "512.0", tcolumn = "t"
AC1O1L3_2 :      RFCW, lsc = "1.0", cell_length = "0.020994", volt = "57638600.0", trwakefile = "cband_T_4pi5_1mm.sdds", lsc_high_
↪frequency_cutoff0 = "0.25", lsc_high_frequency_cutoff1 = "0.3", smoothing = "1.0", zwakefile = "cband_L_4pi5_1mm.sdds", END2_FOCUS␣
↪= "1.0", wzcolumn = "W", END1_FOCUS = "1.0", n_kicks = "20.0", interpolate = "1.0", lsc_interpolate = "1.0", phase = "108.0", freq␣
↪= "5712000000.0", wxcolumn = "W", change_p0 = "1.0", wycolumn = "W", l = "1.68568", lsc_bins = "512.0", tcolumn = "t"
AC1O1L3_3 :      RFCW, lsc = "1.0", cell_length = "0.020994", volt = "57638600.0", trwakefile = "cband_T_4pi5_1mm.sdds", lsc_high_
↪frequency_cutoff0 = "0.25", lsc_high_frequency_cutoff1 = "0.3", smoothing = "1.0", zwakefile = "cband_L_4pi5_1mm.sdds", END2_FOCUS␣
↪= "1.0", wzcolumn = "W", END1_FOCUS = "1.0", n_kicks = "20.0", interpolate = "1.0", lsc_interpolate = "1.0", phase = "108.0", freq␣
↪= "5712000000.0", wxcolumn = "W", change_p0 = "1.0", wycolumn = "W", l = "1.68568", lsc_bins = "512.0", tcolumn = "t"
AC1O1L3_4 :      RFCW, lsc = "1.0", cell_length = "0.020994", volt = "57638600.0", trwakefile = "cband_T_4pi5_1mm.sdds", lsc_high_
↪frequency_cutoff0 = "0.25", lsc_high_frequency_cutoff1 = "0.3", smoothing = "1.0", zwakefile = "cband_L_4pi5_1mm.sdds", END2_FOCUS␣
↪= "1.0", wzcolumn = "W", END1_FOCUS = "1.0", n_kicks = "20.0", interpolate = "1.0", lsc_interpolate = "1.0", phase = "108.0", freq␣
↪= "5712000000.0", wxcolumn = "W", change_p0 = "1.0", wycolumn = "W", l = "1.68568", lsc_bins = "512.0", tcolumn = "t"
AS1O1L1   :      RFCW, lsc = "1.0", cell_length = "0.03499", volt = "45136800.0", trwakefile = "sband_T_10mm.sdds", lsc_high_
↪frequency_cutoff0 = "0.25", lsc_high_frequency_cutoff1 = "0.3", smoothing = "1.0", zwakefile = "sband_L_10mm.sdds", END2_FOCUS =
↪"1.0", wzcolumn = "W", END1_FOCUS = "1.0", n_kicks = "20.0", interpolate = "1.0", lsc_interpolate = "1.0", phase = "37.4", freq =
↪"2856000000.0", wxcolumn = "W", change_p0 = "1.0", wycolumn = "W", l = "2.974132", lsc_bins = "512.0", tcolumn = "t"
AX1O1L1   :      RFCW, lsc = "1.0", cell_length = "0.010934", volt = "13698400.0", trwakefile = "xband_T1080_T_10mm.sdds", lsc_high_
↪frequency_cutoff0 = "0.25", lsc_high_frequency_cutoff1 = "0.3", smoothing = "1.0", zwakefile = "xband_T1080_L_10mm.sdds", END2_
↪FOCUS = "1.0", wzcolumn = "W", END1_FOCUS = "1.0", n_kicks = "20.0", interpolate = "1.0", lsc_interpolate = "1.0", phase = "270.0
↪", freq = "11424000000.0", wxcolumn = "W", change_p0 = "1.0", wycolumn = "W", l = "0.944578", lsc_bins = "512.0", tcolumn = "t"
B1BC1     :  CSRCSBEN, hgap = "0.015", integration_order = "4.0", nonlinear = "1.0", angle = "0.061510638828", n_kicks = "100.0", l =
↪"0.300189261474", edge1_effects = "1.0", edge2_effects = "1.0", block_csr = "0.0", sg_halfwidth = "1.0", e2 = "0.061510638828", e1␣
↪= "0.0", bins = "512.0", csr = "1.0"
B1LH      :  CSRCSBEN, hgap = "0.015", integration_order = "4.0", nonlinear = "1.0", angle = "0.0872664625997", n_kicks = "100.0", l =
↪"0.200254073567", edge1_effects = "1.0", edge2_effects = "1.0", block_csr = "0.0", sg_halfwidth = "1.0", e2 = "0.0872664625997",␣
↪e1 = "0.0", bins = "512.0", csr = "1.0"
```

```
B2BC1     :  CSRCSBEN, hgap = "0.015", integration_order = "4.0", nonlinear = "1.0", angle = "-0.061510638828", n_kicks = "100.0", l =
↪"0.300189261474", edge1_effects = "1.0", edge2_effects = "1.0", block_csr = "0.0", sg_halfwidth = "1.0", e2 = "0.0", e1 = "-0.
↪061510638828", bins = "512.0", csr = "1.0"
B2LH      :  CSRCSBEN, hgap = "0.015", integration_order = "4.0", nonlinear = "1.0", angle = "-0.0872664625997", n_kicks = "100.0", l␣
↪= "0.200254073567", edge1_effects = "1.0", edge2_effects = "1.0", block_csr = "0.0", sg_halfwidth = "1.0", e2 = "0.0", e1 = "-0.
↪0872664625997", bins = "512.0", csr = "1.0"
B3BC1     :  CSRCSBEN, hgap = "0.015", integration_order = "4.0", nonlinear = "1.0", angle = "-0.061510638828", n_kicks = "100.0", l =
↪"0.300189261474", edge1_effects = "1.0", edge2_effects = "1.0", block_csr = "0.0", sg_halfwidth = "1.0", e2 = "-0.061510638828",␣
↪e1 = "0.0", bins = "512.0", csr = "1.0"
B3LH      :  CSRCSBEN, hgap = "0.015", integration_order = "4.0", nonlinear = "1.0", angle = "-0.0872664625997", n_kicks = "100.0", l␣
↪= "0.200254073567", edge1_effects = "1.0", edge2_effects = "1.0", block_csr = "0.0", sg_halfwidth = "1.0", e2 = "-0.0872664625997
↪", e1 = "0.0", bins = "512.0", csr = "1.0"
B4BC1     :  CSRCSBEN, hgap = "0.015", integration_order = "4.0", nonlinear = "1.0", angle = "0.061510638828", n_kicks = "100.0", l =
↪"0.300189261474", edge1_effects = "1.0", edge2_effects = "1.0", block_csr = "0.0", sg_halfwidth = "1.0", e2 = "0.0", e1 = "0.
↪061510638828", bins = "512.0", csr = "1.0"
B4LH      :  CSRCSBEN, hgap = "0.015", integration_order = "4.0", nonlinear = "1.0", angle = "0.0872664625997", n_kicks = "100.0", l =
↪"0.200254073567", edge1_effects = "1.0", edge2_effects = "1.0", block_csr = "0.0", sg_halfwidth = "1.0", e2 = "0.0", e1 = "0.
↪0872664625997", bins = "512.0", csr = "1.0"
BAM01BI1  :  LSCDRIFT, lsc = "1.0", l = "0.15", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
BAM01BI3  :  LSCDRIFT, lsc = "1.0", l = "0.15", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
BAM01L0   :  LSCDRIFT, lsc = "1.0", l = "0.15", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
BLL01     :  LSCDRIFT, lsc = "1.0", l = "0.07", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
BLL02     :  LSCDRIFT, lsc = "1.0", l = "0.09", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
BLL03     :  LSCDRIFT, lsc = "1.0", l = "0.18405", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
BLL04     :  LSCDRIFT, lsc = "1.0", l = "0.09982", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
BPM01BC1  :       MONI, l = "0.075"
BPM01BC2  :       MONI, l = "0.075"
BPM01BI1  :       MONI, l = "0.075"
BPM01BI2  :       MONI, l = "0.075"
BPM01BI3  :       MONI, l = "0.075"
BPM01L0   :       MONI, L = "0.075"
BPM01L1   :       MONI, l = "0.075"
BPM01L2   :       MONI, l = "0.075"
BPM01L3   :       MONI, l = "0.075"
BPM02BC1  :       MONI, l = "0.2"
BPM02BC2  :       MONI, l = "0.075"
BPM02BI1  :       MONI, l = "0.075"
BPM02BI3  :       MONI, l = "0.075"
BPM02L0   :       MONI, L = "0.15"
BPM02L2   :       MONI, l = "0.075"
BPM02L3   :       MONI, l = "0.075"
BPM03BI1  :       MONI, l = "0.075"
BPM03BI3  :       MONI, l = "0.075"
BPM03L0   :       MONI, L = "0.075"
BPM03L3   :       MONI, l = "0.075"
BPM04BI1  :       MONI, l = "0.075"
BPM04L3   :       MONI, l = "0.075"
BPM05BI1  :       MONI, l = "0.075"
BPM05L3   :       MONI, l = "0.075"
BPM06BI1  :       MONI, l = "0.075"
BPM06L3   :       MONI, l = "0.075"
BPM07BI1  :       MONI, l = "0.075"
BPM07L3   :       MONI, l = "0.075"
C         :     CHARGE, total = "5e-10"
CDR01BC1  :  LSCDRIFT, lsc = "1.0", l = "0.4", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
CRR01BC1  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR01BC2  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR01BI1  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR01BI2  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR01BI3  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR01L0   :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR01L1   :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR01L2   :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR01L3   :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR02BC1  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR02BC2  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR02BI1  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR02BI3  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR02L0   :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR02L2   :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR02L3   :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR03BC1  :     KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
```

```
CRR03BI1  :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR03BI3  :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR03L3   :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR04BI1  :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR04L3   :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR05BI1  :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR05L3   :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR06BI1  :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR06L3   :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR07BI1  :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
CRR07L3   :    KICKER, VKICK = "0.0", L = "0.1", HKICK = "0.0"
D01BC1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D01BC2    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D01BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D01BI2    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D01BI3    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D01L0     :  LSCDRIFT, lsc = "1.0", l = "0.534", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D01L1     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D01L2     :  LSCDRIFT, lsc = "1.0", l = "1.598", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D01L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02BC1    :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02BC2    :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02BI1    :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02BI2    :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02BI3    :  LSCDRIFT, lsc = "1.0", l = "0.6", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02L0     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02L1     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02L2     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D02L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03BC1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03BC2    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03BI2    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03BI3    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03L0     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03L1     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03L2     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D03L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D04BC1    :  LSCDRIFT, lsc = "1.0", l = "0.18", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D04BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D04BI2    :  LSCDRIFT, lsc = "1.0", l = "0.515", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D04BI3    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D04DBC2   :  LSCDRIFT, lsc = "1.0", l = "0.6055", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D04L0     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D04L1     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D04L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
```

```
D05BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.1", csr = "1.0"
D05BI1    :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D05BI2    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D05BI3    :  LSCDRIFT, lsc = "1.0", l = "0.6", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D05DBC2   :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.613", csr = "1.0"
D05L0     :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D05L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D06BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.77", csr = "1.0"
D06BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D06BI3    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D06L0     :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.250954959386", csr = "1.0"
D06L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D07BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.2", csr = "1.0"
D07BI1    :  LSCDRIFT, lsc = "1.0", l = "0.305", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D07BI3    :  LSCDRIFT, lsc = "1.0", l = "2.868", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D07L0     :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.05", csr = "1.0"
D07L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D08BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D08DBC1CSR:   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.329909487823", csr = "1.0"
D08DBC1LSC:  LSCDRIFT, lsc = "1.0", l = "0.0", interpolate = "1.0", smoothing = "1.0", LEFFECTIVE = "0.480909487823", high_frequency_
↪cutoff1 = "0.3", high_frequency_cutoff0 = "0.25", bins = "512.0"
D08L0     :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.1", csr = "1.0"
D08L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D09BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.1", csr = "1.0"
D09BI1    :  LSCDRIFT, lsc = "1.0", l = "0.8", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D09L0     :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.250954959386", csr = "1.0"
D09L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D10BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.1", csr = "1.0"
D10BI1    :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D10L0     :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D10L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D11BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D11L0     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D11L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D12BC1CSR :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.1", csr = "1.0"
D12BC1LSC :  LSCDRIFT, lsc = "1.0", l = "0.0", interpolate = "1.0", smoothing = "1.0", LEFFECTIVE = "1.08", high_frequency_cutoff1 =
↪"0.3", high_frequency_cutoff0 = "0.25", bins = "512.0"
D12BI1    :  LSCDRIFT, lsc = "1.0", l = "0.265", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D12L0     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D12L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D13BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.1", csr = "1.0"
D13BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D13L0     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D13L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D14BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D14DBC1CSR:   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.329909487823", csr = "1.0"
D14DBC1LSC:  LSCDRIFT, lsc = "1.0", l = "0.0", interpolate = "1.0", smoothing = "1.0", LEFFECTIVE = "0.480909487823", high_frequency_
↪cutoff1 = "0.3", high_frequency_cutoff0 = "0.25", bins = "512.0"
D14L0     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D14L3     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
```

```
D15BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.2", csr = "1.0"
D15BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D15L0     :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D16BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.77", csr = "1.0"
D16BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D16L0     :  LSCDRIFT, lsc = "1.0", l = "0.2", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D17BC1    :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.1", csr = "1.0"
D17BI1    :  LSCDRIFT, lsc = "1.0", l = "0.265", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D17L0     :  LSCDRIFT, lsc = "1.0", l = "0.67", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D18BC1    :  LSCDRIFT, lsc = "1.0", l = "0.18", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
D18BI1    :  LSCDRIFT, lsc = "1.0", l = "0.4", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D19BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D20BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D21BI1    :  LSCDRIFT, lsc = "1.0", l = "0.4", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
D22BI1    :  LSCDRIFT, lsc = "1.0", l = "0.1", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
DACL2     :  LSCDRIFT, lsc = "1.0", l = "1.78963", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
DACL3     :  LSCDRIFT, lsc = "1.0", l = "1.78963", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
DAWGC1    :  LSCDRIFT, lsc = "1.0", l = "0.051975", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
DAWGC2    :  LSCDRIFT, lsc = "1.0", l = "0.051975", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
DAWGS     :  LSCDRIFT, lsc = "1.0", l = "0.047934", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
DAWGX1    :  LSCDRIFT, lsc = "1.0", l = "0.067711", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
DAWGX2    :  LSCDRIFT, lsc = "1.0", l = "0.067711", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
DBEND01BI1:  LSCDRIFT, lsc = "1.0", l = "0.8", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
DBEND01BI3:  LSCDRIFT, lsc = "1.0", l = "2.0", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
DBEND01L0 :  LSCDRIFT, lsc = "1.0", l = "0.8", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
DBPM      :     DRIFT, l = "0.2"
DBPMLSC   :  LSCDRIFT, lsc = "1.0", l = "0.0", interpolate = "1.0", smoothing = "1.0", LEFFECTIVE = "0.2", high_frequency_cutoff1 = "0.
↪3", high_frequency_cutoff0 = "0.25", bins = "512.0"
DCRR      :     DRIFT, l = "0.025"
DFODO1BI1 :  LSCDRIFT, lsc = "1.0", l = "0.215", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
DFODO2BI1 :  LSCDRIFT, lsc = "1.0", l = "0.7", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_frequency_
↪cutoff0 = "0.25", bins = "512.0"
DVALVBLL  :     DRIFT, L = "0.142"
DZERO     :     DRIFT, l = "0.0"
ICT01BI1  :  LSCDRIFT, lsc = "1.0", l = "0.15", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
ICT01BI3  :  LSCDRIFT, lsc = "1.0", l = "0.15", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
ICT01L0   :  LSCDRIFT, lsc = "1.0", l = "0.15", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
L0WAKE    :      WAKE, N_BINS = "0.0", wColumn = "W", interpolate = "1.0", SMOOTHING = "1.0", factor = "172.0", tColumn = "t",␣
↪INPUTFILE = "sband_L_10mm.sdds"
LM        :     DRIFT, l = "0.5"
LTU_BC11  :  CSRCSBEN, HGAP = "0.015", integration_order = "4.0", nonlinear = "1.0", ANGLE = "0.0546288055874", n_kicks = "10.0",␣
↪output_file = "%s.LTU_BC11.csr", L = "0.2", EDGE1_EFFECTS = "1.0", EDGE2_EFFECTS = "1.0", output_interval = "10.0", sg_halfwidth =
↪"1.0", isr = "0.0", E2 = "0.0546288055874", E1 = "0.0", bins = "600.0", csr = "0.0"
LTU_BC12  :  CSRCSBEN, HGAP = "0.015", integration_order = "4.0", nonlinear = "1.0", ANGLE = "-0.0546288055874", n_kicks = "10.0",␣
↪output_file = "%s.LTU_BC12.csr", L = "0.2", EDGE1_EFFECTS = "1.0", EDGE2_EFFECTS = "1.0", output_interval = "10.0", sg_halfwidth =
↪"1.0", isr = "0.0", E2 = "0.0", E1 = "-0.0546288055874", bins = "600.0", csr = "0.0"
LTU_BC13  :  CSRCSBEN, HGAP = "0.015", integration_order = "4.0", nonlinear = "1.0", ANGLE = "-0.0546288055874", n_kicks = "10.0",␣
↪output_file = "%s.LTU_BC13.csr", L = "0.2", EDGE1_EFFECTS = "1.0", EDGE2_EFFECTS = "1.0", output_interval = "10.0", sg_halfwidth =
↪"1.0", isr = "0.0", E2 = "-0.0546288055874", E1 = "0.0", bins = "600.0", csr = "0.0"
LTU_BC14  :  CSRCSBEN, HGAP = "0.015", integration_order = "4.0", nonlinear = "1.0", ANGLE = "0.0546288055874", n_kicks = "10.0",␣
↪output_file = "%s.LTU_BC14.csr", L = "0.2", EDGE1_EFFECTS = "1.0", EDGE2_EFFECTS = "1.0", output_interval = "10.0", sg_halfwidth =
↪"1.0", isr = "0.0", E2 = "0.0", E1 = "0.0546288055874", bins = "600.0", csr = "0.0"
LTU_BC21  :  CSRCSBEN, HGAP = "0.015", integration_order = "4.0", nonlinear = "1.0", ANGLE = "0.010471975512", n_kicks = "10.0",␣
↪output_file = "%s.LTU_BC21.csr", L = "0.2", EDGE1_EFFECTS = "1.0", EDGE2_EFFECTS = "1.0", output_interval = "10.0", sg_halfwidth =
↪"1.0", isr = "0.0", E2 = "0.010471975512", E1 = "0.0", bins = "600.0", csr = "0.0"
```

```
LTU_BC22  :  CSRCSBEN, HGAP = "0.015", integration_order = "4.0", nonlinear = "1.0", ANGLE = "-0.010471975512", n_kicks = "10.0",␣
↪output_file = "%s.LTU_BC22.csr", L = "0.2", EDGE1_EFFECTS = "1.0", EDGE2_EFFECTS = "1.0", output_interval = "10.0", sg_halfwidth =
↪"1.0", isr = "0.0", E2 = "0.0", E1 = "-0.010471975512", bins = "600.0", csr = "0.0"
LTU_BC23  :  CSRCSBEN, HGAP = "0.015", integration_order = "4.0", nonlinear = "1.0", ANGLE = "-0.010471975512", n_kicks = "10.0",␣
↪output_file = "%s.LTU_BC23.csr", L = "0.2", EDGE1_EFFECTS = "1.0", EDGE2_EFFECTS = "1.0", output_interval = "10.0", sg_halfwidth =
↪"1.0", isr = "0.0", E2 = "-0.010471975512", E1 = "0.0", bins = "600.0", csr = "0.0"
LTU_BC24  :  CSRCSBEN, HGAP = "0.015", integration_order = "4.0", nonlinear = "1.0", ANGLE = "0.010471975512", n_kicks = "10.0",␣
↪output_file = "%s.LTU_BC24.csr", L = "0.2", EDGE1_EFFECTS = "1.0", EDGE2_EFFECTS = "1.0", output_interval = "10.0", sg_halfwidth =
↪"1.0", isr = "0.0", E2 = "0.0", E1 = "0.010471975512", bins = "600.0", csr = "0.0"
LTU_CBD11 :   CSRDRIF, N_kicks = "1.0", L = "0.7", USE_STUPAKOV = "1.0"
LTU_CBD12 :   CSRDRIF, N_kicks = "1.0", L = "0.6", USE_STUPAKOV = "1.0"
LTU_CBD13 :   CSRDRIF, N_kicks = "1.0", L = "0.7", USE_STUPAKOV = "1.0"
LTU_CBD21 :   CSRDRIF, N_kicks = "1.0", L = "0.7", USE_STUPAKOV = "1.0"
LTU_CBD22 :   CSRDRIF, N_kicks = "1.0", L = "0.6", USE_STUPAKOV = "1.0"
LTU_CBD23 :   CSRDRIF, N_kicks = "1.0", L = "0.7", USE_STUPAKOV = "1.0"
LTU_L0    :      DRIF, L = "0.54"
LTU_L1    :      DRIF, L = "0.47"
LTU_L10   :      DRIF, L = "3.575"
LTU_L11   :      DRIF, L = "0.839"
LTU_L12   :      DRIF, L = "3.84"
LTU_L13   :      DRIF, L = "3.84"
LTU_L14   :      DRIF, L = "3.84"
LTU_L15   :      DRIF, L = "3.84"
LTU_L16   :      DRIF, L = "0.865"
LTU_L17   :      DRIF, L = "0.9"
LTU_L18   :      DRIF, L = "0.775"
LTU_L19   :      DRIF, L = "0.91"
LTU_L2    :      DRIF, L = "3.069"
LTU_L20   :      DRIF, L = "3.004"
LTU_L21   :      DRIF, L = "0.775"
LTU_L22   :      DRIF, L = "0.839"
LTU_L3    :      DRIF, L = "3.069"
LTU_L4    :      DRIF, L = "0.55"
LTU_L5    :      DRIF, L = "0.6"
LTU_L6    :      DRIF, L = "12.642"
LTU_L7    :      DRIF, L = "0.55"
LTU_L8    :      DRIF, L = "0.6"
LTU_L9    :      DRIF, L = "2.594"
LTU_Q0    :      QUAD, K1 = "-2.04092178418", L = "0.1"
LTU_Q1    :      QUAD, K1 = "-3.92247060497", L = "0.1"
LTU_Q10   :      QUAD, K1 = "-0.168396320285", L = "0.1"
LTU_Q11   :      QUAD, K1 = "0.795456339652", L = "0.1"
LTU_Q12   :      QUAD, K1 = "0.631804583095", L = "0.1"
LTU_Q13   :      QUAD, K1 = "-2.0", L = "0.1"
LTU_Q14   :      QUAD, K1 = "2.0", L = "0.1"
LTU_Q15   :      QUAD, K1 = "-2.0", L = "0.1"
LTU_Q16   :      QUAD, K1 = "1.24927520914", L = "0.1"
LTU_Q17   :      QUAD, K1 = "-0.0941988441023", L = "0.1"
LTU_Q18   :      QUAD, K1 = "0.901961496764", L = "0.1"
LTU_Q19   :      QUAD, K1 = "-1.22468814231", L = "0.1"
LTU_Q2    :      QUAD, K1 = "3.43160610965", L = "0.1"
LTU_Q20   :      QUAD, K1 = "-1.09435990917", L = "0.1"
LTU_Q21   :      QUAD, K1 = "2.19055626432", L = "0.1"
LTU_Q22   :      QUAD, K1 = "-2.28862467644", L = "0.1"
LTU_Q3    :      QUAD, K1 = "0.651660173421", L = "0.1"
LTU_Q4    :      QUAD, K1 = "1.0", L = "0.1"
LTU_Q5    :      QUAD, K1 = "0.0", L = "0.1"
LTU_Q6    :      QUAD, K1 = "-2.0", L = "0.1"
LTU_Q7    :      QUAD, K1 = "2.0", L = "0.1"
LTU_Q8    :      QUAD, K1 = "1.24003413647", L = "0.1"
LTU_Q9    :      QUAD, K1 = "-3.38716619143", L = "0.1"
LTU_W0    :     WATCH, FILENAME = "C0.out"
PRF01BC1  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF01BC2  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF01BI1  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF01BI2  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF01BI3  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF01L0   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF01L1   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF01L2   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF01L3   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
```

```
PRF02BC1  :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.2", csr = "1.0"
PRF02BC2  :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.115", csr = "1.0"
PRF02BI1  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF02BI3  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF02L0   :  LSCDRIFT, lsc = "1.0", l = "0.14", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF02L2   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF02L3   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF03BI1  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF03BI3  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF03L0   :  LSCDRIFT, lsc = "1.0", l = "0.12", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF03L3   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF04BI1  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF04L0   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF04L3   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF05BI1  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF05L3   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF06BI1  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF06L3   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF07BI1  :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PRF07L3   :  LSCDRIFT, lsc = "1.0", l = "0.115", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
PSTN01    :      MARK, FITPOINT = "1.0"
PSTN02    :      MARK, FITPOINT = "1.0"
PSTN03    :      MARK, FITPOINT = "1.0"
PSTN04    :      MARK, FITPOINT = "1.0"
PSTN05    :      MARK, FITPOINT = "1.0"
PSTN06    :      MARK, FITPOINT = "1.0"
PSTN07    :      MARK, FITPOINT = "1.0"
PSTN08    :      MARK, FITPOINT = "1.0"
PSTN09    :      MARK, FITPOINT = "1.0"
PSTN10    :      MARK, FITPOINT = "1.0"
PSTN11    :      MARK, FITPOINT = "1.0"
PSTN12    :      MARK, FITPOINT = "1.0"
PSTN13    :      MARK, FITPOINT = "1.0"
PSTN14    :      MARK, FITPOINT = "1.0"
PSTN15    :      MARK, FITPOINT = "1.0"
Q01BC1    :      QUAD, K1 = "-4.64573", L = "0.2"
Q01BC2    :      QUAD, K1 = "-2.36377", L = "0.2"
Q01BI1    :      QUAD, K1 = "0.4", L = "0.2"
Q01BI2    :      QUAD, K1 = "-2.44726", L = "0.2"
Q01BI3    :      QUAD, K1 = "1.0", L = "0.2"
Q01L0     :      QUAD, K1 = "3.95", L = "0.1"
Q01L1     :      QUAD, K1 = "-5.62694", L = "0.1"
Q01L2H    :      QUAD, K1 = "-1.02711", L = "0.1"
Q01L3H    :      QUAD, K1 = "-1.08303", L = "0.1"
Q02BC1    :      QUAD, K1 = "4.3335", L = "0.2"
Q02BC2    :      QUAD, K1 = "2.66697", L = "0.2"
Q02BI1    :      QUAD, K1 = "-0.8", L = "0.2"
Q02BI2    :      QUAD, K1 = "2.92352", L = "0.2"
Q02BI3    :      QUAD, K1 = "-0.6", L = "0.2"
Q02L0     :      QUAD, K1 = "-3.75", L = "0.2"
Q02L1     :      QUAD, K1 = "5.59702", L = "0.2"
Q02L3H    :      QUAD, K1 = "0.99491", L = "0.1"
Q03BC1    :      QUAD, K1 = "0.0", L = "0.1"
Q03BI1    :      QUAD, K1 = "-4.12047", L = "0.2"
Q03BI2    :      QUAD, K1 = "0.0971", L = "0.2"
Q03BI3    :      QUAD, K1 = "3.00056", L = "0.2"
Q03L0     :      QUAD, K1 = "3.95", L = "0.1"
Q03L1     :      QUAD, K1 = "-5.62694", L = "0.1"
Q03L3H    :      QUAD, K1 = "-1.00454", L = "0.1"
Q04BI1    :      QUAD, K1 = "3.24914", L = "0.2"
Q04BI3    :      QUAD, K1 = "-2.87761", L = "0.2"
Q04L0     :      QUAD, K1 = "4.27971", L = "0.1"
```

```
Q04L3H    :     QUAD, K1 = "0.99491", L = "0.1"
Q05BC1    :     QUAD, K1 = "0.0", L = "0.1"
Q05BI1    :     QUAD, K1 = "1.6603", L = "0.2"
Q05L0     :     QUAD, K1 = "-4.08602", L = "0.2"
Q05L3H    :     QUAD, K1 = "-1.00454", L = "0.1"
Q06BI1    :     QUAD, K1 = "1.8468", L = "0.2"
Q06L0     :     QUAD, K1 = "4.27971", L = "0.1"
Q06L3H    :     QUAD, K1 = "0.99491", L = "0.1"
Q07BI1    :     QUAD, K1 = "-3.13483", L = "0.2"
Q07L0     :     QUAD, K1 = "0.0", L = "0.1"
Q07L3H    :     QUAD, K1 = "-1.00454", L = "0.1"
Q08BI1H   :     QUAD, K1 = "-4.60769", L = "0.1"
Q08L0     :     QUAD, K1 = "0.0", L = "0.1"
Q09BI1H   :     QUAD, K1 = "4.60769", L = "0.1"
Q10BI1H   :     QUAD, K1 = "-4.60769", L = "0.1"
Q11BI1H   :     QUAD, K1 = "4.60769", L = "0.1"
Q12BI1H   :     QUAD, K1 = "-4.60769", L = "0.1"
Q13BI1H   :     QUAD, K1 = "4.60769", L = "0.1"
Q14BI1H   :     QUAD, K1 = "-4.60769", L = "0.1"
Q15BI1H   :     QUAD, K1 = "4.60769", L = "0.1"
Q16BI1    :     QUAD, K1 = "3.47744", L = "0.2"
Q17BI1    :     QUAD, K1 = "-4.00489", L = "0.2"
Q18BI1    :     QUAD, K1 = "-3.48617", L = "0.2"
Q19BI1    :     QUAD, K1 = "4.21129", L = "0.2"
SLT01BC1  :   CSRDRIF, use_stupakov = "1.0", dz = "0.01", l = "0.21", csr = "1.0"
TDS01BI1  :     RFDF, tilt = "1.5707963268", l = "1.4", frequency = "2856000000.0", voltage = "0.0", n_kicks = "34.0", phase = "90.0"
TDS01BI3  :     RFDF, tilt = "1.5707963268", l = "1.4", frequency = "2856000000.0", voltage = "0.0", n_kicks = "172.0", phase = "90.0"
TDS01L0   :     RFDF, tilt = "1.5707963268", l = "0.656", frequency = "2856000000.0", voltage = "0.0", n_kicks = "34.0", phase = "90.0"
VALV      :  LSCDRIFT, lsc = "1.0", l = "0.072", interpolate = "1.0", smoothing = "1.0", high_frequency_cutoff1 = "0.3", high_
↪frequency_cutoff0 = "0.25", bins = "512.0"
W0        :     WATCH, mode = "coord", filename = "L0.out"
W1        :     WATCH, mode = "coord", filename = "L1.out"
W2        :     WATCH, mode = "coord", filename = "BC1.out"
W3        :     WATCH, mode = "coord", filename = "L2.out"
W4        :     WATCH, mode = "coord", filename = "BC2.out"
W5        :     WATCH, mode = "coord", filename = "L3.out"

! Beamline definitions:
SXFEL     : line = (c, l0wake, dzero, dzero, d01l0, q01l0, d02l0, q02l0, d03l0, q03l0, d04l0, bll01, bpm01l0, crr01l0, dcrr, prf01l0,␣
↪valv, bll01, d05l0, b1lh, d06l0, b2lh, bpm02l0, bll02, prf02l0, d07l0, lm, d07l0, prf03l0, d08l0, b3lh, d09l0, b4lh, d10l0, bll01,␣
↪d11l0, q04l0, d12l0, q05l0, d13l0, q06l0, d14l0, bll01, bpm03l0, crr02l0, dcrr, prf04l0, bll01, tds01l0, bll01, d15l0, q07l0,␣
↪d16l0, q08l0, d17l0, bam01l0, ict01l0, bll01, dbend01l0, bll01, dbpm, w0, dzero, d01l1, q01l1, d02l1, q02l1, d03l1, q03l1, d04l1,␣
↪bll01, bpm01l1, crr01l1, dcrr, dbpmlsc, prf01l1, valv, bll01, dawgs, as1o1l1, dawgs, bll03, dawgs, as1o1l1, dawgs, bll01, valv,␣
↪bll01, dawgx1, ax1o1l1, dawgx2, bll01, valv, pstn01, w1, dzero, d01bc1, q01bc1, d02bc1, q02bc1, d03bc1, bll01, bpm01bc1, crr01bc1,␣
↪dcrr, dbpmlsc, dbpmlsc, prf01bc1, bll01, d04bc1, b1bc1, d05bc1, bll01, d06bc1, q03bc1, d07bc1, crr02bc1, d08dbc1csr, d08dbc1lsc,␣
↪d08dbc1csr, d08dbc1lsc, d08dbc1csr, d08dbc1lsc, d08dbc1csr, d08dbc1lsc, d08dbc1csr, d08dbc1lsc, d08dbc1csr, d08dbc1lsc,␣
↪d08dbc1csr, d08dbc1lsc, d08dbc1csr, d08dbc1lsc, d08dbc1csr, d08dbc1lsc, d08dbc1csr, d08dbc1lsc, bll01, d09bc1, b2bc1, d10bc1,␣
↪bll02, bpm02bc1, bll02, slt01bc1, prf02bc1, bll02, d12bc1csr, d12bc1lsc, b3bc1, d13bc1, bll01, d14dbc1csr, d14dbc1lsc, d14dbc1csr,␣
↪d14dbc1lsc, d14dbc1csr, d14dbc1lsc, d14dbc1csr, d14dbc1lsc, d14dbc1csr, d14dbc1lsc, d14dbc1csr, d14dbc1lsc, d14dbc1csr,␣
↪d14dbc1lsc, d14dbc1csr, d14dbc1lsc, d14dbc1csr, d14dbc1lsc, d14dbc1csr, d14dbc1lsc, crr03bc1, d15bc1, q05bc1, d16bc1, bll01,␣
↪d17bc1, b4bc1, pstn02, d18bc1, bll01, cdr01bc1, w2, dzero, dzero, dzero, d01bi1, q01bi1, d02bi1, q02bi1, d03bi1, valv, bll01,␣
↪tds01bi1, bll01, valv, d04bi1, q03bi1, d05bi1, q04bi1, d06bi1, bll01, bpm01bi1, crr01bi1, dcrr, prf01bi1, d07bi1, q05bi1, d08bi1,␣
↪bll01, bam01bi1, ict01bi1, d09bi1, q06bi1, d10bi1, q07bi1, d11bi1, bll01, bpm02bi1, crr02bi1, dcrr, prf02bi1, d12bi1, q08bi1h,␣
↪pstn04, dzero, q08bi1h, dfodo2bi1, q09bi1h, pstn03, q09bi1h, d13bi1, bll01, bpm03bi1, crr03bi1, dcrr, prf03bi1, dfodo1bi1,␣
↪q10bi1h, pstn03, q10bi1h, dfodo2bi1, q11bi1h, q11bi1h, d14bi1, bll01, bpm04bi1, crr04bi1, dcrr, prf04bi1, dfodo1bi1, q12bi1h,␣
↪q12bi1h, dfodo2bi1, q13bi1h, q13bi1h, d15bi1, bll01, bpm05bi1, crr05bi1, dcrr, prf05bi1, dfodo1bi1, q14bi1h, q14bi1h, dfodo2bi1,␣
↪q15bi1h, q15bi1h, d16bi1, bll01, bpm06bi1, crr06bi1, dcrr, prf06bi1, dzero, d17bi1, q16bi1, d18bi1, q17bi1, d19bi1, bll01,␣
↪bpm07bi1, crr07bi1, dcrr, prf07bi1, dbend01bi1, bll01, d20bi1, q18bi1, d21bi1, q19bi1, d22bi1, pstn05, dzero, dzero, bll01,␣
↪bpm01l2, crr01l2, dcrr, dbpmlsc, prf01l2, valv, d01l2, bll01, dacl2, bll04, dacl2, d02l2, q01l2h, pstn06, q01l2h, d03l2, bll01,␣
↪bpm02l2, crr02l2, dcrr, prf02l2, bll01, dawgc1, ac1o1l2_1, dawgc2, bll04, dawgc1, ac1o1l2_1, dawgc2, bll01, dawgc1, ac1o1l2_2,␣
↪dawgc2, bll04, dawgc1, ac1o1l2_2, dawgc2, pstn07, w3, dzero, dzero, d01bc2, q01bc2, d02bc2, q02bc2, d03bc2, bll01, bpm01bc2,␣
↪crr01bc2, dcrr, dbpmlsc, prf01bc2, valv, bll01, d04dbc2, d04dbc2, d04dbc2, d04dbc2, d04dbc2, d04dbc2, d04dbc2, d04dbc2, d04dbc2,␣
↪d04dbc2, bll01, bpm02bc2, crr02bc2, dcrr, dbpmlsc, prf02bc2, bll01, d05dbc2, d05dbc2, d05dbc2, d05dbc2, d05dbc2, d05dbc2, d05dbc2,␣
↪d05dbc2, d05dbc2, d05dbc2, bll01, pstn08, w4, dzero, dzero, d01bi2, q01bi2, d02bi2, q02bi2, d03bi2, bll01, bpm01bi2, crr01bi2,␣
↪dcrr, dbpmlsc, prf01bi2, d04bi2, q03bi2, d05bi2, dzero, dzero, d01l3, q01l3h, pstn09, q01l3h, d02l3, bll01, bpm01l3, crr01l3,␣
↪dcrr, prf01l3, valv, bll01, dawgc1, ac1o1l3_1, dawgc2, bll04, dawgc1, ac1o1l3_1, dawgc2, d03l3, q02l3h, dzero, pstn10, q02l3h,␣
↪d04l3, bll01, bpm02l3, crr02l3, dcrr, prf02l3, valv, bll01, dawgc1, ac1o1l3_2, dawgc2, bll04, dawgc1, ac1o1l3_2, dawgc2, dzero,␣
↪d05l3, q03l3h, pstn11, q03l3h, d06l3, bll01, bpm03l3, crr03l3, dcrr, prf03l3, valv, bll01, dawgc1, ac1o1l3_3, dawgc2, bll04,␣
↪dawgc1, ac1o1l3_3, dawgc2, dzero, d07l3, q04l3h, pstn12, q04l3h, d08l3, bll01, bpm04l3, crr04l3, dcrr, prf04l3, valv, bll01,␣
↪dawgc1, ac1o1l3_4, dawgc2, bll04, dawgc1, ac1o1l3_4, dawgc2, dzero, d09l3, q05l3h, pstn13, q05l3h, d10l3, bll01, bpm05l3, crr05l3,␣
↪dcrr, prf05l3, valv, bll01, dacl3, bll04, dacl3, dzero, d11l3, q06l3h, pstn14, q06l3h, d12l3, bll01, bpm06l3, crr06l3, dcrr,␣
↪prf06l3, dvalvbll, dacl3, bll04, dacl3, dzero, d13l3, q07l3h, pstn15, q07l3h, d14l3, bll01, bpm07l3, crr07l3, dcrr, prf07l3,␣
↪dvalvbll, dacl3, bll04, dacl3, w5, dzero, dzero, d01bi3, q01bi3, d02bi3, q02bi3, d03bi3, bll01, bpm01bi3, crr01bi3, dcrr,␣
↪prf01bi3, valv, bll01, tds01bi3, bll01, valv, d04bi3, q03bi3, d05bi3, q04bi3, d06bi3, bll01, bpm02bi3, crr02bi3, dcrr, prf02bi3,␣
↪d07bi3, bam01bi3, ict01bi3, bll01, dbend01bi3, bll01, bpm03bi3, crr03bi3, dcrr, prf03bi3, ltu_q0, ltu_l0, ltu_q1, ltu_l1, ltu_q2,␣
↪ltu_l2, ltu_q3, ltu_l3, ltu_q4, ltu_l4, ltu_q5, ltu_l5, ltu_q6, ltu_l6, ltu_q7, ltu_l7, ltu_q8, ltu_l8, ltu_q9, ltu_l9, ltu_q10,␣
↪ltu_l10, ltu_q11, ltu_l11, ltu_q12, ltu_l12, ltu_q13, ltu_l13, ltu_q14, ltu_l14, ltu_q15, ltu_l15, ltu_q16, ltu_l16, ltu_q17, ltu_
↪l17, ltu_q18, ltu_l18, ltu_bc11, ltu_cbd11, ltu_bc12, ltu_cbd12, ltu_bc13, ltu_cbd13, ltu_bc14, ltu_q19, ltu_l19, ltu_q20, ltu_
↪l20, ltu_q21, ltu_l21, ltu_bc21, ltu_cbd21, ltu_bc22, ltu_cbd22, ltu_bc23, ltu_cbd23, ltu_bc24, ltu_q22, ltu_l22, ltu_w0)
```

Note: The grouped highlighted lines indicate the EPICS control configurations of the elements if defined with.

Example 3

Lattice/Twiss matching by using `matchutils` module of `beamline` package for free-electron laser facility.

Module `matchutils` provides classes/functions to handle issues about numerical simulation of free-electron laser (FEL), including Twiss matching, to optimize an FEL, at the simulation stage.

In this example, we will tune the FODO lattice of an high-gain harmonic generation (HGHG) FEL, to figure out the best Twiss parameters when the electron beam enters into the undulator line, such matched Twiss parameter could be used as the lattice matching goal of an matching application.

Here is the code for demostration:

```
#!/usr/bin/python
# -*- coding: utf-8 -*-

from beamline import parseLattice, ParseParams, BeamMatch, FELSimulator


def test():
    testParse = ParseParams('rad.in', 'rad.lat')
    aw0 = testParse.getUndulatorParameter()
    xlamd = testParse.getUndulatorPeriod()
    unitlength = testParse.getUndulatorUnitlength()
    xlamds = testParse.getFELwavelength()
    gamma0 = testParse.getElectronGamma()
    emitx = testParse.getElectronEmitx()
    imagl = testParse.getChicaneMagnetLength()
    idril = testParse.getChicaneDriftLength()
    ibfield = testParse.getChicaneMagnetField()
    """
    print("aw0    = %.3f" % aw0   )
    print("xlamd  = %.3f" % xlamd )
    print("xlamds = %.3e" % xlamds)
    print("gamma0 = %.3f" % gamma0)
    print("emitx  = %.3e" % emitx )
    print("imagl  = %.3f" % imagl )
    print("idril  = %.3f" % idril )
    print("ibfield= %.3f" % ibfield)
    print("unit   = %.3f" % unitlength)
    print parseLattice('fullat.hghg')
    """

    qf, qd = -1, 2

    testMatch = BeamMatch('mod.in', 'rad.in', 'mod.lat', 'rad.lat',
                          'newmod.in', 'newrad.in', 'newrad.lat', qf, qd)
    if testMatch.matchCalculate():
        testMatch.matchPerform()
        #testMatch.matchPrintout()
        fel = FELSimulator()
        fel.run()
        fel.postProcess()
        print fel.getMaxPower()

if __name__ == '__main__':
    test()
```

The FEL physics related files:

- `Modulator input file`
- `Modulator lattice file`
- `Radiator input file`

- `Radiator lattice file`
- `Lattice configuration file`

Wrap it up:

- Create `ParseParams()` instance to get needed parameter values;
- **Create `BeamMatch()` instance to resolve matching issues:**
  - Invoke `mathCalculate()` to figure out the Twiss parameter required;
  - `run()` and `postProcess()` methods of `FELSimulator` to produce the simulation and get output files;
  - `genesis 1.3` is used to handle the FEL simulation.

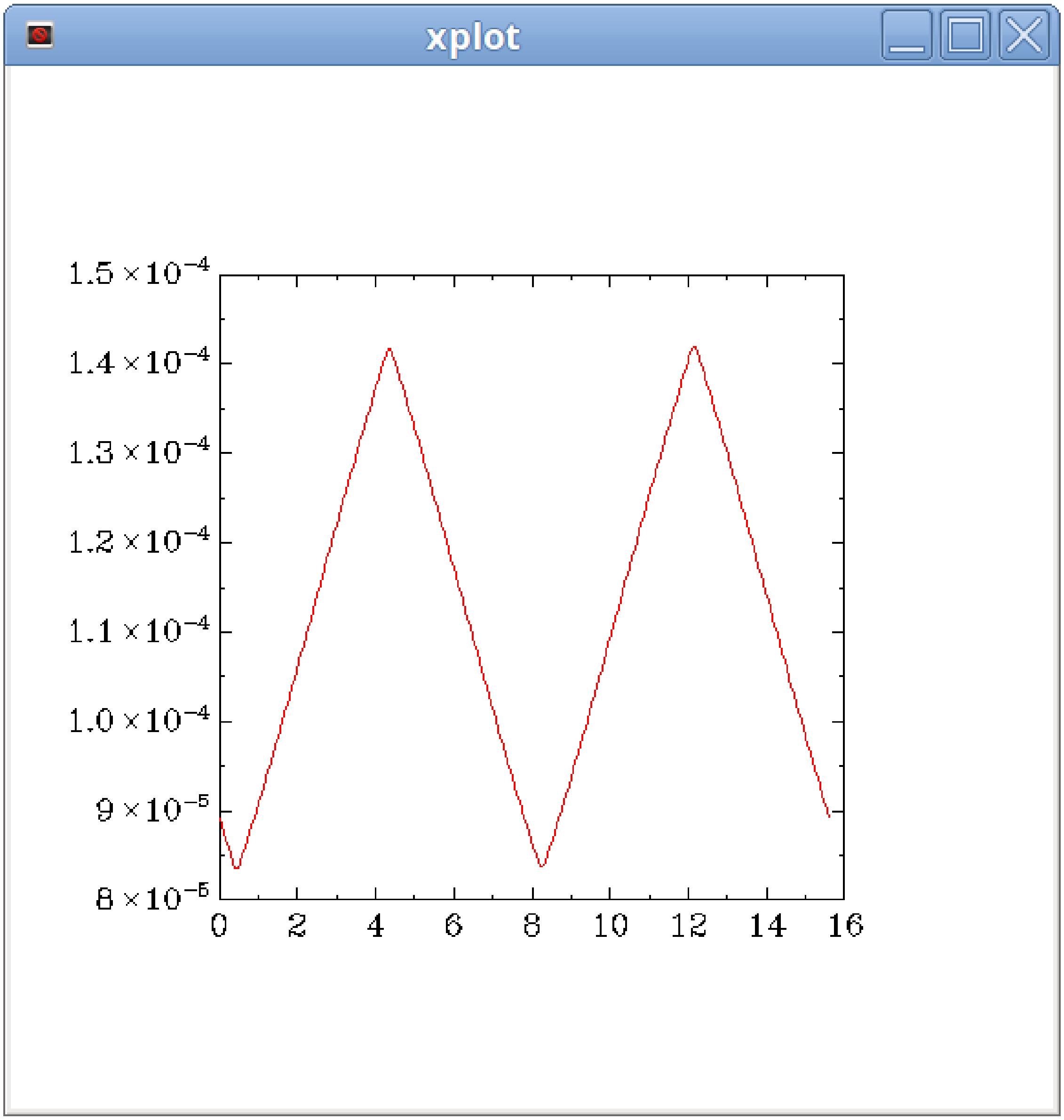
Tips about the post processing with the `genesis 1.3` generated files:

- For steady-stat (single slice) simulation mode, use this shell script (name: `getssdata.sh`)to get well-formated output data by columns:

```
tempdir=/tmp/tmp.$$
outfile="rad.out"
awk '/z\[m\]/,/current/' ${outfile} | tr -d '\r' | sed "/^$/d;/[*,=,cur]/d;s/^[ ,\t]*//;1s/^/#/" > ${tempdir}/zaq
awk '/current/,/\$/' ${outfile} | tr -d '\r' | sed "1d;/^$/d;s/^[ ,\t]*//" > ${tempdir}/outdata
paste ${tempdir}/zaq ${tempdir}/outdata
```

- Show the beam size variation along undulator:

```
getssdata.sh rad.out | awk '{print $1,$13}' | graph -T X -C
```



Todo

Integrate FEL physics manipulation, like Twiss matching into online modeling framework, and develop online-modeling optimization modules.

Example 4

This example presents the scripts/commands that `beamline` provides, demonstrate how to make them useful.

lte2json

Convert `.lte` file into JSON string format.

```
usage: lte2json [-h] [--lte LTEFILE] [--json JSONFILE]

Convert Elegant lte file into json string file

optional arguments:
  -h, --help       show this help message and exit
  --lte LTEFILE    .lte file to read
  --json JSONFILE  .json file to generate, if None or omitted, write to stdout
```

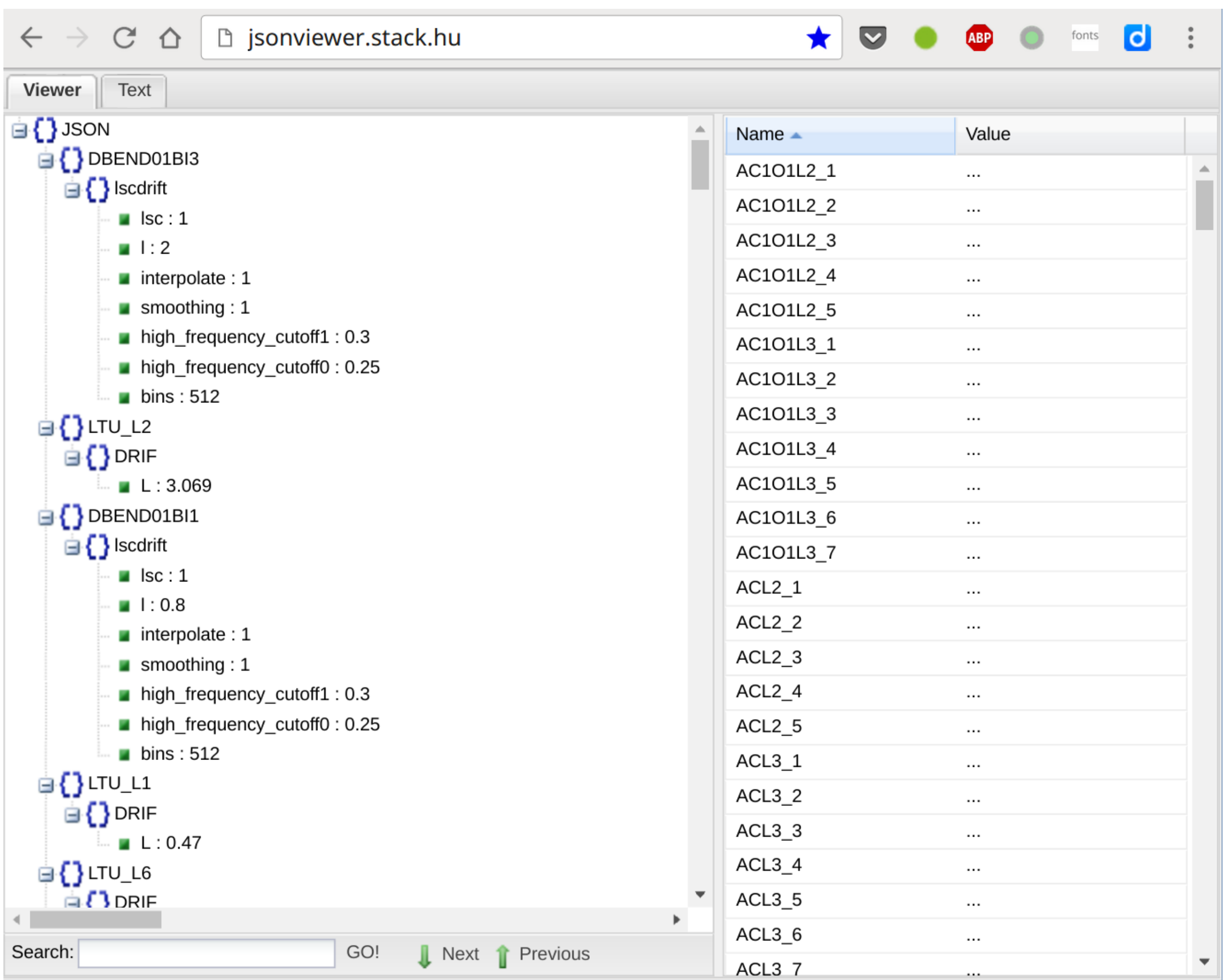
Example

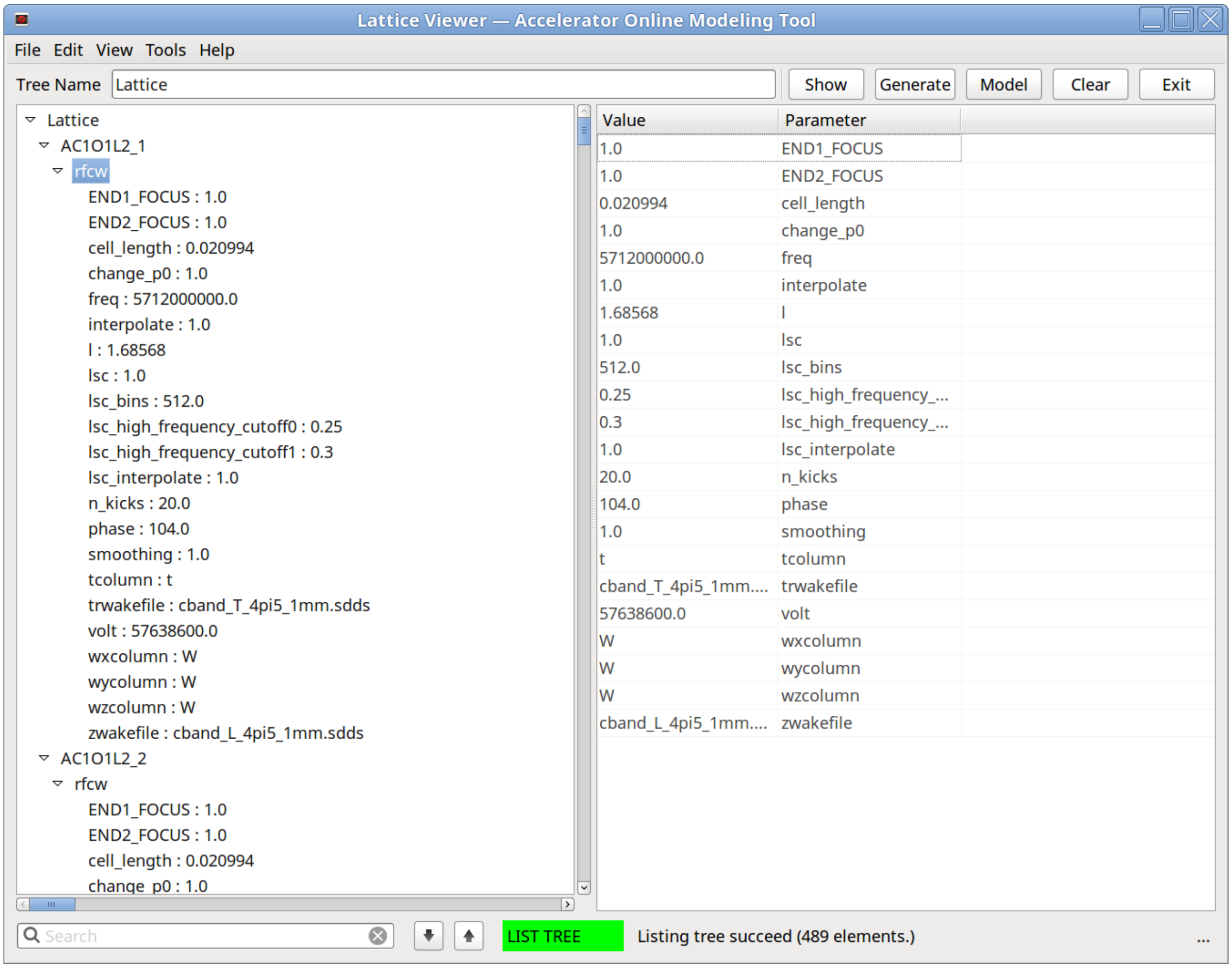
lte2json –lte sxfel_all.lte –json sxfel_all.json

Note:

- Download `sxfel_all.lte` here
- Recommand an online general JSON file viewer: http://jsonviewer.stack.hu/
- Open the generated `.json` file by `latticeviewer`, which is a GUI application distributed with `beamline` package

---





json2lte

```
usage: json2lte [-h] [--json JSONFILE] [--lte LTEFILE] [--bl BLNAME]
                [--show SHOW_FLAG]

Read json file and generate lte file with the given beamline name

optional arguments:
  -h, --help        show this help message and exit
  --json JSONFILE   .json file to read
  --lte LTEFILE     .lte file to generate
  --bl BLNAME       name of beamline
  --show SHOW_FLAG  show all valid beamline names, --json should be valid
```

Example

To inspect the beamline names:

```
json2lte --json sxfel_all.json --show True
```

The output:

```
M1BI3,M1BI1,BLEMIT1,D14DBC1,ASL1,LTU_BC1,LTU_BC2,BL1,ACL2_
↪4,LTU,H07L3,AXL1,L2,L3,L0,L1,H03L3,D08DBC1,D12BC1,H01BI1,H01L3,SXFEL,H05L3,BI2,BI3,H04BI1,LTU_BL1,LTU_
↪BL2,H02BI1,TESTLINE,H03BI1,M2BI1,BC1,BC2,H02L3,BL,FODOBI1,ACL2_1,ACL2_3,ACL2_2,ACL2_5,ACL3_2,ACL3_5,H04L3,BI1,H06L3,ACL3_1,ACL3_
↪3,ACL3_4,ACL3_6,ACL3_7
```

Generate `.lte` file with beamline `SXFEL` exclusively:

```
json2lte --json sxfel_all.json --lte sxfel.lte --bl SXFEL
```

The generate files: `sxfel.lte`, could be read by `Elegant`.

Note: `latticeviewer` could read both `.lte` file and `.json` file, and accomplish the format conversion tasks.

CHAPTER

FOUR

API

beamline package

Python package created for lattice generation, operation, manipulation, visualization and accelerator online modeling, distributed with both console and graphical user interfaces environment.

To evoke the GUI app:

1. run `lv` or `latticeviewer` in terminal after `beamline` package is installed;
2. `beamline.ui_main.run()` in [i]python terminal after `beamline` is imported.

Version 1.3.6

Author Tong Zhang (zhangtong@sinap.ac.cn)

class Drift(*length=2.0*, *angle=0.0*, *link_node=(0.0, 0.0)*, *line_color='black'*, *line_width=1.5*, *_alpha=0.8*)

Bases: `object`

Element drift section

Parameters

- **_length** – drift length, [m]
- **_angle** – angle between drawing line and horizontal plane, [deg]
- **_linkNode** – `(x,y)` coordinates that drawing begins or linked to another element

show(*fignum=1*)

class Rbend(*width=1.0*, *height=2.0*, *angle=0.0*, *link_node=(0.0, 1.0)*, *face_color='red'*, *edge_color='red'*, *line_width=0.1*, *_alpha=0.8*)

Bases: `object`

Element rectangle bend

Parameters

- **_width** – bend width, [m]
- **_height** – bend hight, [m]
- **_angle** – bend angle, [deg]
- **_linkNode** – `(x,y)` coordinates that drawing begins or linked to another element

info()

show(*fignum=1*)

class Undulator(*period_length=2.0, period_number=10, north_color='red', south_color='blue', link_node=(0, 0), ratio=[2.5, 1.5], spacing=1.5, _alpha=0.8*)

Bases: `object`

Element undulator (not included in `element` module)

Parameters

- **period_length** – undulator period length, [m]
- **period_number** – undulator period number
- **north_color** – color of north pole
- **south_color** – color of south pole
- **link_node** – `(x,y)` coordinates that drawing begins or linked to another element
- **ratio** – ratio of pole_height v.s. pole_width, and gap v.s pole_width
- **spacing** – spacing between pole, measured by pole_width,
- **gap** – undulator gap only for visualization, not true magnetic gap

show(*fignum=1*)

class Quadrupole(*width=1.0, angle=75, xysign='x', link_node=(0.0, 1.0), face_color='blue', edge_color='blue', line_width=0.1, _alpha=0.8*)

Bases: `object`

Element quadrupole

Parameters

- **width** – quad width, [m]
- **angle** – angle, [deg]
- **xy_sign** – `x`: x-focusing, `K1>0`; `y`: y-focusing, `K1<0`
- **link_node** – `(x,y)` coordinates that drawing begins or linked to another element

show(*fignum=1*)

plotLattice(*beamlinepatchlist, fignum=1, fig_size=20, fig_ratio=0.5, xranges=(-10, 10), yranges=(-10, 10), zoomfac=1.5*)

function plot beamline defined by `beamlinepatchlist`, which is a set of patches for all elements

Parameters

- **beamlinepatchlist** – generated by function `makeBeamline()`
- **fignum** – figure number, 1 by default
- **fig_size** – figure size, 20 inch by default
- **fig_ratio** – figure ratio, 0.5 by default
- **xranges** – axes x-ranges, `(-10,10)` by default
- **yranges** – axes y-ranges, `(-10,10)` by default
- **zoomfac** – zoom in factor, 1.5 by default

makeBeamline(*beamlinelist, startpoint=(0, 0)*)

function to construct patches for `plotLattice()`, from different elements like `rbend`, `quadrupole`,etc. parsing from lattice file, mad-8. drift sections are calculated from other elements.

Input parameters:

Parameters

- **`beamlinelist`** – list, which elements are dict, each dict is the description for magnetic element, should be returned from module `blparser`, function `madParser()`
- **`startpoint`** – pos to start drawing, `(0,0)` by default

Returns tuple of beamline patches list, xlim and ylim * beamline patches list: patches to be drawn * xlim: data limit along x-axis * ylim: data limit along y-axis

`madParser`(*mad_filename*, *idbl='BL'*)

function to parse beamline with MAD-8 input format

Parameters

- **`mad_filename`** – lattice filename with mad-8 like format
- **`idbl`** – beamline to be used that defined in lattice file, default value is `BL`

Returns list of dict that contains magnetic elements

Return type list

Example

```
>>> import beamline
>>> beamlinelist = beamline.blparser.madParser('LPA.list', 'BL2')
>>> print beamlinelist
>>> [{'type': 'drift', 'l': '0.1', 'ID': 'd0'}, {'type': 'quad', 'k1': '75', 'angle': '75', 'l': '0.1', 'ID': 'q1'}, {'type':
↪'drift', 'l': '0.18', 'ID': 'd3'}, {'type': 'quad', 'k1': '-75', 'angle': '75', 'l': '0.1', 'ID': 'q2'}, {'type': 'drift',
↪'l': '0.27', 'ID': 'd6'}, {'type': 'rbend', 'angle': '10', 'l': '0.1', 'ID': 'b1'}, {'type': 'drift', 'l': '1.0', 'ID': 'd8
↪'}, {'type': 'rbend', 'angle': '-5', 'l': '0.1', 'ID': 'b2'}, {'type': 'drift', 'l': '0.45', 'ID': 'd4'}, {'type': 'quad',
↪'k1': '75', 'angle': '75', 'l': '0.1', 'ID': 'q1'}]
```

Download `LPA.list` for reference.

class `Lattice`(*elements*)

Bases: `object`

class for handling lattice configurations and operations

`dumpAllElements`()

dump all element configuration lines as json format.

`formatElement`(*kw*, *format='elegant'*)

convert json/dict of element configuration into elegant/mad format :param kw: keyword

`generateLatticeFile`(*beamline*, *filename=None*, *format='elegant'*)

generate simulation files for lattice analysis, e.g. ".lte" for elegant, ".madx" for madx

input parameters: :param beamline: keyword for beamline :param filename: name of lte/mad file,

if None, output to stdout; if 'sio', output to a string as return value; other cases, output to filename;

Parameters **`format`** – madx, elegant, 'elegant' by default, generated lattice is for elegant tracking

`generateLatticeLine`(*latname='newline'*, *line=None*)

construct a new lattice line :param latname: name for generated new lattice

`getAllBl`()

return all beamline keywords

getAllEle()
return all element keywords

getAllKws()
extract all keywords into two categories

kws_ele: magnetic elements kws_bl: beamline elements

return (kws_ele, kws_bl)

getBeamline(*beamlineKw*)
get beamline definition from all_elements, return as a list :param beamlineKw: keyword of beamline

getChargeElement()
return charge element name

getElementByName(*beamline*, *name*)
return element list by literal name in beamline each element is tuple like (name, type, order) :param beamline: beamline name :param name: element literal name

getElementByOrder(*beamline*, *type*, *irange*)

return element list by appearance order in beamline, which could be returned by orderLattice(beamline)

param beamline beamline name

param type element type name

param irange selected element range

possible irange definitions: irange = 0, first one 'type' element; irange = -1, last one irange = 0,2,3, the first, third and fourth 'type' element irange = 2:10:1, start:end:setp range irange = 'all', all

getElementConf(*elementKw*, *raw=False*)
return configuration for given element keyword, e.g. getElementConf('Q01') should return dict: {u'k1': 0.0, u'l': 0.05} :param elementKw: element keyword

getElementCtrlConf(*elementKw*)
return keyword's EPICS control configs, if not setup, return {}

getElementList(*bl*)
return the elements list according to the appearance order in beamline named 'bl'

Parameters `bl` – beamline name

getElementProperties(*name*)
return element properties :param name: element name

getElementType(*elementKw*)
return type name for given element keyword, e.g. getElementType('Q01') should return string: 'QUAD'

getFullBeamline(*beamlineKw*, *extend=False*)
get beamline definition from all_elements, expand iteratively with the elements from all_elements e.g. element 'doub1' in chi : line=(DBLL2 , doub1 , DP4FH , DP4SH , DBLL5 , DBD ,

B11 , DB11 , B12 , DB12 , PF2 , DB13 , B13 , DB14 , B14 , DBD , DBLL5 , doub2 , DP5FH , DP5SH , DBLL2 , PSTN1) should be expaned with 'doub1' configuration: doub1 : line=(DQD3, Q05, DQD2, Q06, DQD3)

since: getBeamline('doub1') = [u'dqd3', u'q05', u'dqd2', u'q06', u'dqd3'] = A getBeamline('doub2') = [u'dqd3', u'q05', u'dqd2', u'q06', u'dqd3'] = B getBeamline('chi') = [u'dbll2', u'doub1', u'dp4fh', u'dp4sh', u'dbll5', u'dbd', u'b11', u'db11', u'b12',

u'db12', u'pf2', u'db13', u'b13', u'db14', u'b14', u'dbd', u'dbll5', u'doub2', u'dp5fh', u'dp5sh', u'dbll2', u'pstn1']

thus: getFullBeamline('chi') should return: [u'dbll2', A, u'dp4fh', u'dp4sh', u'dbll5', u'dbd', u'b11', u'db11', u'b12', u'db12', u'pf2', u'db13',

u'b13', u'db14', u'b14', u'dbd', u'dbll5', B, u'dp5fh', u'dp5sh', u'dbll2', u'pstn1']

Parameters **`extend`** – if extend mode should be envoked, by default False

if extend = True, element like '2*D01' would be expended to be D01, D01

`isBeamline`(*kw*)

test if kw is a beamline :param kw: keyword

`makeElement`(*kw*)

return element object regarding the keyword configuration

`manipulateLattice`(*beamline*, *type='quad'*, *irange='all'*, *property='k1'*, *opstr='+0%'*)

manipulate element with type, e.g. quad

input parameters: :param beamline: beamline definition keyword :param type: element type, case insensitive :param irange: slice index, see getElementByOrder() :param property: element property, e.g. 'k1' for 'quad' strength :param opstr: operation, '+[-]n%' or '+[-*/]n'

`orderLattice`(*beamline*)

ordering element type appearance sequence for each element of beamline e.g. after getFullBeamline, lattice list ['q','Q01', 'B11', 'Q02', 'B22'] will return: [(u'q', u'CHARGE', 1),

(u'q01', u'QUAD', 1), (u'b11', u'CSRCSBEN', 1), (u'q02', u'QUAD', 2), (u'b12', u'CSRCSBEN', 2)]

`rinseElement`(*ele*)

resolve element case with multiply format, e.g. rinseElement('10*D01') should return dict {'num': 10; 'name' = 'D01'} :param ele: element string

`showBeamlines`()

show all defined beamlines

class `LteParser`(*infile*, *mode='f'*)

Bases: `object`

Parameters

- **`infile`** – lte filename or list of lines of lte file
- **`mode`** – 'f': treat infile as file, 's': (else) treat as list of lines

`detectAllKws`()

Detect all keyword from infile, return as a list

USAGE: kwslist = detectAllKws()

`dict2json`(*idict*)

convert dict into json

USAGE: rjson = dict2json(idict)

file2json(*jsonfile=None*)
Convert entire lte file into json like format

USAGE: 1: kwsdictstr = file2json() 2: kwsdictstr = file2json(jsonfile = 'somefile')

show pretty format with pipeline: | jshon, or | pjson if jsonfile is defined, dump to defined file before returning json string :param jsonfile: filename to dump json strings

getKw(*kw*)

Extract doc snippet for element configuration,

param kw element name

return instance itself 1 call getKwAsDict() to return config as a dict 2 call getKwAsJson() to return config as json string 3 call getKwAsString() to return config as a raw string

USAGE: getKw('Q10')

getKwAsDict(*kw*)
return keyword configuration as a dict

Usage: rdict = getKwAsDict(kw)

getKwAsJson(*kw*)
return keyword configuration as a json

Usage: rjson = getKwAsJson(kw) :param kw: element keyword

getKwAsString(*kw*)
return keyword configuration as a string

Usage: rstr = getKwAsString(kw)

getKwConfig(*kw*)
return the configuration of kw, dict

USAGE: rdict = getKwConfig(kw)

getKwCtrlConf(*kw*, *fmt='dict'*)
return keyword's control configuration, followed after '!epics' notation :param kw: keyword name :param fmt: return format, 'raw', 'dict', 'json', default is 'dict'

getKwType(*kw*)
return the type of kw, upper cased string

USAGE: rtype = getKwType(kw)

get_rpndict_flag(*rpndict*)
calculate flag set, the value is True or False, if rpndict value is not None, flag is True, or False

if a set with only one item, i.e. True returns, means values of rpndict are all valid float numbers, then finally return True, or False

makeElement(*kw*)
return element object regarding the keyword configuration

resolveEPICS()
extract epics control configs into

resolvePrefix()
extract prefix information into dict with the key of '_prefixstr'

resolve_rpn(*rpndict*)
solve dict of rpn expressions to pure var to val dict :param rpndict: dict of rpn expressions return pure var to val dict

rinse_rpnexp(*rpnexp*, *rpndict*)
replace valid keyword of rpnexp from rpndict e.g. rpnexp = 'b a /', rpndict = {'b': 10} then after rinsing, rpnexp = '10 a /'

return rinsed rpnexp

rpn2val(*rdict*)
Resolve the rpn string into calulated float number

USAGE: rpn2val(rdict)

param rdict json like dict

scanStoVars(*strline*)
scan input string line, replace sto parameters with calculated results.

solve_rpn()
solve rpn string in self.confdict, and update self.confdict

USAGE: ins = LteParser(infile) ins.getKw(kw).toDict().solve_rpn()

str2dict(*rawstr*)
convert str to dict format

USAGE: rdict = str2dict(rawstr) :param rawstr: raw configuration string of element

toDict()
convert self.confstr to dict, could apply chain rule, write to self.confdict

USAGE: ins = LteParser(infile) ins.getKw(kw).toDict()

update_rpndict(*rpndict*)
update rpndict, try to solve rpn expressions as many as possible, leave unsolvable unchanged.

return new dict

class Simulator(*infile=''*)
Bases: `object`

doSimulation()

getOutput(***kws*)

setExec(*execpath*)
set executable for simulation :param execpath: elegant or madx full path

setInputfiles(***infiles*)
input parameters: (elegant mode)

1: lte file 2: ele file

(mad mode) 1: mad file

setMode(*mode='elegant'*)
set simulation mode, define mode parameter of 'elegant' or 'mad' :param mode: simulation mode

setPath(*simpath*)
set simulation path where data should be put into :param simpath: where simulations take place, all data files should be found there

setScript(*fullname*)

set bash shell script full path name for simulation :param fullname: set 'runElegant.sh', which should be available after installed beamline package

class **DataExtracter**(*sddsfile*, **kws*)

Bases: `object`

Extract required data from a SDDS formated file, to put into hdf5 formated file or just dump into RAM for post-processing.

Parameters

- **sddsfile** – filename of SDDS data file
- **kws** – packed tuple/list options, usually sdds column names, e.g. `('s','Sx')`

Example

```
>>> # *sddsquery -col* shows it has 's', 'Sx' data columns
>>> sddsfile = 'output.sdds'
>>> param_list = ('s', 'Sx')
>>> dh = DataExtracter(sddsfile, *param_list)
>>> # *dh* is a newly created DataExtracter instance
```

dump()

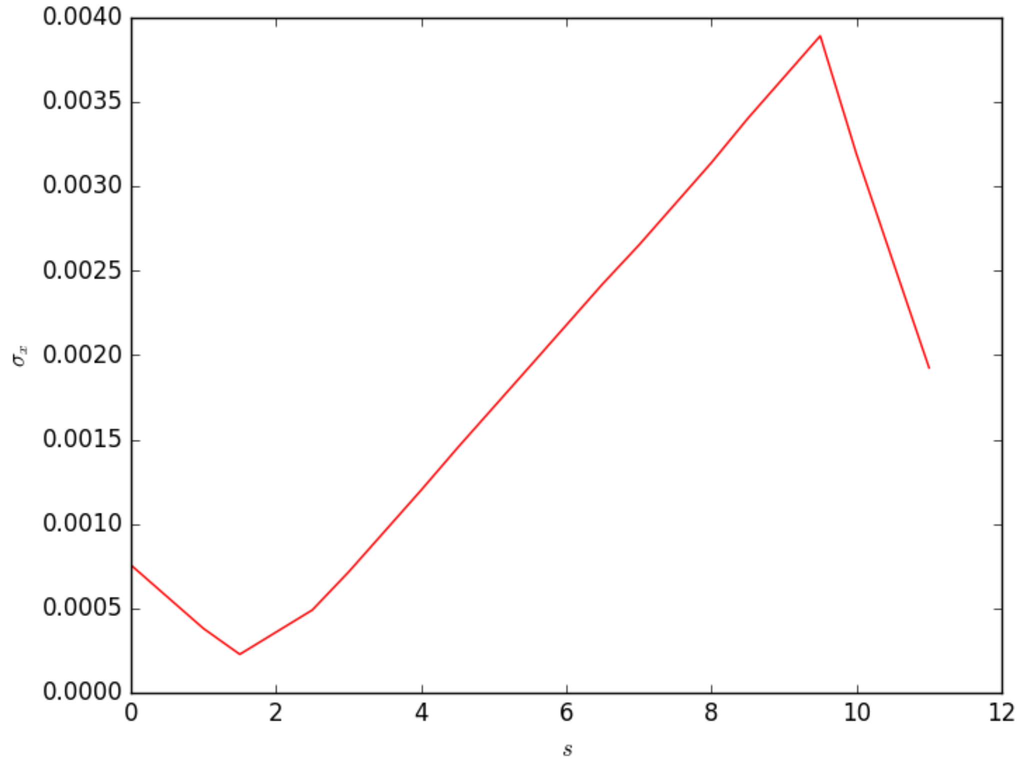
dump extracted data into a single hdf5file,

Returns None

Example

```
>>> # dump data into an hdf5 formated file
>>> datafields = ['s', 'Sx', 'Sy', 'enx', 'eny']
>>> datascript = 'sddsprintdata.sh'
>>> datapath   = './tests/tracking'
>>> hdf5file   = './tests/tracking/test.h5'
>>> A = DataExtracter('test.sig', *datafields)
>>> A.setDataScript(datascript)
>>> A.setDataPath  (datapath)
>>> A.setH5file    (hdf5file)
>>> A.extractData().dump()
>>>
>>> # read dumped file
>>> fd = h5py.File(hdf5file, 'r')
>>> d_s  = fd['s'][:]
>>> d_sx = fd['Sx'][:]
>>>
>>> # plot dumped data
>>> import matplotlib.pyplot as plt
>>> plt.figure(1)
>>> plt.plot(d_s, d_sx, 'r-')
>>> plt.xlabel('$s$')
>>> plt.ylabel('$\sigma_x$')
>>> plt.show()
```

Just like the following figure shows:



extractData()

return *self* with extracted data as *numpy array*

Extract the data of the columns and parameters of *self.kws* and put them in a **:np:func:'array'** with all columns as columns or parameters as columns. If columns and parameters are requested at the same then each column is one row and all parameters are in the last row. This **:np:func:'array'** is saved in `h5data`.

Note: If you mix types (e. g. float and str) then the minimal fitting type is taken for all columns.

Warning: Non float types need *sdds* as an extra dependency

Returns instance of itself

Example

One column of the watch element >>> dh = DataExtracter('test.w1') >>> dh.kwslist = ['Step'] >>> print(dh.extractData().h5data) array([[1]])

Two columns of the watch element >>> dh = DataExtracter('test.w1') >>> dh.kwslist = ['s', 'betax'] >>> print(dh.extractData().h5data) array([[0, 1], [1, 2], [2, 1]])

Two columns of the watch element and one parameter. The columns transform to rows and the parameter row is at the end. Furthermore all elements are strings, because the type of *PreviousElementName* is str and not float. >>> dh = DataExtracter('test.w1') >>> dh.kwslist = ['s', 'PreviousElementName', 'betax'] >>> print(dh.extractData().h5data) array([['0', '1', '2'], ['1', '2', '1'], ['DR01']])

getAllCols(*sddsfile=None*)
get all available column names from sddsfile

Parameters **sddsfile** – sdds file name, if not given, rollback to the one that from `__init__()`

Returns all sdds data column names

Return type list

Example

```
>>> dh = DataExtracter('test.out')
>>> print(dh.getAllCols())
['x', 'xp', 'y', 'yp', 't', 'p', 'particleID']
>>> print(dh.getAllCols('test.twi'))
['s', 'betax', 'alphax', 'psix', 'etax', 'etaxp', 'xAperture', 'betay', 'alphay', 'psiy', 'etay', 'etayp', 'yAperture',
↪'pCentral0', 'ElementName', 'ElementOccurence', 'ElementType']
```

getAllPars(*sddsfile=None*)
get all available parameter names from sddsfile

Parameters **sddsfile** – sdds file name, if not given, rollback to the one that from `__init__()`

Returns all sdds data parameter names

Return type list

Warning: *sdds* needs to be installed as an extra dependency.

Example

```
>>> dh = DataExtracter('test.w1')
>>> print(dh.getAllPars())
['Step', 'pCentral', 'Charge', 'Particles', 'IDSlotsPerBunch', 'SVNVersion', 'Pass', 'PassLength', 'PassCentralTime',
↪'ElapsedCoreTime', 'MemoryUsage', 's', 'Description', 'PreviousElementName']
```

Seealso `getAllCols()`

getH5Data()
return extracted data as numpy array

Returns numpy array after executing `extractData()`

getKws()
return data column fields that defined in constructor, e.g. `('s','Sx')`

Returns data columns keyword

Return type tuple

setDataPath(*path*)
set full dir path of data files

Parameters **path** – data path, usually is the directory where numerical simulation was taken place

Returns None

setDataScript(*fullscriptpath='sddsprintdata.sh'*)
configure script that should be utilized by DataExtracter, to extract data colums from sddsfile.

Parameters **fullscriptpath** – full path of script that handles the data extraction of sddsfile, default value is `sddsprintdata.sh`, which is a script that distributed with `beamline` package.

Returns None

setH5file(*h5filepath*)
set h5file full path name

Parameters **h5filepath** – path for hdf5 file

Returns None

setKws(**kws*)
set keyword list, i.e. sdds field names, update `kwslist` property

Parameters **kws** – packed tuple of sdds datafile column names

Return None

class **DataVisualizer**(*data*)
Bases: `object`

for data visualization purposes, to be implemented

illustrate(*xlabel*, *ylabel*)
plot x, y w.r.t. xlabel and ylabel :param ylabel: xlabel :param xlabel: ylabel

inspectDataFile()
inspect hdf5 data file

saveArtwork(*name='image'*, *fmt='jpg'*)
save figure by default name of image.jpg :param name: image name, 'image' by default :param fmt: image format, 'jpg' by default

class **DataStorage**(*data*)
Bases: `object`

for data storage management, to be implemented. communicate with database like mongodb, mysql, sqlite, etc.

configDatabase()
configure database

getData()
get data from database

putData()
put data into database

class **Models**(*name='BL'*, *mode='simu'*)
Bases: `object`

make lattice configuration (json) for lattice.Lattice return instance as a json string file with all configuration. get lattice name by instance.name.

LatticeDict
show lattice configuration

LatticeList
show lattice element list

addElement(**ele*)
add element to lattice element list

Parameters **ele** – magnetic element defined in element module

return total element number

static **anoteElements**(*ax*, *anotelist*, *showAccName=False*, *efilter=None*, *textypos=None*, ***kwargs*)
annotate elements to axes

Parameters

- **ax** – matplotlib axes object
- **anotelist** – element annotation object list
- **showAccName** – tag name for accelerator tubes? default is False, show acceleration band type, e.g. 'S', 'C', 'X', or for '[S,C,X]D' for cavity
- **efilter** – element type filter, default is None, annotate all elements could be defined to be one type name or type name list/tuple, e.g. filter='QUAD' or filter=('QUAD', 'CSRCSBEN')
- **textypos** – y coordinator of annotated text string
- **kwargs** – alpha=0.8, arrowprops=dict(arrowstyle='->'), rotation=-60, fontsize='small'

return list of annotation objects

draw(*startpoint=(0, 0)*, *mode='plain'*, *showfig=False*)
lattice visualization

Parameters

- **startpoint** – start drawing point coords, default: (0, 0)
- **showfig** – show figure or not, default: False
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

Returns patchlist, anotelist, (xmin0, xmax0), (ymin0, ymax0) patchlist: list of element patches anotelist: list of annotations (xmin0, xmax0) and (ymin0, ymax0) are ploting range

static **flatten**(*ele*)
flatten recursively defined list, e.g. [1,2,3, [4,5], [6,[8,9,[10,[11,'x']]]]]

Parameters **ele** – recursive list, i.e. list in list in list ...

Returns generator object

getAllConfig(*fmt='json'*)
return all element configurations as json string file. could be further processed by beamline.Lattice class

Parameters **fmt** – 'json' (default) or 'dict'

getCtrlConf(*msgout=True*)
get control configurations regarding to the PV names, read PV value

Parameters **msgout** – print information if True (by default)

return updated element object list

getElementsByName(*name*)
get element with given name, return list of element objects regarding to 'name'

Parameters **`name`** – element name, case sensitive, if elements are auto-generated from LteParser, the name should be lower cased.

`initPos`(*startpos=0.0*)
initialize the elements position [m] in lattice, the starting point is 0 [m] for the first element by default.

Parameters **`startpos`** – starting point, 0 [m] by default

static **`makeLatticeDict`**(*ele*)
return lattice dict conf like {"lattice": "(q b d)"}

Parameters **`ele`** – element list

static **`makeLatticeString`**(*ele*)
return string like "lattice = (q b d)"

Parameters **`ele`** – element list

`mode`

`name`

static **`plotElements`**(*ax*, *patchlist*)
plot elements' drawings to axes

Parameters

- **`ax`** – matplotlib axes object
- **`patchlist`** – element patch object list

`printAllElements`()
print out all modeled elements

`putCtrlConf`(*eleobj*, *ctrlkey*, *val*, *type='raw'*)
put the value to control PV field

Parameters

- **`eleobj`** – element object in lattice
- **`ctrlkey`** – element control property, PV name
- **`val`** – new value for ctrlkey
- **`type`** – set in 'raw' or 'real' mode, 'raw' by default 'raw': set PV with the value of 'val', 'real': set PV with the value translated from 'val'

`updateConfig`(*eleobj*, *config*, *type='simu'*)
write new configuration to element

Parameters

- **`eleobj`** – define element object
- **`config`** – new configuration for element, string or dict
- **`type`** – 'simu' by default, could be online, misc, comm, ctrl

class **`ParseParams`**(**infilename*)
Bases: `object`

`getChicaneDriftLength`()

`getChicaneMagnetField`()

`getChicaneMagnetLength`()

getElectronEmitx()

getElectronGamma()

getFELwavelength()

getUndulatorParameter()

getUndulatorPeriod()

getUndulatorUnitlength()

onParseFile()

class **BeamMatch**(*infile_mod*, *infile_rad*, *latfile_mod*, *latfile_rad*, *infile_mod_new*, *infile_rad_new*, *latfile_rad_new*, *qfval*, *qdval*)
Bases: `object`

matchCalculate(*latlengthname='fullat.hghg'*)

matchPerform(*qf_linenum=11*, *qd_linenum=13*)

matchPrintout()

class **FELSimulator**(*mode='HGHG'*, *modinfile='mod.in'*, *radinfile='rad.in'*, *modlatfile='mod.lat'*, *radlatfile='rad.lat'*)
Bases: `object`

getMaxPower()

grepParam(*param='entries'*, *outfile='rad.out'*)

plotPower()

postProcess(*outfile='rad.out'*)

run()

parseLattice(*latlengthname='fullat.hghg'*)

funTransQuadF(*k*, *s*)
Focusing quad in X, defocusing in Y

Parameters

- **k** – k1, in [T/m]
- **s** – width, in [m]

Returns 2x2 numpy array

funTransQuadD(*k*, *s*)
Defocusing quad in X, focusing in Y

Parameters

- **k** – k1, in [T/m]
- **s** – width, in [m]

Returns 2x2 numpy array

funTransDrift(*s*)
Drift space

Parameters **s** – drift length, in [m]

Returns 2x2 numpy array

`funTransUnduH`(*s*)
Planar undulator transport matrix in horizontal direction

Parameters **s** – horizontal width, in [m]

Returns 2x2 numpy array

`funTransUnduV`(*k*, *s*)
Planar undulator transport matrix in vertical direction

Parameters

- **k** – equivalent k1, in [T/m], i.e. natural focusing
- **s** – horizontal width, in [m]

Returns 2x2 numpy array

`funTransEdgeX`(*theta*, *rho*)
Fringe matrix in X

Parameters

- **theta** – fringe angle, in [rad]
- **rho** – bend radius, in [m]

Returns 2x2 numpy array

`funTransEdgeY`(*theta*, *rho*)
Fringe matrix in Y

Parameters

- **theta** – fringe angle, in [rad]
- **rho** – bend radius, in [m]

Returns 2x2 numpy array

`funTransSectX`(*theta*, *rho*)
Sector matrix in X

Parameters

- **theta** – bend angle, in [rad]
- **rho** – bend radius, in [m]

Returns 2x2 numpy array

`funTransSectY`(*theta*, *rho*)
Sector matrix in Y

Parameters

- **theta** – bend angle, in [rad]
- **rho** – bend radius, in [m]

Returns 2x2 numpy array

`funTransChica`(*imagl*, *idril*, *ibfield*, *gamma0*, *xoy='x'*)
Chicane matrix, composed of four rbends, seperated by drifts

Parameters

- **imagl** – rbend width, in [m]

- **idril** – drift length between two adjacent rbends, in [m]
- **ibfield** – rbend magnetic strength, in [T]
- **gamma0** – electron energy, gamma
- **xoy** – 'x' or 'y', matrix in X or Y direction, 'x' by default

Returns 2x2 numpy array

transDrift(*length=0.0*, *gamma=None*)
Transport matrix of drift

Parameters

- **length** – drift length in [m]
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

transQuad(*length=0.0*, *k1=0.0*, *gamma=None*)
Transport matrix of quadrupole

Parameters

- **length** – quad width in [m]
- **k1** – quad k1 strength in [T/m]
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

transSect(*theta=None*, *rho=None*, *gamma=None*)
Transport matrix of sector dipole

Parameters

- **theta** – bending angle in [RAD]
- **rho** – bending radius in [m]
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

transRbend(*theta=None*, *rho=None*, *gamma=None*, *incsym=-1*)
Transport matrix of rectangle dipole

Parameters

- **theta** – bending angle in [RAD]
- **incsym** – incident symmetry, -1 by default, available options:
  - -1: left half symmetry,
  - 0: full symmetry,
  - 1: right half symmetry
- **rho** – bending radius in [m]
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

transFringe(*beta=None*, *rho=None*)
Transport matrix of fringe field

Parameters

- **beta** – angle of rotation of pole-face in [RAD]
- **rho** – bending radius in [m]

Returns 6x6 numpy array

transChicane(*bend_length=None*, *bend_field=None*, *drift_length=None*, *gamma=None*)

Transport matrix of chicane composed of four rbends and three drifts between them

Parameters

- **bend_length** – rbend width in [m]
- **bend_field** – rbend magnetic field in [T]
- **drift_length** – drift length, list or tuple of three elements, in [m] single float number stands for same length for three drifts
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

class Chicane(*bend_length=None*, *bend_field=None*, *drift_length=None*, *gamma=None*)

Bases: `object`

Chicane class transport configuration of a chicane, comprising of four dipole with three drift sections

> **Warning:** it's better to issue `getMatrix()` before `getR()`, `getAngle()`

Parameters

- **bend_length** – bend length, [m]
- **bend_field** – bend field, [T]
- **drift_length** – drift length [m], list: [1,2,1], [1], [1,2], 1
- **gamma** – electron energy, gamma value

getAngle(*mode='deg'*)

return bend angle

Parameters **mode** – 'deg' or 'rad'

Returns deflecting angle in RAD

getBendField()

Returns bend magnetic field

getBendLength()

Returns bend length

getDriftLength()

Returns drift lengths list

getGamma()

Returns gamma value

getMatrix()

get transport matrix with `mflag` flag, if `mflag` is True, return calculated matrix, else return unity matrix

Returns transport matrix

getR(*i=5*, *j=6*)

return transport matrix element, indexed by i, j, be default, return dispersion value, i.e. getR(5,6) in [m]

Parameters

- **i** – row index, with initial index of 1
- **j** – col indx, with initial index of 1

Returns transport matrix element

setBendField(*x*)

set bend magnetic field

Parameters **x** – new bend field to be assigned, [T]

Returns None

setBendLength(*x*)

set bend length

Parameters **x** – new bend length to be assigned, [m]

Returns None

setDriftLength(*x*)

set lengths for drift sections

Parameters **x** – single double or list

Returns None

Example

```
>>> import beamline
>>> chi = beamline.mathutils.Chicane(bend_length=1,bend_field=0.5,drift_length=1,gamma=1000)
>>> chi.getMatrix()
>>> r56 = chi.getR(5,6)  # r56 = -0.432
>>> chi.setDriftLength([2,4,2])
>>> # same effect (to R56) as ``chi.setDriftLength([2,4])`` or ``chi.setDriftLength([2])``
>>> # or ``chi.setDriftLength(2)``
>>> r56 = chi.getR(5,6)  # r56 = -0.620
```

setGamma(*x*)

set electron energy, gamma value

Parameters **x** – new energy, gamma value

Returns None

setParams(*bend_length*, *bend_field*, *drift_length*, *gamma*)

set chicane parameters

Parameters

- **bend_length** – bend length, [m]
- **bend_field** – bend field, [T]
- **drift_length** – drift length, [m], list
- **gamma** – electron energy, gamma

Returns None

class **ElementCharge**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

charge element

Parameters

- **name** – charge element name that could be used in other method, e.g. 'C', 'Q', etc.
- **config** –

Example

```
>>> chconf = {'total': 1.0e-9}  # total charge of 1.0 nC
>>> q = ElementCharge(name='q', config=chconf)
```

class **ElementCsrcsben**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

csrcsben element

calcTransM(*gamma=None*, *type='simu'*, *incsym=-1*)
calculate transport matrix

Parameters

- **gamma** – electron energy measured by mc^2
- **type** – configuration type, 'simu' (simulation mode) or 'online' (online mode)
- **incsym** – incident symmetry, -1, 0, 1

Returns transport matrix

Return type numpy array

field

Returns magnetic field, [T]

rho

Returns bending radius, [m]

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – rotation angle [deg] of drawing central point, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementQuad**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

quad element

calcTransM(*gamma=None*, *type='simu'*)
calculate transport matrix

Parameters **gamma** – electron energy measured by mc^2

Returns transport matrix

Return type numpy array

getK1(*type='simu'*)
get quad k1 value

Parameters **type** – 'simu' or 'online'

Returns quad strength,i.e. k1

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – rotation angle [deg] of drawing central point, angle is rotating from x-axis to be '+' or '-', '+': anticlockwise, '-': clockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

unitTrans(*inval*, *direction='+'*, *transfun=None*)

class **ElementCsrdrift**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

csrdrift element

calcTransM(*gamma=None*)
calculate transport matrix

Parameters **gamma** – electron energy measured by mc^2

Returns transport matrix

Return type numpy array

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

ElementCsrdrif
alias of `ElementCsrdrift`

class **ElementDrift**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

drift element

calcTransM(*gamma=None*)
calculate transport matrix

Parameters **gamma** – electron energy measured by mc^2

Returns transport matrix

Return type numpy array

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

ElementDrif
alias of ElementDrift

class ElementLscdrift(*name=None*, *config=None*)
Bases: beamline.element.MagBlock

lscdrift element

calcTransM(*gamma=None*)
calculate transport matrix

Parameters **gamma** – electron energy measured by mc^2

Returns transport matrix

Return type numpy array

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

ElementLscdrif
alias of ElementLscdrift

class ElementKicker(*name=None*, *config=None*)
Bases: beamline.element.MagBlock

kicker element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementMark**(*name=None*, *config=None*)

Bases: `beamline.element.MagBlock`

mark element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)

set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementWatch**(*name=None*, *config=None*)

Bases: `beamline.element.MagBlock`

watch element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)

set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementMoni**(*name=None*, *config=None*)

Bases: `beamline.element.MagBlock`

moni element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)

set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)

- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementRfcw**(*name=None, config=None*)
Bases: `beamline.element.MagBlock`

rfcw element

setDraw(*p0=(0, 0), angle=0, mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementRfdf**(*name=None, config=None*)
Bases: `beamline.element.MagBlock`

rfdf element

setDraw(*p0=(0, 0), angle=0, mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementWake**(*name=None, config=None*)
Bases: `beamline.element.MagBlock`

wake element

setDraw(*p0=(0, 0), angle=0, mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementBeamline**(*name='bl'*, *config=None*)

Bases: `beamline.element.MagBlock`

beamline element, virtual element, does not present in ELEGANT

Subpackages

beamline.ui package

Submodules

beamline.ui.appui module

class **BeamlineFrame**(*parent*)

Bases: `wx._windows.Frame`

bl_choicebookOnChoicebookPageChanged(*event*)

cancel_btnOnButtonClick(*event*)

ok_btnOnButtonClick(*event*)

class **DataFrame**(*parent*)

Bases: `wx._windows.Frame`

copy_btnOnButtonClick(*event*)

exit_btnOnButtonClick(*event*)

class **DrawFrame**(*parent*)

Bases: `wx._windows.Frame`

bend_ckbOnCheckBox(*event*)

mode_rbOnRadioBox(*event*)

monitor_ckbOnCheckBox(*event*)

quad_ckbOnCheckBox(*event*)

rf_ckbOnCheckBox(*event*)

undulator_ckbOnCheckBox(*event*)

class **LogFrame**(*parent*)

Bases: `wx._windows.Frame`

copy_btnOnButtonClick(*event*)

exit_btnOnButtonClick(*event*)

class **MainFrame**(*parent*)

Bases: `wx._windows.Frame`

about_mitemOnMenuSelection(*event*)

bl_mitemOnMenuSelection(*event*)

clear_btnOnButtonClick(*event*)

collapse_mitemOnMenuSelection(*event*)

dict_mitemOnMenuSelection(*event*)

draw_mitemOnMenuSelection(*event*)

exit_btnOnButtonClick(*event*)

expand_mitemOnMenuSelection(*event*)

generate_btnOnButtonClick(*event*)

guide_mitemOnMenuSelection(*event*)

lte_mitemOnMenuSelection(*event*)

mainview_treeOnLeftDown(*event*)

model_btnOnButtonClick(*event*)

next_bmpbtnOnButtonClick(*event*)

nodeview_lcOnListColRightClick(*event*)

nodeview_lcOnListItemSelected(*event*)

nodeview_lcOnRightUp(*event*)

open_mitemOnMenuSelection(*event*)

previous_bmpbtnOnButtonClick(*event*)

pt_mitemOnMenuSelection(*event*)

quit_mitemOnMenuSelection(*event*)

raw_mitemOnMenuSelection(*event*)

reopen_mitemOnMenuSelection(*event*)

save_mitemOnMenuSelection(*event*)

saveas_mitemOnMenuSelection(*event*)

search_ctrlOnCancelButton(*event*)

search_ctrlOnText(*event*)

search_ctrlOnTextEnter(*event*)

show_btnOnButtonClick(*event*)

showlog_btnOnButtonClick(*event*)

tree_splitterOnIdle(*event*)

treename_tcOnTextEnter(*event*)

beamline.ui.main module

GUI app of latticeviewer, which is meant for accelerator online modeling,

additional machine-related physics application should be implemented and inserted into latticeviewer->Tools menu as a tool plugin.

Tong Zhang 2016-05-27 15:54:59 PM CST

run(*debug=True*, *icon=None*)

beamline.ui.myappframe module

Subclass of MainFrame, which is generated by wxFormBuilder.

class MyAppFrame(*parent*, *title*)

Bases: `beamline.ui.appui.MainFrame`

about_mitemOnMenuSelection(*event*)

add_items(*data_dict*, *root=None*, *target=None*)
add items for tree :param data_dict: dict of tree data :param root: treeitemid of tree root :param target: treectrl obj

bl_mitemOnMenuSelection(*event*)

clear_btnOnButtonClick(*event*)

clear_tree()
return has_tree stat

collapse_mitemOnMenuSelection(*event*)

create_online_model()

dict_mitemOnMenuSelection(*event*)

draw_mitemOnMenuSelection(*event*)

exit_app()

exit_btnOnButtonClick(*event*)

expand_mitemOnMenuSelection(*event*)

expand_tree(*expand_all_flag*)

generate_btnOnButtonClick(*event*)

get_beamlines()
return dict with k:v, k: 'beamline name' v: beamline elements list regarding to 'beamline name'

get_children(*root=None*, *target=None*)
return list of all the children of root of given tree :param root: treectrl item, treeitemid :param target: treectrl object

get_file_ext(*filename*)
return file extension, e.g. file.json, return json

get_filetype(*filename*)
return file type according to extension

get_refresh_flag(*filename*)
set refresh data flag return True or False

json2lte(*filename*)
convert json to lte

return tuple of json, lte file content

lte2json(*filename*)
convert lte to json

return tuple of json, lte file content

lte_mitemOnMenuSelection(*event*)

mainview_treeOnLeftDown(*event*)

model_btnOnButtonClick(*event*)

next_bmpbtnOnButtonClick(*event*)

nodeview_lcOnListColRightClick(*event*)

nodeview_lcOnListItemSelected(*event*)

nodeview_lcOnRightUp(*event*)

onPopOne(*event*)

onPopTwo(*event*)

open_file()

open_mitemOnMenuSelection(*event*)

pack_found_items(*s_text*, *target*)
pack up found items for search ctrl :param target: treectrl obj :param s_text: text to search, lower case return list of found items

previous_bmpbtnOnButtonClick(*event*)

pt_mitemOnMenuSelection(*event*)

quit_mitemOnMenuSelection(*event*)

raw_mitemOnMenuSelection(*event*)

read_json(*filename*)
return dict the first line of json file, which defined by filename

read_lte(*filename*)
parse lte file first, then return dict as read_json() does

reopen_mitemOnMenuSelection(*event*)

saveas_file()

saveas_mitemOnMenuSelection(*event*)

search_ctrlOnCancelButton(*event*)

search_ctrlOnText(*event*)

search_ctrlOnTextEnter(*event*)

set_title()

show_btnOnButtonClick(*event*)

show_data(*item*)
show data key-value in ListCtrl for tree item

show_tree(*has_tree=False*, *force_update=False*)
show tree list

Parameters

- **has_tree** – tree exist or not, False by default, if True, tree should be cleared first
- **force_update** – force update flag, if True, update neglect other flags.

Returns has_tree, True successful, not change when exception

showlog_btnOnButtonClick(*event*)

treename_tcOnTextEnter(*event*)

update_stat(*mode='open'*, *infostr=''*, *stat=''*)
write operation stats to log :param mode: 'open', 'saveas', 'listtree' :param infostr: string to put into info_st :param stat: 'OK' or 'ERR'

getFileToLoad(*parent*, *ext='*'*, *flag='single'*)

getFileToSave(*parent*, *ext='*'*)

beamline.ui.mychoiceframe module

Subclass of BeamlineFrame, which is generated by wxFormBuilder.

class **MyChoiceFrame**(*parent*, *beamline_info_dict*)
Bases: `beamline.ui.appui.BeamlineFrame`

bl_choicebookOnChoicebookPageChanged(*event*)

cancel_btnOnButtonClick(*event*)

init_ui()

ok_btnOnButtonClick(*event*)

beamline.ui.mydataframe module

Subclass of DataFrame, which is generated by wxFormBuilder.

class **MyDataFrame**(*parent*, *data*)
Bases: `beamline.ui.appui.DataFrame`

copy_btnOnButtonClick(*event*)

exit_btnOnButtonClick(*event*)

beamline.ui.mydrawframe module

Subclass of DrawFrame, which is generated by wxFormBuilder.

class **MyDrawFrame**(*parent*, *lattice_model*, *aspect=1*)
Bases: `beamline.ui.appui.DrawFrame`

bend_ckbOnCheckBox(*event*)

mode_rbOnRadioBox(*event*)

quad_ckbOnCheckBox(*event*)

rf_ckbOnCheckBox(*event*)

beamline.ui.mylogframe module

Subclass of LogFrame, which is generated by wxFormBuilder.

class **MyLogFrame**(*parent*, *log*)
Bases: `beamline.ui.appui.LogFrame`

copy_btnOnButtonClick(*event*)

exit_btnOnButtonClick(*event*)

show_log(*log*)

beamline.ui.pltutils module

custom GUI controls

class **LatticePlotPanel**(*parent*, ***kwargs*)
Bases: `beamline.ui.pltutils.MyPlotPanel`

identify_obj(*x*)

on_motion(*event*)

class **MyPlotPanel**(*parent*, *figsize=None*, *dpi=None*, *bgcolor=None*, *type=None*, *toolbar=None*, *aspect=1*, ***kwargs*)
Bases: `wx._windows.Panel`

fit_canvas()
tight fit canvas layout

on_motion(*event*)

on_press(*event*)

on_release(*event*)

on_size(*event*)

set_color(*rgb_tuple*)
set figure and canvas with the same color.

Parameters **rgb_tuple** – rgb color tuple, e.g. (255, 255, 255) for white color

set_layout()
set panel layout

class **MyToolbar**(*canvas*)
Bases: `matplotlib.backends.backend_wxagg.NavigationToolbar2WxAgg`

class **TestFrame**(*parent*, ***kwargs*)
Bases: `wx._windows.Frame`

test()

Submodules

beamline.blparser module

Python module for parsing MAD-8 lattice file, only used by matchwizard app (deprecated).

madParser(*mad_filename*, *idbl='BL'*)
function to parse beamline with MAD-8 input format

Parameters

- **mad_filename** – lattice filename with mad-8 like format
- **idbl** – beamline to be used that defined in lattice file, default value is `BL`

Returns list of dict that contains magnetic elements

Return type list

Example

```
>>> import beamline
>>> beamlinelist = beamline.blparser.madParser('LPA.list', 'BL2')
>>> print beamlinelist
>>> [{'type': 'drift', 'l': '0.1', 'ID': 'd0'}, {'type': 'quad', 'k1': '75', 'angle': '75', 'l': '0.1', 'ID': 'q1'}, {'type':
↪'drift', 'l': '0.18', 'ID': 'd3'}, {'type': 'quad', 'k1': '-75', 'angle': '75', 'l': '0.1', 'ID': 'q2'}, {'type': 'drift',
↪'l': '0.27', 'ID': 'd6'}, {'type': 'rbend', 'angle': '10', 'l': '0.1', 'ID': 'b1'}, {'type': 'drift', 'l': '1.0', 'ID': 'd8
↪'}, {'type': 'rbend', 'angle': '-5', 'l': '0.1', 'ID': 'b2'}, {'type': 'drift', 'l': '0.45', 'ID': 'd4'}, {'type': 'quad',
↪'k1': '75', 'angle': '75', 'l': '0.1', 'ID': 'q1'}]
```

Download `LPA.list` for reference.

main()

beamline.elements module

definition for magnetic elements (deprecated)

class Drift(*length=2.0, angle=0.0, link_node=(0.0, 0.0), line_color='black', line_width=1.5, _alpha=0.8*)

Bases: `object`

Element drift section

Parameters

- **_length** – drift length, [m]
- **_angle** – angle between drawing line and horizontal plane, [deg]
- **_linkNode** – `(x,y)` coordinates that drawing begins or linked to another element

show(*fignum=1*)

class Quadrupole(*width=1.0, angle=75, xysign='x', link_node=(0.0, 1.0), face_color='blue', edge_color='blue', line_width=0.1, _alpha=0.8*)

Bases: `object`

Element quadrupole

Parameters

- **width** – quad width, [m]
- **angle** – angle, [deg]
- **xy_sign** – `x`: x-focusing, `K1>0`; `y`: y-focusing, `K1<0`
- **link_node** – `(x,y)` coordinates that drawing begins or linked to another element

show(*fignum=1*)

class Rbend(*width=1.0, height=2.0, angle=0.0, link_node=(0.0, 1.0), face_color='red', edge_color='red', line_width=0.1, _alpha=0.8*)

Bases: `object`

Element rectangle bend

Parameters

- **_width** – bend width, [m]
- **_height** – bend hight, [m]
- **_angle** – bend angle, [deg]
- **_linkNode** – `(x,y)` coordinates that drawing begins or linked to another element

info()

show(*fignum=1*)

class **Undulator**(*period_length=2.0*, *period_number=10*, *north_color='red'*, *south_color='blue'*, *link_node=(0, 0)*, *ratio=[2.5, 1.5]*, *spacing=1.5*, *_alpha=0.8*)

Bases: `object`

Element undulator (not included in `element` module)

Parameters

- **period_length** – undulator period length, [m]
- **period_number** – undulator period number
- **north_color** – color of north pole
- **south_color** – color of south pole
- **link_node** – `(x,y)` coordinates that drawing begins or linked to another element
- **ratio** – ratio of pole_height v.s. pole_width, and gap v.s pole_width
- **spacing** – spacing between pole, measured by pole_width,
- **gap** – undulator gap only for visualization, not true magnetic gap

show(*fignum=1*)

test()

beamline.pltutils module

functions for lattice visualization.

main()

makeBeamline(*beamlinelist*, *startpoint=(0, 0)*)

function to construct patches for `plotLattice()`, from different elements like `rbend`, `quadrupole`,etc. parsing from lattice file, mad-8. drift sections are calculated from other elements.

Input parameters:

Parameters

- **beamlinelist** – list, which elements are dict, each dict is the description for magnetic element, should be returned from module `blparser`, function `madParser()`
- **startpoint** – pos to start drawing, `(0,0)` by default

Returns tuple of beamline patches list, xlim and ylim * beamline patches list: patches to be drawn * xlim: data limit along x-axis * ylim: data limit along y-axis

plotLattice(*beamlinepatchlist*, *fignum=1*, *fig_size=20*, *fig_ratio=0.5*, *xranges=(-10, 10)*, *yranges=(-10, 10)*, *zoomfac=1.5*)

function plot beamline defined by `beamlinepatchlist`, which is a set of patches for all elements

Parameters

- **beamlinepatchlist** – generated by function `makeBeamline()`
- **fignum** – figure number, 1 by default
- **fig_size** – figure size, 20 inch by default
- **fig_ratio** – figure ratio, 0.5 by default

- **xranges** – axes x-ranges, (-10,10) by default
- **yranges** – axes y-ranges, (-10,10) by default
- **zoomfac** – zoom in factor, 1.5 by default

beamline.datautils module

This module is created for data processing framework, to make rules for data saving, visualization issues, etc.

class DataExtracter(*sddsfile*, **kws*)

Bases: object

Extract required data from a SDDS formated file, to put into hdf5 formated file or just dump into RAM for post-processing.

Parameters

- **sddsfile** – filename of SDDS data file
- **kws** – packed tuple/list options, usually sdds column names, e.g. ('s','Sx')

Example

```
>>> # *sddsquery -col* shows it has 's', 'Sx' data columns
>>> sddsfile = 'output.sdds'
>>> param_list = ('s', 'Sx')
>>> dh = DataExtracter(sddsfile, *param_list)
>>> # *dh* is a newly created DataExtracter instance
```

dump()

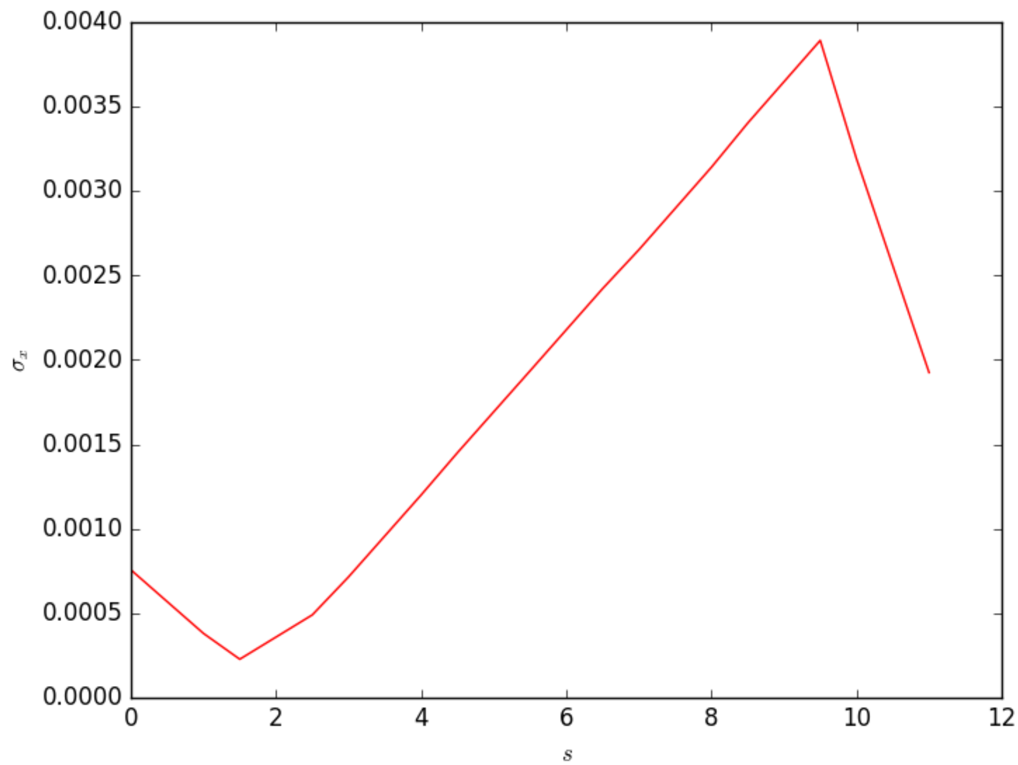
dump extracted data into a single hdf5file,

Returns None

Example

```
>>> # dump data into an hdf5 formated file
>>> datafields = ['s', 'Sx', 'Sy', 'enx', 'eny']
>>> datascript = 'sddsprintdata.sh'
>>> datapath   = './tests/tracking'
>>> hdf5file   = './tests/tracking/test.h5'
>>> A = DataExtracter('test.sig', *datafields)
>>> A.setDataScript(datascript)
>>> A.setDataPath  (datapath)
>>> A.setH5file    (hdf5file)
>>> A.extractData().dump()
>>>
>>> # read dumped file
>>> fd = h5py.File(hdf5file, 'r')
>>> d_s  = fd['s'][:]
>>> d_sx = fd['Sx'][:]
>>>
>>> # plot dumped data
>>> import matplotlib.pyplot as plt
>>> plt.figure(1)
>>> plt.plot(d_s, d_sx, 'r-')
>>> plt.xlabel('$s$')
>>> plt.ylabel('$\sigma_x$')
>>> plt.show()
```

Just like the following figure shows:



`extractData()`

return *self* with extracted data as *numpy array*

Extract the data of the columns and parameters of *self.kws* and put them in a **:np:func:'array'** with all columns as columns or parameters as columns. If columns and parameters are requested at the same then each column is one row and all parameters are in the last row. This **:np:func:'array'** is saved in `h5data`.

Note: If you mix types (e. g. float and str) then the minimal fitting type is taken for all columns.

Warning: Non float types need *sdds* as an extra dependency

Returns instance of itself

Example

One column of the watch element >>> dh = DataExtracter('test.w1') >>> dh.kwslist = ['Step'] >>> print(dh.extractData().h5data) array([[1]])

Two columns of the watch element >>> dh = DataExtracter('test.w1') >>> dh.kwslist = ['s', 'betax'] >>> print(dh.extractData().h5data) array([[0, 1], [1, 2], [2, 1]])

Two columns of the watch element and one parameter. The columns transform to rows and the parameter row is at the end. Furthermore all elements are strings, because the type of *PreviousElementName* is str and not float. >>> dh = DataExtracter('test.w1') >>> dh.kwslist = ['s', 'PreviousElementName', 'betax'] >>> print(dh.extractData().h5data) array([['0', '1', '2'], ['1', '2', '1'], ['DR01']])

getAllCols(*sddsfile=None*)
get all available column names from sddsfile

Parameters **sddsfile** – sdds file name, if not given, rollback to the one that from `__init__()`

Returns all sdds data column names

Return type list

Example

```
>>> dh = DataExtracter('test.out')
>>> print(dh.getAllCols())
['x', 'xp', 'y', 'yp', 't', 'p', 'particleID']
>>> print(dh.getAllCols('test.twi'))
['s', 'betax', 'alphax', 'psix', 'etax', 'etaxp', 'xAperture', 'betay', 'alphay', 'psiy', 'etay', 'etayp', 'yAperture',
↪'pCentral0', 'ElementName', 'ElementOccurence', 'ElementType']
```

getAllPars(*sddsfile=None*)
get all available parameter names from sddsfile

Parameters **sddsfile** – sdds file name, if not given, rollback to the one that from `__init__()`

Returns all sdds data parameter names

Return type list

Warning: *sdds* needs to be installed as an extra dependency.

Example

```
>>> dh = DataExtracter('test.w1')
>>> print(dh.getAllPars())
['Step', 'pCentral', 'Charge', 'Particles', 'IDSlotsPerBunch', 'SVNVersion', 'Pass', 'PassLength', 'PassCentralTime',
↪'ElapsedCoreTime', 'MemoryUsage', 's', 'Description', 'PreviousElementName']
```

Seealso `getAllCols()`

getH5Data()
return extracted data as numpy array

Returns numpy array after executing `extractData()`

getKws()
return data column fields that defined in constructor, e.g. `('s','Sx')`

Returns data columns keyword

Return type tuple

setDataPath(*path*)
set full dir path of data files

Parameters **path** – data path, usually is the directory where numerical simulation was taken place

Returns None

setDataScript(*fullscriptpath='sddsprintdata.sh'*)
configure script that should be utilized by DataExtracter, to extract data colums from sddsfile.

Parameters **fullscriptpath** – full path of script that handles the data extraction of sddsfile, default value is `sddsprintdata.sh`, which is a script that distributed with `beamline` package.

Returns None

setH5file(*h5filepath*)
set h5file full path name

Parameters **h5filepath** – path for hdf5 file

Returns None

setKws(**kws*)
set keyword list, i.e. sdds field names, update `kwslist` property

Parameters **kws** – packed tuple of sdds datafile column names

Return None

class DataStorage(*data*)
Bases: `object`

for data storage management, to be implemented. communicate with database like mongodb, mysql, sqlite, etc.

configDatabase()
configure database

getData()
get data from database

putData()
put data into database

class DataVisualizer(*data*)
Bases: `object`

for data visualization purposes, to be implemented

illustrate(*xlabel*, *ylabel*)
plot x, y w.r.t. xlabel and ylabel :param ylabel: xlabel :param xlabel: ylabel

inspectDataFile()
inspect hdf5 data file

saveArtwork(*name='image'*, *fmt='jpg'*)
save figure by default name of image.jpg :param name: image name, 'image' by default :param fmt: image format, 'jpg' by default

test()

beamline.mathutils module

functions for mathematical calculations:

- transfer matrix for quad, drift, undulator, chicane, etc.

class Chicane(*bend_length=None*, *bend_field=None*, *drift_length=None*, *gamma=None*)
Bases: `object`

Chicane class transport configuration of a chicane, comprising of four dipole with three drift sections

> **Warning:** it's better to issue `getMatrix()` before `getR()`, `getAngle()`

Parameters

- **bend_length** – bend length, [m]
- **bend_field** – bend field, [T]
- **drift_length** – drift length [m], list: [1,2,1], [1], [1,2], 1
- **gamma** – electron energy, gamma value

getAngle(*mode='deg'*)
return bend angle

Parameters **mode** – 'deg' or 'rad'

Returns deflecting angle in RAD

getBendField()

Returns bend magnetic field

getBendLength()

Returns bend length

getDriftLength()

Returns drift lengths list

getGamma()

Returns gamma value

getMatrix()
get transport matrix with `mflag` flag, if `mflag` is True, return calculated matrix, else return unity matrix

Returns transport matrix

getR(*i=5, j=6*)
return transport matrix element, indexed by i, j, be default, return dispersion value, i.e. getR(5,6) in [m]

Parameters

- **i** – row index, with initial index of 1
- **j** – col indx, with initial index of 1

Returns transport matrix element

setBendField(*x*)
set bend magnetic field

Parameters **x** – new bend field to be assigned, [T]

Returns None

setBendLength(*x*)
set bend length

Parameters **x** – new bend length to be assigned, [m]

Returns None

setDriftLength(*x*)
set lengths for drift sections

Parameters **x** – single double or list

Returns None

Example

```
>>> import beamline
>>> chi = beamline.mathutils.Chicane(bend_length=1,bend_field=0.5,drift_length=1,gamma=1000)
>>> chi.getMatrix()
>>> r56 = chi.getR(5,6)  # r56 = -0.432
>>> chi.setDriftLength([2,4,2])
>>> # same effect (to R56) as ``chi.setDriftLength([2,4])`` or ``chi.setDriftLength([2])``
>>> # or ``chi.setDriftLength(2)``
>>> r56 = chi.getR(5,6)  # r56 = -0.620
```

setGamma(*x*)
set electron energy, gamma value

Parameters **x** – new energy, gamma value

Returns None

setParams(*bend_length*, *bend_field*, *drift_length*, *gamma*)
set chicane parameters

Parameters

- **bend_length** – bend length, [m]
- **bend_field** – bend field, [T]
- **drift_length** – drift length, [m], list
- **gamma** – electron energy, gamma

Returns None

funTransChica(*imagl*, *idril*, *ibfield*, *gamma0*, *xoy='x'*)
Chicane matrix, composed of four rbends, seperated by drifts

Parameters

- **imagl** – rbend width, in [m]
- **idril** – drift length between two adjacent rbends, in [m]
- **ibfield** – rbend magnetic strength, in [T]
- **gamma0** – electron energy, gamma
- **xoy** – `'x'` or `'y'`, matrix in X or Y direction, `'x'` by default

Returns 2x2 numpy array

funTransDrift(*s*)
Drift space

Parameters **s** – drift length, in [m]

Returns 2x2 numpy array

funTransEdgeX(*theta*, *rho*)
Fringe matrix in X

Parameters

- **theta** – fringe angle, in [rad]
- **rho** – bend radius, in [m]

Returns 2x2 numpy array

funTransEdgeY(*theta*, *rho*)
Fringe matrix in Y

Parameters

- **theta** – fringe angle, in [rad]
- **rho** – bend radius, in [m]

Returns 2x2 numpy array

funTransQuadD(*k*, *s*)
Defocusing quad in X, focusing in Y

Parameters

- **k** – k1, in [T/m]
- **s** – width, in [m]

Returns 2x2 numpy array

funTransQuadF(*k*, *s*)
Focusing quad in X, defocusing in Y

Parameters

- **k** – k1, in [T/m]
- **s** – width, in [m]

Returns 2x2 numpy array

funTransSectX(*theta*, *rho*)
Sector matrix in X

Parameters

- **theta** – bend angle, in [rad]
- **rho** – bend radius, in [m]

Returns 2x2 numpy array

funTransSectY(*theta*, *rho*)
Sector matrix in Y

Parameters

- **theta** – bend angle, in [rad]
- **rho** – bend radius, in [m]

Returns 2x2 numpy array

funTransUnduH(*s*)
Planar undulator transport matrix in horizontal direction

Parameters **s** – horizontal width, in [m]

Returns 2x2 numpy array

`funTransUnduV`(*k*, *s*)
Planar undulator transport matrix in vertical direction

Parameters

- **k** – equivalent k1, in [T/m], i.e. natural focusing
- **s** – horizontal width, in [m]

Returns 2x2 numpy array

`test`()

`transChicane`(*bend_length=None*, *bend_field=None*, *drift_length=None*, *gamma=None*)

Transport matrix of chicane composed of four rbends and three drifts between them

Parameters

- **bend_length** – rbend width in [m]
- **bend_field** – rbend magnetic field in [T]
- **drift_length** – drift length, list or tuple of three elements, in [m] single float number stands for same length for three drifts
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

`transDrift`(*length=0.0*, *gamma=None*)
Transport matrix of drift

Parameters

- **length** – drift length in [m]
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

`transFringe`(*beta=None*, *rho=None*)
Transport matrix of fringe field

Parameters

- **beta** – angle of rotation of pole-face in [RAD]
- **rho** – bending radius in [m]

Returns 6x6 numpy array

`transQuad`(*length=0.0*, *k1=0.0*, *gamma=None*)
Transport matrix of quadrupole

Parameters

- **length** – quad width in [m]
- **k1** – quad k1 strength in [T/m]
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

`transRbend`(*theta=None*, *rho=None*, *gamma=None*, *incsym=-1*)
Transport matrix of rectangle dipole

Parameters

- **theta** – bending angle in [RAD]
- **incsym** – incident symmetry, -1 by default, available options:
  - -1: left half symmetry,
  - 0: full symmetry,
  - 1: right half symmetry
- **rho** – bending radius in [m]
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

transSect(*theta=None*, *rho=None*, *gamma=None*)
Transport matrix of sector dipole

Parameters

- **theta** – bending angle in [RAD]
- **rho** – bending radius in [m]
- **gamma** – electron energy, gamma value

Returns 6x6 numpy array

beamline.matchutils module

classes to do beam optics matching Tong Zhang Aug. 10, 2015

class BeamMatch(*infile_mod*, *infile_rad*, *latfile_mod*, *latfile_rad*, *infile_mod_new*, *infile_rad_new*, *latfile_rad_new*, *qfval*, *qdval*)
Bases: `object`

matchCalculate(*latlengthname='fullat.hghg'*)

matchPerform(*qf_linenum=11*, *qd_linenum=13*)

matchPrintout()

class FELSimulator(*mode='HGHG'*, *modinfile='mod.in'*, *radinfile='rad.in'*, *modlatfile='mod.lat'*, *radlatfile='rad.lat'*)
Bases: `object`

getMaxPower()

grepParam(*param='entries'*, *outfile='rad.out'*)

plotPower()

postProcess(*outfile='rad.out'*)

run()

class ParseParams(**infilename*)
Bases: `object`

getChicaneDriftLength()

getChicaneMagnetField()

getChicaneMagnetLength()

getElectronEmitx()

getElectronGamma()

getFELwavelength()

getUndulatorParameter()

getUndulatorPeriod()

getUndulatorUnitlength()

onParseFile()

parseLattice(*latlengthname='fullat.hghg'*)

test()

beamline.element module

This module defines all kinds of magnet components/elements.

class **ElementBeamline**(*name='bl'*, *config=None*)
Bases: `beamline.element.MagBlock`

beamline element, virtual element, does not present in ELEGANT

class **ElementCenter**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

center element

class **ElementCharge**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

charge element

Parameters

- **name** – charge element name that could be used in other method, e.g. 'C', 'Q', etc.
- **config** –

Example

```
>>> chconf = {'total': 1.0e-9}  # total charge of 1.0 nC
>>> q = ElementCharge(name='q', config=chconf)
```

class **ElementCsrcsben**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

csrcsben element

calcTransM(*gamma=None*, *type='simu'*, *incsym=-1*)
calculate transport matrix

Parameters

- **gamma** – electron energy measured by mc^2
- **type** – configuration type, 'simu' (simulation mode) or 'online' (online mode)
- **incsym** – incident symmetry, -1, 0, 1

Returns transport matrix

Return type numpy array

field

Returns magnetic field, [T]

rho

Returns bending radius, [m]

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – rotation angle [deg] of drawing central point, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

ElementCsrcsbent
alias of ElementCsrcsben

ElementCsrdrif
alias of ElementCsrdrift

class ElementCsrdrift(*name=None*, *config=None*)
Bases: beamline.element.MagBlock

csrdrift element

calcTransM(*gamma=None*)
calculate transport matrix

Parameters **gamma** – electron energy measured by mc^2

Returns transport matrix

Return type numpy array

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

ElementDrif
alias of ElementDrift

class ElementDrift(*name=None*, *config=None*)
Bases: beamline.element.MagBlock

drift element

calcTransM(*gamma=None*)
calculate transport matrix

Parameters **gamma** – electron energy measured by mc^2

Returns transport matrix

Return type numpy array

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementKicker**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

kicker element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

ElementLscdrif
alias of `ElementLscdrift`

class **ElementLscdrift**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

lscdrift element

calcTransM(*gamma=None*)
calculate transport matrix

Parameters **gamma** – electron energy measured by mc^2

Returns transport matrix

Return type numpy array

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementMark**(*name=None, config=None*)
Bases: `beamline.element.MagBlock`

mark element

setDraw(*p0=(0, 0), angle=0, mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementMoni**(*name=None, config=None*)
Bases: `beamline.element.MagBlock`

moni element

setDraw(*p0=(0, 0), angle=0, mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementQuad**(*name=None, config=None*)
Bases: `beamline.element.MagBlock`

quad element

calcTransM(*gamma=None, type='simu'*)
calculate transport matrix

Parameters **gamma** – electron energy measured by mc^2

Returns transport matrix

Return type numpy array

getK1(*type='simu'*)
get quad k1 value

Parameters **type** – 'simu' or 'online'

Returns quad strength,i.e. k1

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – rotation angle [deg] of drawing central point, angle is rotating from x-axis to be '+' or '-', '+': anticlockwise, '-': clockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

unitTrans(*inval*, *direction='+'*, *transfun=None*)

class **ElementRfcw**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

rfcw element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementRfdf**(*name=None*, *config=None*)
Bases: `beamline.element.MagBlock`

rfdf element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementWake**(*name=None*, *config=None*)

Bases: `beamline.element.MagBlock`

wake element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)

set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis, angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **ElementWatch**(*name=None*, *config=None*)

Bases: `beamline.element.MagBlock`

watch element

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)

set element visualization drawing

Parameters

- **p0** – start drawing position, (x,y)
- **angle** – angle [deg] between x-axis angle is rotating from x-axis to be '+' or '-', '+': clockwise, '-': anticlockwise
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setStyle(***style*)

style

class **MagBlock**(*name=None*)

Bases: `object`

Super class of all elements, part of configuration parameters are defined here:

- objcnt: object counter, if create/add element one by one, objcnt will return the total element number by sumObjNum() method;
- comminfo: the shared common information for all elements, could be defined by calling setCommInfo() static method;
- __styleconfig_dict: style configurations for element drawing, could be defined by setStyleConfig() static method;

New element should inherit MagBlock, and define following methods: __init__(), setStyle(), setDraw()

class constructor :param name: literal name of the element, None by default

comminfo = {}

static **copy_patches**(*ptches0*)

return a list of copied input matplotlib patches

Parameters **ptches0** – list of matploblib.patches objects

Returns copyed patches object

`dumpConfig`(*type='online'*, *format='elegant'*)

dump element configuration to given format, inpurt parameters:

Parameters

- **`type`** – comm, simu, ctrl, misc, all, online (default)
- **`format`** – elegant/mad, elegant by default

`getConfig`(*type='online'*, *format='elegant'*)
only dump configuration part, dict

Parameters

- **`type`** – comm, simu, ctrl, misc, all, online (default)
- **`format`** – elegant/mad, elegant by default

`getLength`()
return element length if valid, or return 0.0

`getMatrix`()
return 6 x6 dims transport matrix

`getPosition`()
return the element position along beamline/lattice, in [m] should be initialized in Models.initPos() method first (by default, will complete after Models.addElement() method) i.e. valid position in [m] would return after lattice modeled.

`getR`(*i*, *j*)

return transport matrix element, indexed by i(row) and j(col), with the initial index of 1

Parameters

- **`i`** – row index
- **`j`** – col index

`name`
element name property :return: element name

`objcnt = 0`

`printConfig`(*type='simu'*)
print information about element

Parameters **`type`** – comm, simu, ctrl, misc, all

static `rot`(*inputArray*, *theta=0*, *pc=(0, 0)*)
rotate input array with angle of theta

Parameters

- **`inputArray`** – input array or list, e.g. np.array([[0,0],[0,1],[0,2]]) or [[0,0],[0,1],[0,2]]
- **`theta`** – rotation angle in degree
- **`pc`** – central point coords (x,y) regarding to rotation

Returns rotated numpy array

static `setCommInfo`(*infostr*)
set common information, update MagBlock.comminfo

Parameters **infostr** – should be met one of the following options:

- infostr is a dict, {k1:v1, k2:v2}
- infostr is a string, with format like: "k1=v1, k2=v2"

setConf(*conf*, *type='simu'*)
set information for different type dict,

Parameters

- **conf** – configuration information, str or dict
- **type** – simu, ctrl, misc

setDraw(*p0=(0, 0)*, *angle=0*, *mode='plain'*)
set element visualization drawing

Parameters

- **angle** – rotation angle
- **p0** – start drawing point coords, (x, y)
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

setPosition(*s*)
set element position along beamline/lattice, in [m]

Parameters **s** – element position measured by meter

setStyle(***style*)
set element style configuration

Parameters **style** – dict of keys: 'color', 'h', 'alpha'

static **setStyleConfig**(*config=None*, *showhelp=False*)
set/update global style configurations for magblock elements update Magblock._styleconfig_dict and _styleconfig_json

Parameters

- **config** – configuration dict or json
- **showhelp** – if True, print showhelp information, default is False

Returns new style config dict

Example

```
>>> MagBlock.setStyleConfig(
    config={'quad':{'fc':'blue', 'ec': 'blue'},
            'bend':{'fc':'red', 'ec': 'red'}})
>>> MagBlock.setStyleConfig(showhelp=True)
The input configuration string should be with the format like:
{"quad": {"h": 0.6, "fc": "red", "ec": "red", "alpha": 0.5},
 "drift": {"color": "black", "h": 0.1, "alpha": 0.75, "lw": 1},
 "bend": {"h": 0.5, "fc": "blue", "ec": "blue", "alpha": 0.5},
 "moni": {"color": "#FF9500", "lw": 1, "alpha": 0.75}}
with all or part of new properties, e.g. {"quad": {"fc": "blue"}}
```

showDraw(*fignum=1*)
show the element drawing

Parameters **fignum** – define figure number to show element drawing

static **str2dict**(*istr*)
translate string into dict

Parameters **`istr`** – string with format like: "k1=v1, k2=v2" ...

Returns dict

static **`sumObjNum`**()

Returns number of defined element object

`unitTrans`(*inval*, *direction='+'*, *transfun=None*)
unit translation between EPICS PV and physical values,

Parameters

- **`inval`** – input val,
- **`direction`** – '+': PV->physical, '-': physical->PV, '+' by default,
- **`transfun`** – userdefined translation function, None by default, could be defined through creating obj.transfun

`test`()

beamline.lattice module

Classes and routines to handle lattice issues for online modeling and runtime calculations.

- class `LteParser`: parse `ELEGANT` lattice definition files for simulation:
  1. convert lte file into dict/json format for further usage;
  2. resolve rpn expressions within element definitions;
  3. retain prefixed information of lte file as '_prefixstr' key in json/dict;
- class `Lattice`: handle lattice issues from json/dict definitions:
  1. instantiate with json/dict lattice definition, e.g. from `LteParser.file2json()`;
  2. generate lte file for elegant simulation;
  3. iteratively expand the beamline definition in lte file;
  4. generate lte file after manipulations.

class **`Lattice`**(*elements*)
Bases: `object`

class for handling lattice configurations and operations

`dumpAllElements`()
dump all element configuration lines as json format.

`formatElement`(*kw*, *format='elegant'*)
convert json/dict of element configuration into elegant/mad format :param kw: keyword

`generateLatticeFile`(*beamline*, *filename=None*, *format='elegant'*)
generate simulation files for lattice analysis, e.g. ".lte" for elegant, ".madx" for madx

input parameters: :param beamline: keyword for beamline :param filename: name of lte/mad file,

if None, output to stdout; if 'sio', output to a string as return value; other cases, output to filename;

Parameters **`format`** – madx, elegant, 'elegant' by default, generated lattice is for elegant tracking

generateLatticeLine(*latname='newline'*, *line=None*)
construct a new lattice line :param latname: name for generated new lattice

getAllBl()
return all beamline keywords

getAllEle()
return all element keywords

getAllKws()
extract all keywords into two categories

kws_ele: magnetic elements kws_bl: beamline elements

return (kws_ele, kws_bl)

getBeamline(*beamlineKw*)
get beamline definition from all_elements, return as a list :param beamlineKw: keyword of beamline

getChargeElement()
return charge element name

getElementByName(*beamline*, *name*)
return element list by literal name in beamline each element is tuple like (name, type, order) :param beamline: beamline name :param name: element literal name

getElementByOrder(*beamline*, *type*, *irange*)

return element list by appearance order in beamline, which could be returned by orderLattice(beamline)

param beamline beamline name

param type element type name

param irange selected element range

possible irange definitions: irange = 0, first one 'type' element; irange = -1, last one irange = 0,2,3, the first, third and fourth 'type' element irange = 2:10:1, start:end:setp range irange = 'all', all

getElementConf(*elementKw*, *raw=False*)
return configuration for given element keyword, e.g. getElementConf('Q01') should return dict: {u'k1': 0.0, u'l': 0.05} :param elementKw: element keyword

getElementCtrlConf(*elementKw*)
return keyword's EPICS control configs, if not setup, return {}

getElementList(*bl*)
return the elements list according to the appearance order in beamline named 'bl'

Parameters `bl` – beamline name

getElementProperties(*name*)
return element properties :param name: element name

getElementType(*elementKw*)
return type name for given element keyword, e.g. getElementType('Q01') should return string: 'QUAD'

getFullBeamline(*beamlineKw*, *extend=False*)
get beamline definition from all_elements, expand iteratively with the elements from

all_elements e.g. element 'doub1' in chi : line=(DBLL2 , doub1 , DP4FH , DP4SH , DBLL5 , DBD ,

B11 , DB11 , B12 , DB12 , PF2 , DB13 , B13 , DB14 , B14 , DBD , DBLL5 , doub2 , DP5FH , DP5SH , DBLL2 , PSTN1)

should be expaned with 'doub1' configuration: doub1 : line=(DQD3, Q05, DQD2, Q06, DQD3)

since: getBeamline('doub1') = [u'dqd3', u'q05', u'dqd2', u'q06', u'dqd3'] = A getBeamline('doub2') = [u'dqd3', u'q05', u'dqd2', u'q06', u'dqd3'] = B getBeamline('chi') = [u'dbll2', u'doub1', u'dp4fh', u'dp4sh', u'dbll5', u'dbd', u'b11', u'db11', u'b12',

u'db12', u'pf2', u'db13', u'b13', u'db14', u'b14', u'dbd', u'dbll5', u'doub2', u'dp5fh', u'dp5sh', u'dbll2', u'pstn1']

thus: getFullBeamline('chi') should return: [u'dbll2', A, u'dp4fh', u'dp4sh', u'dbll5', u'dbd', u'b11', u'db11', u'b12', u'db12', u'pf2', u'db13',

u'b13', u'db14', u'b14', u'dbd', u'dbll5', B, u'dp5fh', u'dp5sh', u'dbll2', u'pstn1']

Parameters **extend** – if extend mode should be envoked, by default False

if extend = True, element like '2*D01' would be expended to be D01, D01

isBeamline(*kw*)
test if kw is a beamline :param kw: keyword

makeElement(*kw*)
return element object regarding the keyword configuration

manipulateLattice(*beamline*, *type='quad'*, *irange='all'*, *property='k1'*, *opstr='+0%'*)
manipulate element with type, e.g. quad

input parameters: :param beamline: beamline definition keyword :param type: element type, case insensitive :param irange: slice index, see getElementByOrder() :param property: element property, e.g. 'k1' for 'quad' strength :param opstr: operation, '+[-]n%' or '+[-*/]n'

orderLattice(*beamline*)
ordering element type appearance sequence for each element of beamline e.g. after getFullBeamline, lattice list ['q','Q01', 'B11', 'Q02', 'B22'] will return: [(u'q', u'CHARGE', 1),

(u'q01', u'QUAD', 1), (u'b11', u'CSRCSBEN', 1), (u'q02', u'QUAD', 2), (u'b12', u'CSRCSBEN', 2)]

rinseElement(*ele*)
resolve element case with multiply format, e.g. rinseElement('10*D01') should return dict {'num': 10; 'name' = 'D01'} :param ele: element string

showBeamlines()
show all defined beamlines

class LteParser(*infile*, *mode='f'*)
Bases: object

Parameters

- **infile** – lte filename or list of lines of lte file
- **mode** – 'f': treat infile as file, 's': (else) treat as list of lines

detectAllKws()
Detect all keyword from infile, return as a list

USAGE: kwslist = detectAllKws()

`dict2json`(*idict*)
convert dict into json

USAGE: rjson = dict2json(idict)

`file2json`(*jsonfile=None*)
Convert entire lte file into json like format

USAGE: 1: kwsdictstr = file2json() 2: kwsdictstr = file2json(jsonfile = 'somefile')

show pretty format with pipeline: | jshon, or | pjson if jsonfile is defined, dump to defined file before returning json string :param jsonfile: filename to dump json strings

`getKw`(*kw*)

Extract doc snippet for element configuration,

param kw element name

return instance itself 1 call getKwAsDict() to return config as a dict 2 call getKwAsJson() to return config as json string 3 call getKwAsString() to return config as a raw string

USAGE: getKw('Q10')

`getKwAsDict`(*kw*)
return keyword configuration as a dict

Usage: rdict = getKwAsDict(kw)

`getKwAsJson`(*kw*)
return keyword configuration as a json

Usage: rjson = getKwAsJson(kw) :param kw: element keyword

`getKwAsString`(*kw*)
return keyword configuration as a string

Usage: rstr = getKwAsString(kw)

`getKwConfig`(*kw*)
return the configuration of kw, dict

USAGE: rdict = getKwConfig(kw)

`getKwCtrlConf`(*kw*, *fmt='dict'*)
return keyword's control configuration, followed after '!epics' notation :param kw: keyword name :param fmt: return format, 'raw', 'dict', 'json', default is 'dict'

`getKwType`(*kw*)
return the type of kw, upper cased string

USAGE: rtype = getKwType(kw)

`get_rpndict_flag`(*rpndict*)
calculate flag set, the value is True or False, if rpndict value is not None, flag is True, or False

if a set with only one item, i.e. True returns, means values of rpndict are all valid float numbers, then finally return True, or False

`makeElement`(*kw*)
return element object regarding the keyword configuration

`resolveEPICS`()
extract epics control configs into

resolvePrefix()
extract prefix information into dict with the key of '_prefixstr'

resolve_rpn(*rpndict*)
solve dict of rpn expressions to pure var to val dict :param rpndict: dict of rpn expressions return pure var to val dict

rinse_rpnexp(*rpnexp*, *rpndict*)
replace valid keyword of rpnexp from rpndict e.g. rpnexp = 'b a /', rpndict = {'b': 10} then after rinsing, rpnexp = '10 a /'

return rinsed rpnexp

rpn2val(*rdict*)
Resolve the rpn string into calulated float number

USAGE: rpn2val(rdict)

param rdict json like dict

scanStoVars(*strline*)
scan input string line, replace sto parameters with calculated results.

solve_rpn()
solve rpn string in self.confdict, and update self.confdict

USAGE: ins = LteParser(infile) ins.getKw(kw).toDict().solve_rpn()

str2dict(*rawstr*)
convert str to dict format

USAGE: rdict = str2dict(rawstr) :param rawstr: raw configuration string of element

toDict()
convert self.confstr to dict, could apply chain rule, write to self.confdict

USAGE: ins = LteParser(infile) ins.getKw(kw).toDict()

update_rpndict(*rpndict*)
update rpndict, try to solve rpn expressions as many as possible, leave unsolvable unchanged.

return new dict

main()

test2()

beamline.models module

This module is written for the purposes of elements modeling for accelerator: 1: manually define magnetic elements one by one and model the machine; 2: interpret lattice file (.lte file) to be modeled elements; 2: update (EPICS databases)/(EPICS PVs) with new configuration.

Author : Tong Zhang Created : 2016-03-18

class Models(*name='BL'*, *mode='simu'*)
Bases: `object`

make lattice configuration (json) for lattice.Lattice return instance as a json string file with all configuration. get lattice name by instance.name.

LatticeDict
show lattice configuration

LatticeList
show lattice element list

addElement(**ele*)
add element to lattice element list

Parameters **ele** – magnetic element defined in element module

return total element number

static **anoteElements**(*ax*, *anotelist*, *showAccName=False*, *efilter=None*, *textypos=None*, ***kwargs*)
annotate elements to axes

Parameters

- **ax** – matplotlib axes object
- **anotelist** – element annotation object list
- **showAccName** – tag name for accelerator tubes? default is False, show acceleration band type, e.g. 'S', 'C', 'X', or for '[S,C,X]D' for cavity
- **efilter** – element type filter, default is None, annotate all elements could be defined to be one type name or type name list/tuple, e.g. filter='QUAD' or filter=('QUAD', 'CSRCSBEN')
- **textypos** – y coordinator of annotated text string
- **kwargs** – alpha=0.8, arrowprops=dict(arrowstyle='->'), rotation=-60, fontsize='small'

return list of annotation objects

draw(*startpoint=(0, 0)*, *mode='plain'*, *showfig=False*)
lattice visualization

Parameters

- **startpoint** – start drawing point coords, default: (0, 0)
- **showfig** – show figure or not, default: False
- **mode** – artist mode, 'plain' or 'fancy', 'plain' by default

Returns patchlist, anotelist, (xmin0, xmax0), (ymin0, ymax0) patchlist: list of element patches anotelist: list of annotations (xmin0, xmax0) and (ymin0, ymax0) are ploting range

static **flatten**(*ele*)
flatten recursively defined list, e.g. [1,2,3, [4,5], [6,[8,9,[10,[11,'x']]]]]

Parameters **ele** – recursive list, i.e. list in list in list ...

Returns generator object

getAllConfig(*fmt='json'*)
return all element configurations as json string file. could be further processed by beamline.Lattice class

Parameters **fmt** – 'json' (default) or 'dict'

getCtrlConf(*msgout=True*)
get control configurations regarding to the PV names, read PV value

Parameters **msgout** – print information if True (by default)

return updated element object list

getElementsByName(*name*)
get element with given name, return list of element objects regarding to 'name'

Parameters **name** – element name, case sensitive, if elements are auto-generated from LteParser, the name should be lower cased.

initPos(*startpos=0.0*)
initialize the elements position [m] in lattice, the starting point is 0 [m] for the first element by default.

Parameters **startpos** – starting point, 0 [m] by default

static **makeLatticeDict**(*ele*)
return lattice dict conf like {"lattice": "(q b d)"}

Parameters **ele** – element list

static **makeLatticeString**(*ele*)
return string like "lattice = (q b d)"

Parameters **ele** – element list

mode

name

static **plotElements**(*ax*, *patchlist*)
plot elements' drawings to axes

Parameters

- **ax** – matplotlib axes object
- **patchlist** – element patch object list

printAllElements()
print out all modeled elements

putCtrlConf(*eleobj*, *ctrlkey*, *val*, *type='raw'*)
put the value to control PV field

Parameters

- **eleobj** – element object in lattice
- **ctrlkey** – element control property, PV name
- **val** – new value for ctrlkey
- **type** – set in 'raw' or 'real' mode, 'raw' by default 'raw': set PV with the value of 'val', 'real': set PV with the value translated from 'val'

updateConfig(*eleobj*, *config*, *type='simu'*)
write new configuration to element

Parameters

- **eleobj** – define element object
- **config** – new configuration for element, string or dict
- **type** – 'simu' by default, could be online, misc, comm, ctrl

test()

test1()

beamline.simulation module

Module designed for online modeling: *elegant tracking with lte/ele files*:

> 1: lte file should be generated from lattice.Lattice.generateLatticeFile() method; 2: take ele file as initialization parameter, but could be changed; 3: output tracking results as hdf5 file (hard drive) and numpy array (memory);

Author : Tong Zhang Created : 2016-03-08 Last updated: 2016-03-08

class Simulator(*infile=''*)

Bases: `object`

doSimulation()

getOutput(***kws*)

setExec(*execpath*)

set executable for simulation :param execpath: elegant or madx full path

setInputfiles(***infiles*)

input parameters: (elegant mode)

1: lte file 2: ele file

(mad mode) 1: mad file

setMode(*mode='elegant'*)

set simulation mode, define mode parameter of 'elegant' or 'mad' :param mode: simulation mode

setPath(*simpath*)

set simulation path where data should be put into :param simpath: where simulations take place, all data files should be found there

setScript(*fullname*)

set bash shell script full path name for simulation :param fullname: set 'runElegant.sh', which should be available after installed beamline package

test()

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

A

B

C

D

E

F

G

I

J

L

M

N

O

P

Q

R

S

T

U