
Lisa Documentation

Release 0.01

Patrick Schreiber

Nov 10, 2017

CONTENTS

1	Availability of Classes	2
2	Simple Usage	3
2.1	SimplePlotter	3
2.2	MultiPlot	3
2.3	File	3
2.4	Data	4
2.5	PhaseSpace	4
2.6	MultiPhaseSpaceMovie	4
3	Inovesa Data Plots	5
4	Inovesa Result Files	10
5	Data with Units	13
6	Plot Configuration	14
7	Helpers	16
8	Create animated plots	17
9	Internal Utilities	18
9.1	How the version handling for attributes works:	18
	Python Module Index	20
	Index	21

Lisa is a Python module to easily plot and interact with Data from Inovesa result files (h5).

AVAILABILITY OF CLASSES

The following Classes are available at top Level (using Lisa.<CLS>):

- core.file.File
- core.file.MultiFile
- core.data.Data
- core.plots.plots.SimplePlotter
- core.plots.plots.MultiPlot
- core.plots.plots.PhaseSpace
- core.plots.plots.MultiPhaseSpaceMovie

Style is available through Lisa.plots.Style

SIMPLE USAGE

This is intended for someone who simply wants to plot something quickly for example.

2.1 SimplePlotter

Each Dataset in the Inovesa generated h5 files is a possible plot. Just call the corresponding function.

Plots for data with multiple axes expect a period as parameter. If this is omitted a meshplot is done. The period is used to generate a slice in the time axis.

```
import Lisa
sp = Lisa.SimplePlotter("/path/to/h5")
sp.bunch_profile(100, xunit='s')
plt.show()
sp.energy_spread()
plt.show()
```

2.2 MultiPlot

MultiPlot is the same as SimplePlotter except it works on multiple files.

```
import Lisa
mp = Lisa.MultiPlot()
mp.add_file("/path/to/h5")
mp.add_file("/path/to/second/h5")
...
mp.energy_spread()  # or any other plot supported by SimplePlotter
```

2.3 File

File is an object encapsulating the h5 file.

It has a method for each of the DataGroups in the h5 file.

Each method takes a parameter specifying the dataset to use. This parameter has to be an attribute from Lisa.Axis (e.g. Lisa.Axis.XAXIS for spaceaxis, Lisa.Axis.DATA for data etc.)

Inovesa Parameters are available via File.parameters. It will return an h5.Attribute instance if no parameter is specified.

For recurring access to data it is a speed improvement to call `File.preload_full("name_of_dataset")`. This will read all the data to memory for faster access.

If one does not specify an axis a `DataContainer` containing all the data there is in the requested `DataGroup` will be returned. This object is iterable, subscriptable and has a get method that accepts `Lisa.Axis` properties.

Usage:

```
import Lisa
file = Lisa.File("/path/to/h5")
bunch_profile_axis = file.bunch_profile(Lisa.Axis.XAXIS)
bunch_profile = file.bunch_profile(Lisa.Axis.DATA)
```

2.4 Data

Data is an object encapsulating a File object. The benefit of this is it converts data to the given unit.

```
import Lisa
data = Lisa.Data("/path/to/h5")
data.bunch_profile(Lisa.Axis.DATA, unit="c") # c for coulomb.
data.bunch_position(Lisa.Axis.XAXIS, unit="m") # m for meter
data.bunch_position(Lisa.Axis.XAXIS, unit="s") # s for meter
```

To get data as it is saved in the h5 file use `unit=None`.

It is possible to pass the unit as second parameter without a keyword. (For raw data (`unit=None`) this is not possible).

2.5 PhaseSpace

PhaseSpace is used to generate PhaseSpace plots or movies.

Use `PhaseSpace.plot_ps` to plot a phasespace or use `PhaseSpace.ps_movie` to generate a PhaseSpace Movie

2.6 MultiPhaseSpaceMovie

This is used to generate PhaseSpace movies from phase spaces in multiple files.

The method to generate the movie is `MultiPhaseSpaceMovie.create_movie`

INOVESA DATA PLOTS

Author Patrick Schreiber

class `plots.Deprecated` (*Exception*)

setup_plots():

Setup Plots to look native in Latex documents. Subject to change.

class `plots.SimplePlotter` (*object*)

A Simple Plot Helper Class.

It takes a Filename to the Constructor.

Each actual plotting function is decorated using `plot` or `meshPlot`. See those for additional parameters to each plot function.

__init__ (*self, filef, unit_connector='in'*)

Initialise a Simple Plotter instance

Parameters

- **filef** – The file name of this Plotter or a `Lisa.File` instance
- **unit_connector** – The string to use between label and unit.

plot (*func*)

Decorator to reuse plotting methods for different data. Calling one of the actual plot functions will result in calling this. This means the following options are available:

General Options (always use as keywords):

Parameters

- **fig** – (optional) the figure to plot in
- **ax** – (optional) the axis to use to plot
- **label** – (optional) the label for this plot (legend)
- **scale_factor** – (optional) a scale factor Note: This does not modify the labels
- **use_offset** – (optional) a bool if one wants an offset on yaxis or not
- **force_exponential_x** – (optional) a bool if one wants to force exponential notation on xaxis or not
- **force_exponential_y** – (optional) a bool if one wants to force exponential notation on yaxis or not
- **fft** – (optional) a bool if one wants to plot `fft(data)` instead of `data` or string for method to use in `numpy.fft`

- **fft_padding** – (optional) an integer to specify how much 0 will be padded to the data before fft default fft
- **abs** – (optional) a boolean to select if plot absolute values or direct data
- **plt_args** – (optional) dictionary with arguments to the displaying function
- **x_log** – (optional) a boolean to set the x axis to log scale
- **y_log** – (optional) a boolean to set the y axis to log scale

meshPlot (*func*)

Decorator to reuse plotting methods and to unify colormesh plots and normal line plots. Calling one of the actual plot functions will result in calling this. This means the following options are available:

General Options (always use as keywords):

Parameters

- **fig** – (optional) the figure to plot in
- **ax** – (optional) the axis to use to plot
- **label** – (optional) the label for this plot (legend) (if line plot)
- **norm** – (optional) the norm to use if pcolormesh (default LogNorm)
- **colormap** – (optional) the colormap for pcolormesh to use (default PuBu)
- **force_bad_to_min** – (optional) force bad values (e.g. negative or zero in LogNorm) of colorbar to minimum color of colorbar
- **force_exponential_x** – (optional) a bool if one wants to force exponential notation on xaxis or not
- **force_exponential_y** – (optional) a bool if one wants to force exponential notation on yaxis or not
- **plt_args** – (optional) dictionary with arguments to the displaying function
- **period** – (optional) the period to use. If not given will plot all data as pcolormesh (use parameters from plot)
- **use_index** – (optional) Use period as index in data and not synchrotron period (default False)
- **mean_range** – (optional) If given plot a normal plot but with data from mean of the given range (use parameters from plot)

energy_spread (*self*, ***kwargs*)

kwargs:

- xunit: possible values: “ts”, “seconds”, “raw”
- yunit: possible values: “eV”, “raw”

bunch_profile (*self*, **args*, ***kwargs*)

Plot the bunch_profile either as line plot or as pcolormesh

to plot as line pass either the period as first argument or a keyword argument period

Note: if yunit is passed period is in that unit, else in default value.

kwargs: (first value is default)

- xunit: possible values: “meters”, “seconds”, “raw”

- yunit: possible values: “ts”, “seconds”, “raw”
- zunit: possible values: “coulomb”, “ampere”, “raw”
- pad_zero: True or False. Pad data to zero to avoid white lines in plot (only considered if period is None or not given)

bunch_length (*self*, ****kwargs**)

kwargs:

- xunit: possible values: “ts”, “seconds”, “raw”
- yunit: possible values: “meters”, “secons”, “raw”

csr_intensity (*self*, ****kwargs**)

kwargs:

- xunit: possible values: “ts”, “seconds”, “raw”
- yunit: possible values: “watt”, “raw”

bunch_position (*self*, ****kwargs**)

kwargs: (first value is default)

- xunit: possible values: “ts”, “seconds”, “raw”
- yunit: possible values: “meters”, “seconds”, “raw”

bunch_population (*self*, ****kwargs**)

kwargs: (first value is default)

- xunit: possible values: “ts”, “seconds”, “raw”
- yunit: possible values: “meters”, “seconds”, “raw”

csr_spectrum (*self*, **args*, ****kwargs**)

Plot the csr_spectrum either as line plot or as pcolormesh
to plot as line pass either the period as first argument or a keyword argument period
Note: if yunit is passed period is in that unit, else in default value.

kwargs: (first value is default)

- xunit: possible values: “ts”, “seconds”, “raw”
- yunit: possible values: “hertz”, “raw”
- zunit: possible values: “watt”, “raw”

energy_profile (*self*, **args*, ****kwargs**)

Plot the energy_profile either as line plot or as pcolormesh
to plot as line pass either the period as first argument or a keyword argument period

kwargs: (first value is default)

- xunit: possible values: “eV”, “raw”
- yunit: possible values: “ts”, “seconds”, “raw”

- **zunit**: possible values: “coulomb”, “ampere”, “raw”

impedance (*self*, *args, **kwargs)

Plot Impedance (Fixed units). Real and Imaginary Part

class `plots.MultiPlot` (*object*)

Combine multiple files into one plot

__init__ (*self*)

Creates MultiPlot Instance

No Parameters (to add files use `add_file`)

clone (*self*)

Return a copy of this instance

add_file (*self*, *filename*, *label=None*)

Add a file to this instance

Parameters

- **filename** – Path to the file
- **label** – (optional) the label for plots with this filename

reset (*self*)

Reset this instance

possible_plots (*self*)

List all possible plots

class `plots.PhaseSpace` (*object*)

Plot PhaseSpaces of a single Inovesa result file

__init__ (*self*, *file*)

eax (*self*)

xax (*self*)

ps_data (*self*, *index*)

clone (*self*)

Return a copy of this instance

This effectively creates a new object with same file object

plot_ps (*self*, *index*)

Plot the phasespace.

Parameters **index** – the index of the dataset (in timeaxis)

center_of_mass (*self*, *xax*, *yax*)

phase_space_movie (*self*, *path*, *fr_idx=-1*, *to_idx=-1*, *fps=20*, *plot_area_width=None*)

Plot a movie of the evolving phasespace

Parameters

- **path** – Path to a movie file to save to
- **fr_idx** – Index in the phasespace data to start video
- **to_idx** – Index in the phasespace data to stop video

- **fps** – Frames per second of the output video
- **plot_area_width** – The width in pixel to plot around com. If None will plot full phasespace. (Will plot from com-plot_area_width/2 to com+plot_area_width/2 in space and energy)

Returns animation object

microstructure_movie (*self, path, fr_idx=-1, to_idx=-1, mean_range=(None, None), fps=20, plot_area_width=None*)

Plot the difference between the mean phasespace and the current snapshot as video.

Parameters

- **path** – Path to a movie file to save to
- **fr_idx** – Index in the phasespace data to start video
- **to_idx** – Index in the phasespace data to stop video
- **mean_range** – The min index and max index to use when calculating the mean of the phasespace
- **fps** – Frames per second of the output video
- **plot_area_width** – The width in pixel to plot around com. If None will plot full phasespace. (Will plot from com-plot_area_width/2 to com+plot_area_width/2 in space and energy)

Returns animation object

class `plots.MultiPhaseSpaceMovie` (*object*)

Create Phasespace of multiple Files

Useful to check the phasespace over multiple currenst (spectrogram)

__init__ (*self, path*)

create_movie (*self, filename, dpi=200, size_inches=(5.5, 5.5), fps=30, autoscale=False*)

Create a Movie of phasespaces

Parameters

- **filename** – The filename to use for the produces video file (if None, a moviepy video object will be returned)
- **dpi (=200)** – the dpi of the produces video
- **5.5) (size_inches (=5.5,)** – tuple used for size in inches of the video
- **fps (=30)** – the frames per second to use
- **autoscale (=False)** – if True will autoscale each frame (will not make sense if you want to compare)

Returns True if file produced, moviepy video instance if None as filename was given

INOVESA RESULT FILES

Author Patrick Schreiber

```
class file.AttributedNPArray (np.ndarray)
    Simple Wrapper around numpy.ndarray to include h5 attrs attribute

file.registered (cls)
    Registeres properties of one class into another class.registered_properties

class file.AxisSelector (object)

    __init__ (self, version)
    all_for (self, group)

class file.DataError (Exception)

class file.DataContainer (object)
```

```
    __init__ (self, data_dict, list_of_elements)
    get (self, item, default)
```

```
class file.File (object)
```

Wrapper around a h5 File created by Inovesa

Each Datagroup is a method of this object. To get a special axis of each group use additional parameters.

If one parameter is supplied the data will be returned as HDF5.Dataset object or, if preloaded, as Attributed-NPArray.

If no or more than one parameter is supplied the data will be returned as DataContainer object containing the HDF5.Dataset or AttributedNPArray objects.

This is not completely true for File.parameters.

```
f = File("path/to/file")
tax = f.bunch_profile(Axis.TIME)
xax = f.bunch_profile(Axis.XAXIS)
data = f.bunch_profile(Axis.DATA)
all = f.bunch_profile()
axes = f.bunch_profile(Axis.TIME, Axis.XAXIS)
```

Method names are mostly the datagroup-names with underscores instead of camelcase (e.g. bunch_profile for BunchProfile)

Available Groups:

- energy_spread

- bunch_length
- bunch_position
- bunch_population
- bunch_profile
- csr_intensity
- csr_spectrum
- energy_profile
- impedance
- particles
- phase_space
- wake_potential
- source_map (only Inovesa version 0.15 and above)
- – parameters
- parameters does not expose the exact same interface. Use File.parameters() to get the HDF5 Attribute Object.

Use File.parameters(param1, param2 ...) to get either a single parameter (with the datatype that this parameter is saved in the hdf5 file) or a DataContainer object depending on how much params were passed.

__init__ (*self, filename*)
Create File object

Parameters filename – The filename of the Inovesa result file

preload_full (*self, what, axis=None*)
Preload Data into memory. This will speed up recurring reads by a lot

class file.File_dep (*object*)
Wrapper around an h5 File created by Inovesa

__init__ (*self, filename*)
Create File object

Parameters filename – The filename of the Inovesa result file

preload_full (*self, what, axis=None*)
DEPRECATED Preload Data into memory. This will speed up recurring reads by a lot

energy_spread (*self, *selectors*)
Return the EnergySpread

bunch_length (*self, *selectors*)
Return the BunchLength

bunch_position (*self, *selectors*)
Return the BunchPosition

bunch_population (*self, *selectors*)
Return the BunchPopulation

bunch_profile (*self, *selectors*)
Return the BunchProfile

Returns [BunchProfile.axis0, BunchProfile.axis1, BunchProfile.data]

csr_intensity (*self*, **selectors*)
Return the CSRIntensity

csr_spectrum (*self*, **selectors*)
Return the CSRSpectrum

energy_profile (*self*, **selectors*)
Return the EnergyProfile

impedance (*self*, **selectors*)
Return the Impedance

particles (*self*, **selectors*)
Return the Particles

phase_space (*self*, **selectors*)
Return the PhaseSpace

wake_potential (*self*, **selectors*)
Return the WakePotential

parameters (*self*)
Return the Inovesa Parameters

Returns h5py.Attribute Instance

class file.**MultiFile** (*object*)
Multiple File container for whole directories

__init__ (*self*, *path*, *pattern=None*, *sorter=None*)

Parameters

- **path** – The path to search in
- **pattern** – The pattern to search for (pattern has to be accepted by glob) (None for no pattern)
- **sorter** – The sorting method to use. (None for default)

set_sorter (*self*, *sorter*)
Set the sorting method

Parameters **sorter** – A callable that accepts a list of files as strings and returns a sorted list of strings

strlst (*self*, *sorted=True*)
Get File list as string list

Parameters **sorted** – True to sort the list

objlst (*self*, *sorted=True*)
Get File list as Lisa.File object list

Parameters **sorted** – True to sort the list

DATA WITH UNITS

Author Patrick Schreiber

class `data.Data` (*object*)

Get Data from Inovesa Result Datafiles with specified unit.

Use the same attributes that are available for `Lisa.File` objects.

Usage:

```
d = Lisa.Data("/path/to/file")
d.bunch_profile(Lisa.Axis.DATA, unit="c") # unit="c" for coulomb
d.bunch_profile(Lisa.Axis.XAXIS, unit="s") # unit="s" for second XAXIS for space_
↪axis
```

See also:

`Lisa.File`

`__init__` (*self*, *p_file*)

Parameters *p_file* – either a filename of a `Lisa.File` object

`__getattr__` (*self*, *attr*)

Convert to correct unit.

Parameters

- **idx** – An axis specification from `Lisa.Axis`
- **unit** (*string*) – Use this as second argument or kwarg
- **sub_idx** – (kwarg) the index in the h5object (if `File` returns [`Dataset1`, `Dataset2`] then `idx=0` and `sub_idx` is given will result in `Dataset1[sub_idx]`) This will speed up if only part of the data is used.

PLOT CONFIGURATION

Author Patrick Schreiber

class `config.Alias`

Alias to another method

class `config.StyleError` (*Exception*)

class `config.Style` (*dict*)

Style for Matplotlib axes. It is essentially a dictionary with overridden methods.

Usage (assume ax is an matplotlib axes object):

```
style = Style()
style.apply_to_ax(ax)
style.update({selector: value}) # or use style.update_style
style['selector']=value # this also updates the axes object
# if you have done some modifications to the plot yourself and want to update the
↪ style again use
style.reapply()
```

Note: instead of working with the Style object directly you can use `ax.current_style` after you applied a style

Note: You can apply one Style object to more than one axes object. Just call `apply_to_ax` multiple times or with a list of axes. If you applied one Style to multiple axes objects and you use `axes.current_style` instead of the style object directly the updates/changes will nonetheless be applied to all applied axes objects

apply_to_ax (*self*, *ax*)

Apply this style to the given axes object

You can apply to multiple axes objects by calling this multiple times or with a list of objects

Parameters **ax** – matplotlib axes object or list of those

apply_to_fig (*self*, *fig*)

Apply this style to the given figure and its axes (and new axes)

You can apply to multiple figures by calling this multiple times or with a list of figures

Parameters **fig** – matplotlib figure or list of those

update (*self*, *E=None*, ***F*)

Update the value in the internal dictionary and update the axes object

Parameters **E** – dictionary with key/value pairs to update or name of a predefined style

update_style (*self*, *E*)

more explicit alias to update

reapply (*self*)

reapply this style to the axes object

selectors (*self*, *print_out=True*)

Print or Get a list of possible selectors

HELPERS

Author Patrick Schreiber

`helpers.display_video` (*mov*)

Display Video certain video objects in Jupyter Notebooks

CREATE ANIMATED PLOTS

`animation.create_animation` (*figure, frame_generator, frames, fps=None, bitrate=18000, dpi=100, path=None, blit=False, clear_between=False, save_args=None*)

Create an animation.

Parameters

- **figure** – The figure to use.
- **frame_generator** – Callable that generates a new frame on figure!
- **frames** – List with index for each frame
- **fps** – The frames per seconds.
- **bitrate** – Bitrate of video
- **dpi** – Dpi of video
- **path** – Path to save the video to. If None does not save.

`animation.data_frame_generator` (*fig, xdata, ydata, label_only_once=False*)

Generate a frame generator for simple data.

Parameters

- **fig** – The figure to draw in
- **xdata** – The xdata as iterable
- **ydata** – The ydata as iterable
- **label_only_once** – True to only show x and y tick labels only once (otherwise will be drawn over each other if axis present (not so nice))

INTERNAL UTILITIES

Author Patrick Schreiber

9.1 How the version handling for attributes works:

InovesaVersion is a object that implements comparators for inovesa versions

unit_to_attr_map is a dictionary that maps physical units to inovesa attributes. Since these attributes change over different inovesa versions first a dictionary for “older” versions (13 and up, maybe some versions below match too) then it is modified into *unit_to_attr_map_v15* for versions 15 and up (until the next change in inovesa is made).

attr_to_unit_map and *attr_to_unit_map_v15* are just reversed dictionaries to make it possible to find the corresponding readable/printable unit for an attribute

Then *alias_map* is generated that makes it possible to not only use the exact specification used in *unit_to_attr_map* but also the ones specified in *alias_map*. The reason for this is that the user does not have to remember the exact unit string.

Then a *complete_unit_map* (and *complete_unit_map_v15*) is generated where all the entries in *alias_map* are mapped to attributes

There are 3 ‘public’ methods that should be used.

- *attr_from_unit*: Get the h5 attribute from the given unit
- *unit_from_attr*: Get the readable/printable unit from a h5 attribute
- *unit_from_spec*: Get the readable/printable unit from a unit specifier (an alias)

class `utils.UnitError` (*Exception*)

class `utils.InovesaVersion` (*object*)
Object to make comparison of versions easier

`utils.attr_from_unit` (*unit, version*)
Get the h5 attribute from a unit specification

Parameters

- **unit** – specification to get the attribute for
- **version** – inovesa version as InovesaVersion Object

Returns a string with the h5 attribute for the given unit

Raises `UnitError` – when a unit is not known

`utils.unit_from_attr` (*attr, version*)
Get a printable/readable unit from an h5 attribute

Parameters

- **attr** – h5 attribute to get the unit for
- **version** – inovesa version as InovesaVersion Object

Returns a string with the requested unit

Raises *UnitError* – when an attribute is not known

`utils.unit_from_spec(spec)`

Get a printable/readable unit from a unit specification

Parameters **spec** – the unit specification

Returns a string with the requested unit

Raises *UnitError* – when the specification is not known

`utils.color_map_inferno()`

PYTHON MODULE INDEX

a

animation, [16](#)

c

config, [13](#)

d

data, [12](#)

f

file, [9](#)

h

helpers, [15](#)

p

plots, [4](#)

u

utils, [17](#)

Symbols

[__getattr__\(\)](#) (data.Data method), 13
[__init__\(\)](#) (data.Data method), 13
[__init__\(\)](#) (file.AxisSelector method), 10
[__init__\(\)](#) (file.DataContainer method), 10
[__init__\(\)](#) (file.File method), 11
[__init__\(\)](#) (file.File_dep method), 11
[__init__\(\)](#) (file.MultiFile method), 12
[__init__\(\)](#) (plots.MultiPhaseSpaceMovie method), 9
[__init__\(\)](#) (plots.MultiPlot method), 8
[__init__\(\)](#) (plots.PhaseSpace method), 8
[__init__\(\)](#) (plots.SimplePlotter method), 5

A

[add_file\(\)](#) (plots.MultiPlot method), 8
[Alias](#) (class in config), 14
[all_for\(\)](#) (file.AxisSelector method), 10
[animation](#) (module), 16
[apply_to_ax\(\)](#) (config.Style method), 14
[apply_to_fig\(\)](#) (config.Style method), 14
[attr_from_unit\(\)](#) (in module utils), 18
[AttributedNPArray](#) (class in file), 10
[AxisSelector](#) (class in file), 10

B

[bunch_length\(\)](#) (file.File_dep method), 11
[bunch_length\(\)](#) (plots.SimplePlotter method), 7
[bunch_population\(\)](#) (file.File_dep method), 11
[bunch_population\(\)](#) (plots.SimplePlotter method), 7
[bunch_position\(\)](#) (file.File_dep method), 11
[bunch_position\(\)](#) (plots.SimplePlotter method), 7
[bunch_profile\(\)](#) (file.File_dep method), 11
[bunch_profile\(\)](#) (plots.SimplePlotter method), 6

C

[center_of_mass\(\)](#) (plots.PhaseSpace method), 8
[clone\(\)](#) (plots.MultiPlot method), 8
[clone\(\)](#) (plots.PhaseSpace method), 8
[color_map_inferno\(\)](#) (in module utils), 19
[config](#) (module), 13
[create_animation\(\)](#) (in module animation), 17
[create_movie\(\)](#) (plots.MultiPhaseSpaceMovie method), 9

[csr_intensity\(\)](#) (file.File_dep method), 12
[csr_intensity\(\)](#) (plots.SimplePlotter method), 7
[csr_spectrum\(\)](#) (file.File_dep method), 12
[csr_spectrum\(\)](#) (plots.SimplePlotter method), 7

D

[Data](#) (class in data), 13
[data](#) (module), 12
[data_frame_generator\(\)](#) (in module animation), 17
[DataContainer](#) (class in file), 10
[DataError](#) (class in file), 10
[Deprecated](#) (class in plots), 5
[display_video\(\)](#) (in module helpers), 16

E

[eax\(\)](#) (plots.PhaseSpace method), 8
[energy_profile\(\)](#) (file.File_dep method), 12
[energy_profile\(\)](#) (plots.SimplePlotter method), 7
[energy_spread\(\)](#) (file.File_dep method), 11
[energy_spread\(\)](#) (plots.SimplePlotter method), 6

F

[File](#) (class in file), 10
[file](#) (module), 9
[File_dep](#) (class in file), 11

G

[get\(\)](#) (file.DataContainer method), 10

H

[helpers](#) (module), 15

I

[impedance\(\)](#) (file.File_dep method), 12
[impedance\(\)](#) (plots.SimplePlotter method), 8
[InovesaVersion](#) (class in utils), 18

M

[meshPlot\(\)](#) (plots.SimplePlotter method), 6
[microstructure_movie\(\)](#) (plots.PhaseSpace method), 9
[MultiFile](#) (class in file), 12

MultiPhaseSpaceMovie (class in plots), 9
 MultiPlot (class in plots), 8

O

objlst() (file.MultiFile method), 12

P

parameters() (file.File_dep method), 12
 particles() (file.File_dep method), 12
 phase_space() (file.File_dep method), 12
 phase_space_movie() (plots.PhaseSpace method), 8
 PhaseSpace (class in plots), 8
 plot() (plots.SimplePlotter method), 5
 plot_ps() (plots.PhaseSpace method), 8
 plots (module), 4
 possible_plots() (plots.MultiPlot method), 8
 preload_full() (file.File method), 11
 preload_full() (file.File_dep method), 11
 ps_data() (plots.PhaseSpace method), 8

R

reapply() (config.Style method), 15
 registered() (in module file), 10
 reset() (plots.MultiPlot method), 8

S

selectors() (config.Style method), 15
 set_sorter() (file.MultiFile method), 12
 SimplePlotter (class in plots), 5
 strlst() (file.MultiFile method), 12
 Style (class in config), 14
 StyleError (class in config), 14

U

unit_from_attr() (in module utils), 18
 unit_from_spec() (in module utils), 19
 UnitError (class in utils), 18
 update() (config.Style method), 14
 update_style() (config.Style method), 15
 utils (module), 17

W

wake_potential() (file.File_dep method), 12

X

xax() (plots.PhaseSpace method), 8