

---

# **FLAME Documentation**

***Release 1.5.1***

**Kei Fukushima**

**Jun 13, 2018**



**CONTENTS:**

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Build from source code . . . . .	3
<b>2</b>	<b>Tutorial</b>	<b>5</b>
2.1	1. Basic usage . . . . .	5
2.2	2. Lattice parameter control . . . . .	6
2.3	3. Run for the selected section . . . . .	8
2.4	4. Example: Quadrupole scan . . . . .	8
<b>3</b>	<b>Lattice File</b>	<b>11</b>
3.1	General parameter . . . . .	11
3.2	Beam parameter . . . . .	12
3.3	Lattice elements . . . . .	12
3.4	Rf cavity data format . . . . .	18
<b>4</b>	<b>Class Library</b>	<b>21</b>
4.1	Machine class . . . . .	21
4.2	State Class . . . . .	23
<b>5</b>	<b>Indices</b>	<b>27</b>
	<b>Index</b>	<b>29</b>



## FLAME - Fast Linear Accelerator Model Engine

FLAME is high-speed envelope tracking code developed in FRIB.

### *Remarkable Features*

- Envelope tracking with multiple charge states
- Support general lattice elements and asymmetric rf cavity by Thin-Lens-Model
- Transfer matrix caching for iterative running
- Python interface (include ipython-notebook)



## INSTALLATION

### 1.1 Build from source code

Git clone from FLAME repository.

```
$ git clone *repository-address*
```

Pre-requisites (may need to apt-get with sudo)

```
$apt-get install libboost-dev libboost-system-dev \  
  libboost-thread-dev libboost-filesystem-dev \  
  libboost-regex-dev libboost-program-options-dev \  
  libboost-test-dev \  
  build-essential cmake bison flex cppcheck git libhdf5-dev \  
  python-numpy python-nose python3-numpy python3-nose
```

FLAME supports python 2.7 and 3.4, EPICS interface is optional.

Make build directory and compile with CMake.

```
$ cd flame  
$ mkdir build  
$ cd build  
$ cmake ..  
$ make
```

Test FLAME (include the beam dynamics test).

```
$ make test
```

Install with proper permissions.

```
$ make install # may need to install with sudo
```





## TUTORIAL

### 2.1 1. Basic usage

In Python interface (include IPython-notebook), user can import flame *Machine* class.

```
>>> from flame import Machine
```

Create *Machine* object with input file.

```
>>> with open('lattice_file.lat', 'rb') as f :  
>>>     M = Machine(f)
```

Allocate the beam state. - *Machine.allocState()*, *State*

```
>>> S = M.allocState({})
```

Run envelope tracking simulation. - *Machine.propagate()*

```
>>> M.propagate(S)
```

The beam state has the finite state beam information. - *State()*

```
>>> S                # centroid vector  
State: moment0 mean=[7] (3.18839,0.00871355,-12.0779,-0.00254204,-35.2039,0.000489827,  
↪1)  
>>> S.ref_IonEk      # reference energy  
11969.995341581
```

*The attribute list of the beam state can be found here.*

User can observe the beam state history by using observe keyword in *propagate()*.

```
>>> result = M.propagate(S, observe=range(len(M))) # observe the beam state in all_  
↪elements
```

It returns enumerated list of the beam state.

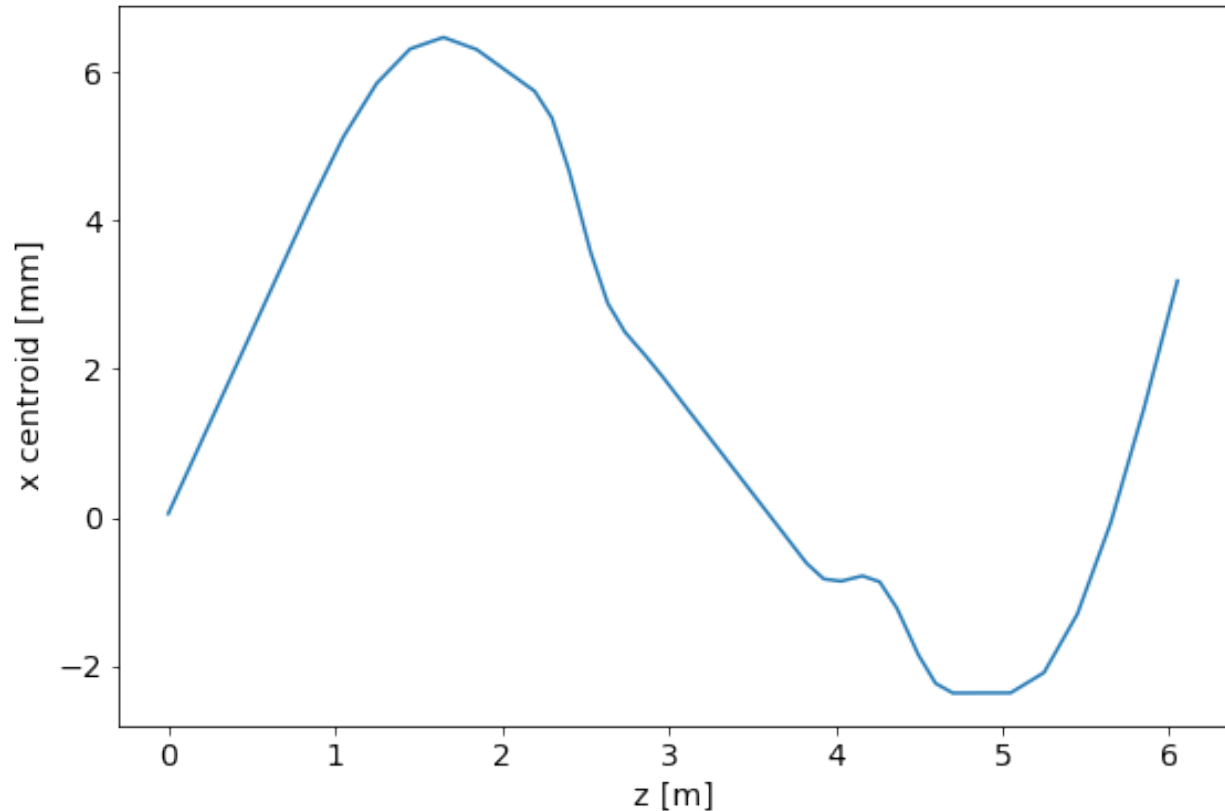
```
>>> result[3]  
(3, State: moment0 mean=[7] (2.2532,0.00489827,2.2532,0.00489827,-2.7162,0.000489827,  
↪1))
```

User can generate the history data from the list of beam states,

```
>>> z = [s[1].pos for s in result] # reference beam position history
>>> x = [s[1].moment0_env[0] for s in result] # x centroid history
```

and plot.

```
>>> import matplotlib.pyplot as plt
>>> plt.plot(z, x)
>>> plt.ylabel('x centroid [mm]')
>>> plt.xlabel('z [m]')
>>> plt.show()
```



## 2.2 2. Lattice parameter control

`conf()` returns initial machine parameter.

```
>>> M.conf()
OrderedDict([('AMU', 931494320.0),
             ('BaryCenter0',
              array([ 0.1 ,  0.01 ,  0.1 ,  0.01 ,  0.001,  0.001,  1.   ])),
             ('BaryCenter1', array([ 0.,  0.,  0.,  0.,  0.,  0.,  1.])),
             ('IonChargeStates', array([ 0.13865546,  0.14285714])),
             ('IonEk', 11969.995341581),
             ('IonEs', 931494320.0),
             ('IonW', 931506289.9953415),
             ('IonZ', 0.13865546218487396),
             ('NCharge', array([ 10111., 10531.])),
```

```

('S0',
 array([ 3.68800000e+02,  2.50000000e-02,  0.00000000e+00,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         0.00000000e+00,  2.50000000e-02,  2.88097000e-05,
         0.00000000e+00,  0.00000000e+00,  0.00000000e+00,
         ...

```

User can *find* the element index by *element type* or *element name*.

```

>>> M.find(type='solenoid')
[15, 16, 18, 19, 21, 22, 27, 28, 30, 31, 33, 34]
>>> M.find(name='qlh_1')
[15]

```

*conf(index)* returns all parameters of the element.

```

>>> M.conf(15).keys() # parameter keywords
['AMU', 'B2', 'BaryCenter0', 'BaryCenter1', 'IonChargeStates', 'IonEk', 'IonEs', 'IonW
↪', 'IonZ', 'L', 'NCharge', 'S0', 'S1', 'aper', 'name', 'sim_type', 'type']
>>> M.conf(15)['B2'] # quadrupole strength
0.942438547187938

```

Change the parameter by using *reconfigure()*.

```

>>> M.reconfigure(15, {'B2': 0.8})

```

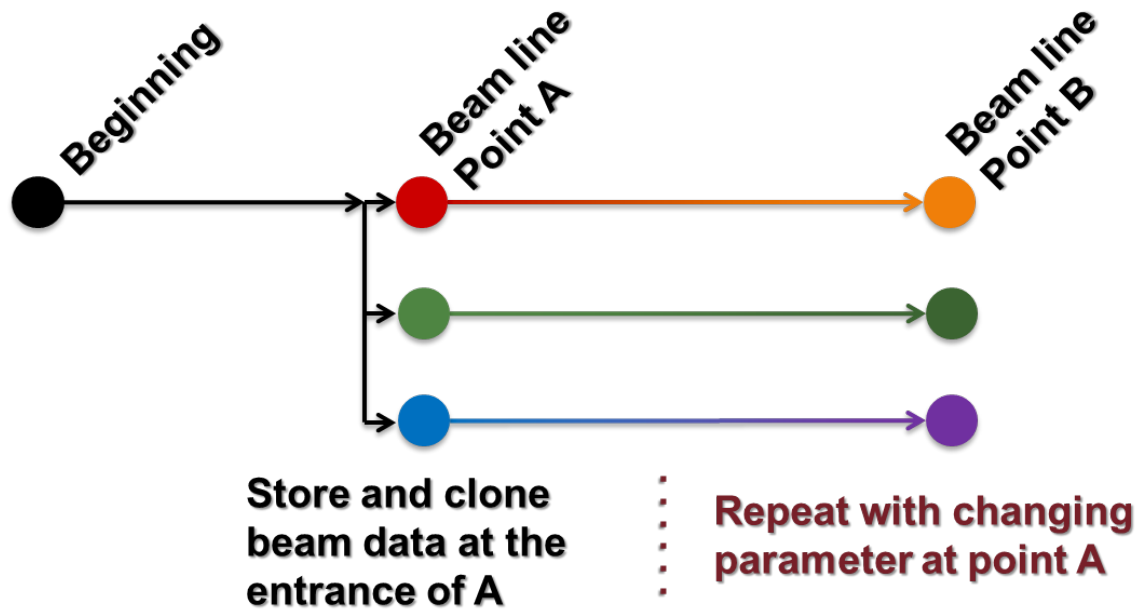
Check new parameter of the solenoid.

```

>>> M.conf(15)['B2']
0.8

```

## 2.3 3. Run for the selected section



User can input *start-point index* and *end-point index* to *propagate*.

```
>>> M.propagate(S, 0, 10) # simulate from 0th to the entrance of 10th element
>>> S1 = S.clone() # clone the beam state
>>> M.propagate(S1, 10, -1) # simulate from 10th to the last element
```

In this case, “S” has the beam state after the 9th element, and “S1” has the finite beam state.

## 2.4 4. Example: Quadrupole scan

Run simulation up to the target element.

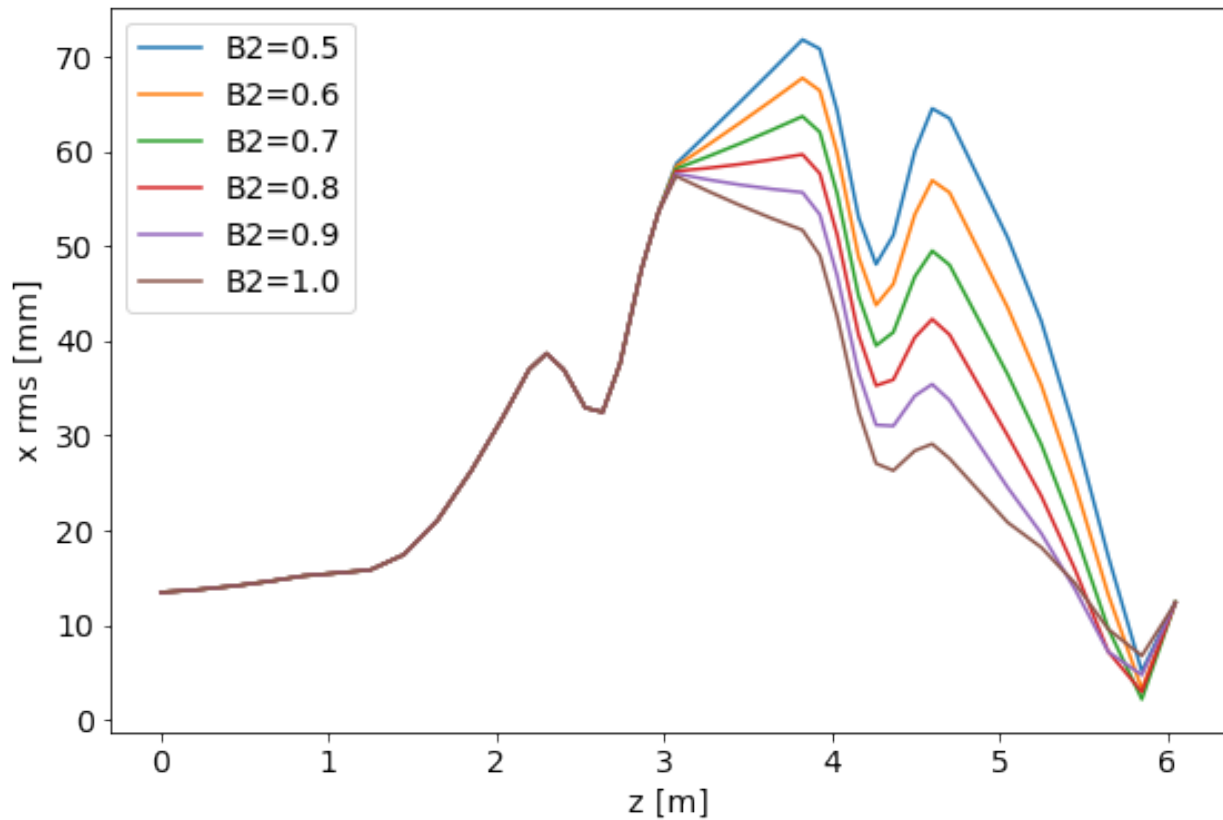
```
>>> M.find(name='q3h_6') # get index of the target element
[22]
>>> ini = M.conf(22) ['B2'] # store the initial quadrupole strength
>>> ini
0.853489750615018
>>> SA = M.allocState({})
>>> rA = M.propagate(SA, 0, 22, observe=range(len(M))) # propagate 22 elements from 0
```

Scan parameters by using simple loop.

```
>>> b2lst = [0.5, 0.6, 0.7, 0.8, 0.9, 1.0]
>>> rlst = []
>>> for b2 in b2lst:
>>>     SB = SA.clone()
>>>     M.reconfigure(22, {'B2':b2})
>>>     rt = M.propagate(SB, 22, -1, observe=range(len(M)))
>>>     rlst.append(rt)
```

Plot the scan result.

```
>>> zA = [s[1].pos for s in rA]
>>> xA = [s[1].moment0_rms[0] for s in rA] # get the x rms size
>>>
>>> for b2,rt in zip(b2lst,rlst):
>>>     zt = zA + [s[1].pos for s in rt] # join the history result
>>>     xt = xA + [s[1].moment0_rms[0] for s in rt]
>>>     plt.plot(zt, xt, label='B2='+str(b2))
>>>
>>> plt.ylabel('x rms [mm]')
>>> plt.xlabel('z [m]')
>>> plt.legend(loc='best')
>>> plt.show()
```





## LATTICE FILE

### 3.1 General parameter

Basic format of the general parameters are,

```
keyword1 = "value1";  
keyword2 = "value2";  
...
```

keyword	value	description
<b>sim_type</b>	"MomentMatrix"	Simulation mode. FRIB simulation uses the particular mode "MomentMatrix".
<b>MpoleLevel</b>	"0", "1", or "2"	Multipole term controller for the rf cavities. "0" - only include focusing and defocusing effect "1" - include dipole terms "2" - include dipole and quadrupole terms
<b>EmitGrowth</b>	"0" or "1"	Flag for cross-cavity emittance growth effect. "0" - False (no emittance growth) "1" - True (calculate emittance growth)
<b>HdipoleFitMode</b>	"0" or "1"	Flag for auto-adjustment of bending element "0" - use "bg" or "beta" for the bending strength "1" - auto-adjust the bending strength

## 3.2 Beam parameter

Basic format of the beam parameters are,

```
keyword1 = value1;
keyword2 = [value2, value3]; # list input
...
```

keyword	value	description
<b>IonEs</b>	float	Nucleon mass of the reference beam. [eV/u]
<b>IonEk</b>	float	Initial kinetic energy of the reference beam. [eV/u]
<b>IonChargeStates</b>	list of float	List of charge to mass ratios of the all charge states. [1] The first element is used as the reference beam.
<b>NCharge</b>	list of float	List of macro weights of the all charge states. [1]
<b><math>\{\text{vector\_variable}\}_n</math></b>	vector[7]	Initial centroid vector of the <b>n</b> -th charge state. $\{\text{vector\_variable}\}$ is defined in <a href="#">source</a> . $[x, x', y, y', \phi, E_k, 1]$ with [mm, rad, mm, rad, rad, MeV/u, 1]
<b><math>\{\text{matrix\_variable}\}_n</math></b>	vector[49]	Flattened initial envelope matrix of the <b>n</b> -th charge state. $\{\text{matrix\_variable}\}$ is defined in <a href="#">source</a> . Cartisan product of $[x, x', y, y', \phi, E_k, 1]^2$ with [mm, rad, mm, rad, rad, MeV/u, 1] <sup>2</sup>
<b>Eng_Data_Dir</b>	string	Directory path of the rf cavity data. <code>dir(path)</code> supports relative path.

## 3.3 Lattice elements

Basic format of the one lattice element is,



```
name_of_element1: element_type, parameter1 = value1, parameter2 = value2, ... ;
```

After writing down the all lattice elements, user need to specify the lattice cell and the cell to USE.

```
# define the cell
name_of_cell: LINE = ( name_of_element1, name_of_element2, name_of_element3, ... );

# set the cell to USE
USE: name_of_cell;
```

element_type	description
<i>source</i>	Starting point of the simulation.
<i>marker</i>	Marker element.
<i>stripper</i>	Chage stripper element.
<i>tmatrix</i>	User input transfer matrix.
<i>orbtrim</i>	Orbit trim element.
<i>drift</i>	Drift space element.
<i>solenoid</i>	Solenoid magnet element.
<i>quadrupole</i>	Magnetic quadrupole element.
<i>equad</i>	Electrostatic quadrupole element.
<i>sbend</i>	Magnetic bend element.
<i>edipole</i>	Electrostatic dipole element.
<i>rfcavity</i>	RF cavity element.

### 3.3.1 Special element

#### type source

Starting point of the simulation. Initial beam state parameters are set at this element.

**Parameters** **vector\_variable**: string

Name key of the initial centroid vector.

**matrix\_variable**: string

Name key of the initial envelope matrix.

#### type marker

Marker element. Nothing to do.

#### type stripper

Stripper element.

**Parameters** **IonChargeStates**: list of float

List of charge to mass ratios after the charge stripper. [1]

**charge\_model**: string

Macro weight model for stripper.

- **“baron”** (default): Use Baron formula for the macro weights.

- **“off”**: Use NCharge parameter for the macro weights.

**NCharge**: list of float

List of macro weights after the charge stripper. [1]

This list length must be same as the IonChargeStates

This parameter is used only in the case of `charge_model = "baron"`.

**Stripper\_IonZ:** float (optional, default is **78.0/238.0**)

Charge to mass ratio of the reference beam. [1]

**Stripper\_IonMass:** float (optional, default is **238.0**)

Ion mass of the reference beam. [amu]

**Stripper\_IonProton:** float (optional, default is **92.0**)

Proton number of the reference beam. [1]

**Stripper\_E1Para:** float (optional, default is **2.8874e-3**)

Constant part of the energy straggling parameter of the charge stripper. [MeV/u]

**Stripper\_lambda:** float (optional, default is **5.5740**)

Momentum spread factor  $\lambda$  of the charge stripper. [1]

**Stripper\_upara:** float (optional, default is **2.6903**)

Momentum spread factor  $U$  of the charge stripper. [1]

The momentum spread is defined as  $\sqrt{(U/\lambda^2)}$  [mrad].

**Stripper\_E0Para:** vector[3] (optional, default is **[16.348e6, 1.00547, -0.10681]**)

Energy loss parameters due to the ionization.

[Constant\_part, Energy\_dependence, Thickness\_dependence] with [eV/u, 1, 1]

**Stripper\_Para:** vector[3] (optional, default is **[3.0, 20.0, 16.623e6]**)

Stripper foil parameters.

[Thickness, Thickness\_variation, reference\_energy] with [um, %, eV/u]

**type tmatrix**

User input transfer matrix element.

**Parameter matrix:** vector[49]

Flattened  $7 \times 7$  transfer matrix.

### 3.3.2 Optical element

**type orbtrim**

Orbit trim element. This can be use as steering magnet.

**Parameters realpara:** int

Realistic input parameter flag for the beam kick angle.

**0** - use `theta_x` and `theta_y` for the beam kick.

**1** - use `tm_xkick` and `tm_ykick` for the beam kick.

**theta\_x:** float

Horizontal beam kick angle. [rad]

**theta\_y:** float

Vertical beam kick angle. [rad]

**tm\_xkick:** float

Magnetic field strength for the horizontal beam kick. [T\*m]

**tm\_ykick:** float

Magnetic field strength for the vertical beam kick. [T\*m]

**xyrotate:** float

Transverse rotation angle of the beam. [deg]

---

**Note:** In the case of user puts both “beam kick information” and “transverse rotation angle” to the ONE orbtrim element, the process order is, beam kick -> transverse rotation. In other words, the beam kick is effected BEFORE the transverse rotation.

---

#### **type drift**

Drift space element.

**Parameters L:** float

Length of the lattice element. [m]

#### **type solenoid**

Solenoid magnet element.

**Parameters L:** float

Length of the lattice element. [m]

**B:** float

Solenoid strength ( $B_z$ ). [T]

**dx:** float (default: 0.0)

Misalignment of horizontal shift. [m]

**dy:** float (default: 0.0)

Misalignment of vertical shift. [m]

**pitch:** float (default: 0.0)

Misaglnment of pitch angle. [rad]

**yaw:** float (default: 0.0)

Misaglnment of yaw angle. [rad]

**roll:** float (default: 0.0)

Misaglnment of roll angle. [rad]

#### **type quadrupole**

Magnetic quadrupole element.

**Parameters L:** float

Length of the lattice element. [m]

**B2:** float

Quadrupole field gradient. [T/m]

Positive value means horizontal focusing.

**dx, dy, pitch, yaw, roll:** float

Misalignment parameters. See *solenoid* case.

#### **type equad**

Electrostatic quadrupole element.

**Parameters** **L:** float

Length of the lattice element. [m]

**V:** float

Electrostatic quadrupole pole tip voltage. [V]

Positive value means horizontal focusing.

**radius:** float

Electrostatic quadrupole pole tip radius. [m]

**dx, dy, pitch, yaw, roll:** float

Misalignment parameters. See *solenoid* case.

#### **type sbend**

Magnetic bend element.

**Parameters** **L:** float

Length of the lattice element. [m]

**phi:** float

Bend angle. [deg]

**phi1:** float

Front pole face angle. [deg]

**phi2:** float

Back pole face angle. [deg]

**bg:** float (optional: Used in the case of “*HdipoleFitMode*” is 0.)

Lorentz  $\beta\gamma$  for the reference beam. [1]

This parameter is correspond to the bend field strength.

**dx, dy, pitch, yaw, roll:** float

Misalignment parameters. See *solenoid* case.

#### **type edipole**

Electrostatic dipole (bend) element.

**Parameters** **L:** float

Length of the lattice element. [m]

**phi:** float

Bend angle. [deg]

**beta:** float (optional: Used in the case of “*HdipoleFitMode*” is 0.)

Lorentz  $\beta$  for the reference beam. [1]

This parameter is correspond to the bend field strength.

**fringe\_x**: float

Horizontal fringe term. [rad/mm]

**fringe\_y**: float

Vertical fringe term. [rad/mm]

**asymfac**: float

Characteristic parameter of the kinetic energy change due to the middle point potential deviation from ground. [1]

**spher**: int

Flag for the electrostatic dipole shape.

**0** - cylindrical electrostatic dipole

**1** - spherical electrostatic dipole

**ver**: int

Flag for the bending direction.

**0** - horizontal bend

**1** - vertical bend

**dx, dy, pitch, yaw, roll**: float

Misalignment parameters. See *solenoid* case.

**type rfcavity**

RF cavity element.

**Parameters L**: float

Length of the lattice element. [m]

**cavtype**: string

Cavity type. Supports “Generic”, “0.041QWR”, “0.085QWR”, “0.29HWR”, and “0.53HWR”. *The file format is described here.*

**f**: float

RF frequency of the cavity. [Hz]

**phi**: float

Input phase of the cavity. [deg]

**syncflag**: int

Flag for synchronous phase input (for above parameter **phi**).

**0** for driven phase input.

**1** for synchronous phase input. (default)

**scl\_fac**: float

Scaling factor of the field. [1]

**datafile**: string (optional: Used in the case of `cavtype = “Generic”`)

File path of the rf cavity data.

**Rm**: float (optional: Used in the case of `cavtype = “Generic”`)

Characteristic radial length of the multipole expansion. [mm]

**dx, dy, pitch, yaw, roll:** float

Misalignment parameters. See *solenoid* case.

## 3.4 Rf cavity data format

FLAME using Thin-Lens-Model for rf cavity calculation. Rf cavity data is composed of “Longitudinal axis data”, “Multipole lattice data”, “Multipole field data”, and “TTF fitting data”.

### 3.4.1 Hard-coded FRIB cavity models

For typical rf cavity in FRIB, the “TTF fitting data” is hard-coded in FLAME. Following files are required for each rf cavity type.

cavtype	Longitudinal axis data	Multipole lattice data	Multipole field data
“0.041QWR”	“axisData_41.txt”	“Multipole41/thinlenlon_41.txt”	“Multipole41/CaviMlp_41.txt”
“0.085QWR”	“axisData_85.txt”	“Multipole85/thinlenlon_85.txt”	“Multipole85/CaviMlp_85.txt”
“0.29HWR”	“axisData_29.txt”	“Multipole29/thinlenlon_29.txt”	“Multipole29/CaviMlp_29.txt”
“0.53HWR”	“axisData_53.txt”	“Multipole53/thinlenlon_53.txt”	“Multipole53/CaviMlp_53.txt”

### 3.4.2 Generic rf cavity model

FLAME supports *lattice format* input for the generic rf cavity model.

The basic format of the rf cavity data is similar to the main lattice file,

```
Rm = value1;

Ez = [
z1, Ez1,
z2, Ez2,
z3, Ez3,
...
];

name_of_element1: element_type, parameter1 = value1, parameter2 = value2, ... ;
...

cell: LINE =(name_of_element1, ...);
USE: cell;
```

keyword	value	description
<b>Rm</b>	float	Characteristic radial length of the multipole expansion. [mm]
<b>Ez</b>	vector[2*n]	On axis $E_z$ data. The odd index (1,3,5,...) is z position. [mm] The even index (2,4,6,...) is Electric field strength. [V/m]

### Lattice element for the rf cavity data

Drift space is the same format as the main lattice but unit of  $L$  is [mm] - *drift*

#### type **EDipole**

Dipole term generated by the electric field.

**Parameters** **L**: float

Length of the lattice element. [mm]

This parameter should be 0.0 in Thin-Lens-Model.

**V0**: float

Amplitude of the multipole term. [MV]

**attr**: vector[20]

TTF fitting parameter. ([see here](#))

1 to 10 - fitting parameter for  $T$

11 to 20 - fitting parameter for  $S$

#### type **EFocus**

Constant focusing term generated by the electric field.

Parameters are the same as *EDipole*.

#### type **EQuad**

Quadrupole term generated by the electric field.

Parameters are the same as *EDipole*.

#### type **HMono**

Dipole term generated by the magnetic field.

**Parameters** **L**: float

Length of the lattice element. [mm]

This parameter should be 0.0 in Thin-Lens-Model.

**V0**: float

Amplitude of the multipole term. [MA]

**attr**: vector[20]

TTF fitting parameter. ([see here](#))

1 to 10 - fitting parameter for  $T$

11 to 20 - fitting parameter for  $S$

**type HFocus**

Constant focusing term generated by the magnetic field.

Parameters are the same as [HMono](#).

**type HQuad**

Quadrupole term generated by the magnetic field.

Parameters are the same as [HMono](#).

**type AccGap**

Acceleration gap term by the longitudinal electric field.

**Parameters L:** float

Length of the lattice element. [mm]

This parameter should be 0.0 in Thin-Lens-Model.

**V0:** float

Amplitude of the multipole term. [MV]

**attr:** vector[23]

TTF fitting parameter. ([see here](#))

1 to 10 - fitting parameter for  $T$

11 to 20 - fitting parameter for  $S$

21 to 23 - fitting parameter for the synchronous phase

---

**Note:** FLAME is using TTF-calculation acceleration technique to boost cavity modeling speed. TTF factor  $T$  and  $S$  are pre-calculated and fitted using 9th order polynomial function according to different particle phase speed  $k$ .  $n$ -th fitting parameter  $p_n$  is listed as,

$$T(k), S(k) = \sum_{n=0}^9 p_n k^{(9-n)}.$$

The driven-phase calculation is also boosted by similar method. The phase transferring factor  $\varphi_c$  is fitted by  $p_{i=0,1,2}$

$$\varphi_c = p_0 E^{p_1} + p_2.$$

Here,  $E$  is the kinetic energy. The driven phase  $\varphi_d$  is calculated by using  $\varphi_c$ ,

$$\varphi_d = \varphi_s - \varphi_c - m\varphi_{\text{abs}}$$

where,  $\varphi_s$  is the synchronous phase in input,  $\varphi_{\text{abs}}$  is absolute phase in front of the rf cavity, and  $m$  is the harmonic number.

---



## CLASS LIBRARY

### 4.1 Machine class

**class Machine** (*config*)

FLAME Machine class for Python API.

**Parameter config:** dict, list of tuples, or byte buffer

Input lattice data.

**conf** (*index=None*)

Check configuration of the Machine object.

**Parameter index:** int (optional)

Index of the lattice element.

**Returns**

dict

Configuration of the lattice element

---

**Note:** In the case of *index* is *None*, *conf()* returns *initial* configuration of the lattice.

---

**allocState** (*config=None*)

Allocate the beam state object.

**Parameter config :** dict

Input lattice data. Empty dict is required as dummy data.

**Returns**

*State* object

Beam state object (see here)

**propagate** (*state*, *start*=0, *max*=-1, *observe*=None)

Run envelope tracking simulation.

**Parameters** *state*: *State* object

Allocated beam state object

**start**: int (optional)

Index of the starting lattice element.

**max**: int (optional)

Index of the max (ending) lattice element. It propagates to the entrance of the max-th element.

**observe**: list of int (optional)

List of indexes for observing the beam state.

**Returns**

list

List of the beam states at *observe* points. Each tuple has (*index*, *State*).

**reconfigure** (*index*, *config*)

Reconfigure the lattice element configuration.

**Parameters** *index*: int

Index of the lattice element.

**config**: dict

New configuration of the lattice element parameter.

**find** (*name*=None, *type*=None)

Find the indexes of the lattice elements by *name* or *type*.

**Parameter** *name*: str or unicode

Name of the lattice element to find.

**type**: str or unicode

Type of the lattice element to find.

**Returns**

list

List of matched element indexes.

## 4.2 State Class

**class State** (*object*)

FLAME beam state class for Python API.

**clone** ()

Clone the beam state object.

**Returns** *State* object

- **Attributes - reference beam**

<i>pos</i>	z position [m]
<i>ref_beta</i>	Lorentz $\beta$ [1]
<i>ref_bg</i>	Lorentz $\beta\gamma$ [1]
<i>ref_gamma</i>	Lorentz $\gamma$ [1]
<i>ref_IonEk</i>	Kinetic energy [eV/u]
<i>ref_IonEs</i>	Nucleon mass [eV/u]
<i>ref_IonQ</i>	Macro weight [1]
<i>ref_IonW</i>	Total energy [eV/u]
<i>ref_IonZ</i>	Charge to mass ratio [1]
<i>ref_phis</i>	Absolute phase [rad]
<i>ref_SampleIonK</i>	Phase speed [rad]
<i>last_caviphi0</i>	Synchronous phase of the last rf cavity [deg]

- **Attributes - actual beam**

<i>beta</i>	Lorentz $\beta$ [1]
<i>bg</i>	Lorentz $\beta\gamma$ [1]
<i>gamma</i>	Lorentz $\gamma$ [1]
<i>IonEk</i>	Kinetic energy [eV/u]
<i>IonEs</i>	Nucleon mass [eV/u]
<i>IonQ</i>	Macro weight [1]
<i>IonW</i>	Total energy [eV/u]
<i>IonZ</i>	Charge to mass ratio [1]
<i>phis</i>	Absolute phase [rad]
<i>SampleIonK</i>	Phase speed [rad]
<i>moment0</i>	Centroids of the all charge states.
<i>moment0_env</i>	Total centroid of the all charge states.
<i>moment0_rms</i>	Total rms size of the all charge states.
<i>moment1</i>	Envelope matrixes of the all charge states.
<i>moment1_env</i>	Total envelope matrixes of the all charge states.

**pos**

**float:** z position of the reference beam. [m]

**ref\_beta**

**float:** Lorentz  $\beta$  of the reference beam. [1]

**ref\_bg**

**float:** Lorentz  $\beta\gamma$  of the reference beam. [1]

**ref\_gamma**

**float:** Lorentz  $\gamma$  of the reference beam. [1]

**ref\_IonEk**  
float: Kinetic energy of the reference beam. [eV/u]

**ref\_IonEs**  
float: Nucleon mass of the reference beam. [eV/u]

**ref\_IonQ**  
float: Macro weight of the reference beam. [1]

**ref\_IonW**  
float: Total energy of the reference beam. [eV/u]

**ref\_IonZ**  
float: Charge to mass ratio of the reference beam. [1]

**ref\_phis**  
float: Absolute synchrotron phase of the reference beam. [rad]

**ref\_SampleIonK**  
float: Phase speed of the reference beam. [rad]

**last\_caviphi0**  
float: Synchronous phase of the last rf cavity. [deg]

**beta**  
list of float: Lorentz  $\beta$  of the all charge states. [1]

**bg**  
list of float: Lorentz  $\beta\gamma$  of the all charge states. [1]

**gamma**  
list of float: Lorentz  $\gamma$  of the all charge states. [1]

**IonEk**  
list of float: Kinetic energy of the all charge states. [eV/u]

**IonEs**  
list of float: Nucleon mass of the all charge states. [eV/u]

**IonQ**  
list of float: Macro weight of the all charge states. [1]

**IonW**  
list of float: Total energy of the all charge states. [eV/u]

**IonZ**  
list of float: Charge to mass ratio of the all charge states. [1]

**phis**  
list of float: Absolute synchrotron phase of the all charge states. [rad]

**SampleIonK**  
list of float: Phase speed of the all charge states. [rad]

**moment0**  
Centroids of the all charge states.  
list of vector[7]:  $[x, x', y, y', \phi, E_k, 1]$  with [mm, rad, mm, rad, rad, MeV/u, 1].

**moment0\_env**  
Total centroid of the all charge states.  
vector[7]:  $[x, x', y, y', \phi, E_k, 1]$  with [mm, rad, mm, rad, rad, MeV/u, 1].

**moment0\_rms**

Total rms size (2nd order moments) of the all charge states.

**vector[7]:** rms of  $[x, x', y, y', \phi, E_k, 1]$  with [mm, rad, mm, rad, rad, MeV/u, 1].

**moment1**

Envelope matrixes of the all charge states.

**list of matrix[7,7]:**

Cartisan product of  $[x, x', y, y', \phi, E_k, 1]^2$  with [mm, rad, mm, rad, rad, MeV/u, 1]<sup>2</sup>.

**moment1\_env**

Total envelope matrix of the all charge states.

**matrix[7,7]:**

Cartisan product of  $[x, x', y, y', \phi, E_k, 1]^2$  with [mm, rad, mm, rad, rad, MeV/u, 1]<sup>2</sup>.



## INDICES

- `genindex`





## A

AccGap (C++ type), 20

## B

beta, 24

bg, 24

## D

drift (C++ type), 15

## E

EDipole (C++ type), 19

edipole (C++ type), 16

EFocus (C++ type), 19

EQuad (C++ type), 19

equad (C++ type), 16

## G

gamma, 24

## H

HFocus (C++ type), 20

HMono (C++ type), 19

HQuad (C++ type), 20

## I

IonEk, 24

IonEs, 24

IonQ, 24

IonW, 24

IonZ, 24

## L

last\_caviphi0, 24

## M

Machine (built-in class), 21

Machine.allocState() (built-in function), 21

Machine.conf() (built-in function), 21

Machine.find() (built-in function), 22

Machine.propagate() (built-in function), 22

Machine.reconfigure() (built-in function), 22

marker (C++ type), 13

moment0, 24

moment0\_env, 24

moment0\_rms, 24

moment1, 25

moment1\_env, 25

## O

orbtrim (C++ type), 14

## P

phis, 24

pos, 23

## Q

quadrupole (C++ type), 15

## R

ref\_beta, 23

ref\_bg, 23

ref\_gamma, 23

ref\_IonEk, 23

ref\_IonEs, 24

ref\_IonQ, 24

ref\_IonW, 24

ref\_IonZ, 24

ref\_phis, 24

ref\_SampleIonK, 24

rvcavity (C++ type), 17

## S

SampleIonK, 24

sbend (C++ type), 16

solenoid (C++ type), 15

source (C++ type), 13

State (built-in class), 23

State.clone() (built-in function), 23

stripper (C++ type), 13

## T

tmatrix (C++ type), 14