

---

# **Masci-tools Documentation**

***Release 0.6.2***

**The JuDFT team**

**Oct 10, 2021**



# CONTENTS

<b>1</b>	<b>Requirements to use this code:</b>	<b>3</b>
<b>2</b>	<b>Installation Instructions:</b>	<b>5</b>
<b>3</b>	<b>Acknowledgments:</b>	<b>7</b>
<b>4</b>	<b>User's Guide</b>	<b>9</b>
4.1	User guide . . . . .	9
4.1.1	Using the Fleur input/output parsers . . . . .	9
4.1.1.1	Parser for the Fleur inp.xml file . . . . .	9
4.1.1.2	Parser for the Fleur out.xml file . . . . .	10
4.1.1.3	XML getter functions . . . . .	11
4.1.1.4	Using the schema_dict_util functions . . . . .	11
4.1.2	FleurXMLModifier . . . . .	12
4.1.2.1	Description . . . . .	12
4.1.2.2	Usage . . . . .	12
4.1.2.3	User Methods . . . . .	12
4.1.2.4	Modifying the density matrix for LDA+U calculations . . . . .	14
4.1.3	General HDF5 file reader . . . . .	15
4.1.3.1	Basic Usage . . . . .	15
4.1.3.2	Structure of recipes for the HDF5Reader . . . . .	16
4.1.4	Plotting Fleur DOS/bandstructures . . . . .	21
4.1.4.1	Bandstructures . . . . .	21
4.1.4.2	Density of States . . . . .	25
4.1.5	General Plotting routines . . . . .	30
4.1.5.1	Available Routines . . . . .	30
4.1.5.2	Providing Data . . . . .	32
4.1.5.3	Customizing Plots . . . . .	32
<b>5</b>	<b>Developer's Guide</b>	<b>43</b>
5.1	Developers Guide . . . . .	43
5.1.1	Updating or adapting the Fleur Parsers . . . . .	43
5.1.1.1	Adding/modifying a Fleur Schema: . . . . .	43
5.1.1.2	Adapting the outxml_parser: . . . . .	43
5.1.1.3	Migrating the parsing tasks . . . . .	45
5.1.2	Using the Plotter class . . . . .	45
5.1.2.1	Description . . . . .	45
5.1.2.2	Writing a plotting function . . . . .	46
5.1.3	Using the PlotData class . . . . .	54
5.1.3.1	Description . . . . .	54

5.1.3.2	Initializing PlotData without a mapping . . . . .	55
5.1.3.3	Available routines on PlotData . . . . .	55
<b>6</b>	<b>Module reference (API)</b>	<b>57</b>
6.1	Source code Documentation (API reference) . . . . .	57
6.1.1	Visualisation and Plotting . . . . .	57
6.1.1.1	Fleur specific Plotting . . . . .	57
6.1.1.2	KKR specific Plotting . . . . .	59
6.1.1.3	General Plotting . . . . .	60
6.1.2	Calculation tools . . . . .	103
6.1.3	IO helper functions and file parsers . . . . .	110
6.1.3.1	KKR related IO . . . . .	110
6.1.3.2	Fleur related IO . . . . .	114
6.1.3.3	General HDF5 parser . . . . .	132
6.1.3.4	Definition of default parsing tasks for fleur out.xml . . . . .	140
6.1.4	Commandline interface (CLI) . . . . .	153
6.1.4.1	masci_tools . . . . .	153
6.1.5	Utility Functions/Classes . . . . .	161
6.1.5.1	Custom Datatypes . . . . .	161
6.1.5.2	Common XML utility . . . . .	164
6.1.5.3	XML Setter functions . . . . .	169
6.1.5.4	XML Getter functions . . . . .	190
6.1.5.5	Basic IO helper functions . . . . .	195
6.1.5.6	Logging Utility . . . . .	198
6.1.5.7	Fleur parser utility . . . . .	199
6.1.6	Basic Fleur Schema parser functions . . . . .	212
6.1.7	Defined constants . . . . .	220
<b>7</b>	<b>Indices and tables</b>	<b>249</b>
	<b>Python Module Index</b>	<b>251</b>
	<b>Index</b>	<b>253</b>

This package was developed in the process of developing the [AiiDA-FLEUR](#) and [AiiDA\\_KKR](#) plugins to [AiiDA](#). It contains helper functions that can help with common pre- and postprocessing steps of the [FLEUR](#) and [KKR](#) codes developed at the Forschungszentrum Jülich (see also the [juDFT](#) website for more information).

If you use this package please cite: ...



## REQUIREMENTS TO USE THIS CODE:

- lxml
- h5py
- ase
- pymatgen
- numpy
- scipy
- more\_itertools





## **INSTALLATION INSTRUCTIONS:**

Install from pypi the latest release:

```
$ pip install masci-tools
```

or from the masci-tools source folder any branch:

```
$ pip install .  
# or which is very useful to keep track of the changes (developers)  
$ pip install -e .
```



## ACKNOWLEDGMENTS:

We acknowledge partial support from the EU Centre of Excellence “MaX – Materials Design at the Exascale” (<http://www.max-centre.eu>). (Horizon 2020 EINFRA-5, Grant No. 676598) We thank the AiiDA team for their help and work. Also the vial exchange with developers of AiiDA packages for other codes was inspiring.



## USER'S GUIDE

### 4.1 User guide

This is the maschi-tools user's guide.

#### 4.1.1 Using the Fleur input/output parsers

##### Contents

- *Using the Fleur input/output parsers*
  - *Parser for the Fleur inp.xml file*
  - *Parser for the Fleur out.xml file*
  - *XML getter functions*
  - *Using the schema\_dict\_util functions*

##### 4.1.1.1 Parser for the Fleur inp.xml file

The fleur `inp.xml` contains all the information about the setup of a fleur calculation. To use this information in external scripts or aiida-fleur, the information needs to be parsed from the `.xml` format somehow.

For this purpose the `inpxml_parser()` is implemented. The usage is shown below. The input file is parsed recursively and all information is put into the dictionary.

```
from maschi_tools.io.parsers.fleur import inpxml_parser

input_dict = inpxml_parser('/path/to/random/inp.xml')

#The call below will output warnings about failed conversions in the warnings dictionary
warnings = {'parser_warnings': []}
input_dict = inpxml_parser('/path/to/random/inp.xml', parser_info_out=warnings)
```

The conversion of each attribute or text is done according to the FleurInputSchema for the same version, which is stored in this repository for versions from `0.27` to `0.34`. The following table shows the version compatibility of the input parser.

File version	Compatible?
0.27 - 0.34	
0.35 -	

#### 4.1.1.2 Parser for the Fleur out.xml file

For the `out.xml` file a similar parser is implemented. However, since the output file contains a lot more information, which is not always useful the `outxml_parser()` is defined a lot more selectively. But the usage is almost completely identical to the input file.

```
from maschi_tools.io.parsers.fleur import outxml_parser

#The default is that only the last stable iteration is parsed
output_dict = outxml_parser('/path/to/random/out.xml')

#Here all iterations are parsed
output_dict = outxml_parser('/path/to/random/out.xml', iteration_to_parse='all')

#Or the 5.
output_dict = outxml_parser('/path/to/random/out.xml', iteration_to_parse=5)

#The call below will output warnings about failed conversions in the warnings dictionary
warnings = {'parser_warnings': []}
output_dict = outxml_parser('/path/to/random/out.xml', parser_info_out=warnings)
```

For each iteration the parser decides based on the type of fleur calculation, what things should be parsed. For a more detailed explanation refer to the [Developers Guide](#).

The following table shows the version compatibility of the output parser. For versions before 0.34 the file version corresponds to the input version, since the output version is 0.27 for all versions before this point.

File version	Compatible?
0.27 - 0.29	
0.30 - 0.31	
0.32	
0.33	
0.34	
0.35 -	

#### 4.1.1.3 XML getter functions

There are a number of functions for extracting specific parts of the XML files in the `xml_getters` module. The following are available:

- `get_fleur_modes()`: Get information about the mode of the fleur calculation
- `get_nkpts()`: Get the (for older versions approximate if not `kPointList` is used) number of kpoints to be used in the calculation
- `get_cell()`: Get the Bravais matrix of the system
- `get_parameter_data()`: Get the information about the calculation parameters needed to reproduce a calculation starting from the inpgen
- `get_structure_data()`: Get the structure from the xml file (atom positions + unit cell)
- `get_kpoints_data()`: Get the defined kpoint sets (single/multiple) from the xml file (kpoints + weights + unit cell)
- `get_relaxation_information()`: Get the relaxation history and current displacements
- `get_symmetry_information()`: Get the symmetry operations used in the calculation

All of these are used in the same way:

```
from masci_tools.io.io_fleurxml import load_inpxml
from masci_tools.util.xml.xml_getters import get_fleur_modes

xmltree, schema_dict = load_inpxml('/path/to/inp.xml')

fleur_modes = get_fleur_modes(xmltree, schema_dict)
print(fleur_modes)
```

#### 4.1.1.4 Using the schema\_dict\_util functions

If only a small amount of information is required from the input or output files of fleur the full parsers might be overkill. But there are a number of utility functions allowing easy access to information from the `.xml` files without knowing the exact xpath expressions for each version of the input/output. A code example extracting information from a input file is given below.

```
from masci_tools.io.io_fleurxml import load_inpxml
from masci_tools.util.schema_dict_util import evaluate_attribute, eval_simple_xpath

#First we create a xml-tree from the input file and load the desired input schema_
↳ dictionary
xmltree, schema_dict = load_inpxml('/path/to/inp.xml')
root = xmltree.getroot()

#Here an example of extracting some attributes. The interface to all functions in
#schema_dict_util is the same

#Number of spins
spins = evaluate_attribute(root, schema_dict, 'jspins')

#Planewave cutoff (notice the names are case-insensitive, 'KMAX' would work as well)
kmax = evaluate_attribute(root, schema_dict, 'kmax')
```

(continues on next page)

(continued from previous page)

```

#Some attributes need to be specified further for a distinct path
#`radius` exists both for atom species and atom groups so we give a phrase to distinguish
↳ them
mt_radii = evaluate_attribute(root, schema_dict, 'radius', contains='species')

#But we can also make implicit constraints
# 1. Get some element in the xml tree, where the path is more specified. In the example
↳ lets
#   get the element containing all species
# 2. If we evaluate the `radius` attribute now on the species elements, we do not need
#   the contains parameter, since from the point of the species element there is only
↳ one possibility
#   for the `radius` attribute

species = eval_simple_xpath(root, schema_dict, 'atomSpecies')
mt_radii = evaluate_attribute(species, schema_dict, 'radius')

```

## 4.1.2 FleurXMLModifier

### 4.1.2.1 Description

The *FleurXMLModifier* class can be used if you want to change anything in a *inp.xml* file in an easy and robust way. It will validate all the changes you wish to do and apply all these changes to a given *inp.xml* and produce a new xmltree.

### 4.1.2.2 Usage

To modify an existing *inp.xml*, a *FleurXMLModifier* instance has to be initialised. After that, a user should register certain modifications which will be cached. They will be applied on a given *inp.xml*. However, the provided *inp.xml* will not be changed but only a modified xmltree is returned, which you can store in a new *.xml* file.

```

from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()                                # Initialise
↳ FleurXMLModifier class
fm.set_inpchanges({'dos' : True, 'Kmax': 3.9 })         # Add changes
new_xmltree = fm.modify_xmlfile('/path/to/original/inp.xml') #Apply

```

### 4.1.2.3 User Methods

#### General methods

- *modify\_xmlfile()*: Applies the registered changes to a given *inp.xml* (and optional *n\_mmp\_mat* file)
- *changes()*: Displays the current list of changes.
- *undo()*: Removes the last task or all tasks from the list of changes.



## Modification registration methods

The registration methods can be separated into two groups. First of all, there are XML methods that require deeper knowledge about the structure of an `inp.xml` file. All of them require an xpath input and start their method names start with `xml_`:

- `xml_set_attr_value_no_create()`: Set attributes on the result(s) of the given xpath
- `xml_set_text_no_create()`: Set text on the result(s) of the given xpath
- `xml_create_tag()`: Insert an xml element in the xml tree on the result(s) of the given xpath.
- `xml_delete_tag()`: Delete an xml element in the xml tree on the result(s) of the given xpath.
- `xml_delete_att()`: Delete an attribute in the xml tree on the result(s) of the given xpath.
- `xml_replace_tag()`: Replace an xml element on the result(s) of the given xpath.

On the other hand, there are shortcut methods that already know some paths:

- `set_species()`: Specific user-friendly method to change species parameters.
- `set_atomgroup()`: Specific method to change atom group parameters.
- `set_species_label()`: Specific user-friendly method to change a species of an atom with a certain label.
- `set_atomgroup_label()`: Specific method to change atom group parameters of an atom with a certain label.
- `set_nkpts()`: user-friendly method for setting the *kPointCount* (**Only for MaX4 and older**)
- `set_kpath()`: user-friendly method for setting the path for a bandstructure calculations (**Only for MaX4 and older**)
- `set_kpointlist()`: user-friendly method for setting/creating a *kPointlist* from lists
- `switch_kpointset()`: user-friendly method for switching the used kpoint set in a calculation (**Only for MaX5 and newer**)
- `set_inpchanges()`: Specific user-friendly method for easy changes of attribute key value type.
- `shift_value()`: Specific user-friendly method to shift value of an attribute.
- `shift_value_species_label()`: Specific user-friendly method to shift value of an attribute of an atom with a certain label.
- `set_attr_value()`: user-friendly method for setting attributes in the xml file by specifying their name
- `set_first_attr_value()`: user-friendly method for setting the first occurrence of an attribute in the xml file by specifying its name
- `add_number_to_attr()`: user-friendly method for adding to or multiplying values of attributes in the xml file by specifying their name
- `add_number_to_first_attr()`: user-friendly method for adding to or multiplying values of the first occurrence of the attribute in the xml file by specifying their name
- `set_text()`: user-friendly method for setting text on xml elements in the xml file by specifying their name
- `set_first_text()`: user-friendly method for setting the text on the first occurrence of an xml element in the xml file by specifying its name
- `set_simple_tag()`: user-friendly method for creating and setting attributes on simple xml elements (only attributes) in the xml file by specifying its name
- `set_complex_tag()`: user-friendly method for creating complex tags in the xml file by specifying its name
- `create_tag()`: User-friendly method for inserting a tag in the right place by specifying it's name

- `delete_tag()`: User-friendly method for delete a tag by specifying it's name
- `delete_att()`: User-friendly method for deleting an attribute from a tag by specifying it's name
- `replace_tag()`: User-friendly method for replacing a tag by another by specifying its name
- `set_nmmpmat()`: Specific method for initializing or modifying the density matrix file for a LDA+U calculation (details see below)
- `rotate_nmmpmat()`: Specific method for rotating a block of the density matrix file for a LDA+U calculation (details see below) in real space

#### 4.1.2.4 Modifying the density matrix for LDA+U calculations

The above mentioned `set_nmmpmat()` and `rotate_nmmpmat()` take a special role in the modification registration methods, as the modifications are not done on the `inp.xml` file but the density matrix file `n_mmp_mat` used by Fleur for LDA+U calculations. The resulting new `n_mmp_mat` file is returned next to the new `inp.xml` by the `modify_xmlfile()`.

The code example below shows how to use this method to add a LDA+U procedure to an atom species and provide an initial guess for the density matrix.

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()                                # Initialise
↳FleurXMLModifier class
fm.set_species('Nd-1', {'ldaU':                        # Add LDA+U
↳procedure
                        {'l': 3, 'U': 6.76, 'J': 0.76, 'l_amf': 'F'}})
fm.set_nmmpmat('Nd-1', orbital=3, spin=1, occStates=[1,1,1,1,0,0,0]) # Initialize n_mmp_
↳mat file with the states
                                                    # m = -3 to m = 0
↳occupied for spin up
                                                    # spin down is
↳initialized with 0 by default
new_xmltree, nmmp_content = fm.modify_xmlfile('/path/to/original/inp.xml') #
↳Apply
```

---

**Note:** The `n_mmp_mat` file is a simple text file with no knowledge of which density matrix block corresponds to which LDA+U procedure. They are read in the same order as they appear in the `inp.xml`. For this reason the `n_mmp_mat` file can become invalid if one adds/removes a LDA+U procedure to the `inp.xml` after the `n_mmp_mat` file was initialized. Therefore any modifications to the `n_mmp_mat` file should be done after adding/removing or modifying the LDA+U configuration.

---

### 4.1.3 General HDF5 file reader

Fleur uses the HDF5 library for output files containing large datasets. The maschi-tools library provides the [HDF5Reader](#) class to extract and transform information from these files. The h5py library is used to get information from .hdf files

#### 4.1.3.1 Basic Usage

The specifications of what to extract and how to transform the data are given in the form of a python dictionary. Let us look at a usage example; extracting data for a bandstructure calculation from the `banddos.hdf` file produced by Fleur.

```
from maschi_tools.io.parsers.hdf5 import HDF5Reader
from maschi_tools.io.parsers.hdf5.recipes import FleurBands

#The HDF5Reader is used with a contextmanager to safely handle
#opening/closing the h5py.File object that is produced to extract information
with HDF5Reader('/path/to/banddos.hdf') as h5reader:
    datasets, attributes = h5reader.read(recipe=FleurBands)
```

The method `read()` produces two python dictionaries. In the case of the `FleurBands` recipe these contain the following information.

- **datasets**
  - Eigenvalues converted to eV shifted to  $E_F=0$  (if available in the `banddos.hdf`) and split up into spin-up/down and flattened to one dimension
  - The kpath projected to 1D and reshaped to same length as weights/eigenvalues
  - The weights (flattened) of the interstitial region, each atom, each orbital on each atom for all eigenvalues
- **attributes**
  - The coordinates of the used kpoints
  - Positions, atomic symbols and indices of symmetry equivalent atoms
  - Dimensions of eigenvalues (`nkpts` and `nbands`)
  - Bravais matrix/Reciprocal cell of the system
  - Indices and labels of special k-points
  - Fermi energy
  - Number of spins in the calculation

The following pre-defined recipes are stored in [recipes](#):

- Recipe for `banddos.hdf` for bandstructure calculations
- Recipe for `banddos.hdf` for standard density of states calculations
- Different DOS modes are also supported (`jdOS`, `orbcomp`, `mcd`)

If no recipe is provided to the [HDF5Reader](#), it will create the `datasets` and `attributes` as two nested dictionaries, exactly mirroring the structure of the .hdf file and converting datasets into numpy arrays.

For big datasets it might be useful to keep the dataset as a reference to the file and not load the dataset into memory. To achieve this you can pass `move_to_memory=False`, when initializing the reader. Notice that most of the transformations will still implicitly create numpy arrays and after the hdf file is closed the datasets will no longer be available.

### 4.1.3.2 Structure of recipes for the HDF5Reader

The recipe for extracting bandstructure information from the `banddos.hdf` looks like this:

```

1  def bands_recipe_format(group, simple=False):
2      """
3      Format for bandstructure calculations retrieving weights from the given group
4
5      :param group: str of the group the weights should be taken from
6      :param simple: bool, if True no additional weights are retrieved with the produced
7      ↪ recipe
8
9      :returns: dict of the recipe to retrieve a bandstructure calculation
10     """
11
12     if group == 'Local':
13         atom_prefix = 'MT:'
14     elif group == 'jDOS':
15         atom_prefix = 'jDOS:'
16     elif group == 'Orbcomp':
17         atom_prefix = 'ORB:'
18     elif group == 'MCD':
19         atom_prefix = 'At'
20     else:
21         raise ValueError(f'Unknown group: {group}')
22
23     recipe = {
24         'datasets': {
25             'eigenvalues': {
26                 'h5path':
27                 f'/{group}/BS/eigenvalues',
28                 'transforms': [
29                     AttribTransformation(name='shift_by_attribute',
30                                         attrib_name='fermi_energy',
31                                         args=(),
32                                         kwargs={
33                                             'negative': True,
34                                         }),
35                     Transformation(name='multiply_scalar', args=(HTR_TO_EV,), kwargs={}),
36                     Transformation(name='split_array',
37                                   args=(),
38                                   kwargs={
39                                       'suffixes': ['up', 'down'],
40                                       'name': 'eigenvalues'
41                                   }),
42                     Transformation(name='flatten_array', args=(), kwargs={})
43                 ],
44             },
45             'unpack_dict':
46             True
47         },
48         'kpath': {
49             'h5path':
50             '/kpts/coordinates',
51         },
52     }

```

(continues on next page)

(continued from previous page)

```

49         'transforms': [
50             AttribTransformation(name='multiply_by_attribute',
51                                 attrib_name='reciprocal_cell',
52                                 args=(),
53                                 kwargs={'transpose': True}),
54             Transformation(name='calculate_norm', args=(), kwargs={'between_
↪neighbours': True}),
55             Transformation(name='cumulative_sum', args=(), kwargs={})),
56             AttribTransformation(name='repeat_array_by_attribute', attrib_name=
↪'nbands', args=(), kwargs={})),
57         ]
58     },
59     'attributes': {
60         'group_name': {
61             'h5path': f'/{group}',
62             'transforms': [
63                 Transformation(name='get_name', args=(), kwargs={})),
64             ],
65         },
66         'kpoints': {
67             'h5path': '/kpts/coordinates',
68         },
69         'nkpts': {
70             'h5path':
71             '/Local/BS/eigenvalues',
72             'transforms': [
73                 Transformation(name='get_shape', args=(), kwargs={})),
74                 Transformation(name='index_dataset', args=(1,), kwargs={})),
75             ],
76         },
77         'nbands': {
78             'h5path':
79             '/Local/BS/eigenvalues',
80             'transforms': [
81                 Transformation(name='get_shape', args=(), kwargs={})),
82                 Transformation(name='index_dataset', args=(2,), kwargs={})),
83             ],
84         },
85         'atoms_elements': {
86             'h5path': '/atoms/atomicNumbers',
87             'description': 'Atomic numbers',
88             'transforms': [Transformation(name='periodic_elements', args=(), kwargs=
↪{})]
89         },
90     },
91     'n_types': {
92         'h5path':
93         '/atoms',
94         'description':
95         'Number of atom types',
96         'transforms': [
97             Transformation(name='get_attribute', args=('nTypes',), kwargs={})),

```

(continues on next page)

(continued from previous page)

```

98         Transformation(name='get_first_element', args=(), kwargs={})
99     ]
100 },
101 'atoms_position': {
102     'h5path': '/atoms/positions',
103     'description': 'Atom coordinates per atom',
104 },
105 'atoms_groups': {
106     'h5path': '/atoms/equivAtomsGroup'
107 },
108 'reciprocal_cell': {
109     'h5path': '/cell/reciprocalCell'
110 },
111 'bravais_matrix': {
112     'h5path': '/cell/bravaisMatrix',
113     'description': 'Coordinate transformation internal to physical for atoms
↪ ',
114     'transforms': [Transformation(name='multiply_scalar', args=(BOHR_A,), ↪
↪ kwargs={})]
115 },
116 'special_kpoint_indices': {
117     'h5path': '/kpts/specialPointIndices',
118     'transforms': [Transformation(name='shift_dataset', args=(-1,), kwargs={})
↪ ]
119 },
120 'special_kpoint_labels': {
121     'h5path': '/kpts/specialPointLabels',
122     'transforms': [Transformation(name='convert_to_str', args=(), kwargs={})]
123 },
124 'fermi_energy': {
125     'h5path':
126     '/general',
127     'description':
128     'fermi_energy of the system',
129     'transforms': [
130         Transformation(name='get_attribute', args=('lastFermiEnergy',), ↪
↪ kwargs={}),
131         Transformation(name='get_first_element', args=(), kwargs={})
132     ]
133 },
134 'spins': {
135     'h5path':
136     '/general',
137     'description':
138     'number of distinct spin directions in the system',
139     'transforms': [
140         Transformation(name='get_attribute', args=('spins',), kwargs={}),
141         Transformation(name='get_first_element', args=(), kwargs={})
142     ]
143 }
144 }
145 }
```

(continues on next page)

(continued from previous page)

```

146
147     if simple:
148         return recipe
149
150     recipe['datasets']['weights'] = {
151         'h5path':
152         f'/{group}/BS',
153         'transforms': [
154             Transformation(name='get_all_child_datasets', args=(), kwargs={'ignore': [
155 ↪ 'eigenvalues', 'kpts']})),
156             AttribTransformation(name='add_partial_sums',
157                                 attrib_name='atoms_groups',
158                                 args=(f'{atom_prefix}{{{}}}'.format,),
159                                 kwargs={'make_set': True}),
160             Transformation(name='split_array', args=(), kwargs={'suffixes': ['up', 'down
161 ↪ ']})),
162             Transformation(name='flatten_array', args=(), kwargs={})
163         ],
164         'unpack_dict':
165         True
166     }
167
168     return recipe

```

Each recipe can define the *datasets* and *attributes* entry (if one is not defined, a empty dict is returned in its place). Each entry in these sections has the same structure.

*#Example entry from the FleurBands recipe*

```

'fermi_energy': {
    'h5path':
    '/general',
    'description':
    'fermi_energy of the system',
    'transforms': [
        Transformation(name='get_attribute', args=('lastFermiEnergy',), kwargs={}),
        Transformation(name='get_first_element', args=(), kwargs={})
    ]
}

```

All entries must define the key `h5path`. This gives the initial dataset for this key, which will be extracted from the given `.hdf` file. The key of the entry corresponds to the key under which the result will be saved to the output dictionary.

If the dataset should be transformed in some way after reading it, there are a number of defined transformations in *transforms*. These are added to an entry by adding a list of namedtuples (*Transformation* for general transformations; *AttribTransformation* for attribute transformations) under the key *transforms*. General Transformations can be used in all entries, while transformations using an attribute value can only be used in the *datasets* entries. Each namedtuple takes the name of the transformation function and the positional (*args*), and keyword arguments (*kwargs*) for the transformation. Attribute transformations also take the name of the attribute, whose value should be passed to the transformation in *attrib\_name*.

At the moment the following transformation functions are pre-defined:

#### General Transformations:

- `get_first_element()`: Get the index 0 of the dataset
- `index_dataset()`: Get the index `index` of the dataset
- `slice_dataset()`: Slice the given dataset with the given argument
- `get_shape()`: Get the shape of the dataset
- `tile_array()`: Use `np.tile` to repeat dataset a given amount of times
- `repeat_array()`: Use `np.repeat` to repeat each element in the dataset a given amount of times
- `get_all_child_datasets()`: extract all datasets contained in the current hdf group and enter them into a dict
- `merge_subgroup_datasets()`: extract all datasets contained in the subgroups of the current hdf group and enter them into a dict in a list (or one numpy array)
- `stack_datasets()`: Stack the given datasets in the dictionary along a given axis
- `shift_dataset()`: Shift the given dataset with a scalar value
- `multiply_scalar()`: Multiply the given dataset with a scalar value
- `multiply_array()`: Mutiply the given dataset with a given array
- `convert_to_complex_array()`: Convert real dataset to complex array
- `calculate_norm()`: Calculate norm of list of vectors (either absolute or difference between subsequent entries)
- `cumulative_sum()`: Calculative cumulative sum of dataset
- `get_attribute()`: Get the value of one given attribute on the dataset
- `attributes()`: Get all defined attributes on the dataset as a dict
- `move_to_memory()`: Convert dataset to numpy array (if not already done implicitly)
- `flatten_array()`: Create copy of dataset flattened into one dimension
- `split_array()`: Split the given dataset along its first index and store result in a dictionary with keys with suffixes
- `convert_to_str()`: Convert datatype of dataset to string
- `periodic_elements()`: Convert atomic numbers to their atomic symbols

#### Transformations using an attribute:

- `multiply_by_attribute()`: Multiply dataset by value of attribute (both scalar and matrix)
- `shift_by_attribute()`: Shift the given dataset with the value of an attribute
- `repeat_array_by_attribute()`: Call `repeat_array()` with the value of an attribute as argument
- `tile_array_by_attribute()`: Call `tile_array()` with the value of an attribute as argument
- `add_partial_sums()`: Sum over entries in dictionary datasets with given patterns in the key (Pattern is formatted with given attribute value)

Custom transformation functions can also be defined using the `hdf5_transformation()` decorator. For some transformation, e.g. `get_all_child_datasets()`, the result will be a subdictionary in the `datasets` or `attributes` dictionary. If this is not desired the entry can include `'unpack_dict': True`. With this all keys from the resulting dict will be extracted after all transformations and put into the root dictionary.



### 4.1.4 Plotting Fleur DOS/bandstructures

This section discusses how to obtain plots of data in the `banddos.hdf` for density of states and bandstructure calculations.

The process here is divided in two parts. First we extract and transform the data in a way to make it easy to plot via the `HDF5Reader`. For a detailed explanation of the capabilities of this tool refer to *General HDF5 file reader*. Here we show the basic usage:

```
#Example: Bandstructure calculation

from masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurBands

with HDF5Reader('/path/to/banddos.hdf') as h5reader:
    data, attributes = h5reader.read(recipe=FleurBands)
```

In the following bandstructure and DOS plots are explained. Each section leads with the names of the recipes from the *recipes* module that can be used with the explained visualization function.

All Fleur specific plotting routines are found in *fleur* have implementations for both the matplotlib and bokeh plotting libraries and can be customized heavily. For an explanation on customizing plots refer to *General Plotting routines*.

#### 4.1.4.1 Bandstructures

Compatible Recipes for the `HDF5Reader`:

- `FleurBands`: Default recipe reading in the kpoints, eigenvalues and weights for atom and orbital contributions
- `FleurSimpleBands`: Reads in only the kpoints and eigenvalues and now weights
- `FleurOrbcompBands`: In addition to the eigenvalues the weights from an orbital decomposition calculation are read in
- `FleurjDOSBands`: In addition to the eigenvalues the weights from a jDOS calculation are read in
- `FleurMCDBands`: In addition to the eigenvalues the weights from a MCD calculation are read in
- `get_fleur_bands_specific_weights()`: Function to generate a recipe for reading in the eigenvalues+a provided list of weights

The bandstructure visualization `plot_fleur_bands()` can be used to plot

1. Non-spinpolarized/spinpolarized bandstructures
2. Bandstructures with emphasized weights on all eigenvalues (Also non-spinpolarized and spinpolarized)

#### Standard bandstructure

To plot a simple bandstructure without any weighting we just have to pass the data, that the `HDF5Reader` provided to the `plot_fleur_bands()`

The two examples below show the resulting plots for a non-spinpolarized system (bulk Si) and a spin-polarized system (Fe fcc). For both systems the necessary code is exactly the same and is shown above the plots. The shown plots are the ones for the matplotlib plotting backend:

```

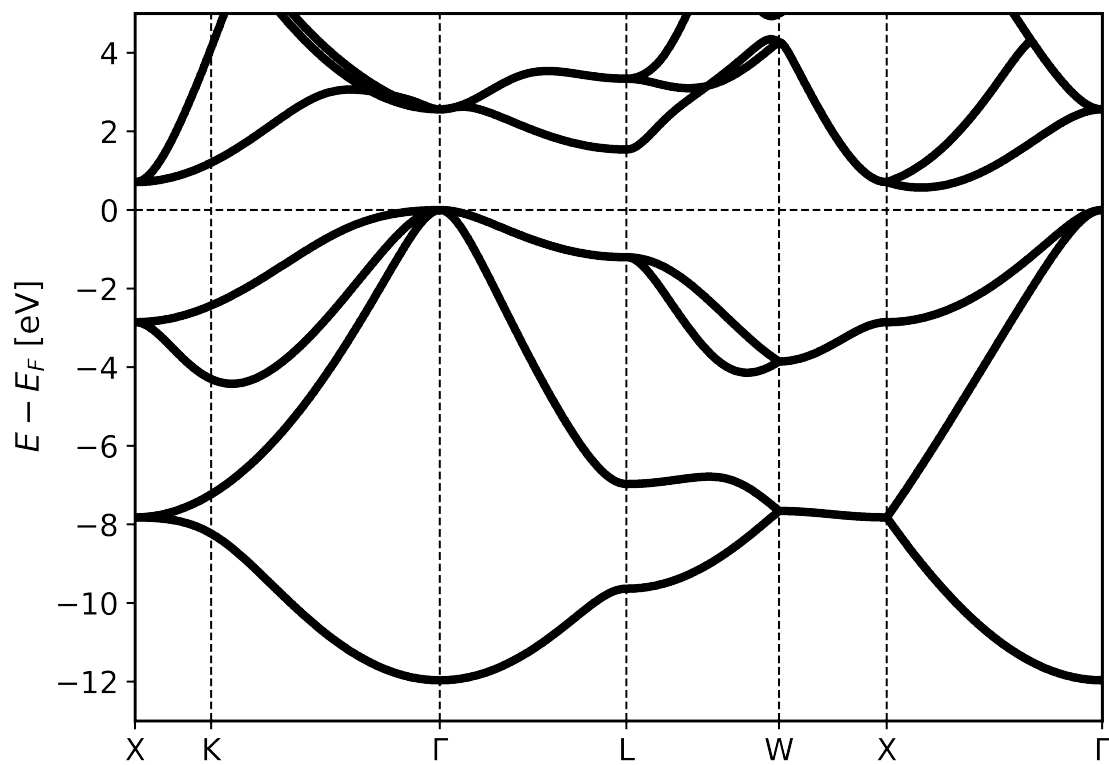
from maschi_tools.io.parsers.hdf5 import HDF5Reader
from maschi_tools.io.parsers.hdf5.recipes import FleurBands
from maschi_tools.vis.fleur import plot_fleur_bands

#Read in data
with HDF5Reader('/path/to/banddos.hdf') as h5reader:
    data, attributes = h5reader.read(recipe=FleurBands)

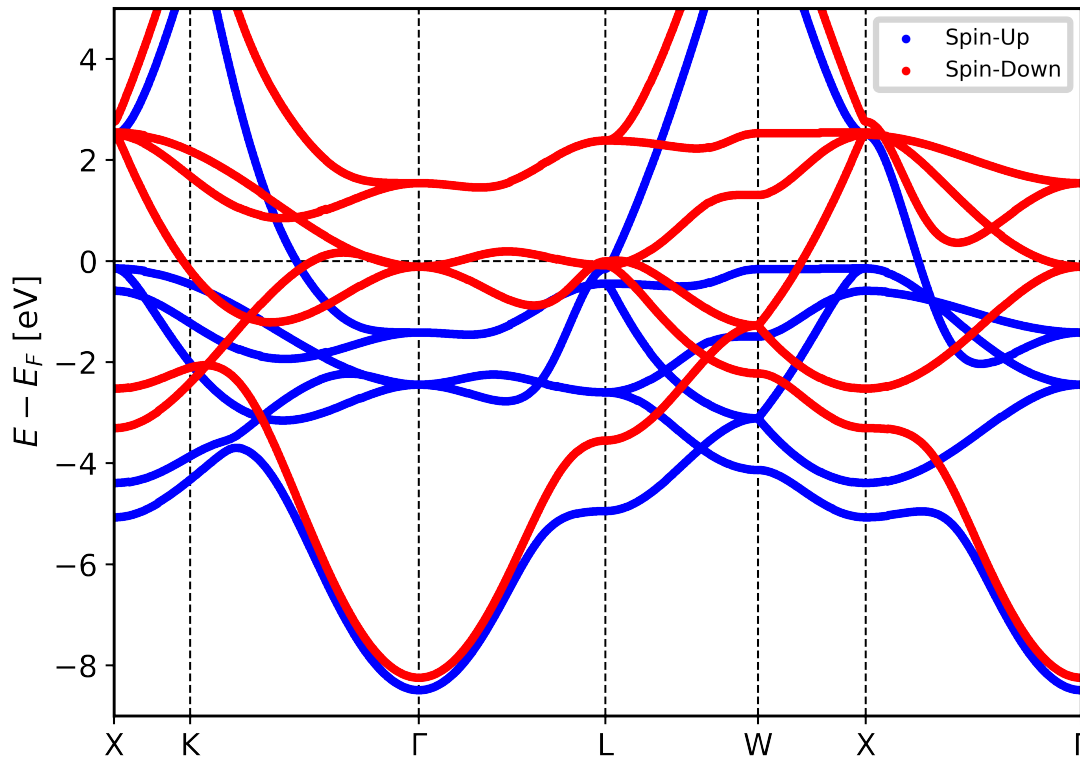
#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_bands(data, attributes)

```

### Non spinpolarized bandstructure



## Spinpolarized bandstructure



## Bandstructure with weights

To plot a simple bandstructure with weighting we do the same procedure as above, but we pass in the entry we want to use for weights. These correspond to the entries in the `banddos.hdf` file (for example the weight for the s-orbital on the first atom type is called `MT:1s`)

The weights will be used to change the size and color (according to a colormap) to indicate regions of high weight.

The two examples below show the resulting plots for a non-spinpolarized system (bulk Si) weighted for the s-orbital on the first atom and a spin-polarized system (Fe fcc) with weights for the d-orbital on the first atom type. For both systems the necessary code is exactly the same and is shown above the plots. The shown plots are the ones for the matplotlib plotting backend:

```
from maschi_tools.io.parsers.hdf5 import HDF5Reader
from maschi_tools.io.parsers.hdf5.recipes import FleurBands
from maschi_tools.vis.fleur import plot_fleur_bands

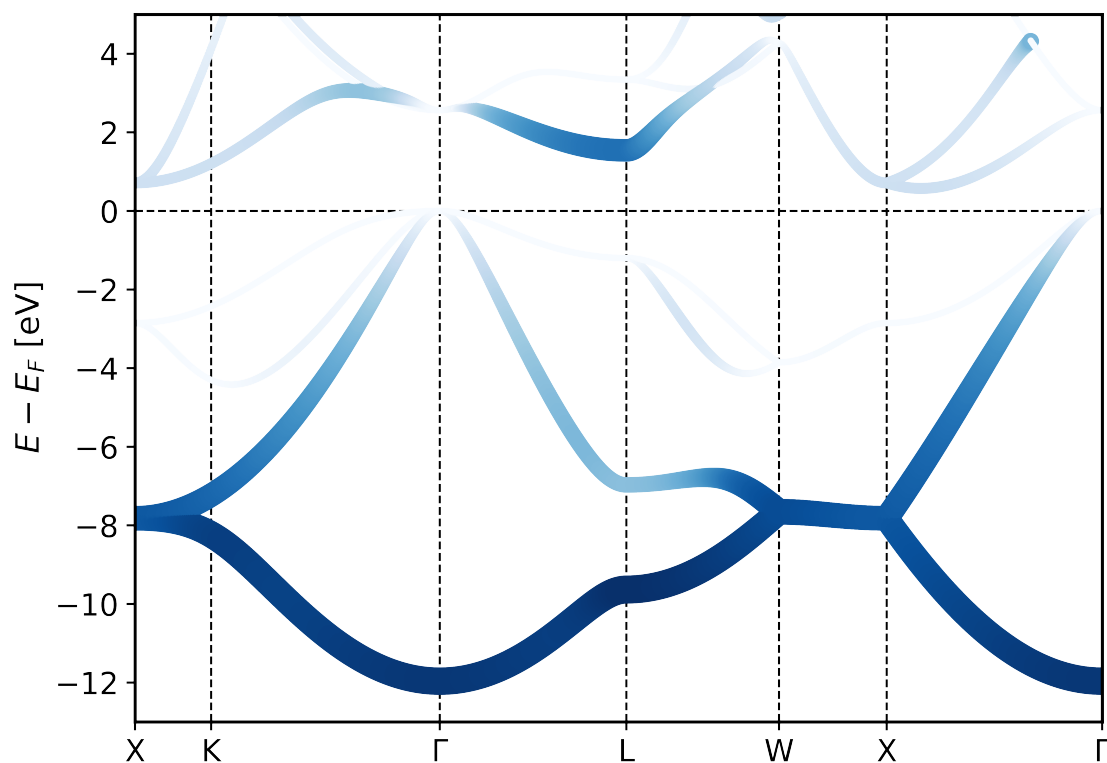
#Read in data
with HDF5Reader('/path/to/banddos.hdf') as h5reader:
    data, attributes = h5reader.read(recipe=FleurBands)
```

(continues on next page)

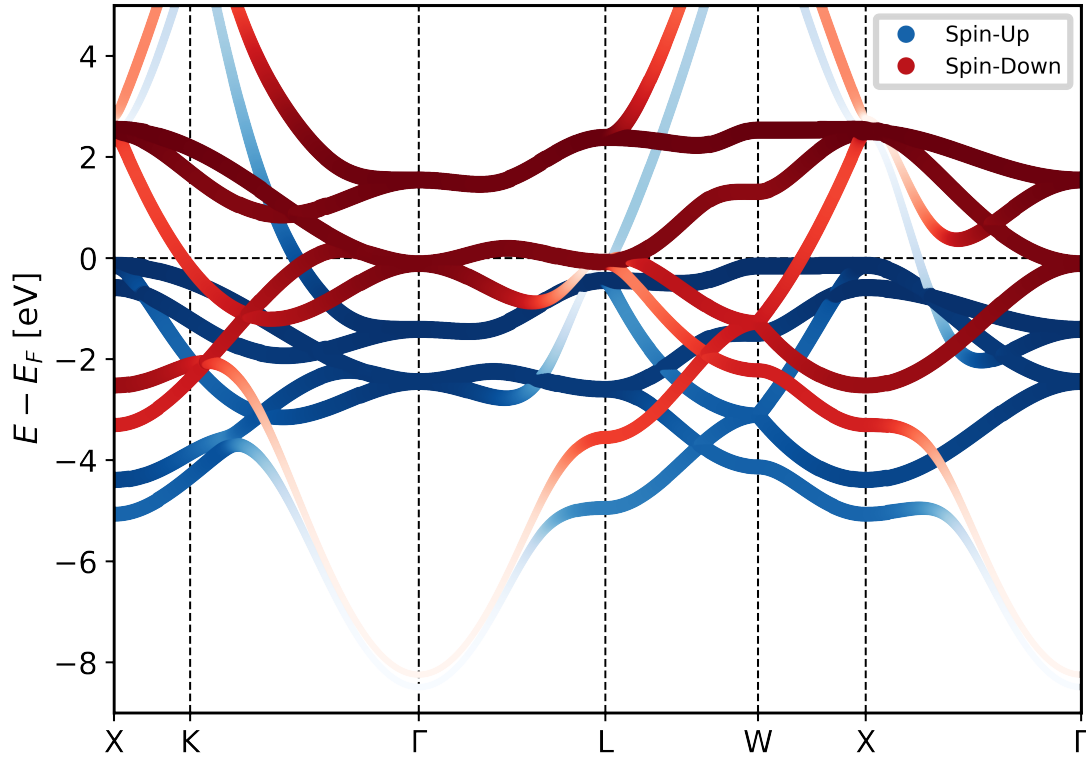
(continued from previous page)

```
#Plot the data  
#Notice that you get the axis object of this plot is returned  
#if you want to make any special additions  
ax = plot_fleur_bands(data, attributes, weight='MT:1s')
```

### Non spinpolarized bandstructure (weights for s-orbital)



### Spinpolarized bandstructure (weights for d-orbital)



#### 4.1.4.2 Density of States

Compatible Recipes for the [HDF5Reader](#):

- FleurDOS: Default recipe reading in the total, interstitial, vacuum, atom and l-channel resolved DOS
- FleurORBCOMP: Read in the DOS from an orbital decomposition calculation
- FleurJDOS: Read in the DOS from a jDOS calculation
- FleurMCD: Read in the DOS from a MCD calculation

The dos visualization `plot_fleur_dos()` can be used to plot non spinpolarized and spinpolarized DOS, with selection of which components to plot.

## Standard density of states plot

```

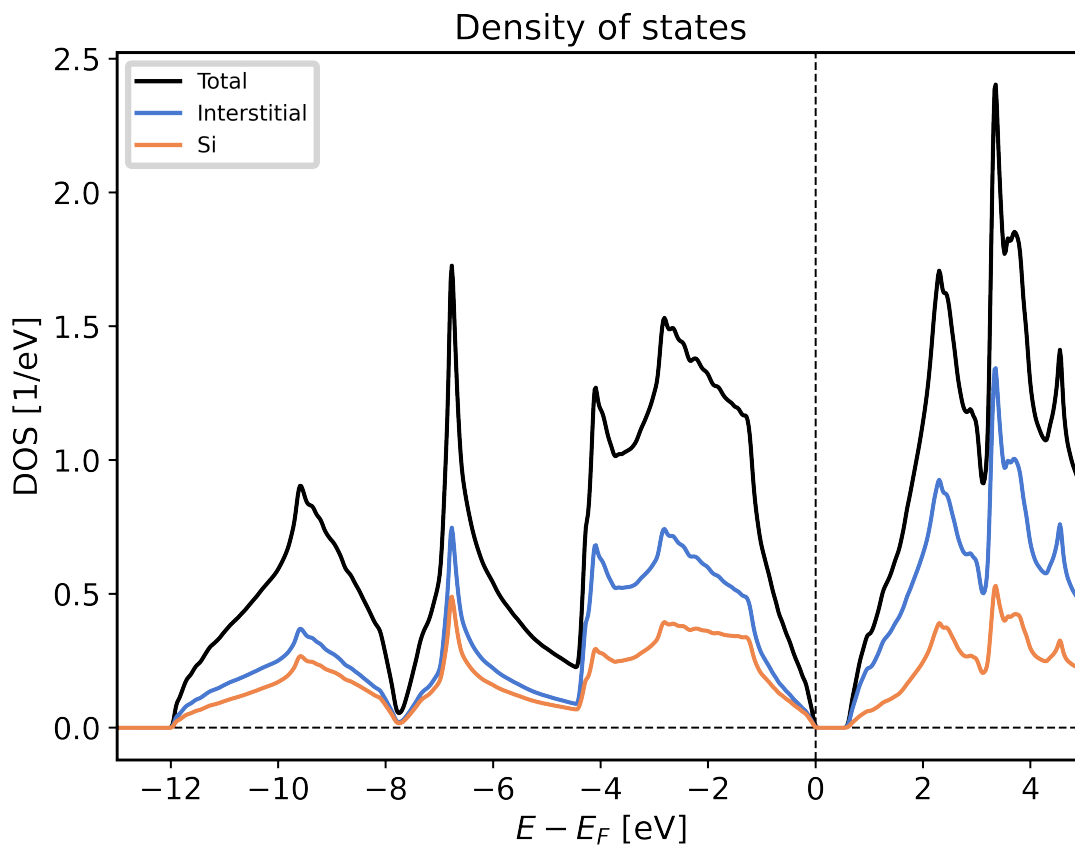
from maschi_tools.io.parsers.hdf5 import HDF5Reader
from maschi_tools.io.parsers.hdf5.recipes import FleurDOS
from maschi_tools.vis.fleur import plot_fleur_dos

#Read in data
with HDF5Reader('/path/to/banddos.hdf') as h5reader:
    data, attributes = h5reader.read(recipe=FleurDOS)

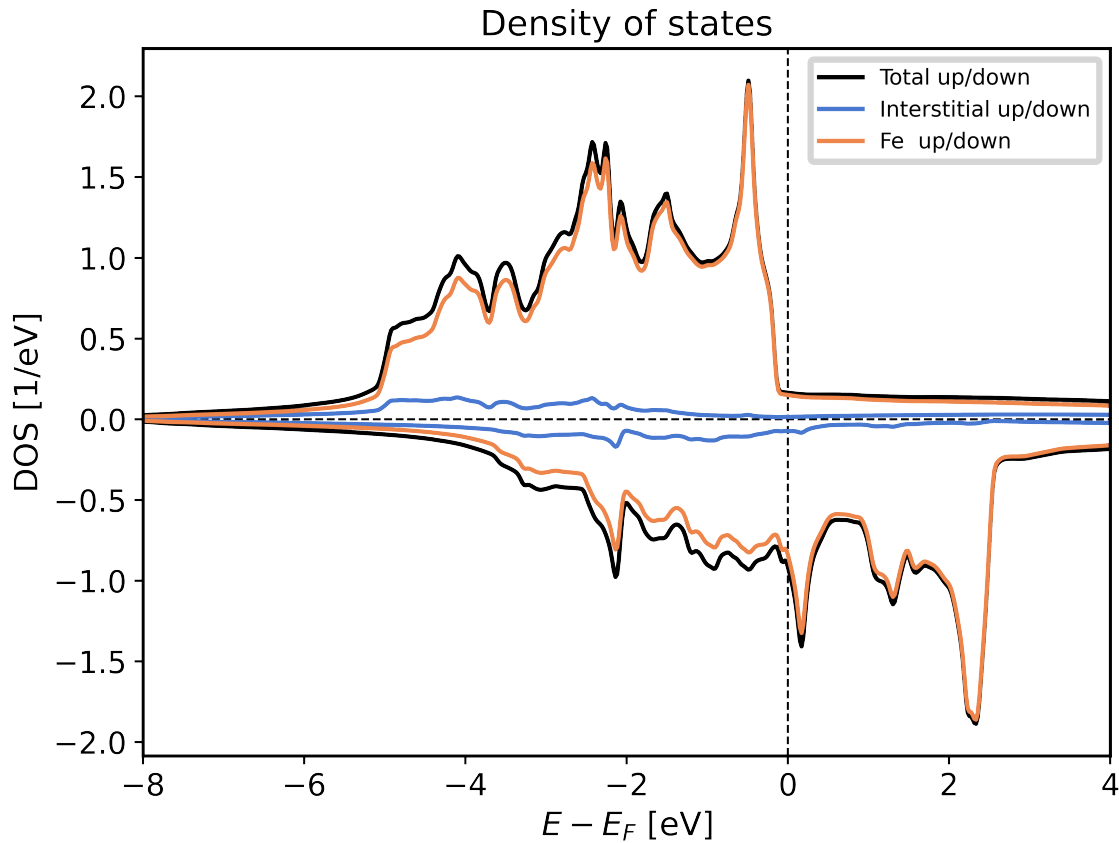
#Plot the data
#Notice that you get the axis object of this plot is returned
#if you want to make any special additions
ax = plot_fleur_dos(data, attributes)

```

## Non spinpolarized DOS



## Spinpolarized DOS



### Plotting options for DOS plots

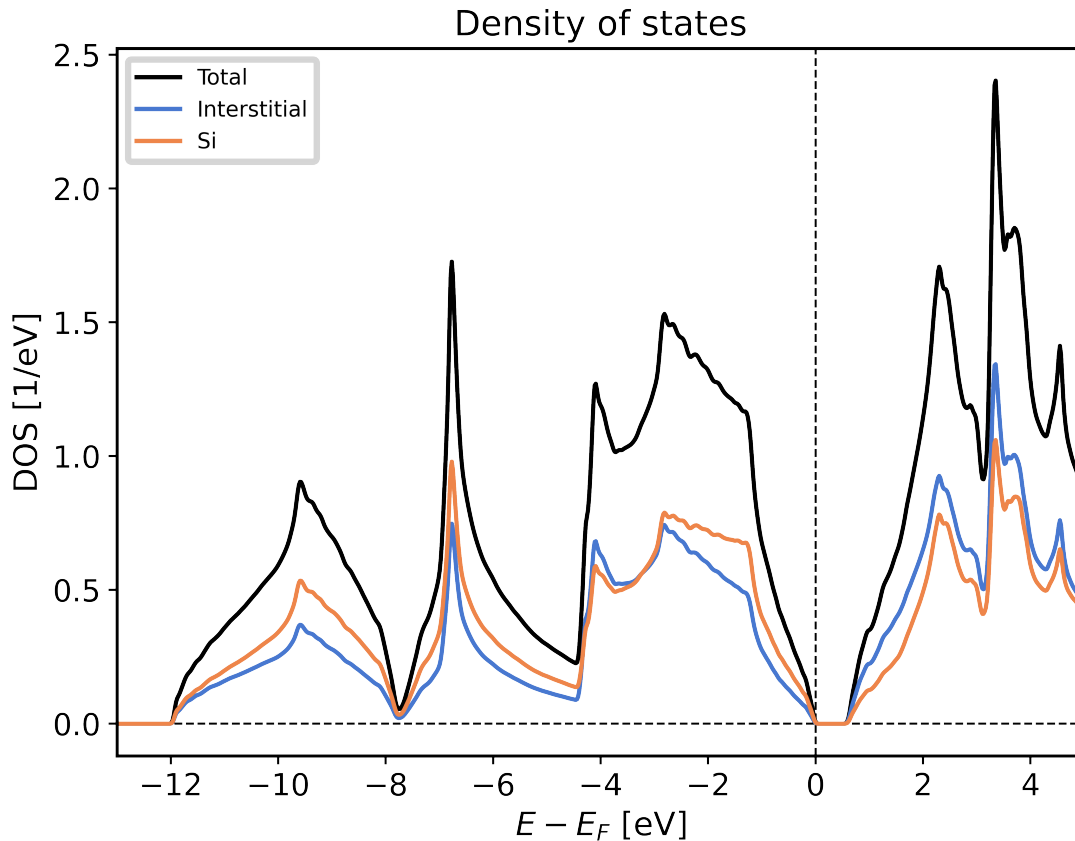
The `plot_fleur_dos()` function has a couple of options to modify, what is being displayed from the `banddos.hdf` file. Below we show a few examples of ways to use these options, together with examples of resulting plots.

### DOS with atom components scaled with equivalent atoms

When you look at the example plot for the non spin-polarized DOS, you might notice that the interstitial component and the atom projected components do not add up to the total density of states. This system has two symmetry equivalent *Si* atoms. By default the atom projected density of states corresponds to only one of these atoms.

If you wish to show the atom projected components of the DOS scaled with the number of symmetry equivalent atoms you can provide the option `multiply_by_equiv_atoms=True` option to the plotting function.

```
ax = plot_fleur_dos(data, attributes, multiply_by_equiv_atoms=True)
```



### Selecting specific DOS components

The DOS is made up of a lot of contributions that can be displayed separately.

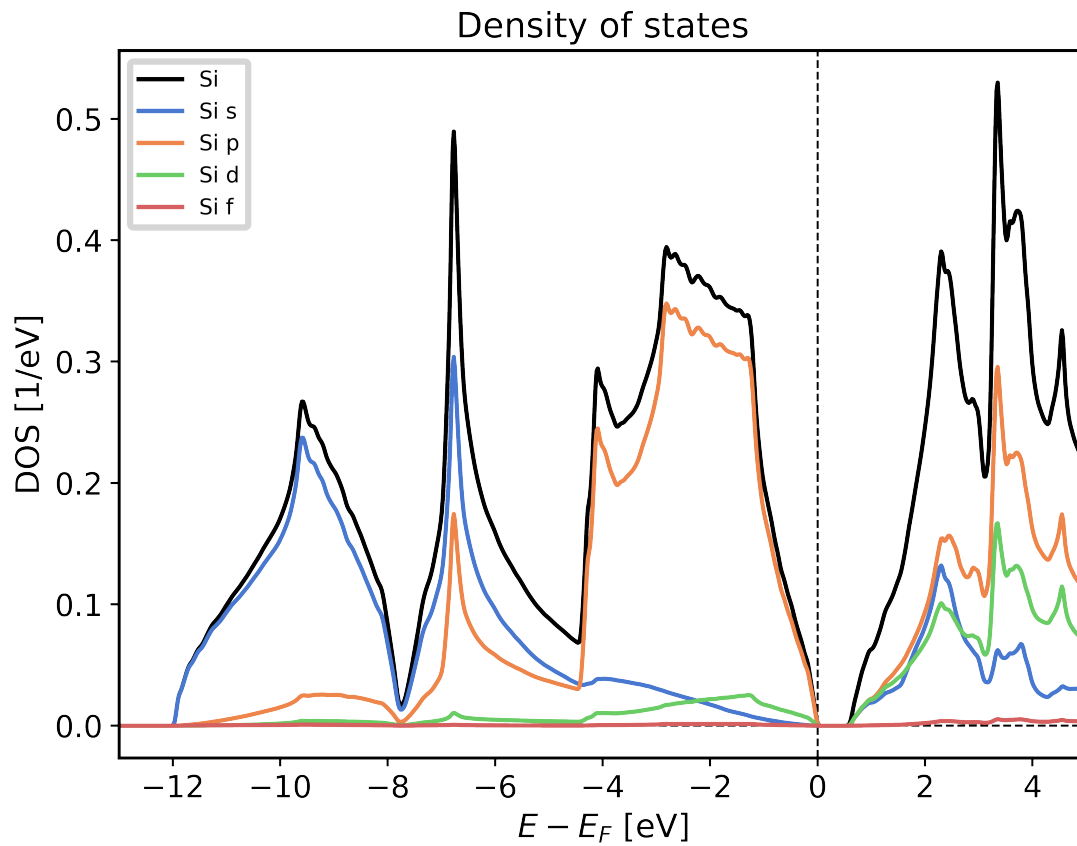
Here we list the options that are available and show example plots for only selecting the atom projected components of the density of states

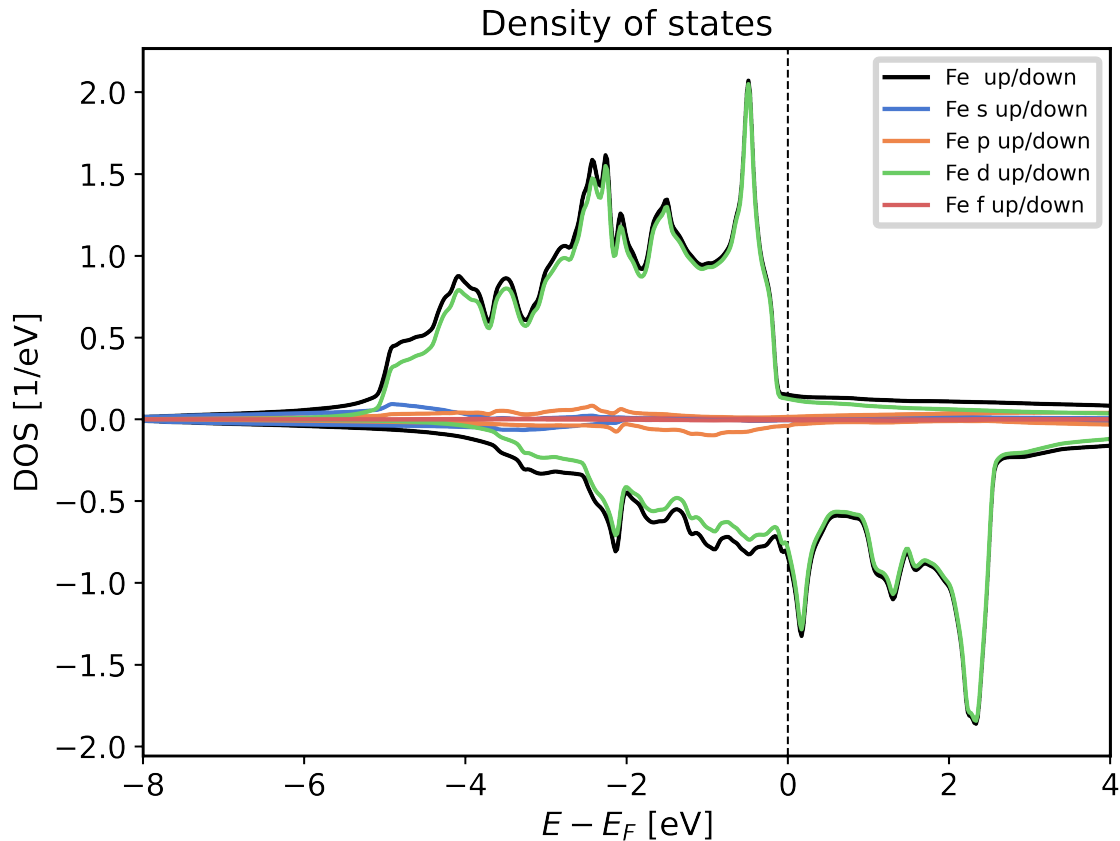
- **plot\_keys** Can be used to provide an explicit list of keys you want to display (Same format as in the `banddos.hdf`)
- **show\_total** Control, whether to show the total density of states (default `True`)
- **show\_interstitial** Control, whether to show the interstitial contribution of the density of states (default `True`)
- **show\_atoms** Control, which total atom projected DOS to show. Can be either the string `all` (All components are shown), the value `None` (no components are shown) or a list of the integer indices of the atom types that should be displayed (default `all`)
- **show\_iresolved** Control, on which atoms to show the orbital projected DOS. Can be either the string `all` (All components are shown), the value `None` (no components are shown) or a list of the integer indices of the atom types for which to display the orbital components (default `None`)

Below an example of only displaying the atom projected DOS together with their orbital contributions is shown.



```
ax = plot_fleur_dos(data, attributes,
                    show_total=False,
                    show_interstitial=False,
                    show_lresolved='all')
```





### 4.1.5 General Plotting routines

The plotting of data is always a common task that needs to be performed. However, there is a lot of variation in how someone might want plots to look or be arranged. Some plots might also need to be interactive to be of a real use.

For these reasons the `masci_tools` library provides utility for general plotting and template functions for common plots made when working with DFT methods. There are two plotting backends available:

- **matplotlib** Mainly used for non-interactive plots
- **bokeh** Mainly used for interactive plots

#### 4.1.5.1 Available Routines

For both of these there are a lot of plotting routines available (both general or specific to a problem). All of these routines will return the used `Axes` object in the case of `matplotlib` or the `figure` produced by `bokeh` for custom modifications.

- **common (Can be used for both backends):**
  - `scatter()`: Make a scatterplot with varying size and color of the points for multiple sets of data
  - `line()`: Make a lineplot with multiple sets of data
  - `dos()`: Plot a general density of states (non-spinpolarized)
  - `spinpol_dos()`: Plot a general density of states (spinpolarized)

- `bands()`: Plot a general bandstructure (non-spinpolarized)
- `spinpol_bands()`: Plot a general bandstructure (spinpolarized)
- **matplotlib:**
  - `single_scatterplot()`: Make a scatterplot with lines for a single set of data
  - `multiple_scatterplots()`: Make a scatterplot with lines for multiple sets of data
  - `multi_scatter_plot()`: Make a scatterplot with varying size and color for the points for multiple sets of data
  - `colormesh_plot()`: Make 2D plot with the data represented as color
  - `waterfall_plot()`: Make 3D plot with the `scatter3D` function of matplotlib
  - `surface_plot()`: Make 3D plot with the `plot_surface` function of matplotlib
  - `multiplot_moved()`: Plot multiple sets of data above each other with a configurable shift
  - `histogram()`: Make a histogram plot
  - `barchart()`: Make a barchart plot
  - `multiaxis_scatterplot()`: Make a plot containing multiple sets of data distributed over multiple subplots in a grid
  - `plot_convex_hull2d()`: Make a 2D plot of a convex hull
  - `plot_residuen()`: Make a residual plot for given real and fit data. Can also produce a histogram of the deviations
  - `plot_convergence_results()`: Plot the convergence behaviour of charge density distances and energies of a single calculation
  - `plot_convergence_results_m()`: Plot the convergence behaviour of charge density distances and energies of multiple calculations
  - `plot_lattice_constant()`: Plot the energy curve with changing unit cell volume
  - `plot_dos()`: Plot a general density of states (non-spinpolarized)
  - `plot_spinpol_dos()`: Plot a general density of states (spinpolarized)
  - `plot_bands()`: Plot a general bandstructure (non-spinpolarized)
  - `plot_spinpol_bands()`: Plot a general bandstructure (spinpolarized)
- **bokeh:**
  - `bokeh_scatter()`: Make a scatterplot for a single set of data
  - `bokeh_multi_scatter()`: Make a scatterplot for a multiple sets of data
  - `bokeh_line()`: Make a line plot for multiple sets of data
  - `bokeh_dos()`: Plot a general density of states (non-spinpolarized)
  - `bokeh_spinpol_dos()`: Plot a general density of states (spinpolarized)
  - `bokeh_bands()`: Plot a general bandstructure (non-spinpolarized)
  - `bokeh_spinpol_bands()`: Plot a general bandstructure (spinpolarized)
  - `periodic_table_plot()`: Make a interactive plot of data for the periodic table

If you have ideas for new useful and beautiful plotting routines you are welcome to contribute. Refer to the sections *Using the `Plotter` class* and *Using the `PlotData` class* for a guide on how to get started.

#### 4.1.5.2 Providing Data

Data can be provided to plotting functions in two main ways:

1. The first arguments and data arguments are given the keys in a mapping, which should be used. The corresponding mapping is provided via the data keyword argument
2. The first arguments and data arguments are given the data that should be plotted against each other.

The following two code blocks are equivalent in terms of the provided data.

```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-10,10,100)
y1 = x**2
y2 = 20*np.sin(x)

#The data is split up according to fixed rules that the plot function defines.
#The default behaviour is that a list of lists is interpreted as multiple separate plots
ax = multiple_scatterplots(x, [y1, y2])
```

```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-10,10,100)
y1 = x**2
y2 = 20*np.sin(x)
data = {'x': x, 'y1': y1, 'y2': y2}

ax = multiple_scatterplots('x', ['y1', 'y2'], data=data)
```

#### 4.1.5.3 Customizing Plots

You might want to change the parameters of your plot. From changing the color, linestyle or shape of the markers there are a million options to tweak.

These can be set by simply passing the keyword arguments with the desired parameters to the plotting function. The names of these parameters mostly correspond to the same names as in the plotting library that is used in the plotting function. However, there are some deviations and also some special keywords that you can use. We will go over the most important ones in this section accompanied with concrete code examples. For a reference of the defaults defined in the `masci_tools` library you can refer to [MatplotlibPlotter](#) and [BokehPlotter](#) for a complete reference.

The most important special keywords are listed below. If there are deviating names for these in `matplotlib` and `bokeh` plotting functions both names are written in the order `matplotlib` or `bokeh`:

- **limits** This is used to set the bounds of the axis specifically. Provided in form of a dictionary. For example passing `limits={'x': (-5,5)}` will set the x-axis limits between -5 and 5 and `limits={'x': (-5,5), 'y': (0,10)}` will set the y-axis limits in addition
- **scale** Used to set the scaling of the axis in the plots. Also provided in form of a dictionary. For example passing `scale={'x': 'log', 'y': 'log'}` will set both axis to logarithmic scaling `scale={'y': 'log'}` will only to it for the y-axis
- **lines or straight\_lines** Easy way to draw help lines into the plot. Provided in form of a dictionary. For example passing `lines={'vertical': 0}` will draw a vertical line at `x=0`

`lines={'horizontal': [1,5,10]}` will draw three horizontal lines at  $y=1$ ,  $5$  or  $10$  respectively

- **plot\_labels or legend\_labels** Defines labels for the legend of a plot
- **labels for axis** Normally called `xlabel` or `ylabel`, but specialized plot routines might have different names
- **title** Title for the produced plot
- **saving options** `show=True` call the plotting library specific show routines (default). For matplotlib you can also specify `saveas='filename'` and `save_plots=True` to save the plot to file

In the following we will look at examples using the matplotlib plotting functions in [plot\\_methods](#). The options are the same for the bokeh plotting routines in [bokeh\\_plots](#).

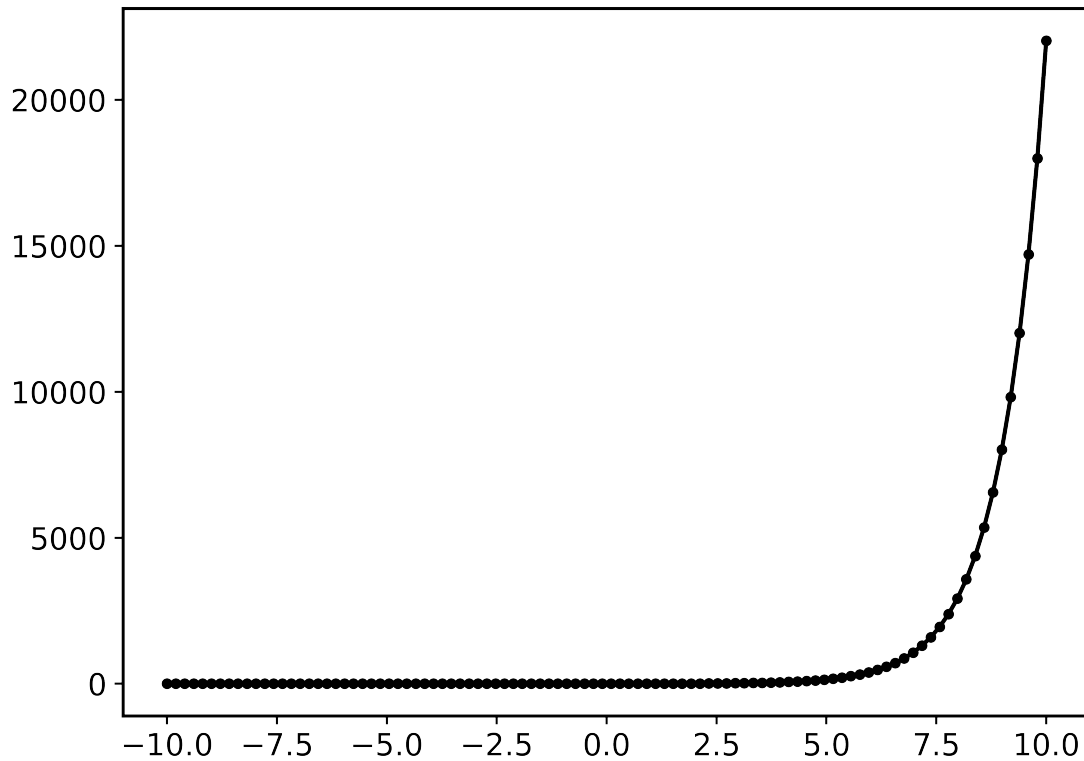
## Single plots

We start from the default result of calling the `single_scatterplot()` function with an exponential function. Afterwards we go through examples of modifying this call in one particular way. All of these can be combined to customize the plot to your desire

```
from masci_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y)
```

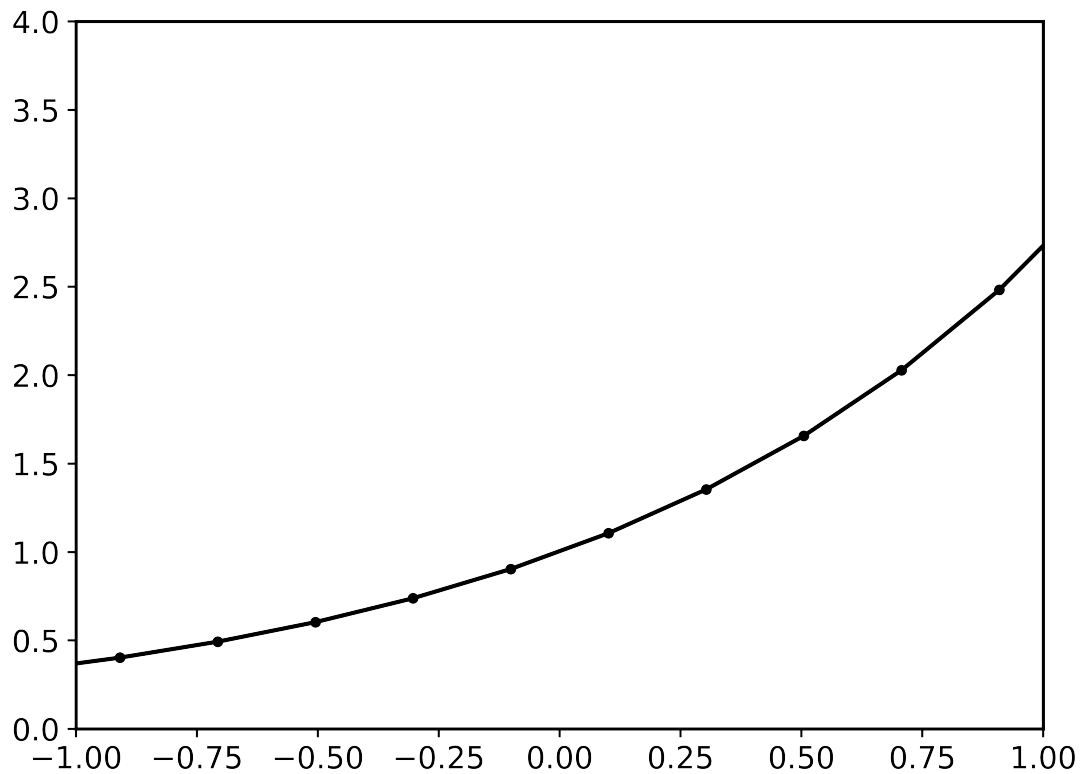


### Setting limits

```
from maschi_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y, limits={'x': (-1,1), 'y': (0,4)})
```

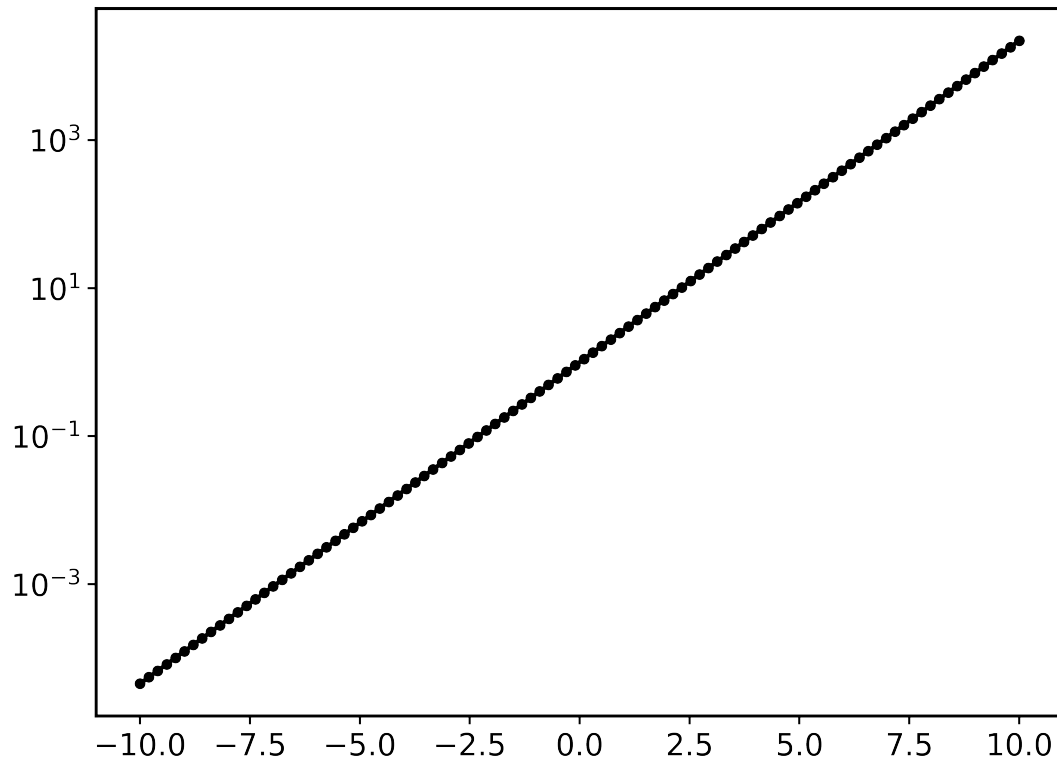


### Modifying the scale of the axis

```
from maschi_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y, scale={'y': 'log'})
```



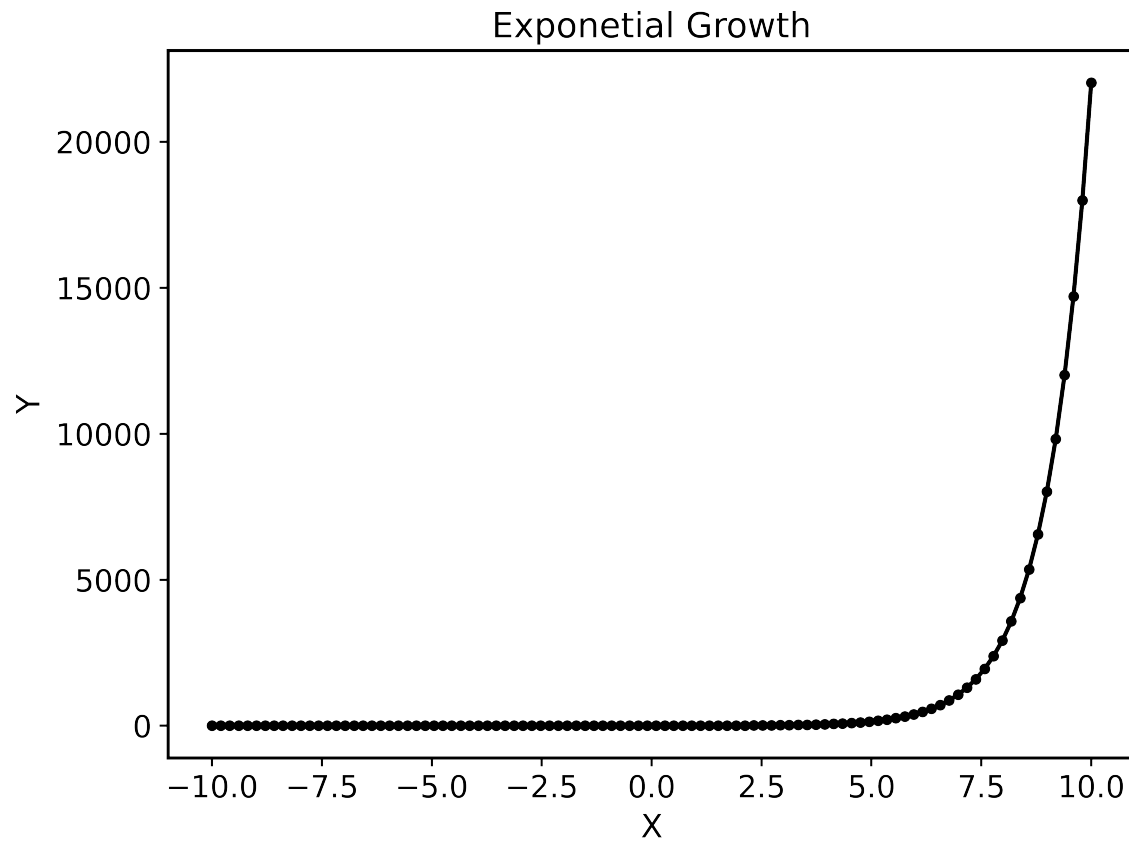
### Setting labels on the axis and a title

```
from maschi_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y, xlabel='X', ylabel='Y', title='Exponential Growth')
```





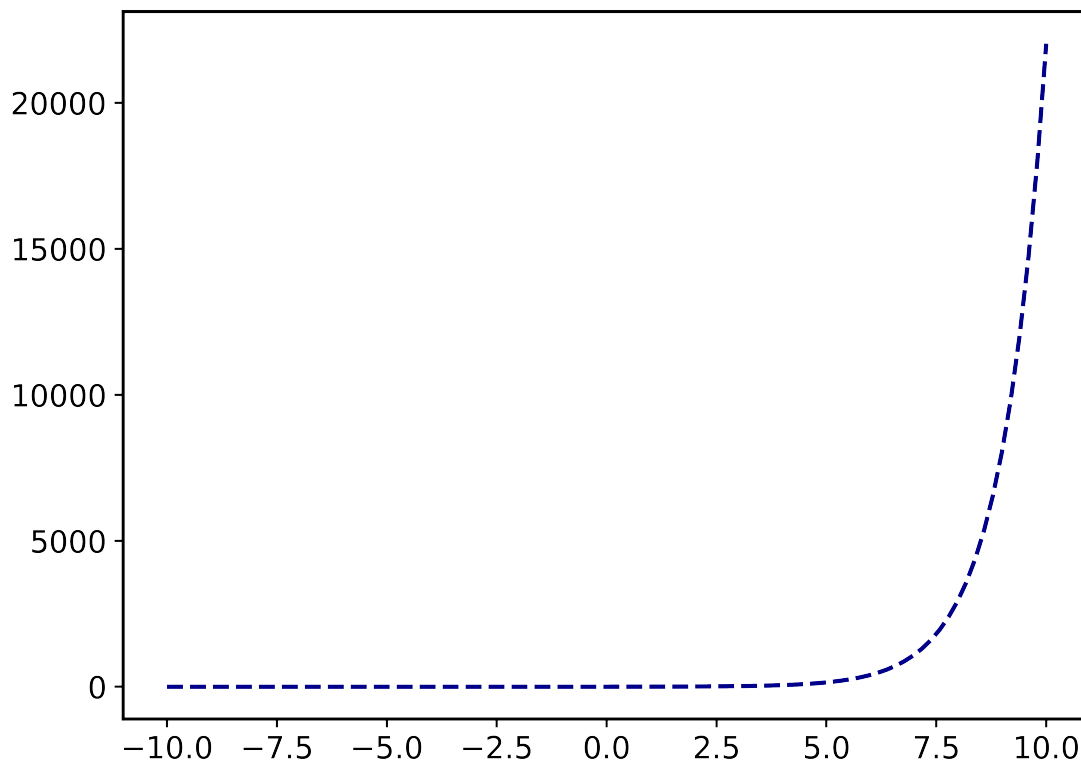
### Modifying plot parameters

See the [matplotlib](#) documentation for complete references of possible options

```
from masci_tools.vis.plot_methods import single_scatterplot
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

ax = single_scatterplot(x,y, color='darkblue', linestyle='--', marker=None)
```



### Setting user defaults

If you wish to change some parameters for all the plots you want to do, you can use the functions `set_mpl_plot_defaults()` or `set_bokeh_plot_defaults()` for the matplotlib and bokeh plotting library respectively. These functions accept the same keyword arguments as above and they will be applied to all the next plots that you do.

You can reset the changes to the defaults with `reset_mpl_plot_defaults()` or `reset_bokeh_plot_defaults()`

---

**Note:** You can still override these defaults by simply passing in another value for the parameter you wish to overwrite in the call to a plotting function

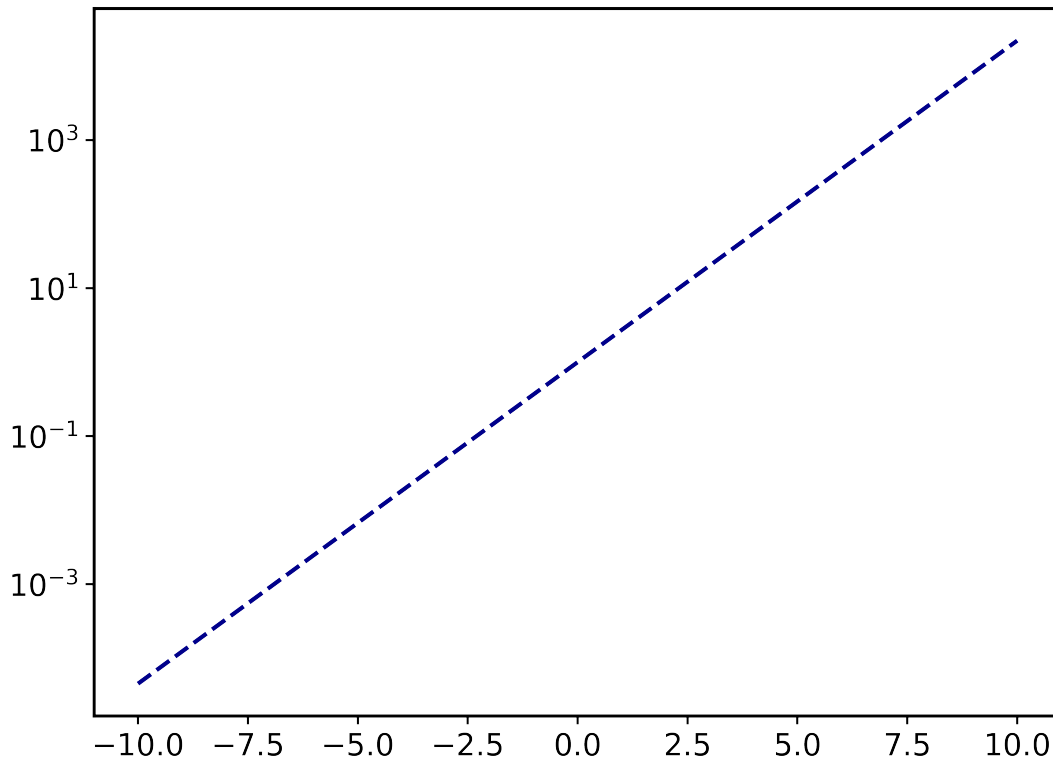
---

```
from maschi_tools.vis.plot_methods import single_scatterplot, set_mpl_plot_defaults
import numpy as np

x = np.linspace(-10, 10, 100)
y = np.exp(x)

set_mpl_plot_defaults(color='darkblue', linestyle='--', marker=None)

ax = single_scatterplot(x,y, scale={'y': 'log'})
```



## Multiple plots

Many plotting routines accept multiple sets of data to plot. An example of this is the `multiple_scatterplots()` function. The usage of these is essentially the same. However, some parameters can be changed for each data set to plot. These include but are not limited to `linestyle`, `linewidth`, `marker`, `markersize` and `color`. These parameters can either be set to a single value applying it to all data sets, or can be specified for some/all data sets with unspecified values being replaced with the current defaults. This second way can be done in two ways (Both of the below examples have the same effect):

1. List of values (None for unspecified values) Example: `linestyle=['-', None, '--']`
2. Dictionary with integer indices Example: `linestyle={0:'-', 2:'--'}`

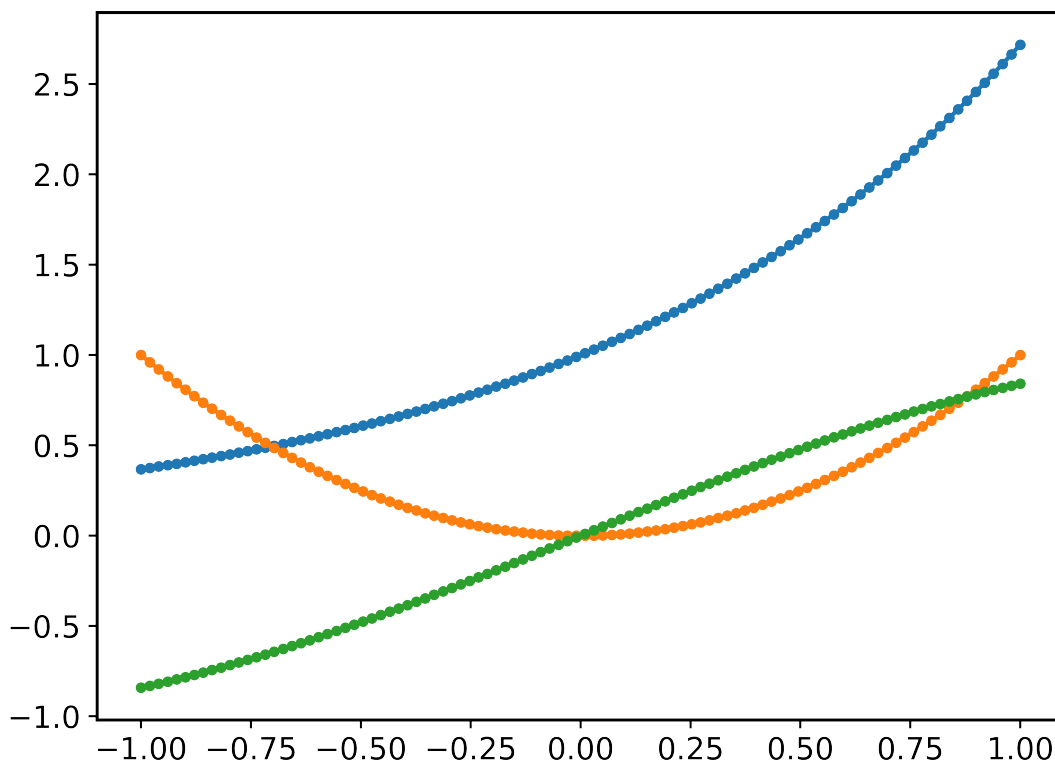
**Warning:** Specifying parameters for multiple data sets is only valid for the parameters passed into the function. Setting defaults with values for multiple data sets is not supported

## Default plot

```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-1,1,100)
y = np.exp(x)
y2 = x**2
y3 = np.sin(x)

ax = multiple_scatterplots([x, x, x], [y, y2, y3])
```



## Changing parameters on all plots

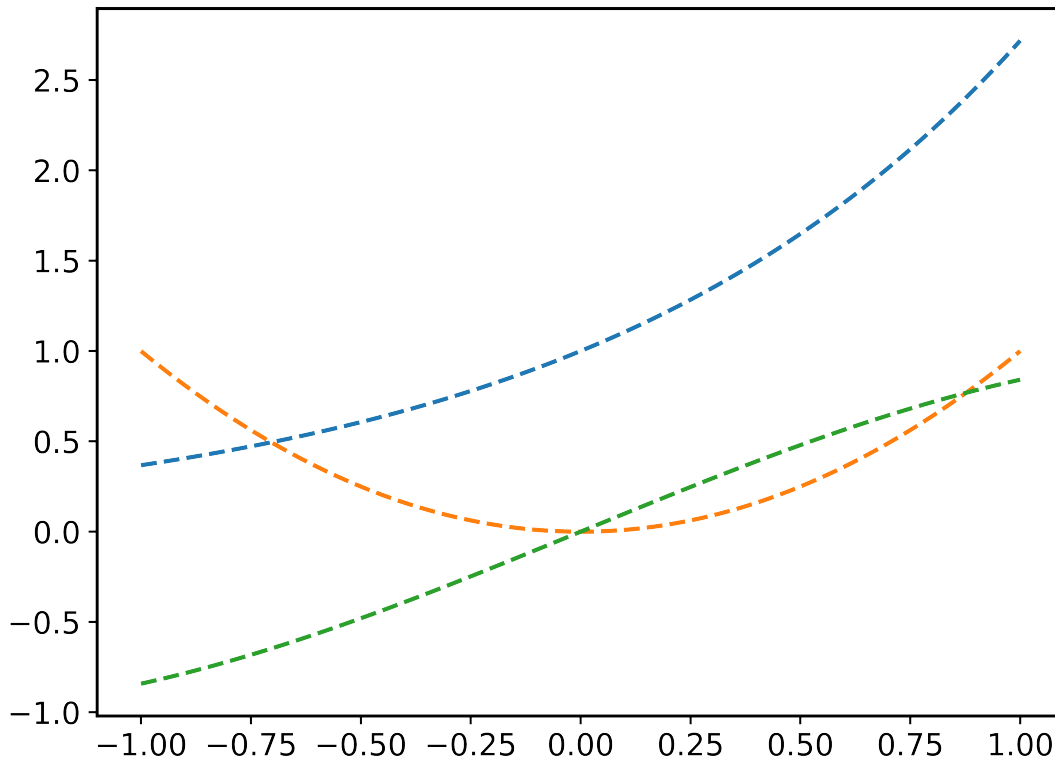
```
from masci_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-1,1,100)
y = np.exp(x)
y2 = x**2
y3 = np.sin(x)
```

(continues on next page)

(continued from previous page)

```
ax = multiple_scatterplots([x, x, x], [y, y2, y3], linestyle='--', marker=None)
```

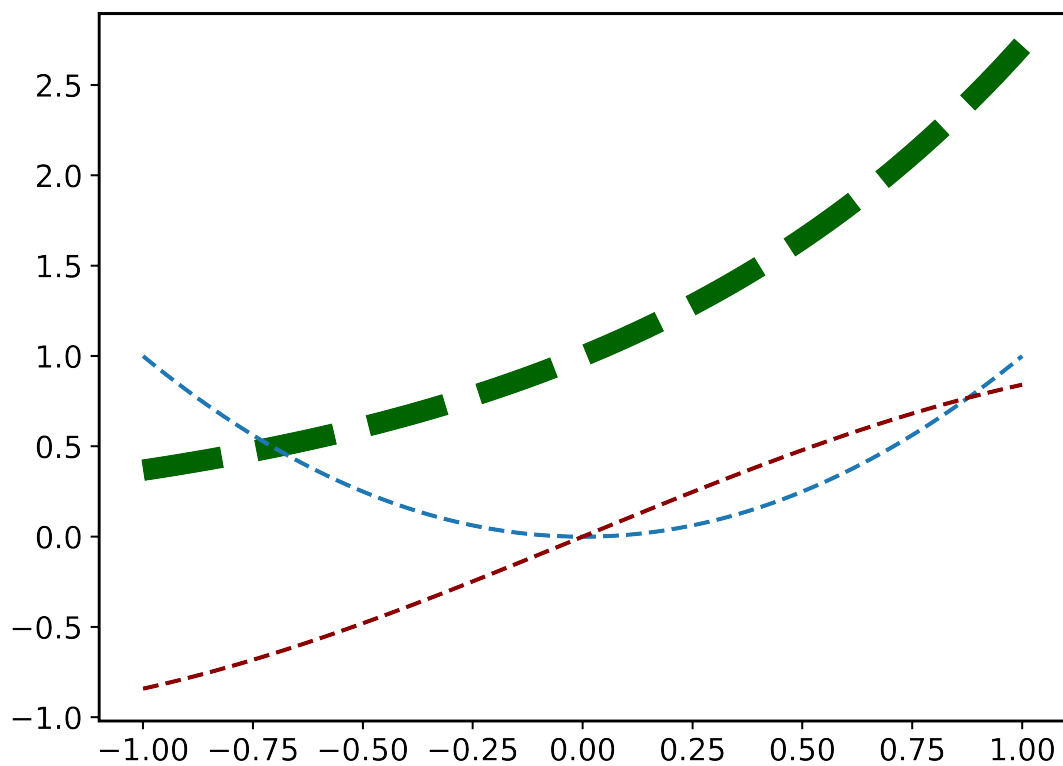


### Changing parameters on specific plots

```
from maschi_tools.vis.plot_methods import multiple_scatterplots
import numpy as np

x = np.linspace(-1,1,100)
y = np.exp(x)
y2 = x**2
y3 = np.sin(x)

ax = multiple_scatterplots([x, x, x], [y, y2, y3],
                           linestyle='--',
                           marker=None,
                           color=['darkgreen', None, 'darkred'],
                           linewidth={0: 10})
```



## DEVELOPER'S GUIDE

## 5.1 Developers Guide

This is the developers guide for masçi-tools

### 5.1.1 Updating or adapting the Fleur Parsers

Each input and output file for Fleur has a corresponding XML-Schema, where the structure of these files are defined.

To be able to parse such files efficiently and without hardcoding their structure we extract all necessary information about the schemas in the modules under *fleur\_schema*. The resulting python dictionaries can be accessed via the classes *InputSchemaDict* and *OutputSchemaDict*. The easiest way to instantiate one of these objects is to use the *fromVersion()* or *fromVersion()* methods by providing the desired version string.

#### 5.1.1.1 Adding/modifying a Fleur Schema:

The command `masçi-tools fleur-schema add <path-to-schema-file>` can be used to add the schema to the repository. If the schema for the specified version already exists an error is raised. To ignore this error the option `--overwrite` can be used.

#### 5.1.1.2 Adapting the outxml\_parser:

In contrast to the input file parser *inpxml\_parser()*, which parses all information available, the *outxml\_parser()* has to be more flexible. The out file has much more information which might not be always useful for users. Therefore the selection of what is parsed has to be much more specific.

**This selection is expressed in the context of tasks. In general this corresponds to things like:**

- Total energy
- Charge density distances
- Magnetic moments
- and so on ...

These are expressed in a definition in form of a dictionary. Below a simple example (Total energy) is shown, which parses the ``value`` and ``units`` attribute of the ``totalEnergy`` tag. The hardcoded known parsing tasks can be found in *default\_parse\_tasks*

```
total_energy_definition = {
    'energy_hartree': {
        'parse_type': 'singleValue',
        'path_spec': {
            'name': 'totalEnergy'
        }
    },
}
```

The definition of a task can consist of multiple keys (in this case only `energy_hartree`), which by default correspond to the keys in the resulting output dictionary. Each key has to contain the `parse_type` and `path_spec` key. The `parse_type` defines the method used to extract the information.

The following are possible:

- attrib** Will parse the value of the given attribute
- text** Will parse the text of the given tag
- numberNodes** Will return the number of nodes for the given tag
- exists** Will return, whether the given tag exists
- attrib\_exists** Will return, whether the given attribute exists
- allAttribs** Will parse all known attributes at the given tag into a dictionary
- parentAttribs** Will parse all known attributes at the given tag into a dictionary, but for the parent of the tag
- singleValue** Special case of allAttribs to parse value and units attribute for the given tag

The `path_spec` key specifies how the key can be uniquely identified.

It can contain the following specifications:

- name** Name of the wanted tag/attribute
- contains** A phrase, which has to occur in the path
- not\_contains** A phrase, which has to not occur in the path
- exclude** list of str. Only valid for attributes (these are sorted into different categories `unique`, `unique_path` and `other`). This attribute can exclude one or more of these categories

All except the `name` key are optional and should be constructed so that there is only one possible choice. Otherwise an exception is raised. There are other keywords, which can be entered here. These control how the parsed data is entered into the output dictionary. For a definition of these keywords, please refer to [default\\_parse\\_tasks](#).

Each task can also contain a number of control keys, determining when to perform the tasks. Each of these keys begins with an underscore. All of these are optional. The following are valid:

- \_general** bool, if True (default False) the task is not performed for each iteration but once on the root of the file
- \_minimal** bool, if True the task is performed even when `minimal_mode = True` is given
- \_modes** list of tuples specifying requirements on the `fleur_modes` for the task. For example `[('jspins', 2), ('soc', True)]` will only perform the task for a magnetic SOC calculation
- \_conversions** list of str, giving the names of functions to call after this task. Functions given here have to be decorated with the [conversion\\_function\(\)](#) decorator



`_special` bool, if True (default False) this task is NEVER added automatically and has to be added by hand

### 5.1.1.3 Migrating the parsing tasks

These task definitions might have to be adapted for new fleur versions. Some changes might be possible to make in `default_parse_tasks` directly without breaking backwards compatibility. If this is not possible there is a decorator `register_migration()` to define a function that is recognized by the class `ParseTasks` to convert between versions. A usage example is shown below.

```
from maschi_tools.util.parse_tasks_decorators import register_migration
import copy

@register_migration(base_version='0.33', target_version='0.34')
def migrate_033_to034(definition_dict):
    """
    Fictitious migration from 0.33 to 0.34
    Moves the `number_of_atom_types` attribute from reading a simple
    attribute to counting the number of atomGroups in the input section
    And removes orbital_magnetic_moments task
    """

    #IMPORTANT: First copy the original dict
    new_dict = copy.deepcopy(definition_dict)

    #If a task is incompatible remove it from the definition_dict
    new_dict.pop('orbital_magnetic_moments')

    new_dict['general_out_info'].pop('number_of_atom_types')
    new_dict['general_inp_info']['number_of_atom_types'] = {
        'parse_type': 'numberNodes',
        'path_spec': {
            'name': 'atomGroup'
        }
    }

    return new_dict
```

## 5.1.2 Using the Plotter class

### 5.1.2.1 Description

The `Plotter` class aims to provide a framework, which can be used to handle default values and collect common codeblocks needed for different plotting frameworks.

The `Plotter` class is a base class that should be subclassed for different Plotting backends. See `MatplotlibPlotter` or `BokehPlotter` for examples. The Subclass provides a dictionary of all the keys that should be handled by the plotter class. The Plotter class provides a hierarchy of overwriting these parameters (Higher numbers take precedence).

1. Function defaults set with `set_defaults()` with `default_type='function'`
2. Global defaults set with `set_defaults()`
3. Parameters set with `set_parameters()`

The subclasses should then also provide the plotting backend specific useful code snippets. For example showing colorbars, legends, and so on ...

For a list of these functions you can look at the respective documentation ([MatplotlibPlotter](#) or [BokehPlotter](#))

### 5.1.2.2 Writing a plotting function

In the following we will go through a few examples of how to write a simple plotting function using the [Plotter](#) class. We will be focusing on the [MatplotlibPlotter](#), but all of this is very similar for other plotting backends.

#### Local instance

Even though the [Plotter](#) class is meant to be used globally or on the module level, it can also be useful locally for simplifying simple plotting scripts. Here we have an example of a function producing a single plot with the given data for the x and y coordinates.

```
def plot_with_defaults(x,y,**kwargs):
    from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter

    #First we instantiate the MatplotlibPlotter class
    plot_params = MatplotlibPlotter()

    #Now we process the given arguments
    plot_params.set_parameters(**kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()

    ax.plot(x, y, **plot_kwargs)

    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)

    return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

#Some examples
plot_with_defaults(x, y)
plot_with_defaults(x, y, limits={'x': (0,1)})
plot_with_defaults(x, y, marker='s', markersize=20)
```

## Global/Module level instance

The local instance already gives us reusable code snippets to avoid common pitfalls when doing matplotlib/bokeh plots. But when instantiating the `Plotter` class locally we have no way of letting the user modify the global defaults.

However, when handling global state we need to be careful to not leave the instance of the `Plotter` class in an inconsistent state. If an error is thrown inside the plotting routine the parameters would stay set and may lead to very unexpected results. For this reason every plotting function using a global or module level instance of these plotters should be decorated with the `ensure_plotter_consistency()` decorator. This does two things:

1. If an error occurs in the decorated function the parameters will be reset before the error is raised
2. It makes sure that nothing inside the plotting routine changed the user defined defaults

Let us take the previous example and convert it to use a global instance

```
from maschi_tools.vis.matplotlib_plotter import MatplotlibPlotter
from maschi_tools.vis import ensure_plotter_consistency

#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_with_defaults(x,y,**kwargs):

    #Now we process the given arguments
    plot_params.set_parameters(**kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()

    ax.plot(x, y, **plot_kwargs)

    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)

    return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

#Some examples
plot_with_defaults(x, y)
plot_params.set_defaults(marker='s', markersize=20)
```

(continues on next page)

(continued from previous page)

```
plot_with_defaults(x, y, limits={'x': (0,1)})
plot_with_defaults(x, y)
```

The `masci_tools.vis.parameters.Plotter.set_defaults()` method is exposed in the two main modules for plotting `masci_tools.vis.plot_methods` `masci_tools.vis.bokeh_plots` as the functions `masci_tools.vis.plot_methods.set_mpl_plot_defaults()` and `masci_tools.vis.bokeh_plots.set_bokeh_plot_defaults()` specific to the plotter instance that is used in these modules.

## Function defaults

Some functions may want to set function specific defaults, that make sense inside the function, but may not be useful globally. The following example sets the default linewidth for our function to 6.

---

**Note:** Function defaults are also reset by the `ensure_plotter_consistency()` decorator, when the plotting function terminates successfully or in an error

---

```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis import ensure_plotter_consistency

#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_with_defaults(x,y,**kwargs):

    #Set the function defaults
    plot_params.set_defaults(default_type='function', linewidth=6)

    #Now we process the given arguments
    plot_params.set_parameters(**kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()

    ax.plot(x, y, **plot_kwargs)

    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)

    return ax
```

(continues on next page)

(continued from previous page)

```
import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

#Some examples
plot_with_defaults(x, y)
plot_params.set_defaults(marker='s', markersize=20)
plot_with_defaults(x, y, limits={'x': (0,1)})
plot_with_defaults(x, y)
```

## Passing keyword arguments directly to plot calls

The plotter classes have a restricted set of keys that they recognize as valid parameters. This set is of course not complete, since there is a vast number of parameters you can set for all plotting backends. In our previous examples unknown keys will immediately lead to an error in the call to `set_parameters()`. To enable this functionality we can provide the `continue_on_error=True` as an argument to this method.

Then the unknown keys are ignored and are returned in a dictionary. Additionally you can explicitly bypass the plotter object if you provide arguments in a dictionary with the name `extra_kwargs` it will be ignored, unpacked and returned along with the unknown keys

**Warning:** Be careful with the this feature and especially the `extra_kwargs`, since there is no check for name clashes with this argument. You might also run into situations, where arguments of different names collide with arguments provided by the *Plotter*

```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis import ensure_plotter_consistency

#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_with_defaults(x,y,**kwargs):

    #Set the function defaults
    plot_params.set_defaults(default_type='function', linewidth=6)

    #Now we process the given arguments (unknown ones are returned)
    kwargs = plot_params.set_parameters(continue_on_error=True, **kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()
```

(continues on next page)

(continued from previous page)

```
ax.plot(x, y, **plot_kwargs, **kwargs)

#The MatplotlibPlotter has a lot of small helper functions
#In this case we just want to set the limits and scale of the
#axis if they were given
plot_params.set_scale(ax)
plot_params.set_limits(ax)

return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

#The key markerfacecolor is not known to the MatplotlibPlotter
plot_with_defaults(x, y, markerfacecolor='red', markersize=20)
```

## Multiple plotting calls

The plotter classes also provide support for multiple plotting calls with different data sets in a single plotting function. To enable this feature we need to set two properties on the `masci_tools.vis.parameters.Plotter`; `single_plot` to `False` and `num_plots` to the number of plot calls made in this function. The plot specific parameters can then be specified in two ways. Shown behind the two ways is the way to set the color of the second data set to red.

1. List of values (None for unspecified values) [None, 'red']
2. Dict with integer indices for the specified values {1: 'red'}

Unspecified values are replaced with the previously set defaults.

---

**Note:** The `num_plots` and `single_plot` properties are also reset by the `ensure_plotter_consistency()`

---

```
from masci_tools.vis.matplotlib_plotter import MatplotlibPlotter
from masci_tools.vis import ensure_plotter_consistency

#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_2lines_with_defaults(x,y,**kwargs):

    plot_params.single_plot = False
    plot_params.num_plots = 2

    #Set the function defaults
    plot_params.set_defaults(default_type='function', linewidth=6)
```

(continues on next page)

(continued from previous page)

```

#Now we process the given arguments (unknown ones are returned)
kwargs = plot_params.set_parameters(continue_on_error=True, **kwargs)

#Set up the axis, on which to plot the data
ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

#The plot_kwargs provides a way to get the keyword arguments for the
#actual plotting call to `plot` in this case.
#For multiple plots this will be a list of dicts
#of length `num_plots`
plot_kwargs = plot_params.plot_kwargs()

ax.plot(x[0], y[0], **plot_kwargs[0], **kwargs)
ax.plot(x[1], y[1], **plot_kwargs[1], **kwargs)

#The MatplotlibPlotter has a lot of small helper functions
#In this case we just want to set the limits and scale of the
#axis if they were given
plot_params.set_scale(ax)
plot_params.set_limits(ax)

return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2
y2 = x**3

#The key markerfacecolor is not known to the MatplotlibPlotter
plot_2lines_with_defaults([x,x], [y,y2])
plot_2lines_with_defaults([x,x], [y,y2],
                           color={1:'red'}, linestyle=['--',None])

```

## Custom function specific parameters

You might have situations, where you want to have some function specific parameters, that should pull from the previously set defaults or even a custom default value.

The `add_parameter()` method is implemented exactly for this purpose. It creates a new key to be handled by the plotter class and with the arguments `default_from` or `default_value` we can specify what the defaults should be. `default_value` sets a specific value, `default_from` specifies a key from the plotter class from which to take the default value.

The `plot_kwargs()` method then can take keyword arguments to replace the arguments to take with your custom parameters

**Note:** These added parameters live on the function defaults and parameters level, meaning they will be removed by the `ensure_plotter_consistency()` decorator after the function finishes

```
from maschi_tools.vis.matplotlib_plotter import MatplotlibPlotter
from maschi_tools.vis import ensure_plotter_consistency

#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

#The decorator needs to get the plotter object
#that is used inside the function
@ensure_plotter_consistency(plot_params)
def plot_shifted_with_defaults(x,y,**kwargs):

    #Set the function defaults
    plot_params.set_defaults(default_type='function', linewidth=6)

    plot_params.add_parameter('linestyle_shifted',
                             default_from='linestyle')

    #Now we process the given arguments (unknown ones are returned)
    kwargs = plot_params.set_parameters(continue_on_error=True, **kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()
    ax.plot(x, y, **plot_kwargs, **kwargs)

    #This call replaces the parameter linestyle with our custom
    #parameter linestyle_shifted
    plot_kwargs = plot_params.plot_kwargs(linestyle='linestyle_shifted')
    ax.plot(x, y+2, **plot_kwargs, **kwargs)

    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)

    return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

plot_shifted_with_defaults(x, y)
plot_shifted_with_defaults(x, y, linestyle_shifted='--')
```



## Nested plotting functions

More complex plotting routines might want to call other plotting routines to simplify their structure. However, this has a side-effect when working with the `Plotter` class and the `ensure_plotter_consistency()` decorator. Since the decorator resets the parameters and function defaults after a plotting function has been called you lose everything that you might have modified in the enclosing plotting function.

If you do need access to these parameters after calling a nested plotting function the `NestedPlotParameters()` contextmanager is implemented. It defines a local scope, in which a plotting function can change the parameters and function defaults. After exiting the local scope the parameters and function defaults are always in the same state as when the `with` block was entered (Even if an error is raised). The nested plotting function will also start with the state that was set before.

Usage is shown here:

```
from maschi_tools.vis.matplotlib_plotter import MatplotlibPlotter
from maschi_tools.vis import ensure_plotter_consistency
from maschi_tools.vis import NestedPlotParameters

#First we instantiate the MatplotlibPlotter class
plot_params = MatplotlibPlotter()

@ensure_plotter_consistency(plot_params)
def nested_plot_function(x, y, **kwargs):

    plot_params.set_defaults(default_type='function',
                             linewidth=10, linestyle='--')

    #The contextmanager also needs a reference to the plotter object
    #to manage
    with NestedPlotParameters(plot_params):
        ax = plot_with_defaults(x,y,**kwargs)

    #Will plot with the above set defaults
    plot_kwargs = plot_params.plot_kwargs()
    ax.plot(x, y+2, **plot_kwargs)

@ensure_plotter_consistency(plot_params)
def plot_with_defaults(x,y,**kwargs):

    #Set the function defaults
    plot_params.set_defaults(default_type='function', linewidth=6)

    #Now we process the given arguments
    plot_params.set_parameters(**kwargs)

    #Set up the axis, on which to plot the data
    ax = plot_params.prepare_plot(xlabel='X', ylabel='Y', title='Single Scatterplot')

    #The plot_kwargs provides a way to get the keyword arguments for the
    #actual plotting call to `plot` in this case.
    plot_kwargs = plot_params.plot_kwargs()
```

(continues on next page)

(continued from previous page)

```

ax.plot(x, y, **plot_kwargs)

#The MatplotlibPlotter has a lot of small helper functions
#In this case we just want to set the limits and scale of the
#axis if they were given
plot_params.set_scale(ax)
plot_params.set_limits(ax)

return ax

import numpy as np

x = np.linspace(-1, 1, 10)
y = x**2

nested_plot_function(x, y)
nested_plot_function(x, y, linewidth=1)

```

## 5.1.3 Using the PlotData class

### 5.1.3.1 Description

The *PlotData* class simplifies supporting data to plotting functions in multiple ways, while keeping the plotting functions themselves simple and easy to understand.

The basic idea of *PlotData* is to mimic the behaviour of the data argument in *matplotlib* or the source argument in *bokeh*. Suppose we have our data for a plot in a dictionary *d*, which has the keys *x*, *y1* and *y2*. If we now want to plot both *y* keys against *x* we can do this in the following way.

```

from masci_tools.vis.data import PlotData

plot_data = PlotData(d, x='x', y=['y1', 'y2'])

for entry, source in plot_data.items():
    #entry has the keys needed to get the data from the source
    #and source is the mapping to use

    print(entry.x, entry.y) #Yields x, y1 in the first loop and x, y2 in the second

    #Now we can plot the data
    #for example plt.plot(entry.x, entry.y, data=source)

```

The keys are automatically expanded to be of the same length, if this is possible. There are three iteration modes, with the same names as for dicts:

- **keys:** Yields *namedtuple* with the keys for each plot
- **values:** Yields *namedtuple* with the values corresponding to the keys for each plot
- **items:** Yields the keys and their corresponding mapping for each plot

All of these functions have an argument *first*, which will only return the first element if it is given as *True*.

---

**Note:** The names `x` and `y` in the example above are completely arbitrary. The names for the columns and the fields on the `namedtuple` are determined by the keyword arguments given to `PlotData` at initialization

---



---

**Note:** At the moment the types of mappings accepted in the `PlotData` class are limited to `dict`, `pd.DataFrame` and `ColumnDataSource` (`bokeh`) objects

---

### 5.1.3.2 Initializing PlotData without a mapping

Users might want to provide data directly as arrays. If this should be allowed, there is a function `process_data_arguments()` to allow for this option. This function can either take a `data` argument with a mapping and the same keyword arguments as the `PlotData`.

```
from masci_tools.vis.data import process_data_arguments

plot_data = process_data_arguments(data=d, x='x', y=['y1', 'y2'])
```

Or you can provide the arrays directly without a `data` argument

```
from masci_tools.vis.data import process_data_arguments

#x,y1,y2 are the actual arrays
plot_data = process_data_arguments(x=x, y=[y1,y2])
```

**If no `data` argument is given the keyword arguments are assumed to contain the data and they will be processed according to the**

1. If the data is a multidimensional array (list of lists, etc.) and it is not forbidden by the given argument the first dimension of the array is iterated over and interpreted as separate entries (if the data was previously split up into multiple sets a length check is performed)
2. If the data is a one-dimensional array and of a different length than the number of defined data sets it is added to all previously existing entries
3. If the data is a one-dimensional array and of the same length as the number of defined data sets each entry is added to the corresponding data set

---

**Note:** List or array in this context refers to `list`, `np.array` and `pd.Series`

---

### 5.1.3.3 Available routines on PlotData

There are a couple of routines for mutating/copying or getting information about the data in a `PlotData` instance. These are not meant to be used heavily and should be used for typical simple work done for plot data processing, i.e. scaling, shifting, getting limits, ...

---

**Note:** The term `data key` in the following section refers to the keys of the keyword arguments given to `PlotData` at initialization or the fields on the `namedtuples` returned by iterating over an instance

---

- `get_keys()`: Get all the keys for a given `data key` in a list

- `get_values()`: Get all the values for a given data key in a list
- `min()`: Get the minimum value for a given data key. A mask can be passed to further select the data. If `separate=True` is passed a list of minimum values for each plot is returned
- `max()`: Get the maximum value for a given data key. A mask can be passed to further select the data. If `separate=True` is passed a list of maximum values for each plot is returned
- `apply()`: Apply a lambda function to transform the data of a given data key (in-place!!)
- `get_function_result()`: Apply a function to a given data key and return the results (Does not change the data)
- `sort_data()`: Sort the data by the given data keys
- `group_data()`: Group the data by the given data keys
- `shift_data()`: Shift the data of a given data key either globally or with different shifts for each plot
- `copy_data()`: Copy data to a of one data key to a new data key
- `distinct_datasets()`: Return how many different datasets exist for a given data key

<p><b>Warning:</b> The methods <code>sort_data()</code> and <code>group_data()</code> will always convert the data sources to <code>pd.DataFrame</code> objects if they are not already.</p>
--

## MODULE REFERENCE (API)

### 6.1 Source code Documentation (API reference)

#### 6.1.1 Visualisation and Plotting

##### 6.1.1.1 Fleur specific Plotting

Plotting routine for fleur density of states and bandstructures

```
masci_tools.vis.fleur.plot_fleur_bands(bandsdata, bandsattributes, spinpol=True, only_spin=None,  
                                       backend=None, weight=None, **kwargs)
```

Plot the bandstructure previously extracted from a *banddos.hdf* via the [HDF5Reader](#)

This routine expects datasets and attributes read in with a *FleurBands* recipe from [recipes](#) or something producing equivalent data

##### Parameters

- **bandsdata** – dataset dict produced by the *FleurBands* recipe
- **attributes** – attributes dict produced by the *FleurBands* recipe
- **spinpol** – bool, if True (default) use the plot for spin-polarized bands if the data is spin-polarized
- **only\_spin** – optional str, if given only the specified spin components are plotted
- **backend** – specify which plotting library to use ('matplotlib' or 'bokeh')
- **weight** – str, name of the weight (without spin suffix *\_up* or *\_dn*) you want to emphasize

All other Kwargs are passed on to the underlying plot routines

- Matplotlib: [plot\\_bands\(\)](#), [plot\\_spinpol\\_bands\(\)](#)
- Bokeh: [bokeh\\_bands\(\)](#), [bokeh\\_spinpol\\_bands\(\)](#)

```
masci_tools.vis.fleur.plot_fleur_bands_characterize(bandsdata, bandsattributes, weight_names,  
                                                    weight_colors, spinpol=True, only_spin=None,  
                                                    backend=None, **kwargs)
```

Plot the bandstructure previously extracted from a *banddos.hdf* via the [HDF5Reader](#) with points colored according to the maximum weight from a selection of weights. Can be used to show what character dominates each band

This routine expects datasets and attributes read in with a *FleurBands* recipe from [recipes](#) or something producing equivalent data

### Parameters

- **bandsdata** – dataset dict produced by the *FleurBands* recipe
- **attributes** – attributes dict produced by the *FleurBands* recipe
- **weight\_names** – list of str with the names of the weights that should be considered in the characterization
- **weight\_color** – list of colors associated with each weight. If spin-polarized bandstructures should be shown with different colors the list should be twice as long as the weights
- **spinpol** – bool, if True (default) use the plot for spin-polarized bands if the data is spin-polarized
- **only\_spin** – optional str, if given only the specified spin components are plotted
- **backend** – specify which plotting library to use ('matplotlib' or 'bokeh')

All other Kwargs are passed on to [plot\\_fleur\\_bands\(\)](#)

```
masci_tools.vis.fleur.plot_fleur_dos(dosdata, attributes, spinpol=True, multiply_by_equiv_atoms=True,
                                     plot_keys=None, show_total=True, show_interstitial=True,
                                     show_sym=False, show_atoms='all', show_lresolved=None,
                                     key_mask=None, backend=None, **kwargs)
```

Plot the density of states previously extracted from a *banddos.hdf* via the [HDF5Reader](#)

This routine expects datasets and attributes read in with the *FleurDOS* (Or related DOS modes) recipe from [recipes](#) or something producing equivalent data

The limits for the axes can be specified either with **x** and **y** or **energy** and **dos**. Mixing the two options is not possible

### Parameters

- **dosdata** – dataset dict produced by the *FleurDOS* recipe
- **attributes** – attributes dict produced by the *FleurDOS* recipe
- **spinpol** – bool, if True (default) use the plot for spin-polarized dos if the data is spin-polarized
- **backend** – specify which plotting library to use ('matplotlib' or 'bokeh')

### Arguments for selecting the DOS components to plot:

**param plot\_keys** optional str list of str, defines the labels you want to plot

**param show\_total** bool, if True (default) the total DOS is shown

**param show\_interstitial** bool, if True (default) the interstitial DOS is shown

**param show\_atoms** either 'all', None, or int or list of ints, defines, which total atom projections to show

**param show\_atoms** either 'all', None, or int or list of ints, defines, which total atom projections to show

**param key\_mask** list of bools of the same length as the number of datasets, alternative way to specify, which entries to plot

All other Kwargs are passed on to the underlying plot routines

- Matplotlib: [plot\\_dos\(\)](#), [plot\\_spinpol\\_dos\(\)](#)
- Bokeh: [bokeh\\_dos\(\)](#), [bokeh\\_spinpol\\_dos\(\)](#)

```
masci_tools.vis.fleur.sum_weights_over_atoms(data, attributes, atoms_to_sum, entry_name)
```

Create sums of atom components over specified atoms. They are entered with the same suffixes as in the original data, but with the given entry\_name as prefix

#### Parameters

- **data** – datasets dict produced by the HDF5Reader with a recipe for DOS or bandstructure
- **attributes** – attributes dict produced by the HDF5Reader with a recipe for DOS or bandstructure
- **atoms\_to\_sum** – list of ints for the atoms, which should be summed
- **entry\_name** – str prefix to be entered for the summed entries

**Returns** dict with the summed entries

### 6.1.1.2 KKR specific Plotting

```
masci_tools.vis.kkr_plot_FS_qdos.FSqdos2D(p0='.', totonly=True, s=20, ls_ef=':', lw_ef=1, color='',
                                           reload_data=False, clrbar=True, atoms=[], ax=None,
                                           nosave=False, noalat=False,
                                           cmap=<matplotlib.colors.LinearSegmentedColormap
                                           object>, noplot=False, return_data=False, pclrmesh=False,
                                           logscale=True, ef=None)
```

plotting routine for dos files

dispersionplot function for plotting KKR bandstructures (i.e. qdos) files

```
masci_tools.vis.kkr_plot_bandstruc_qdos.dispersionplot(p0='.', totonly=True, s=20, ls_ef=':',
                                                        lw_ef=1, units='eV_rel', noefline=False,
                                                        color='', reload_data=False, clrbar=True,
                                                        logscale=True, nosave=False, atoms=None,
                                                        ratios=False, atoms2=None, noscale=False,
                                                        newfig=False, cmap=None, alpha=1.0,
                                                        qcomponent=-2, clim=None, xscale=1.0,
                                                        raster=True, atoms3=None,
                                                        alpha_reverse=False, return_data=False,
                                                        xshift=0.0, yshift=0.0, plotmode='pcolor',
                                                        ptitle=None, ef=None,
                                                        as_e_dimension=None,
                                                        scale_alpha_data=False,
                                                        shading='gouraud')
```

plotting routine for qdos files - dispersion (E vs. q)

```
masci_tools.vis.kkr_plot_dos.dosplot(p0='.', totonly=True, color='', label='', marker='', lw=2, ms=5,
                                      ls='-', ls_ef=':', lw_ef=1, units='Ry', noefline=False, interpol=False,
                                      allatoms=False, onespins=False, atoms=[], lmdos=False, lm=[],
                                      nofig=False, scale=1.0, shift=0, normalized=False, xyswitch=False,
                                      efcolor='', return_data=False, xscale=1.0, xshift=0.0, yshift=0.0,
                                      filled=False, spins=2)
```

plotting routine for dos files

```
masci_tools.vis.kkr_plot_shapefun.change_zoom(ax, zoom_range, center=[0, 0, 0])
```

Change the zoom of a 3d plot

**Author** Philipp Ruessmann

#### Parameters

- **ax** – axis which is zoomed
- **zoom\_range** – range to which the image is zoomed, total range from center-zoom\_range to center+zoom\_range
- **center** – center of the zoomed region (optional, defaults to origin)

`masci_tools.vis.kkr_plot_shapefun.plot_shapefun(pos, out, mode)`

Creates a simple matplotlib image to show the shapefunctions given it's positions in the unit cell, the atoms's vertices in *ut* and the plotting mode

**Author** Philipp Ruessmann

**Parameters**

- **pos** – positions of the centers of the cells
- **verts** – array of vertices of the shapefunction (outlines of shapes)
- **mode** – 'all' or 'single' determines whether or not all shapes are combined in a single figure or plotted as individual figures

**Returns** **ax** return the axis in which the plot was done (useful to pass to 'change\_zoom' and 'zoom\_in' functions of this module)

`masci_tools.vis.kkr_plot_shapefun.zoom_in(ax, atm, pos, zoom_range=10)`

Zoom into shapefun of a single atom

**Author** Philipp Ruessmann

**Parameters**

- **ax** – axis in which shapefun plot is found
- **atm** – atom index whose shapefunction is zoomed
- **pos** – array of positions of centers of the shapes (needed to shift center of zommed region to correct atom)
- **zoom\_range** – range of the zoomed region (optional, defaults to 10)

### 6.1.1.3 General Plotting

Here basic functionality is provided for setting default parameters for plotting and ensuring consistent values for these

`masci_tools.vis.parameters.NestedPlotParameters(plotter_object)`

Contextmanager for nested plot function calls Will reset function defaults and parameters to previous values after exiting

**Parameters** **plotter\_object** – Plotter instance

**class** `masci_tools.vis.parameters.Plotter(default_parameters, general_keys=None, key_descriptions=None, **kwargs)`

Base class for handling parameters for plotting methods. For different plotting backends a subclass can be created to represent the specific parameters of the backend.

**Args:**

**param** **default\_parameters** dict with hardcoded default parameters

**param** **general\_keys** set of str optional, defines parameters which are not allowed to change for each entry in the plot data



Kwargs in the `__init__` method are forwarded to `Plotter.set_defaults()` to change the current defaults away from the hardcoded parameters.

The Plotter class creates a hierarchy of dictionaries for lookups on this object utilizing the `ChainMap` from the `collections` module.

**The hierarchy is as follows (First entries take precedence over later entries):**

- *parameters*: set by `set_parameters()` (usually arguments passed into function)
- *user defaults*: set by `set_defaults()`
- *function defaults*: set by `set_defaults()` with `default_type='function'`
- *global defaults*: Hardcoded as fallback

Only the *parameters* can represent parameters for multiple sets of plot calls. All others are used as fallback for specifying non-specified values for single plots

The current parameters can be accessed by bracket indexing the class. A example of this is shown below.

```
parameter_dict = {'fontsize': 16, 'linestyle': '-'}

params = Plotter(parameter_dict)

#Accessing a parameter
print(params['fontsize']) # 16

#Modifying a parameter
params['fontsize'] = 20
print(params['fontsize']) # 20

#Creating a parameter set for multiple plots

#1. Set the properties to the correct values
params.single_plot = False
params.num_plots = 3

#2. Now we can set a property either by providing a list or a integer indexed dict
# Both of the following examples set the linestyle of the second and third plot.
# to '--'
params['linestyle'] = [None, '--', '--']
params['linestyle'] = {1: '--', 2: '--'}

# Not specified values are replaced with the default value for a single plot
print(params['linestyle']) # ['-', '--', '--']

#In lists properties can also be indexed via tuples
print(params[('linestyle', 0)]) # '-'
print(params[('linestyle', 1)]) # '--'

#Changes to the parameters and properties are reset
params.reset_parameters()

print(params['linestyle']) # '-'
```

**add\_parameter**(name, default\_from=None, default\_val=None)

Add a new parameter to the parameters dictionary.

#### Parameters

- **name** – str name of the parameter
- **default\_from** – str (optional), if given a entry is created in the curent defaults with the name and the default value of the key *default\_from*

**static convert\_to\_complete\_list**(*given\_value, single\_plot, num\_plots, default=None, key=""*)

Converts given value to list with length num\_plots with None for the non-specified values

#### Parameters

- **given\_value** – value passed in, for multiple plots either list or dict with integer keys
- **single\_plot** – bool, if True only a single parameter is allowed
- **num\_plots** – int, if single\_plot is False this defines the number of plots
- **default** – default value for unspecified entries
- **key** – str of the key to process

**static dict\_of\_lists\_to\_list\_of\_dicts**(*dict\_of\_lists, single\_plot, num\_plots*)

Converts dict of lists and single values to list of length num\_plots or single dict for single\_plot=True

#### Parameters

- **dict\_of\_lists** – dict to be converted
- **single\_plot** – boolean, if True only a single parameter set is allowed
- **num\_plots** – int of the number of allowed plots

**Returns** list of dicts

**expand\_parameters**(*original\_length, \*\*kwargs*)

Expand parameters to a bigger number of plots. New length has to be a multiple of original length. Only lists of length <= orignal\_length are expanded. Also expands function defaults

#### Parameters

- **original\_length** – int of the old length
- **kwargs** – arguments to expand

**Returns** expanded kwargs

**get\_description**(*key*)

Get the description of the given key

**Parameters** **key** – str of the key, for which the description should be printed

**get\_dict**()

Return the dictionary of the current defaults. For use of printing

**get\_multiple\_kwargs**(*keys, ignore=None*)

Get multiple parameters and return them in a dictionary

#### Parameters

- **keys** – set of keys to process
- **ignore** – str or list of str (optional), defines keys to ignore in the creation of the dict

**is\_general**(*key*)

Return, whether the key is general (meaning only related to the whole plots)

**Parameters** **key** – str of the key to check

**Returns** bool, whether the key is general

**load\_defaults**(*filename*='plot\_defaults.json')

Load defaults from a json file.

**Parameters** **filename** – filename, from where the defaults should be taken

**property num\_plots**

Integer property for number of plots produced

**remove\_added\_parameters**()

Remove the parameters added via `Plotter.add_parameter()`

**reset\_defaults**()

Resets the defaults to the hardcoded defaults in `_PLOT_DEFAULTS`.

**reset\_parameters**()

Reset the parameters to the current defaults. The properties `single_plot` and `num_plots` are also set to default values

**save\_defaults**(*filename*='plot\_defaults.json', *save\_complete*=False)

Save the current defaults to a json file.

**Parameters**

- **filename** – filename, where the defaults should be stored
- **save\_complete** – bool if True not only the overwritten user defaults but also the unmodified hardcoded defaults are stored

**set\_defaults**(*continue\_on\_error*=False, *default\_type*='global', *\*\*kwargs*)

Set the current defaults. This method will only work if the parameters are not changed from the defaults. Otherwise a error is raised. This is because after changing the defaults the changes will be propagated to the parameters to ensure consistency.

**Parameters** **continue\_on\_error** – bool, if True unknown key are simply skipped

**Default\_type** either 'global' or 'function'. Specifies, whether to set the global defaults (not reset after function) or the function defaults

Kwargs are used to set the defaults.

**set\_parameters**(*continue\_on\_error*=False, *\*\*kwargs*)

Set the current parameters.

**Parameters** **continue\_on\_error** – bool, if True unknown key are simply skipped and returned

Kwargs are used to set the defaults.

**set\_single\_default**(*key*, *value*, *default\_type*='global')

Set default value for a single key/value pair

**Parameters**

- **key** – str of the key to set
- **value** – value to set the key to

**Default\_type** either 'global' or 'function'. Specifies, whether to set the global defaults (not reset after function) or the function defaults

**property single\_plot**

Boolean property if True only a single Plot parameter set is allowed

`maschi_tools.vis.parameters.ensure_plotter_consistency(plotter_object)`

Decorator for plot functions to ensure that the Parameters are reset even if an error occurs in the function. Additionally checks are performed that the parameters are reset after execution and the defaults are never changed in a plot function

**Parameters** `plotter_object` – Plotter instance to be checked for consistency

This module contains classes and functions to make plotting functions more flexible with respect to the used data. This way plotting functions can both allow the flexible usage of lists, arrays directly or dataframes together with the keys that should be used

**class** `maschi_tools.vis.data.ColumnDataSourceWrapper(wrapped)`

Wrapper around `bokeh.models.ColumnDataSource` to give it a `__getitem__` and `__setitem__` method

Used in the `PlotDataIterator` for easier handling of these types

**class** `maschi_tools.vis.data.PlotData(data, use_column_source=False, same_length=False, strict_data_keys=False, copy_data=False, **kwargs)`

Class for iterating over the data in a dict or dataframe with automatic filling in of single defined keys to get a list of keys to extract.

The iteration allows for implicit definition of data for multiple plot sets, without excessive copying of the given data

Usage Example

```
from maschi_tools.vis.data import PlotData
import numpy as np

#Let's say we have one energy grid and a couple of functions
#defined on this energy grid.
#We collect these in a dict

x = np.linspace(-10,10,100)
data = {'x': x, 'y1': np.sin(x), 'y2': np.cos(x), 'y3', x**2}

p = PlotData(data, x='x', y=['y1', 'y2', 'y3'])

#If we now iterate over this object it will result in the data
#for y being returned together with the x data (The same would work the other way_
↪around)
for entry in p:
    print(entry.x) #x' entry
    print(entry.y) #y1' then 'y2' and finally 'y3'

#Additionally data for z, color and size can be defined
```

**Parameters** `data` – object or list of objects which can be bracket indexed with the given keys e.g. dicts, pandas dataframes, ...

**Same\_length** bool if True and any sources are dicts it will be checked for same dimensions in (ALL) entries (not only for keys plotted against each other)

**Strict\_data\_keys** bool if True no new data keys are allowed to be entered via `copy_data()`

Kwargs are used to specify the columns in a namedtuple. If a list is given for any of the keys the data will be expanded to a list of namedtuple with the same length

**add\_data\_key**(*data\_key*, *keys=None*)

Add a new column of data keys

#### Parameters

- **data\_key** – string of the new data key to add
- **keys** – None, Index into data or list of index into the data to initialize the values to

**apply**(*data\_key*, *lambda\_func*, *apply\_to\_whole\_array=True*, *\*\*kwargs*)

Apply a function to a given data column for all entries

**Warning:** This operation is done in-place. Meaning if there are multiple data entries pointing to the same data set and only one should be modified by this method, the data needs to be copied beforehand using `copy_data()`

#### Parameters

- **data\_key** – name of the data key to apply the function
- **lambda\_func** – function to apply to the data

**copy\_data**(*data\_key\_from*, *data\_key\_to*, *prefix=None*, *rename\_original=False*, *force=False*)

Copy the data for a given data key to another one

#### Parameters

- **data\_key\_from** – data key to copy from
- **data\_key\_to** – data key to copy to
- **prefix** – optional prefix to use for the renamed data entries. Can be used to avoid name clashes. If not given the data keys are used
- **rename\_original** – optional bool (default False). If True the original entries are renamed instead of the ones under `data_key_to`

**property data\_keys**

Return the registered data keys for this instance

**distinct\_datasets**(*data\_key*)

Return how many different data sets are present for the given data key

**Parameters** **data\_key** – The data key to analyse

**Returns** int of the number of different datasets

**get\_function\_result**(*data\_key*, *func*, *list\_return=False*, *as\_numpy\_array=False*, *\*\*kwargs*)

Apply a function to a given data column for all entries and return the results

#### Parameters

- **data\_key** – name of the data key to apply the function to
- **func** – function to apply to the data to get the results if func is a string then it will be used to get the attribute with the corresponding name from the source and call it

**get\_keys**(*data\_key*)

Get the keys for a given data column for all entries

**Parameters** **data\_key** – name of the data key to return the keys

**Returns** list of keys, corresponding to the entries for the given data in the sources

**get\_mask**(*mask*, *data\_key=None*)

Get mask list for use with the Data in this instance

**Parameters**

- **mask** – either list of callable, if it is callable it is used in `get_function_result()` together with the `data_key` argument
- **data\_key** – str to be used for the data key if mask is a callable

:param

**get\_values**(*data\_key*)

Get the values for a given data column for all entries

**Parameters** **data\_key** – name of the data key to return the values

**Returns** list of values, corresponding to the entries for the given data in the sources

**group\_data**(*by*, *\*\*kwargs*)

Group the data by the given data\_key(s) or other arguments for groupby

---

**Note:** This function will convert the data arguments to `pd.DataFrame` objects

---

**Parameters** **by** – str or list of str of the data\_keys to sort by or other valid arguments for `by` in `pd.DataFrame.groupby()`

Kwargs are passed on to `pd.DataFrame.groupby()`

**items**(*first=False*, *mappable=False*)

Iterate over PlotData items. Returns the key and corresponding source for the data

**Parameters**

- **first** – bool, if True only the first entry is returned
- **mappable** – bool, if True only the data `ColumnDataSources` are wrapped to be mappable

**keys**(*first=False*)

Iterate over PlotData keys. Returns the keys for the corresponding sources

**Parameters** **first** – bool, if True only the first entry is returned

**mask\_data**(*mask*, *data\_key=None*)

Apply a given mask to the data inplace.

---

**Note:** This function will convert the data arguments to `pd.DataFrame` objects

---

**Parameters**

- **mask** – mask rgument passed to `get_mask()`
- **data\_key** – data\_key argument used by `get_mask()`

**max**(*data\_key*, *separate=False*, *mask=None*, *mask\_data\_key=None*)

Get the maximum value for a given data column for all entries

**Parameters**

- **data\_key** – name of the data key to determine the maximum

- **separate** – bool if True the maximum will be determined and returned for all entries separately
- **mask** – optional mask to select specific rows from the data entries
- **mask\_data\_key** – optional data key to be used when **mask** is a function

**Returns** maximum value for all entries either combined or as a list

**min**(*data\_key*, *separate=False*, *mask=None*, *mask\_data\_key=None*)

Get the minimum value for a given data column for all entries

**Parameters**

- **data\_key** – name of the data key to determine the minimum
- **separate** – bool if True the minimum will be determined and returned for all entries separately
- **mask** – optional mask to select specific rows from the data entries
- **mask\_data\_key** – optional data key to be used when **mask** is a function

**Returns** minimum value for all entries either combined or as a list

**shift\_data**(*data\_key*, *shifts*, *shifted\_data\_key=None*, *separate\_data=True*, *negative=False*)

Apply shifts to a given data column for all entries

**Parameters**

- **data\_key** – name of the data key to shift
- **shifts** – float or array of floats with the shifts to apply
- **shifted\_data\_key** – optional string, if given the data will be copied to this data key
- **separate\_data** – bool, if True and **shifted\_data\_key** is not given the data will be copied to itself (This separates the data for all columns)
- **negative** – bool if True the shifts are applied with a minus sign

**sort\_data**(*by\_data\_keys*, *\*\*kwargs*)

Sort the data by the given data\_key(s)

---

**Note:** This function will convert the data arguments to `pd.DataFrame` objects

---



---

**Note:** If there are multiple plot sets and only one data source. This function will expand the data to be one data source sorted according to the `data_keys` for each plot

---

**Parameters** **by\_data\_keys** – str or list of str of the `data_keys` to sort by

Kwargs are passed on to `pd.DataFrame.sort_values()`

**values**(*first=False*)

Iterate over `PlotData` values. Returns the values for the data

**Parameters** **first** – bool, if True only the first entry is returned

**class** `masci_tools.vis.data.PlotDataIterator`(*plot\_data*, *mode='values'*, *mappable=False*)

Class containing the iteration behaviour over the `PlotData` class. Can be used in three modes:

- *keys*: Returns the keys to be entered in the corresponding data sources for each entry

- *values*: Returns the data for each entry
- *items*: Returns the keys and the data sources in a tuple

The keys and values are always returned in a `namedtuple` with fields corresponding to the set data keys

```
masci_tools.vis.data.normalize_list_or_array(data, key, out_data, flatten_np=False,
                                             forbid_split_up=False)
```

Split up a given list/numpy array or pd.Series to be used in the plotting methods

#### Parameters

- **data** – The (array-like) data to be normalized
- **key** – key under which to enter the new data
- **out\_data** – dict containing previously normalized data
- **flatten\_np** – bool, if True multidimensional numpy arrays are flattened
- **forbid\_split\_up** – bool, if True multidimensional arrays are not split up

#### The rules are the following:

- if **data** is a multidimensional array (list of lists, etc.) and it is not forbidden by the given argument the first dimension of the array is iterated over and interpreted as separate entries (if the data was previously split up into multiple sets a length check is performed)
- if **data** is a one-dimensional array and of a different length than the number of defined data sets it is added to all previously existing entries
- if **data** is a one-dimensional array and of the same length as the number of defined data sets each entry is added to the corresponding data set

**Returns** list of dicts or dict containing the normalized data

```
masci_tools.vis.data.process_data_arguments(data=None, single_plot=False, use_column_source=False,
                                             flatten_np=False, forbid_split_up=None,
                                             same_length=False, copy_data=False, **kwargs)
```

Initialize PlotData from np.arrays or lists of np.arrays or lists or a already given data argument, i.e. mapping

#### Parameters

- **data** – either None or Mapping to be used as the data in the PlotData class
- **single\_plot** – bool, if True only a single dataset is allowed
- **use\_column\_source** – bool, if True all data arguments are converted to ColumnDataSource of bokeh
- **flatten\_np** – bool, if True multidimensional numpy arrays are flattened (Only if data not given)
- **forbid\_split\_up** – set of keys for which not to split up multidimensional arrays
- **same\_length** – bool if True and any sources are dicts it will be checked for same dimensions in (ALL) entries (not only for keys plotted against each other)
- **copy\_data** – bool, if True the data argument will be copied

Kwargs define which keys belong to which data entries if data is given or they contain the data to be normalized

The following two example calls will both create a PlotData object with the same two plot data sets with the entries **x** and **y**:



```
import numpy as np

x = np.linspace(-10,10,100)
y1 = y**2
y2 = np.sin(x)

#Use a predefined data argument (a dict in this case) and the keys in the kwargs
p = process_data_arguments({'x': x, 'y1': y1, 'y2': y2}, x='x', y=['y1','y2'])

#Let the function normalize the given arrays
p = process_data_arguments=(x=x,y=[y1, y2])
```

**Returns** A *PlotData* object corresponding to the given data

This modules provides common plotting functions dispatching to different plotting backends. At the moment the following backends are used:

- matplotlib ('mpl', 'matplotlib')
- bokeh ('bokeh')

The underlying plotting routines collected here should have the same signature for the data arguments; keyword arguments can be different.

**class** maschi\_tools.vis.common.*PlotBackend*(value)

Enumeration containing the possible names for each plotting backend Initialize using the *from\_str()* method

At the moment the following are supported (case-insensitive)

- matplotlib: either 'mpl' or 'matplotlib'
- bokeh: 'bokeh'

**static default()**

Return a *PlotBackend* instance corresponding to the current default backend

**static from\_str(label)**

Initialize the *PlotBackend* from a given string

**Parameters** *label* – str to use to initialize the backend if it is *None* the default is returned

**Returns** *PlotBackend* instance corresponding to the label

maschi\_tools.vis.common.**bands**(kpath, eigenvalues, backend=None, data=None, \*\*kwargs)

Plot the provided data for a bandstructure (non spin-polarized) Non-weighted, weighted, as a line plot or scatter plot, color-mapped or fixed colors are all possible options

**Parameters**

- **kpath** – data for the kpoints path (flattened to 1D)
- **eigenvalues** – data for the eigenvalues
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: *plot\_bands()*
- bokeh: *bokeh\_bands()*

**Returns** Figure object for the used plotting backend

`masci_tools.vis.common.dos(energy_grid, dos_data, backend=None, data=None, **kwargs)`

Plot the provided data as a density of states (not spin-polarized). Can be done horizontally or vertical via the switch `xyswitch`

**Parameters**

- **energy\_grid** – data for the energy grid of the DOS
- **dos\_data** – data for all the DOS components to plot
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: `plot_dos()`
- bokeh: `bokeh_dos()`

**Returns** Figure object for the used plotting backend

`masci_tools.vis.common.get_help(key, backend=None)`

Get a help string for a given parameter.

**Parameters**

- **key** – name of the parameter to get the parameter for
- **backend** – For which backend to get the description of the parameter

`masci_tools.vis.common.get_plotter(backend=None)`

Get the instance of the `Plotter` subclass used for the given plotting backend

**Parameters** **backend** – For which backend to get the Plotter instace

`masci_tools.vis.common.line(xdata, ydata, backend=None, data=None, **kwargs)`

Plot the provided data as a line plot. Multiple data sets are possible

**Parameters**

- **xdata** – data for the x-axis
- **ydata** – data for the y-axis
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: `multiple_scatterplots()`
- bokeh: `bokeh_line()`

**Returns** Figure object for the used plotting backend

`masci_tools.vis.common.load_defaults(backend=None, filename='plot_defaults.json')`

Load defaults for the plot parameters from a file and set the contained defaults.

**Parameters**

- **backend** – For which backend to save the parameters

- **filename** – str of the filename to load the defaults from

`masci_tools.vis.common.reset_defaults(backend=None)`  
Reset the defaults for theplot parameters to the original state.

**Parameters** **backend** – For which backend to reset the parameters

`masci_tools.vis.common.save_defaults(backend=None, filename='plot_defaults.json', save_complete=False)`

Save the defaults for the plot parameters.

**Parameters**

- **backend** – For which backend to save the parameters
- **filename** – str of the filename to save the defaults to
- **save\_complete** – bool, if True also the hardcoded defaults are included

`masci_tools.vis.common.scatter(xdata, ydata, backend=None, data=None, **kwargs)`  
Plot the provided data as a scatter plot. Varying size and color are possible. Multiple data sets are possible

**Parameters**

- **xdata** – data for the x-axis
- **ydata** – data for the y-axis
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: `multi_scatter_plot()`
- bokeh: `bokeh_multi_scatter()`

**Returns** Figure object for the used plotting backend

`masci_tools.vis.common.set_default_backend(backend)`  
Sets the default backend used when no explicit backend is specified.

**Parameters** **backend** – Name of the backend to use

`masci_tools.vis.common.set_defaults(backend=None, **kwargs)`  
Sets defaults for the plot parameters.

**Parameters** **backend** – For which backend to set the parameters

The Kwargs are used to set the parameters of the specified backend

`masci_tools.vis.common.show_defaults(backend=None)`  
Show the current set defaults for the plot parameters.

**Parameters** **backend** – For which backend to show the parameters

`masci_tools.vis.common.spinpol_bands(kpath, eigenvalues_up, eigenvalues_dn, backend=None, data=None, **kwargs)`

Plot the provided data for a bandstructure (spin-polarized) Non-weighted, weighted, as a line plot or scatter plot, color-mapped or fixed colors are all possible options

**Parameters**

- **kpath** – data for the kpoints path (flattened to 1D)
- **eigenvalues\_up** – data for the eigenvalues for spin-up

- **eigenvalues\_dn** – data for the eigenvalues for spin-down
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: `plot_spinpol_bands()`
- bokeh: `bokeh_spinpol_bands()`

**Returns** Figure object for the used plotting backend

```
masci_tools.vis.common.spinpol_dos(energy_grid, dos_data_up, dos_data_dn, backend=None, data=None,
                                   **kwargs)
```

Plot the provided data as a density of states (spin-polarized). Can be done horizontally or vertical via the switch *xyswitch*

#### Parameters

- **energy\_grid** – data for the energy grid of the DOS
- **dos\_data\_up** – data for all the DOS components to plot for spin-up
- **dos\_data\_dn** – data for all the DOS components to plot for spin-down
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **backend** – name of the backend to use (uses a default if None is given)

Kwargs are passed on to the backend plotting functions:

- matplotlib: `plot_spinpol_dos()`
- bokeh: `bokeh_spinpol_dos()`

**Returns** Figure object for the used plotting backend

## Matplotlib

This module contains a subclass of *Plotter* for the matplotlib library

```
class masci_tools.vis.matplotlib_plotter.MatplotlibPlotter(**kwargs)
```

Class for plotting parameters and standard code snippets for plotting with the matplotlib backend.

Kwargs in the `__init__` method are forwarded to setting default values for the instance

For specific documentation about the parameter/defaults handling refer to *Plotter*.

Below the current defined default values are shown:

Table 1: Plot Parameters

Name	Description	Default value
<code>title_fontsize</code>	Fontsize for the title of the figure	16

continues on next page

Table 1 – continued from previous page

Name	Description	Default value
<code>figure_kwargs</code>	Arguments passed to <code>plt.figure</code> when creating the figure. Includes things like <code>figsize</code> , <code>dpi</code> , background color, ...	<code>{'figsize': (8, 6), 'dpi': 100, 'facecolor': 'w', 'edgecolor': 'k', 'constrained_layout': False}</code>
<code>alpha</code>	Float specifying the transparency of the title	1
<code>axis_linewidth</code>	Linewidth of the lines for the axis	1.5
<code>use_axis_formatter</code>	If True the labels will always not be formatted with an additive constant at the top	False
<code>set_powerlimit</code>	If True the threshold for switching to scientific notation is adjusted to 0,3	True
<code>xticks</code>	Positions of the ticks on the x axis	No Default
<code>xticklabels</code>	Labels for the ticks on the x-axis	No Default
<code>yticks</code>	Positions for the ticks on the y-axis	No Default
<code>yticklabels</code>	Labels for the ticks on the y-axis	No Default
<code>invert_xaxis</code>	If True the direction of the x-axis is inverted	False
<code>invert_yaxis</code>	If True the direction of the y-axis is inverted	False
<code>color_cycle</code>	If set this will override the default color cycle of matplotlib. Can be given as name of a colormap cycle or list of colors	No Default
<code>sub_colormap</code>	If a colormap is used this can be used to cut out a part of the colormap. For example (0.5,1.0) will only use the upper half of the colormap	No Default
<code>linewidth</code>	Linewidth for the plot(s)	2.0
<code>linestyle</code>	Linestyle for the plot(s)	-
<code>marker</code>	Shape of the marker to use for the plot(s)	o
<code>markersize</code>	Size of the markers to use in the plot(s)	4.0
<code>color</code>	Color to use in the plot(s)	No Default
<code>zorder</code>	z-position to use for the plot(s) (Is used to define fore- and background)	No Default
<code>repeat_colors</code>	If True the colors will be repeated after the given number of plots. Only implemented for <code>multiple_scatterplots</code>	No Default
<code>edgecolor</code>	Edgecolor to use in the plot(s)	No Default
<code>facecolor</code>	Facecolor to use in the plot(s)	No Default
<code>plot_label</code>	Label to use in the plot(s) for the legend	No Default
<code>area_plot</code>	If True <code>fill_between(x)</code> will be used to produce the plot(s)	False
<code>area_vertical</code>	Determines, whether to use <code>fill_between</code> or <code>fill_betweenx</code> for area plots	False
<code>area_enclosing</code>	If True an enclosing line will be drawn around the area	True
<code>area_alpha</code>	Transparency to use for the area in the area plot(s)	1.0
<code>area_linecolor</code>	Color for the enclosing line in the area plot(s)	No Default
<code>plot_alpha</code>	Transparency to use for the plot(s)	1.0
<code>cmap</code>	Colormap to use for scatter/pcolormesh or 3D plots	viridis
<code>norm</code>	If set this norm will be used to normalize data for the colormap-ping	No Default
<code>shading</code>	Shading to use for pcolormesh plots	gouraud
<code>rasterized</code>	Rasterize the pcolormesh when drawing vector graphics.	True

continues on next page

Table 1 – continued from previous page

Name	Description	Default value
scale	Dict specifying the scales of the axis, e.g {'y': 'log'} will create a logarithmic scale on the y-axis	No Default
limits	Dict specifying the limits of the axis, e.g {'x': (-5,5)}	No Default
labelfontsize	Fontsize for the labels on the axis	15
lines	Dict specifying straight help-lines to draw. For example {'vertical': 0, 'horizontal': [-1,1]} will draw a vertical line at 0 and two horizontal at -1 and 1	No Default
line_options	Color, width, and more options for the help-lines	<pre>{'linestyle': '-- →', 'color': 'k', 'linewidth': 1.0}</pre>
font_options	Default font options that can be used for text annotations	<pre>{'family': 'serif →', 'color': 'black', 'weight': 'normal →', 'size': 16}</pre>
tick_paramsx	Parameters for major ticks on the x-axis (Size, fontsize, ...)	<pre>{'size': 4.0, 'width': 1.0, 'labelsize': 14, 'length': 5, 'labelrotation': ↵ →0}</pre>
tick_paramsy	Parameters for major ticks on the y-axis (Size, fontsize, ...)	<pre>{'size': 4.0, 'width': 1.0, 'labelsize': 14, 'length': 5, 'labelrotation': ↵ →0}</pre>
tick_paramsx_minor	Parameters for minor ticks on the x-axis (Size, fontsize, ...)	<pre>{'size': 2.0, 'width': 1.0, 'labelsize': 0, 'length': 2.5}</pre>
tick_paramsy_minor	Parameters for minor ticks on the y-axis (Size, fontsize, ...)	<pre>{'size': 2.0, 'width': 1.0, 'labelsize': 0, 'length': 2.5}</pre>
colorbar	If True and the function implements color mapping, a colorbar is shown	True

continues on next page

Table 1 – continued from previous page

Name	Description	Default value
colorbar_padding	Specifies the space between plot and colorbar	0.1
legend	If True a legend for the plot is shown	False
legend_show_dataframe_labels	If True the column names from the data argument are shown if not overwritten	False
legend_remove_duplicates	If True duplicate legend labels are removed	False
legend_options	Parameters for displaying the legend (Fontsize, location, ...)	<code>{'fontsize': 'large', 'linewidth': 3.0, 'loc': 'best', 'fancybox': True}</code>
save_plots	if True the plots will be saved to file	False
save_format	Formats to save the plots to, can be single or list of formats	png
save_options	Additional options for saving the plots to file	<code>{'transparent': True}</code>
tightlayout	If True the tight layout will be used (NOT IMPLEMENTED)	False
show	If True plt.show will be called at the end of the routine	True
save_raw_plot_data	If True the data for the plot is saved to file (NOT IMPLEMENTED)	False
raw_plot_data_format	Format in which to save the data for the plot (NOT IMPLEMENTED)	txt

### draw\_lines(ax)

Draw horizontal and vertical lines specified in the lines argument

**Parameters** **ax** – Axes object on which to perform the operation

**plot\_kwargs** (*ignore=None, extra\_keys=None, plot\_type='default', post\_process=True, list\_of\_dicts=True, \*\*kwargs*)

Creates a dict or list of dicts (for multiple plots) with the defined parameters for the plotting calls for matplotlib

#### Parameters

- **ignore** – str or list of str (optional), defines keys to ignore in the creation of the dict
- **extra\_keys** – optional set for additional keys to retrieve
- **post\_process** – bool, if True the parameters are cleaned up for inserting them directly into matplotlib plotting functions

Kwargs are used to replace values by custom parameters:

Example for using a custom markersize:

```
p = MatplotlibPlotter()
p.add_parameter('marker_custom', default_from='marker')
p.plot_kwargs(marker='marker_custom')
```

This code snippet will return the standard parameters for a plot, but the value for the marker will be taken from the key *marker\_custom*

**prepare\_plot**(*title=None, xlabel=None, ylabel=None, zlabel=None, axis=None, minor=False, projection=None*)

Prepares the figure of a matplotlib plot, setting the labels/titles, ticks, ...

**Parameters**

- **title** – str for the title of the figure
- **xlabel** – str for the label on the x-axis
- **ylabel** – str for the label on the y-axis
- **zlabel** – str for the label on the z-axis
- **axis** – matplotlib axes object, optional, if given the operations are performed on the object otherwise a new figure and subplot are created
- **minor** – bool, if True minor tick parameters are set
- **projection** – str, passed on to the add\_subplot call

**Returns** the created or modified axis object

**save\_plot**(*saveas*)

Save the current figure or show the current figure

**Parameters** **saveas** – str, filename for the resulting file

**set\_limits**(*ax*)

Set limits of the axis

**Parameters** **ax** – Axes object on which to perform the operation

**set\_scale**(*ax*)

Set scale of the axis (for example 'log')

**Parameters** **ax** – Axes object on which to perform the operation

**show\_colorbar**(*ax*)

Print a colorbar for the plot

**Parameters** **ax** – Axes object on which to perform the operation

**show\_legend**(*ax, leg\_elems=None*)

Print a legend for the plot

**Parameters** **ax** – Axes object on which to perform the operation

**static truncate\_colormap**(*cmap, minval=0.0, maxval=1.0, n=256*)

Cut off parts of colormap

**Parameters**

- **cmap** – cmap to truncate
- **minval** – minimum value of new colormap
- **maxval** – maximum value of new colormap
- **n** – number of colors in new colormap

**Returns** colormap truncated to only hold colors between minval and maxval from old colormap

In this module are plot routines collected to create default plots out of certain output nodes from certain workflows with matplotlib lib.



Comment: Do not use any aiida methods, otherwise the methods in here can become tricky to use inside a virtual environment. Make the user extract thing out of aiida objects before hand or write something on top. Since usually parameter nodes, or files are plotted, parse a dict or filepath.

Each of the plot\_methods can take keyword arguments to modify parameters of the plots There are keywords that are handled by a special class for defaults. All other arguments will be passed on to the matplotlib plotting calls

For the definition of the defaults refer to [MatplotlibPlotter](#)

```
masci_tools.vis.plot_methods.CDF_voigt_profile(x, fwhm_g, fwhm_l, mu)
```

Cumulative distribution function of a voigt profile implementation of formula found here: [https://en.wikipedia.org/wiki/Voigt\\_profile](https://en.wikipedia.org/wiki/Voigt_profile) # TODO: is there an other way then to calc 2F2? # or is there an other way to calc the integral of wofz directly, or use different error functions.

```
class masci_tools.vis.plot_methods.PDF(pdf, size=(200, 200))
```

Display a PDF file inside a Jupyter notebook.

```
masci_tools.vis.plot_methods.asymmetric_lorentz(x, fwhm, mu, alpha=1.0, beta=1.5)
```

asymetric lorentz function

$L^\alpha$  for  $x \leq \mu$   $L^\beta$  for  $x > \mu$  See casexps LA

```
masci_tools.vis.plot_methods.asymmetric_lorentz_gauss_conv(x, mu, fwhm_l, fwhm_g, alpha=1.0,
                                                            beta=1.5)
```

asymmetric Lorentzian with Gauss convoluted

```
masci_tools.vis.plot_methods.asymmetric_lorentz_gauss_sum(x, mu, fwhm_l, fwhm_g, alpha=1.0,
                                                           beta=1.5)
```

asymmetric Lorentzian with Gauss convoluted

```
masci_tools.vis.plot_methods.barchart(positions, heights, *, width=0.35, xlabel='x', ylabel='y', title='',
                                       bottom=None, alignment='vertical', saveas='barchart',
                                       bar_type='stacked', axis=None, xerr=None, yerr=None,
                                       data=None, copy_data=False, **kwargs)
```

Create a standard bar chart plot (this should be flexible enough) to do all the basic bar chart plots.

#### Parameters

- **positions** – arraylike data for the positions of the bars
- **heights** – arraylike data for the heights of the bars
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **width** – float determines the width of the bars
- **axis** – Axes object where to add the plot
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **xerr** – optional data for errorbar in x-direction
- **yerr** – optional data for errorbar in y-direction
- **bottom** – bottom values for the lowest end of the bars
- **bar\_type** – type of the barchart plot. Either stacked, grouped or independent
- **alignment** – which direction the bars should be plotted (horizontal or vertical)

- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function `bar`

TODO: grouped barchart (meaning not stacked)

```
masci_tools.vis.plot_methods.colormesh_plot(xdata, ydata, cdata, *, xlabel="", ylabel="", title="",
                                             data=None, saveas='colormesh', axis=None,
                                             copy_data=False, **kwargs)
```

Create plot with pcolormesh

#### Parameters

- **xdata** – arraylike, data for the x coordinate
- **ydata** – arraylike, data for the y coordinate
- **cdata** – arraylike, data for the color values with a colormap
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **title** – str, title of the figure
- **saveas** – str specifying the filename (without file format)
- **axis** – Axes object, if given the plot will be applied to this object
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function `pcolormesh`

```
masci_tools.vis.plot_methods.construct_corelevel_spectrum(coreleveldict, natom_typesdict,
                                                         exp_references=None, scale_to=-1,
                                                         fwhm_g=0.6, fwhm_l=0.1,
                                                         energy_range=None, xspec=None,
                                                         energy_grid=0.2, peakfunction='voigt',
                                                         alpha_l=1.0, beta_l=1.5)
```

Constructs a corelevel spectrum from a given corelevel dict

#### Params

**Returns** list: [xdata\_spec, ydata\_spec, ydata\_single\_all, xdata\_all, ydata\_all, xdata\_label]

```
masci_tools.vis.plot_methods.default_histogram(*args, **kwargs)
```

Create a standard looking histogram (DEPRECATED)

```
masci_tools.vis.plot_methods.doniach_sunjic(x, scale=1.0, E_0=0, gamma=1.0, alpha=0.0)
```

Doniach Sunjic asymmetric peak function. tail to higher binding energies.

param x: list values to evaluate this function  
 param scale: multiply the function with this factor  
 param E\_0: position of the peak  
 param gamma, 'lifetime' broadening  
 param alpha: 'asymmetry' parameter

See Doniach S. and Sunjic M., J. Phys. 4C31, 285 (1970) or [http://www.casaxps.com/help\\_manual/line\\_shapes.htm](http://www.casaxps.com/help_manual/line_shapes.htm)

```
masci_tools.vis.plot_methods.gauss_one(x, fwhm, mu)
```

Returns a Lorentzian line shape at x with FWHM fwhm and mean mu

```
masci_tools.vis.plot_methods.gaussian(x, fwhm, mu)
```

Returns Gaussian line shape at x with FWHM fwhm and mean mu

```
masci_tools.vis.plot_methods.get_mpl_help(key)
```

Print the decription of the given key in the matplotlib backend

Available defaults can be seen in [MatplotlibPlotter](#)

```
masci_tools.vis.plot_methods.histogram(xdata, density=False, histtype='bar', align='mid',
                                     orientation='vertical', log=False, axis=None, title='hist',
                                     xlabel='bins', ylabel='counts', saveas='histogram',
                                     return_hist_output=False, data=None, copy_data=False,
                                     **kwargs)
```

Create a standard looking histogram

#### Parameters

- **xdata** – arraylike, Data for the histogram
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **density** – bool, if True the histogram is normed and a normal distribution is plotted with the same mu and sigma as the data
- **histtype** – str, type of the histogram
- **align** – str, defines where the bars for the bins are aligned
- **orientation** – str, is the histogram vertical or horizontal
- **log** – bool, if True a logarithmic scale is used for the counts
- **axis** – Axes object where to add the plot
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **return\_hist\_output** – bool, if True the data output from hist will be returned
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to [masci\\_tools.vis.matplotlib\\_plotter.MatplotlibPlotter](#). If the arguments are not recognized they are passed on to the matplotlib function *hist*

```
masci_tools.vis.plot_methods.hyp2f2(a, b, z)
```

Calculation of the  ${}_2F_2()$  hypergeometric function, since it is not part of scipy with the identity 2. from here: [https://en.wikipedia.org/wiki/Generalized\\_hypergeometric\\_function](https://en.wikipedia.org/wiki/Generalized_hypergeometric_function) a, b,z array like inputs TODO: not clear to me how to do this... the identity is only useful if we mange the adjust the arguments in a way that we can use them... also maybe go for the special case we need first: 1,1,3/2;2;-z2

```
masci_tools.vis.plot_methods.load_mpl_defaults(filename='plot_mpl_defaults.json')
```

Load defaults for the matplotlib backend from a json file.

**Parameters** **filename** – filename,from where the defaults should be taken

```
masci_tools.vis.plot_methods.lorentzian(x, fwhm, mu)
```

Returns a Lorentzian line shape at x with FWHM fwhm and mean mu

```
masci_tools.vis.plot_methods.lorentzian_one(x, fwhm, mu)
```

Returns a Lorentzian line shape at x with FWHM fwhm and mean mu

```
masci_tools.vis.plot_methods.multi_scatter_plot(xdata, ydata, *, size_data=None, color_data=None,
                                                xlabel="", ylabel="", title="", data=None,
                                                saveas='mscatterplot', axis=None, copy_data=False,
                                                exclude_points_outside_plot_area=False, **kwargs)
```

Create a scatter plot with varying marker size Info: x, y, size and color data must have the same dimensions.

#### Parameters

- **xdata** – str or arraylike, data for the x coordinate
- **ydata** – str or arraylike, data for the y coordinate
- **size\_data** – str or arraylike, data for the markersizes (optional)
- **color\_data** – str or arraylike, data for the color values with a colormap (optional)
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **title** – str, title of the figure
- **data** – Mapping giving the data for the plots (required if data arguments are str)
- **saveas** – str specifying the filename (without file format)
- **axis** – Axes object, if given the plot will be applied to this object
- **xerr** – optional data for errorbar in x-direction
- **yerr** – optional data for errorbar in y-direction
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to [`masci\_tools.vis.matplotlib\_plotter.MatplotlibPlotter`](#). If the arguments are not recognized they are passed on to the matplotlib function `scatter`

```
masci_tools.vis.plot_methods.multiaxis_scatterplot(xdata, ydata, *, axes_loc, xlabel="", ylabel="",
                                                    title="", num_cols=1, num_rows=1,
                                                    saveas='mscatterplot', **kwargs)
```

Create a scatter plot with multiple axes.

#### Parameters

- **xdata** – list of arraylikes, passed on to the plotting functions for each axis (x-axis)
- **ydata** – list of arraylikes, passed on to the plotting functions for each axis (y-axis)
- **axes\_loc** – list of tuples of two integers, location of each axis
- **xlabel** – str or list of str, labels for the x axis
- **ylabel** – str or list of str, labels for the y-axis
- **title** – str or list of str, titles for the subplots
- **num\_rows** – int, how many rows of axis are created
- **num\_cols** – int, how many columns of axis are created
- **saveas** – str filename of the saved file

#### Special Kwargs:

**param subplot\_params** dict with integer keys, can contain all valid kwargs for `multiple_scatterplots()` with the integer key denoting to which subplot the changes are applied

**param axes\_kwargs** dict with integer keys, additional arguments to pass on to `subplot2grid` for the creation of each axis (e.g colspan, rowspan)

Other Kwargs will be passed on to all `multiple_scatterplots()` calls (If they are not overwritten by parameters in `subplot_params`).

```
masci_tools.vis.plot_methods.multiple_scatterplots(xdata, ydata, *, xlabel="", ylabel="", title="",
                                                    data=None, saveas='mscatterplot', axis=None,
                                                    xerr=None, yerr=None, area_curve=0,
                                                    copy_data=False,
                                                    exclude_points_outside_plot_area=False,
                                                    **kwargs)
```

Create a standard scatter plot with multiple sets of data (this should be flexible enough) to do all the basic plots.

### Parameters

- **xdata** – str or arraylike, data for the x coordinate
- **ydata** – str or arraylike, data for the y coordinate
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, title of the figure
- **data** – Mapping giving the data for the plots (required if xdata and ydata are str)
- **saveas** – str specifying the filename (without file format)
- **axis** – Axes object, if given the plot will be applied to this object
- **xerr** – optional data for errorbar in x-direction
- **yerr** – optional data for errorbar in y-direction
- **area\_curve** – if an area plot is made this arguments defines the other enclosing line defaults to 0
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib functions (`errorbar` or `fill_between`)

```
masci_tools.vis.plot_methods.multiplot_moved(xdata, ydata, *, xlabel="", ylabel="", title="", data=None,
                                              scale_move=1.0, min_add=0, saveas='mscatterplot',
                                              copy_data=False, **kwargs)
```

Plots all the scatter plots above each other. It adds an arbitrary offset to the ydata to do this and calls `multiple_scatterplots`. Therefore you might not want to show the yaxis ticks

### Parameters

- **xdata** – arraylike, data for the x coordinate
- **ydata** – arraylike, data for the y coordinate
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis

- **ylabel** – str, label written on the y axis
- **title** – str, title of the figure
- **scale\_move** – float,  $\max \times \text{scale\_move}$  determines size of the shift
- **min\_add** – float, minimum shift
- **saveas** – str specifying the filename (without file format)
- **copy\_data** – bool, if True the data argument will be copied

Kwargs are passed on to the `multiple_scatterplots()` call

```
masci_tools.vis.plot_methods.plot_bands(kpath, bands, *, data=None, size_data=None,
                                         color_data=None, special_kpoints=None, e_fermi=0, xlabel="",
                                         ylabel='$E-E_F$ [eV]', title="", saveas='bandstructure',
                                         markersize_min=0.5, markersize_scaling=5.0,
                                         scale_color=True, separate_bands=False, line_plot=False,
                                         band_index=None, copy_data=False, **kwargs)
```

Plot the provided data for a bandstructure (non spin-polarized). Can be used to illustrate weights on bands via `size_data`

#### Parameters

- **kpath** – arraylike data for the kpoint data
- **bands** – arraylike data for the eigenvalues
- **size\_data** – arraylike data the weights to emphasize (optional)
- **color\_data** – str or arraylike, data for the color values with a colormap (optional)
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **e\_fermi** – float (default 0), place the line for the fermi energy at this value
- **special\_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **markersize\_min** – minimum value used in scaling points for weight
- **markersize\_scaling** – factor used in scaling points for weight
- **scale\_color** – bool, if True (default) the weight will be additionally shown via a colormap-ping
- **line\_plot** – bool, if True the bandstructure will be plotted with lines Here no weights are supported
- **separate\_bands** – bool, if True the bandstructure will be separately plotted for each band allows more specific parametrization
- **band\_index** – data for which eigenvalue belongs to which band (needed for line\_plot and separate\_bands)
- **copy\_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the `multi_scatter_plot()` call

```
masci_tools.vis.plot_methods.plot_bands_and_dos()
```

Plot a Bandstructure with a density of states on the right side.

```
masci_tools.vis.plot_methods.plot_certain_bands()
```

Plot only certain bands from a bands.1 file from FLEUR

```
masci_tools.vis.plot_methods.plot_colortable(colors, title, sort_colors=False, emptycols=0)
```

Plot a legend of named colors.

Reference: [https://matplotlib.org/3.1.0/gallery/color/named\\_colors.html](https://matplotlib.org/3.1.0/gallery/color/named_colors.html)

#### Parameters

- **colors** (`Dict`) – a dict color\_name : color\_value (hex str, rgb tuple, ...)
- **title** (`str`) – plot title
- **sort\_colors** (`bool`) – True: sort legend entries not by dict position, but by color hue, saturation, value.
- **emptycols** (`int`) –

**Returns** figure

```
masci_tools.vis.plot_methods.plot_convergence(iteration, distance, total_energy, *, data=None,
                                              saveas_energy='energy_convergence',
                                              saveas_distance='distance_convergence',
                                              axis_energy=None, axis_distance=None,
                                              xlabel='Iteration', ylabel_energy='Total energy
                                              difference [Htr]', ylabel_distance='Distance
                                              [me/bohr^3]', title_energy='Total energy difference over
                                              scf-Iterations', title_distance='Convergence (log)',
                                              copy_data=False, drop_last_iteration=False,
                                              **kwargs)
```

Plot the total energy differences versus the scf iteration and plot the distance of the density versus iterations.

#### Parameters

- **iteration** – data for the number of iterations
- **distance** – data of distances
- **total\_energy** – data of total energies
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label for the x-axis of both plots
- **saveas\_energy** – str, filename for the energy convergence plot
- **axis\_energy** – Axes object for the energy convergence plot
- **title\_energy** – str, title for the energy convergence plot
- **ylabel\_energy** – str, label for the y-axis for the energy convergence plot
- **saveas\_distance** – str, filename for the distance plot
- **axis\_distance** – Axes object for the distance plot
- **title\_distance** – str, title for the distance plot
- **ylabel\_distance** – str, label for the y-axis for the distance plot
- **copy\_data** – bool if True the data argument is copied
- **drop\_last\_iteration** – bool if True the last iteration is dropped for the distance plot

Other Kwargs will be passed on to all `multiple_scatterplots()` calls

```
masci_tools.vis.plot_methods.plot_convergence_results(iteration, distance, total_energy, *,
                                                    saveas1='t_energy_convergence',
                                                    axis1=None, saveas2='distance_convergence',
                                                    axis2=None, **kwargs)
```

DEPRECATED Plot the total energy versus the scf iteration and plot the distance of the density versus iterations.

#### Parameters

- **iteration** – array for the number of iterations
- **distance** – array of distances
- **total\_energy** – array of total energies
- **saveas1** – str, filename for the energy convergence plot
- **axis1** – Axes object for the energy convergence plot
- **saveas2** – str, filename for the distance plot
- **axis2** – Axes object for the distance plot

Other Kwargs will be passed on to all `single_scatterplot()` calls

```
masci_tools.vis.plot_methods.plot_convergence_results_m(iterations, distances, total_energies, *,
                                                        modes, nodes=None,
                                                        saveas1='t_energy_convergence',
                                                        saveas2='distance_convergence',
                                                        axis1=None, axis2=None, **kwargs)
```

DEPRECATED Plot the total energy versus the scf iteration and plot the distance of the density versus iterations.

#### Parameters

- **iterations** – array for the number of iterations
- **distances** – array of distances
- **total\_energies** – array of total energies
- **modes** – list of convergence modes (if ‘force’ is in the list the last distance is removed)
- **saveas1** – str, filename for the energy convergence plot
- **axis1** – Axes object for the energy convergence plot
- **saveas2** – str, filename for the distance plot
- **axis2** – Axes object for the distance plot

Other Kwargs will be passed on to all `multiple_scatterplots()` calls

```
masci_tools.vis.plot_methods.plot_convex_hull2d(hull, *, title='Convex Hull', xlabel='Atomic
                                                Percentage', ylabel='Formation energy / atom [eV]',
                                                saveas='convex_hull', axis=None, **kwargs)
```

Plot method for a 2d convex hull diagramm

#### Parameters

- **hull** – pyhull.Convexhull #scipy.spatial.ConvexHull
- **axis** – Axes object where to add the plot
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis



- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot

#### Function specific parameters:

**param marker\_hull** defaults to *marker*, marker type for the hull plot

**param markersize\_hull** defaults to *markersize*, markersize for the hull plot

**param color\_hull** defaults to *color*, color for the hull plot

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib functions `plot`

```
masci_tools.vis.plot_methods.plot_corelevel_spectra(coreleveldict, natom_typesdict,
                                                    exp_references=None, scale_to=-1,
                                                    show_single=True, show_ref=True,
                                                    energy_range=None, title="", fwhm_g=0.6,
                                                    fwhm_l=0.1, energy_grid=0.2,
                                                    peakfunction='voigt', linestyle_spec='-',
                                                    marker_spec='o', color_spec='k',
                                                    color_single='g', xlabel='Binding energy [eV]',
                                                    ylabel='Intensity [arb] (natoms*nelectrons)',
                                                    saveas=None, xspec=None, alpha_l=1.0,
                                                    beta_l=1.0, **kwargs)
```

Plotting function of corelevel in the form of a spectrum.

Convention: Binding energies are positiv!

**Args:** `coreleveldict`: dict of corelevels with a list of corelevel energy of atomtypes # (The given corelevel accounts for a weight (number of electrons for full occupied corelevel) in the plot.) `natom_typesdict`: dict with number of atom types for each entry

**Kwargs:** `exp_references`: dict with experimental refereces, will be plotted as vertical lines `show_single` (bool): plot all single peaks. `scale_to` float: the maximum 'intensity' will be scaled to this value (useful for experimental comparisons) `title` (string): something for labeling `fwhm` (float): full width half maximum of peaks (gaus, lorentz or voigt\_profile) `energy_grid` (float): energy resolution `linetyp_spec` : linetype for spectrum `peakfunction` (string): what the peakfunction should be {'voigt', 'pseudo-voigt', 'lorentz', 'gaus'}

**example:** `coreleveldict = {'u'Be': {'1s1/2' : [-1.0220669053033051, -0.3185614920138805,-0.7924091040092139]}}` `n_atom_types_Be12Ti = {'Be' : [4,4,4]}`

```
masci_tools.vis.plot_methods.plot_corelevels(coreleveldict, compound="", axis=None,
                                              saveas='scatterplot', **kwargs)
```

Plotting function to visualize corelevels and corelevel shifts

```
masci_tools.vis.plot_methods.plot_dos(energy_grid, dos_data, *, data=None, saveas='dos_plot',
                                       energy_label='$E-E_F$ [eV]', dos_label='DOS [1/eV]',
                                       title='Density of states', xyswitch=False, e_fermi=0,
                                       copy_data=False, **kwargs)
```

Plot the provided data for a density of states (not spin-polarized). Can be done horizontally or vertical via the switch `xyswitch`

#### Parameters

- **energy\_grid** – arraylike data for the energy grid of the DOS
- **dos\_data** – arraylike data for all the DOS components to plot
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)

- **title** – str, Title of the plot
- **energy\_label** – str, label for the energy-axis
- **dos\_label** – str, label for the DOS-axis
- **saveas** – str, filename for the saved plot
- **e\_fermi** – float (default 0), place the line for the fermi energy at this value
- **xyswitch** – bool if True, the enrgy axis will be plotted vertically
- **copy\_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the `multiple_scatterplots()` call

```
maschi_tools.vis.plot_methods.plot_lattice_constant(scaling, total_energy, *, fit_data=None,
                                                    data=None, relative=True, ref_const=None,
                                                    title='Equation of states',
                                                    saveas='lattice_constant', axis=None,
                                                    copy_data=False, **kwargs)
```

Plot a lattice constant versus Total energy Plot also the fit. On the x axis is the scaling, it

#### Parameters

- **scaling** – arraylike, data for the scaling factor
- **total\_energy** – arraylike, data for the total energy
- **fit\_data** – arraylike, optional data of fitted data
- **relative** – bool, scaling factor given (True), or lattice constants given?
- **ref\_const** – float (optional), or list of floats, lattice constant for scaling 1.0
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **copy\_data** – bool if True the data argument will be copied

#### Function specific parameters:

**param marker\_fit** defaults to *marker*, marker type for the fit data

**param markersize\_fit** defaults to *markersize*, markersize for the fit data

**param linewidth\_fit** defaults to *linewidth*, linewidth for the fit data

**param plotlabel\_fit** str label for the fit data

Other Kwargs will be passed on to `multiple_scatterplots()`

```
maschi_tools.vis.plot_methods.plot_one_element_corelv(corelevel_dict, element, compound='',
                                                       axis=None, saveas='scatterplot', **kwargs)
```

This routine creates a plot which visualizes all the binding energies of one element (and currently one corelevel) for different atomtypes.

**example:** corelevels = { 'W' : { '4f7/2' : [123, 123.3, 123.4, 123.1], '4f5/2' : [103, 103.3, 103.4, 103.1] }, 'Be' : { '1s' : [118, 118.2, 118.4, 118.1, 118.3] } }

```
maschi_tools.vis.plot_methods.plot_relaxation_results()
```

Plot from the result node of a relaxation workflow, All forces of every atom type versus relaxation cycle. Average force of all atom types versus relaxation cycle. Absolut relaxation in Angstroem of every atom type. Relative realxation of every atom type to a reference structure. (if none given use the structure from first relaxation cycle as reference)

```
masci_tools.vis.plot_methods.plot_residuen(xdata, fitdata, realdata, *, errors=None, xlabel='Energy
[eV]', ylabel='cts/s [arb]', title='Residuen',
saveas='residuen', hist=True, return_residuen_data=True,
**kwargs)
```

Calculates and plots the residuen for given xdata fit results and the real data.

If hist=True also the normed residual distribution is plotted with a normal distribution.

#### Parameters

- **xdata** – arraylike data for the x-coordinate
- **fitdata** – arraylike fitted data for the y-coordinate
- **realdata** – arraylike data to plot residuen against the fit
- **errors** – dict, can be used to provide errordata for the x and y direction
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **title** – str, title for the plot
- **saveas** – str, filename for the saved plot
- **hist** – bool, if True a normed residual distribution is plotted with a normal distribution.
- **return\_residuen\_data** – bool, if True in addition to the produced axis object also the residuen data is returned

#### Special Kwargs:

**param hist\_kwargs** dict, these arguments will be passed on to the [histogram\(\)](#) call (if hist=True)

Other Kwargs will be passed on to all [single\\_scatterplot\(\)](#) call

```
masci_tools.vis.plot_methods.plot_spectral_function(kpath, energy_grid, spectral_function, *,
data=None, special_kpoints=None, e_fermi=0,
xlabel="", ylabel='$E-E_F$ [eV]', title="",
saveas='spectral_function', copy_data=False,
**kwargs)
```

Create a colormesh plot of a spectral function

#### Parameters

- **kpath** – data for the kpoint coordinates
- **energy\_grid** – data for the energy grid
- **spectral\_function** – 2D data for the spectral function
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **e\_fermi** – float (default 0), place the line for the fermi energy at this value

- **special\_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **copy\_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the `colormesh_plot()` call

```
maschi_tools.vis.plot_methods.plot_spinpol_bands(kpath, bands_up, bands_dn, *, size_data=None,
                                                color_data=None, data=None, show_spin_pol=True,
                                                special_kpoints=None, e_fermi=0, xlabel="",
                                                ylabel='$E-E_F$ [eV]', title="",
                                                saveas='bandstructure', markersize_min=0.5,
                                                markersize_scaling=5.0, scale_color=True,
                                                line_plot=False, separate_bands=False,
                                                band_index=None, copy_data=False, **kwargs)
```

Plot the provided data for a bandstrucuture (spin-polarized). Can be used to illustrate weights on bands via `size_data`

#### Parameters

- **kpath** – arraylike data for the kpoint data
- **bands\_up** – arraylike data for the eigenvalues (spin-up)
- **bands\_dn** – arraylike data for the eigenvalues (spin-dn)
- **size\_data** – arraylike data the weights to emphasize BOTH SPINS (optional)
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **e\_fermi** – float (default 0), place the line for the fermi energy at this value
- **special\_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **markersize\_min** – minimum value used in scaling points for weight
- **markersize\_scaling** – factor used in scaling points for weight
- **show\_spin\_pol** – bool, if True (default) the two different spin channles will be shown in blue and red by default
- **scale\_color** – bool, if True (default) the weight will be additionally shown via a colormap-ping
- **line\_plot** – bool, if True the bandstructure will be plotted with lines Here no weights are supported
- **separate\_bands** – bool, if True the bandstructure will be separately plotted for each band allows more specific parametrization
- **band\_index** – data for which eigenvalue belongs to which band (needed for line\_plot and separate\_bands)
- **copy\_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the `multi_scatter_plot()` call

```
masci_tools.vis.plot_methods.plot_spinpol_dos(energy_grid, spin_up_data, spin_dn_data, *,
                                              data=None, saveas='spinpol_dos_plot',
                                              energy_label='$E-E_F$ [eV]', dos_label='DOS [1/eV]',
                                              title='Density of states', xyswitch=False, e_fermi=0,
                                              spin_dn_negative=True, spin_arrows=True,
                                              copy_data=False, **kwargs)
```

Plot the provided data for a density of states (spin-polarized). Can be done horizontally or vertical via the switch *xyswitch*

#### Parameters

- **energy\_grid** – arraylike data for the energy grid of the DOS
- **spin\_up\_data** – arraylike data for all the DOS spin-up components to plot
- **spin\_dn\_data** – arraylike data for all the DOS spin-down components to plot
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **energy\_label** – str, label for the energy-axis
- **dos\_label** – str, label for the DOS-axis
- **saveas** – str, filename for the saved plot
- **e\_fermi** – float (default 0), place the line for the fermi energy at this value
- **xyswitch** – bool if True, the enrgy axis will be plotted vertically
- **energy\_grid\_dn** – arraylike data for the energy grid of the DOS of the spin-down component (optional)
- **spin\_dn\_negative** – bool, if True (default) the spin-down components are plotted downwards
- **spin\_arrows** – bool, if True (default) small arrows will be plotted on the left side of the plot indicating the spin directions (if spin\_dn\_negative is True)
- **copy\_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the [multiple\\_scatterplots\(\)](#) call

```
masci_tools.vis.plot_methods.pseudo_voigt_profile(x, fwhm_g, fwhm_l, mu, mix=0.5)
```

Linear combination of gaussian and loretzian instead of convolution

**Args:** x: array of floats fwhm\_g: FWHM of gaussian fwhm\_l: FWHM of Lorentzian mu: Mean mix: ratio of gaus to lorentz, mix\* gaus, (1-mix)\*Lorentz

```
masci_tools.vis.plot_methods.reset_mpl_plot_defaults()
```

Reset the defaults for matplotlib backend to the hardcoded defaults

Available defaults can be seen in [MatplotlibPlotter](#)

```
masci_tools.vis.plot_methods.save_mpl_defaults(filename='plot_mpl_defaults.json',
                                              save_complete=False)
```

Save the current defaults for the matplotlib backend to a json file.

#### Parameters

- **filename** – filename, where the defaults should be stored
- **save\_complete** – bool if True not only the overwritten user defaults but also the unmodified hardcoded defaults are stored

`masci_tools.vis.plot_methods.set_mpl_plot_defaults(**kwargs)`

Set defaults for matplotlib backend according to the given keyword arguments

Available defaults can be seen in [MatplotlibPlotter](#)

`masci_tools.vis.plot_methods.show_mpl_plot_defaults()`

Show the currently set defaults for matplotlib backend to the hardcoded defaults

Available defaults can be seen in [MatplotlibPlotter](#)

`masci_tools.vis.plot_methods.single_scatterplot(xdata, ydata, *, xlabel="", ylabel="", title="", data=None, saveas='scatterplot', axis=None, xerr=None, yerr=None, area_curve=0, copy_data=False, **kwargs)`

Create a standard scatter plot (this should be flexible enough) to do all the basic plots.

#### Parameters

- **xdata** – str or arraylike, data for the x coordinate
- **ydata** – str or arraylike, data for the y coordinate
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **title** – str, title of the figure
- **data** – Mapping giving the data for the plot (required if xdata and ydata are str)
- **saveas** – str specifying the filename (without file format)
- **axis** – Axes object, if given the plot will be applied to this object
- **xerr** – optional data for errorbar in x-direction
- **yerr** – optional data for errorbar in y-direction
- **area\_curve** – if an area plot is made this arguments defines the other enclosing line defaults to 0
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib functions (*errorbar* or *fill\_between*)

`masci_tools.vis.plot_methods.surface_plot(xdata, ydata, zdata, *, xlabel="", ylabel="", zlabel="", title="", data=None, saveas='surface_plot', axis=None, copy_data=False, **kwargs)`

Create a standard surface plot

#### Parameters

- **xdata** – arraylike, data for the x coordinate
- **ydata** – arraylike, data for the y coordinate
- **zdata** – arraylike, data for the z coordinate
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **zlabel** – str, label written on the z axis

- **title** – str, title of the figure
- **axis** – Axes object, if given the plot will be applied to this object
- **saveas** – str specifying the filename (without file format)
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function `plot_surface`

`masci_tools.vis.plot_methods.voigt_profile(x, fwhm_g, fwhm_l, mu)`

Return the Voigt line shape at x with Lorentzian component FWHM `fwhm_l` and Gaussian component FWHM `fwhm_g` and mean `mu`. There is no closed form for the Voigt profile, but it is related to the real part of the Faddeeva function (`wofz`), which is used here.

`masci_tools.vis.plot_methods.waterfall_plot(xdata, ydata, zdata, *, xlabel="", ylabel="", zlabel="", title="", data=None, saveas='waterfallplot', axis=None, copy_data=False, **kwargs)`

Create a standard waterfall plot

#### Parameters

- **xdata** – arraylike, data for the x coordinate
- **ydata** – arraylike, data for the y coordinate
- **zdata** – arraylike, data for the z coordinate
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label written on the x axis
- **ylabel** – str, label written on the y axis
- **zlabel** – str, label written on the z axis
- **title** – str, title of the figure
- **axis** – Axes object, if given the plot will be applied to this object
- **saveas** – str specifying the filename (without file format)
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function `scatter3D`

## Bokeh

Here the `masci_tools.vis.parameters.Plotter` subclass for the bokeh plotting backend is defined with default values and many helper methods

**class** `masci_tools.vis.bokeh_plotter.BokehPlotter(**kwargs)`

Class for plotting parameters and standard code snippets for plotting with the bokeh backend.

Kwargs in the `__init__` method are forwarded to setting default values for the instance

For specific documentation about the parameter/defaults handling refer to `Plotter`.

Below the current defined default values are shown:

Table 2: Plot Parameters

Name	Description	Default value
figure_kwarg	Parameters for creating the bokeh figure. Includes things like axis type (x and y), tools plot width/height	<code>{'tools': 'pan, →poly_select,tap, →wheel_zoom,box_ →zoom,redo,undo, →reset,save, →crosshair,zoom_ →out,zoom_in', 'y_axis_type': →'linear', 'x_axis_type': →'linear', 'active_inspect': →None, 'toolbar_location →': 'right'}</code>
additional_tools	tools to add to the tools already specified in figure_kwarg (Has to be in the same format)	No Default
show_tooltips	Switch whether to add hover tooltips	True
global_tooltips	Switch whether to add individual (for each renderer) or global hover tooltips.	False
format_tooltips	Switch whether to enable the processing of formatted strings in tooltips.	True
tooltips	List of tuples specifying the tooltips. For more information refer to the bokeh documentation. Strings can contain format specifiers with the data keys of the function e.g. '@ `{x}` '. Here the `{x}` will be replaced by the entry for x. If there are formatting specifications for bokeh they need to be escaped with double curly braces or format_tooltips=False.	<code>[('X', '@{x}'), ( 'Y', '@{y}')]</code>
additional_tooltips	tooltips to add to the already defined tooltips (See above)	No Default
axis_linewidth	linewidth for the lines of the axis	2
label_fontsize	Fontsize for the labels of the axis	18pt
tick_label_fontsize	Fontsize for the ticks on the axis	16pt
background_color	Color of the background of the plot	#ffffff
x_axis_formatter	Set this formatter will be used for the ticks on the x-axis	No Default
y_axis_formatter	Set this formatter will be used for the ticks on the y-axis	No Default
x_ticks	Tick specification for the x-axis	No Default
x_ticklabels	Override the labels for the ticks on the x-axis	No Default
y_ticks	Tick specification for the y-axis	No Default
y_ticklabels	Override the labels for the ticks on the y-axis	No Default
x_range_padding	Specifies the amount of padding on the edges of the x-axis	No Default
y_range_padding	Specifies the amount of padding on the edges of the y-axis	No Default
limits	Dict specifying the limits of the axis, e.g {'x': (-5,5)}	No Default
legend_location	Location of the legend inside the plot area	top_right
legend_click_policy	Policy for what happens when labels are clicked in the legend	hide
legend_orientation	Orientation of the legend	vertical
legend_font_size	Fontsize for the labels inside the legend	14pt
legend_outside_plot	If True the legend will be placed outside of the plot area	False
color_palette	Color palette to use for the plot(s)	No Default

continues on next page



Table 2 – continued from previous page

Name	Description	Default value
color	Specific colors to use for the plot(s)	No Default
legend_label	Labels to use for the legend of the plot(s)	No Default
alpha	Transparency to use for the plot(s)	1.0
name	Name used for identifying elements in the plot (not shown only internally)	No Default
line_color	Color to use for line plot(s)	No Default
line_alpha	Transparency to use for line plot(s)	1.0
line_dash	Dash styles to use for line plot(s)	No Default
line_width	Line width to use for line plot(s)	2.0
marker	Type of marker to use for scatter plot(s)	circle
marker_size	Marker size to use for scatter plot(s)	6
area_plot	If True h(v)area will be used to produce the plot(s)	False
area_vertical	Determines, whether to use harea (False) or varea (True) for area plots	False
fill_alpha	Transparency to use for the area in area plot(s)	1.0
fill_color	Color to use for the area in area plot(s)	No Default
level	Can be used to specified, which elements are fore- or background	No Default
straight_lines	Dict specifying straight help-lines to draw. For example {'vertical': 0, 'horizontal': [-1,1]} will draw a vertical line at 0 and two horizontal at -1 and 1	No Default
straight_line_options	Options, and more options for the help-lines	<pre>{'line_color':   ↳'black', 'line_width': 1.0, 'line_dash':   ↳'dashed'}</pre>
save_plots	If True plots will be saved to file (Configuration beforehand is needed)	False
save_format	Formats to save the plots to, can be single or list of formats (html, png or svg)	html
show	If True bokeh.io.show will be called after the plotting routine	True

**add\_tooltips**(fig, renderers, columns=None, toggleable=False)

Add Hover tooltips to the given renderers and figure

#### Parameters

- **fig** – bokeh figure to apply changes to
- **renderers** – bokeh renderers to activate the tooltips for
- **columns** – namedtuple containing the data keys used for evtl. formatting
- **toggleable** – bool, if True these tooltips will be toggleable in the toolbar

**draw\_straight\_lines**(fig)

Draw horizontal and vertical lines specified in the lines argument

**Parameters fig** – bokeh figure on which to perform the operation

**plot\_kwargs**(ignore=None, extra\_keys=None, plot\_type='default', post\_process=True, list\_of\_dicts=True, \*\*kwargs)

Creates a dict or list of dicts (for multiple plots) with the defined parameters for the plotting calls for matplotlib

### Parameters

- **ignore** – str or list of str (optional), defines keys to ignore in the creation of the dict
- **extra\_keys** – optional set for additional keys to retrieve
- **post\_process** – bool, if True the parameters are cleaned up for inserting them directly into bokeh plotting functions

Kwargs are used to replace values by custom parameters:

Example for using a custom markersize:

```
p = BokehPlotter()
p.add_parameter('marker_custom', default_from='marker')
p.plot_kwargs(marker='marker_custom')
```

This code snippet will return the standard parameters for a plot, but the value for the marker will be taken from the key *marker\_custom*

**prepare\_figure**(title, xlabel, ylabel, figure=None)

Create a bokeh figure according to the set parameters or modify an existing one

### Parameters

- **title** – title of the figure
- **xlabel** – label on the x-axis
- **ylabel** – label on the y-axis
- **figure** – bokeh figure, optional, if given the operations are performed on the object otherwise a new figure is created

**Returns** the created or modified bokeh figure

**save\_plot**(figure, saveas)

Show/save the bokeh figure

**Parameters** **figure** – bokeh figure on which to perform the operation

**set\_color\_palette\_by\_num\_plots**()

Set the colormap for the configured number of plots according to the set colormap or color

copied from [https://github.com/PatrikHlobil/Pandas-Bokeh/blob/master/pandas\\_bokeh/plot.py](https://github.com/PatrikHlobil/Pandas-Bokeh/blob/master/pandas_bokeh/plot.py) credits to PatrikHlobil modified for use in this Plotter class

**set\_legend**(fig)

Set legend options for the figure

**Parameters** **fig** – bokeh figure on which to perform the operation

**set\_limits**(fig)

Set limits of the figure

**Parameters** **fig** – bokeh figure on which to perform the operation

Here are general and special bokeh plots to use

```
masci_tools.vis.bokeh_plots.bokeh_bands(kpath, bands=None, *, data=None, size_data=None,
                                         color_data=None, xlabel="", ylabel='E-E_F [eV]', title="",
                                         special_kpoints=None, markersize_min=3.0,
                                         markersize_scaling=10.0, saveas='bands_plot',
                                         scale_color=True, separate_bands=False, line_plot=False,
                                         band_index=None, copy_data=False, **kwargs)
```

Create an interactive bandstructure plot (non-spinpolarized) with bokeh Can make a simple plot or weight the size and color of the points against a given weight

#### Parameters

- **kpath** – arraylike or key data for the kpoint data
- **bands** – arraylike or key data for the eigenvalues
- **size\_data** – arraylike or key data the weights to emphasize (optional)
- **color\_data** – str or arraylike, data for the color values with a colormap (optional)
- **data** – source for the bands data (optional) of the plot (pandas Dataframe for example)
- **xlabel** – label for the x-axis (default no label)
- **ylabel** – label for the y-axis
- **title** – title of the figure
- **special\_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **e\_fermi** – float, determines, where to put the line for the fermi energy
- **markersize\_min** – minimum value used in scaling points for weight
- **markersize\_scaling** – factor used in scaling points for weight
- **outfilename** – filename of the output file
- **scale\_color** – bool, if True (default) the weight will be additionally shown via a colormap-ping
- **line\_plot** – bool, if True the bandstructure will be plotted with lines Here no weights are supported
- **separate\_bands** – bool, if True the bandstructure will be separately plotted for each band allows more specific parametrization
- **band\_index** – data for which eigenvalue belongs to which band (needed for line\_plot and separate\_bands)
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to [bokeh\\_multi\\_scatter\(\)](#) or [bokeh\\_line\(\)](#)

```
masci_tools.vis.bokeh_plots.bokeh_dos(energy_grid, dos_data=None, *, data=None, energy_label='E-E_F
[eV]', dos_label='DOS [1/eV]', title='Density of states',
xyswitch=False, e_fermi=0, saveas='dos_plot', copy_data=False,
**kwargs)
```

Create an interactive dos plot (non-spinpolarized) with bokeh Both horizontal or vertical orientation are possible

#### Parameters

- **energy\_grid** – arraylike or key data for the energy grid
- **spin\_up\_data** – arraylike or key data for the DOS

- **data** – source for the DOS data (optional) of the plot (pandas Dataframe for example)
- **energy\_label** – label for the energy-axis
- **dos\_label** – label for the dos-axis
- **title** – title of the figure
- **xyswitch** – bool if True, the energy will be plotted along the y-direction
- **e\_fermi** – float, determines, where to put the line for the fermi energy
- **outfilename** – filename of the output file
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to [`bokeh\_line\(\)`](#)

```
masci_tools.vis.bokeh_plots.bokeh_line(x, y=None, *, data=None, figure=None, xlabel='x', ylabel='y',
                                       title="", saveas='line', plot_points=False, area_curve=0,
                                       copy_data=False, set_default_legend=True, **kwargs)
```

Create an interactive multi-line plot with bokeh

#### Parameters

- **x** – arraylike or key for data for the x-axis
- **y** – arraylike or key for data for the y-axis
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – label for the x-axis
- **ylabel** – label for the y-axis
- **title** – title of the figure
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure
- **outfilename** – filename of the output file
- **plot\_points** – bool, if True also plot the points with a scatterplot on top
- **copy\_data** – bool, if True the data argument will be copied
- **set\_default\_legend** – bool if True the data names are used to generate default legend labels

Kwargs will be passed on to [`masci\_tools.vis.bokeh\_plotter.BokehPlotter`](#). If the arguments are not recognized they are passed on to the bokeh function [`line`](#)

```
masci_tools.vis.bokeh_plots.bokeh_multi_scatter(x, y=None, *, data=None, figure=None, xlabel='x',
                                                ylabel='y', title="", saveas='scatter', copy_data=False,
                                                set_default_legend=True, **kwargs)
```

Create an interactive scatter (multiple data sets possible) plot with bokeh

#### Parameters

- **x** – arraylike or key for data for the x-axis
- **y** – arraylike or key for data for the y-axis
- **data** – source for the data of the plot (pandas Dataframe for example)
- **xlabel** – label for the x-axis
- **ylabel** – label for the y-axis
- **title** – title of the figure

- **figure** – bokeh figure (optional), if provided the plot will be added to this figure
- **outfilename** – filename of the output file
- **copy\_data** – bool, if True the data argument will be copied
- **set\_default\_legend** – bool if True the data names are used to generate default legend labels

Kwargs will be passed on to `masci_tools.vis.bokeh_plotter.BokehPlotter`. If the arguments are not recognized they are passed on to the bokeh function `scatter`

```
masci_tools.vis.bokeh_plots.bokeh_scatter(x, y=None, *, xlabel='x', ylabel='y', title='', figure=None,
                                          data=None, saveas='scatter', copy_data=False, **kwargs)
```

Create an interactive scatter plot with bokeh

#### Parameters

- **x** – arraylike or key for data for the x-axis
- **y** – arraylike or key for data for the y-axis
- **data** – source for the data of the plot (pandas Dataframe for example)
- **xlabel** – label for the x-axis
- **ylabel** – label for the y-axis
- **title** – title of the figure
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure
- **outfilename** – filename of the output file
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to `masci_tools.vis.bokeh_plotter.BokehPlotter`. If the arguments are not recognized they are passed on to the bokeh function `scatter`

```
masci_tools.vis.bokeh_plots.bokeh_spectral_function(kpath, energy_grid, spectral_function, *,
                                                    data=None, special_kpoints=None, e_fermi=0,
                                                    xlabel='', ylabel='$E-E_F$ [eV]', title='',
                                                    saveas='spectral_function', copy_data=False,
                                                    figure=None, **kwargs)
```

Create a colormesh plot of a spectral function

#### Parameters

- **kpath** – data for the kpoint coordinates
- **energy\_grid** – data for the energy grid
- **spectral\_function** – 2D data for the spectral function
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **xlabel** – str, label for the x-axis
- **ylabel** – str, label for the y-axis
- **saveas** – str, filename for the saved plot
- **e\_fermi** – float (default 0), place the line for the fermi energy at this value
- **special\_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label

- **copy\_data** – bool, if True the data argument will be copied

All other Kwargs are passed on to the image call of bokeh

```
maschi_tools.vis.bokeh_plots.bokeh_spinpol_bands(kpath, bands_up=None, bands_dn=None, *,
                                                size_data=None, color_data=None, data=None,
                                                xlabel="", ylabel='E-E_F [eV]', title="",
                                                special_kpoints=None, markersize_min=3.0,
                                                markersize_scaling=10.0, saveas='bands_plot',
                                                scale_color=True, line_plot=False,
                                                separate_bands=False, band_index=None,
                                                copy_data=False, **kwargs)
```

Create an interactive bandstructure plot (spinpolarized) with bokeh Can make a simple plot or weight the size and color of the points against a given weight

#### Parameters

- **kpath** – arraylike or key data for the kpoint data
- **bands\_up** – arraylike or key data for the eigenvalues spin-up
- **bands\_dn** – arraylike or key data for the eigenvalues spin-dn
- **size\_data** – arraylike or key data the weights to emphasize (optional)
- **color\_data** – str or arraylike, data for the color values with a colormap (optional)
- **data** – source for the bands data (optional) of the plot (pandas Dataframe for example)
- **xlabel** – label for the x-axis (default no label)
- **ylabel** – label for the y-axis
- **title** – title of the figure
- **special\_kpoints** – list of tuples (str, float), place vertical lines at the given values and mark them on the x-axis with the given label
- **e\_fermi** – float, determines, where to put the line for the fermi energy
- **markersize\_min** – minimum value used in scaling points for weight
- **markersize\_scaling** – factor used in scaling points for weight
- **outfilename** – filename of the output file
- **scale\_color** – bool, if True (default) the weight will be additionally shown via a colormap-ping
- **line\_plot** – bool, if True the bandstructure will be plotted with lines Here no weights are supported
- **separate\_bands** – bool, if True the bandstructure will be separately plotted for each band allows more specific parametrization
- **band\_index** – data for which eigenvalue belongs to which band (needed for line\_plot and separate\_bands)
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to [bokeh\\_multi\\_scatter\(\)](#) or [bokeh\\_line\(\)](#)

```
masci_tools.vis.bokeh_plots.bokeh_spinpol_dos(energy_grid, spin_up_data=None, spin_dn_data=None,
*, data=None, spin_dn_negative=True,
energy_label='E-E_F [eV]', dos_label='DOS [1/eV]',
title='Density of states', xyswitch=False, e_fermi=0,
spin_arrows=True, saveas='dos_plot', copy_data=False,
**kwargs)
```

Create an interactive dos plot (spinpolarized) with bokeh Both horizontal or vertical orientation are possible

#### Parameters

- **energy\_grid** – arraylike or key data for the energy grid
- **spin\_up\_data** – arraylike or key data for the DOS spin-up
- **spin\_dn\_data** – arraylike or key data for the DOS spin-dn
- **data** – source for the DOS data (optional) of the plot (pandas Dataframe for example)
- **spin\_dn\_negative** – bool, if True (default), the spin down components are plotted downwards
- **energy\_label** – label for the energy-axis
- **dos\_label** – label for the dos-axis
- **title** – title of the figure
- **xyswitch** – bool if True, the energy will be plotted along the y-direction
- **e\_fermi** – float, determines, where to put the line for the fermi energy
- **spin\_arrows** – bool, if True (default) small arrows will be plotted on the left side of the plot indicating the spin directions (if spin\_dn\_negative is True)
- **outfilename** – filename of the output file
- **copy\_data** – bool, if True the data argument will be copied

Kwargs will be passed on to [bokeh\\_line\(\)](#)

```
masci_tools.vis.bokeh_plots.get_bokeh_help(key)
Print the decription of the given key in the bokeh backend
```

Available defaults can be seen in [BokehPlotter](#)

```
masci_tools.vis.bokeh_plots.load_bokeh_defaults(filename='plot_bokeh_defaults.json')
Load defaults for the bokeh backend from a json file.
```

**Parameters filename** – filename, from where the defaults should be taken

```
masci_tools.vis.bokeh_plots.periodic_table_plot(source, display_values=[], display_positions=[],
                                                color_value=None, tooltips=[('Name', '@name'),
('Atomic number', '@{atomic number}'), ('Atomic
mass', '@{atomic mass}'), ('CPK color', '$color[hex,
swatch]:CPK'), ('Electronic configuration',
'@{electronic configuration}')], title="",
outfilename='periodictable.html',
value_color_range=[None, None], log_scale=0,
color_map=None, bokeh_palette='Plasma256',
toolbar_location=None, tools='hover',
blank_color='#c4c4c4', blank_outsiders=[True,
True], include_legend=True, copy_source=True,
legend_labels=None, color_bar_title=None,
show=True)
```

Plot function for an interactive periodic table plot. Heat map and hover tool. source must be a panda dataframe containing, period, group,

param source: pandas dataframe containing everything param tooltips: what is shown with hover tool. values have to be in source example:

```
Keys of panda DF. group, period symbol and atomic number or required...
Index([u'atomic number', u'symbol', u'name', u'atomic mass', u'CPK',
u'electronic configuration', u'electronegativity', u'atomic radius',
u'ion radius', u'van der Waals radius', u'IE-1', u'EA',
u'standard state', u'bonding type', u'melting point', u'boiling point',
u'density', u'metal', u'year discovered', u'group', u'period',
u'rmt_mean', u'rmt_std', u'number_of_occ', u'type_color', u'c_value'],
dtype='object')

tooltips_def = [("Name", "@name"),
("Atomic number", "@{atomic number}"),
("Atomic mass", "@{atomic mass}"),
("CPK color", "$color[hex, swatch]:CPK"),
("Electronic configuration", "@{electronic configuration}")]
```

param display\_values: list of strings, have to match source. Values to be displayed on the element rectangles example: ["rmt\_mean", "rmt\_std", "number\_of\_occ"] param display\_positions: list of floats, length has to match display\_values, At which y offset the display values should be displayed.

```
masci_tools.vis.bokeh_plots.plot_convergence(iteration, distance, total_energy, *, data=None,
                                              saveas_energy='energy_convergence',
                                              saveas_distance='distance_convergence',
                                              figure_energy=None, figure_distance=None,
                                              xlabel='Iteration', ylabel_energy='Total energy difference
[Htr]', ylabel_distance='Distance [me/bohr^3]',
title_energy='Total energy difference over scf-Iterations',
title_distance='Convergence (log)', copy_data=False,
drop_last_iteration=False, **kwargs)
```

Plot the total energy differences versus the scf iteration and plot the distance of the density versus iterations.

#### Parameters

- **iteration** – data for the number of iterations
- **distance** – data of distances
- **total\_energy** – data of total energies



- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **xlabel** – str, label for the x-axis of both plots
- **saveas\_energy** – str, filename for the energy convergence plot
- **figure\_energy** – Axes object for the energy convergence plot
- **title\_energy** – str, title for the energy convergence plot
- **ylabel\_energy** – str, label for the y-axis for the energy convergence plot
- **saveas\_distance** – str, filename for the distance plot
- **figure\_distance** – Axes object for the distance plot
- **title\_distance** – str, title for the distance plot
- **ylabel\_distance** – str, label for the y-axis for the distance plot
- **copy\_data** – bool if True the data argument is copied
- **drop\_last\_iteration** – bool if True the last iteration is dropped for the distance plot

Other Kwargs will be passed on to all `bokeh_line()` calls

```
masci_tools.vis.bokeh_plots.plot_convergence_results(iteration, distance, total_energy, *,
                                                    saveas='convergence', **kwargs)
```

Plot the total energy versus the scf iteration and plot the distance of the density versus iterations. Uses `bokeh_line` and `bokeh_scatter`

#### Parameters

- **iteration** – list of Int
- **distance** – list of floats
- **show** – bool, if True call show

**Total\_energy** list of floats

Kwargs will be passed on to `bokeh_line()`

**Returns** `grid` bokeh grid with figures

```
masci_tools.vis.bokeh_plots.plot_convergence_results_m(iterations, distances, total_energies, *,
                                                    link=False, nodes=None, modes=None,
                                                    plot_label=None, saveas='convergence',
                                                    **kwargs)
```

Plot the total energy versus the scf iteration and plot the distance of the density versus iterations in a bokeh grid for several SCF results.

#### Parameters

- **distances** – list of lists of floats
- **iterations** – list of lists of Int
- **link** – bool, optional default=False:
- **nodes** – list of node uuids or pks important for links
- **saveas1** – str, optional default='t\_energy\_convergence', save first figure as
- **saveas2** – str, optional default='distance\_convergence', save second figure as
- **figure\_kwargs** – dict, optional default={'plot\_width': 600, 'plot\_height': 450}, gets parsed to `bokeh_line`

- **kwargs** – further key-word arguments for `bokeh_line`

**Total\_energies** list of lists of floats

**Returns grid** bokeh grid with figures

```
masci_tools.vis.bokeh_plots.plot_convex_hull2d(hull, title='Convex Hull', xlabel='Atomic Procentage',
                                              ylabel='Formation energy / atom [eV]', linestyle='-',
                                              marker='o', legend=True, legend_option={},
                                              saveas='convex_hull', limits=[None, None],
                                              scale=[None, None], axis=None, color='k',
                                              color_line='k', linewidth=2, markersize=8,
                                              marker_hull='o', markersize_hull=8, **kwargs)
```

Plot method for a 2d convex hull diagram

**Parameters hull** – `scipy.spatial.ConvexHull`

```
masci_tools.vis.bokeh_plots.plot_lattice_constant(scaling, total_energy, *, fit_data=None,
                                                  data=None, figure=None, relative=True,
                                                  ref_const=None, title='Equation of states',
                                                  saveas='lattice_constant', copy_data=False,
                                                  **kwargs)
```

Plot a lattice constant versus Total energy Plot also the fit. On the x axis is the scaling, it

**Parameters**

- **scaling** – arraylike, data for the scaling factor
- **total\_energy** – arraylike, data for the total energy
- **fit\_data** – arraylike, optional data of fitted data
- **relative** – bool, scaling factor given (True), or lattice constants given?
- **ref\_const** – float (optional), or list of floats, lattice constant for scaling 1.0
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **copy\_data** – bool if True the data argument will be copied
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure

**Function specific parameters:**

**param marker\_fit** defaults to *marker*, marker type for the fit data

**param marker\_size\_fit** defaults to *marker\_size*, markersize for the fit data

**param line\_width\_fit** defaults to *line\_width*, linewidth for the fit data

**param legend\_label\_fit** str label for the fit data

Other Kwargs will be passed on to [`bokeh\_line\(\)`](#)

```
masci_tools.vis.bokeh_plots.reset_bokeh_plot_defaults()
```

Reset the defaults for bokeh backend to the hardcoded defaults

Available defaults can be seen in [`BokehPlotter`](#)

```
masci_tools.vis.bokeh_plots.save_bokeh_defaults(filename='plot_bokeh_defaults.json',
                                              save_complete=False)
```

Save the current defaults for the matplotlib backend to a json file.

**Parameters**

- **filename** – filename, where the defaults should be stored
- **save\_complete** – bool if True not only the overwritten user defaults but also the unmodified hardcoded defaults are stored

`masci_tools.vis.bokeh_plots.set_bokeh_plot_defaults(**kwargs)`  
Set defaults for bokeh backend according to the given keyword arguments

Available defaults can be seen in [BokehPlotter](#)

`masci_tools.vis.bokeh_plots.show_bokeh_plot_defaults()`  
Show the currently set defaults for bokeh backend

Available defaults can be seen in [BokehPlotter](#)

## 6.1.2 Calculation tools

This file contains a class to compute the crystalfield coefficients from convoluting the charge density with the potential which produces the crystalfield. This is both compatible with the Yttrium-Analogue approximation and self-consistent calculation of the potential

**class** `masci_tools.tools.cf_calculation.CFCalculation`(*radial\_points=4000, reference\_radius='pot', pot\_cutoff=0.001, only\_m0=False, quiet=False*)

Class for calculating Crystal Field coefficients using the procedure described in C.E. Patrick, J.B. Staunton: J. Phys.: Condens. Matter 31, 305901 (2019)

**Using the formula:**

$$B_{lm} = \sqrt{\frac{2l+1}{4\pi}} \int_0^{R_{MT}} dr r^2 V_{lm}(r) n_{4f}(r)$$

The read in quantities are interpolated from logarithmic meshes to equidistant meshes

The function constructs an equidistant mesh between 0 and the muffin tin radius defined in `self.reference_radius` and with `self.radial_points` points

**Parameters:**

**param radial\_points** int, number of radial points in the interpolated mesh

**param reference\_radius** str or float; Either 'pot' or 'cdn' or explicit number. Defines which muffin-tin radius is used for the equidistant mesh. IMPORTANT! If txt files are used the muffin-tin radius has to be provided explicitly

**param pot\_cutoff** float Defines minimum value that has to appear in potentials to not be omitted (Only HDF)

**param only\_m0** bool, Ignores coefficients with m!=0 if True

**param quiet** bool, supresses print statements if True

**performIntegration**(*convert=True*)

Performs the integration to obtain the crystal field coefficients If the data was not already interpolated, the interpolation will be performed beforehand

**Parameters:**

**param convert** bool, converts to Steven's coefficients (if True)

**Returns** list of CFCoefficient objects (namedtuple), with all the necessary information

**prefactor**(*l*, *m*)

Gives the *lm* dependent prefactor for conversion between Blm and Alm coefficients

**Args:**

**param l** int; orbital quantum number

**param m** int; magnetic quantum number

**Returns** float prefactor for conversion to Steven's Coefficients

**readCDN**(*file*, *\*\*kwargs*)

Reads in the normed charge density for the CF coefficient calculation If hdf files are given also the muffin tin radius is read in

**Parameters:**

**param file** Expects string filename for the charge density to read in The function expects either HDF files or txt files with the format (rmesh,cdn). The charge density should be given as  $r^{2n}(r)$  and normed to 1

**kwargs:**

**param atomType** int, Defines the atomType to read in (only for HDF files)

**param header** int, Define how many lines to skip in the beginning of txt file

**readPot**(*\*args*, *lm=None*, *\*\*kwargs*)

Reads in the potentials for the CF coefficient calculation If hdf files are given also the muffin tin radius is read in

**Parameters:**

**param args** Expects string filenames for the potentials to read in The function expects either HDF files or txt files with the format (rmesh,vlmup,vlmdn)

**param lm** list of tuples, Defines the l and m indices for the given txt files

**kwargs:**

**param atomType** int, Defines the atomType to read in (only for HDF files)

**param header** int, Define how many lines to skip in the beginning of txt file

**param complexData** bool, Define if the data in the text file is complex

**Raises:** ValueError: lm indices list length has to match number of files read in

**class** masci\_tools.tools.cf\_calculation.CFCoefficient(*l*, *m*, *spin\_up*, *spin\_down*, *unit*, *convention*)

**convention**

Alias for field number 5

**1**

Alias for field number 0

**m**

Alias for field number 1

**spin\_down**

Alias for field number 3

**spin\_up**

Alias for field number 2

### unit

Alias for field number 4

```
masci_tools.tools.cf_calculation.plot_crystal_field_calculation(cfcalc,
                                                                filename='crystal_field_calc',
                                                                pot_title='Potential',
                                                                cdn_title='Density', xlabel='$R$
                                                                (Bohr)', pot_ylabel='$V_{pot}$
                                                                (Hartree)', cdn_ylabel='Density',
                                                                fontsize=12, labelsz=12,
                                                                pot_colors=None, save=False,
                                                                show=True)
```

Plot the given potentials and charge densities

### Parameters:

- param cfcalc** CFcalculation containing the data to plot
- param filename** str, Define the filename to save the figure
- param pot\_title** Title for the potential subplot
- param cdn\_title** Title for the charge density subplot
- param xlabel** label for the x axis of both subplots
- param pot\_ylabel** label for the y axis of the potential subplot
- param cdn\_ylabel** label for the y axis of the charge density subplot
- param fontsize** fontsize for titles and labels on the axis
- param labelsz** fontsize for the ticks on the axis,

```
masci_tools.tools.cf_calculation.plot_crystal_field_potential(cfcoeffs, file-
                                                                name='crystal_field_potential_areaplot',
                                                                spin='avg', phi=0.0, save=False,
                                                                show=True)
```

Plots the angular dependence of the calculated CF potential as well as a plane defined by phi.

### Parameters:

- param cfcoeffs** list of CFCoefficients to construct the potential
- param filename** str, defines the filename to save the figure
- param spin** str; Either 'up', 'dn' or 'avg'. Which spin direction to plot ('avg' -> ('up'+'dn')/2.0)
- param phi** float, defines the phi angle of the plane

**Raises `AssertionError`** – When coefficients are provided as wrong types or in the wrong convention

This module contains utility and functions to work with Green's functions calculated and written to `greensf.hdf` files by fleur

```
class masci_tools.tools.greensfunction.GreensFunction(element, data, attributes)
```

Class for working with Green's functions calculated by the fleur code

### Parameters

- **element** (*GreensfElement*) – *GreensfElement* namedtuple containing the information about the element

- **data** (`Dict[str, Any]`) – datasets dict produced by one of the hdf recipes for reading Green’s functions
- **attributes** (`Dict[str, Any]`) – attributes dict produced by one of the hdf recipes for reading Green’s functions

**energy\_dependence**(*\**, *m=None*, *mp=None*, *spin=None*, *imag=True*, *both\_contours=False*)

Select data with energy dependence

**Parameters**

- **m** (`Optional[int]`) – optional integer magnetic quantum number between -1 and 1
- **mp** (`Optional[int]`) – optional integer magnetic quantum number between -lp and lp
- **spin** (`Optional[int]`) – optional integer spin between 1 and nspins
- **both\_contours** (`bool`) – bool id True the data is not added for both energy contours
- **imag** (`bool`) – bool if True and both\_contours is False the imaginary part  $1/2i(G(z)-G(z^*))$  is returned otherwise the real part  $1/2(G(z)+G(z^*))$

**Return type** `ndarray`

**Returns** numpy array with the selected data

**classmethod fromFile**(*file*, *index=None*, *\*\*selection\_params*)

Classmethod for creating a [GreensFunction](#) instance directly from a hdf file

**Parameters**

- **file** (`Any`) – path or opened file handle to a greensf.hdf file
- **index** (`Optional[int]`) – optional int index of the element to read in

If index is not given Keyword arguments with the keys being the names of the fields of [GreensfElement](#) can be given to select the right Green’s function. The specification has to match only one element in the file

**Return type** [GreensFunction](#)

**get\_coefficient**(*name*, *spin=None*, *radial=False*)

Get the coefficient with the given name from the data attribute

**Parameters**

- **name** (`Literal['sphavg', 'uu', 'ud', 'du', 'dd', 'ulou', 'uulo', 'ulod', 'dulo', 'uloulo']`) – name of the coefficient
- **radial** (`bool`) – if the Green’s function is stored by coefficient and radial is True it is multiplied by the corresponding radial function otherwise the scalar product is multiplied
- **spin** (`Optional[int]`) – integer index of the spin to retrieve

**Return type** `ndarray`

**Returns** `numpy.ndarray` for the given coefficient and spin

**property nspins:** `int`

Return the number of spins of the current element. If mperp is True for the element it is 4 otherwise it is determined by the spins attribute

**Return type** `int`

**static to\_m\_index**(*m*)

Convert between magnetic quantum numbers between -1 and 1 to 0 and 2l+1 for easier indexing

**Parameters** `m (int)` – int magnetic quantum number to convert

**Return type** `int`

**Returns** converted magnetic quantum number

**static** `to_spin_indices(spin)`

Convert between spin index (0 to 3) to the corresponding two spin indices (0 or 1)

**Parameters** `spin (int)` – int spin index to convert

**Return type** `Tuple[int, int]`

**Returns** tuple of spin indices

**trace\_energy\_dependence(spin, imag=True)**

Select trace of data with energy dependence

**Parameters**

- **spin (int)** – integer spin between 1 and nspins
- **imag (bool)** – bool if True the imaginary part  $1/2i(G(z)-G(z^*))$  is returned otherwise the real part  $1/2(G(z)+G(z^*))$

**Return type** `ndarray`

**Returns** numpy array with the selected and traced over data

**class** `masci_tools.tools.greensfunction.GreensfElement(l, lp, atomType, atomTypep, sphavg, onsite, kresolved, contour, nLO, atomDiff)`

**atomDiff**

Alias for field number 9

**atomType**

Alias for field number 2

**atomTypep**

Alias for field number 3

**contour**

Alias for field number 7

**kresolved**

Alias for field number 6

**l**

Alias for field number 0

**lp**

Alias for field number 1

**nLO**

Alias for field number 8

**onsite**

Alias for field number 5

**sphavg**

Alias for field number 4

**class** `masci_tools.tools.greensfunction.colors`

Color strings for coloring terminal output

You may need to change color settings in iPython

`masci_tools.tools.greensfunction.intersite_shell_indices(elements, reference_atom, show=False)`  
Construct the green's function pairs to calculate the Jij exchange constants for a given reference atom from a list of *GreensfElement*

#### Parameters

- **elements** (`List[GreensfElement]`) – list of GreenfElements to use
- **reference\_atom** (`int`) – integer of the atom to calculate the Jij's for (corresponds to the i)
- **show** (`bool`) – if True the elements belonging to a shell are printed in a shell

**Return type** `List[Tuple[float, List[Tuple[int, int]]]`

**Returns** list of tuples with distance and all indices of pairs in the shell

`masci_tools.tools.greensfunction.intersite_shells(greensfunctions, reference_atom, show=False)`  
Construct the green's function pairs to calculate the Jij exchange constants for a given reference atom from a list of given *GreensFunction*

#### Parameters

- **greensfunctions** (`List[GreensFunction]`) – List of Greens Function to use
- **reference\_atom** (`int`) – integer of the atom to calculate the Jij's for (corresponds to the i)
- **show** (`bool`) – if True the elements belonging to a shell are printed in a shell

**Return type** `Iterator[Tuple[float, GreensFunction, GreensFunction]]`

**Returns** flat iterator with distance and the two corresponding *GreensFunction* instances for each Jij calculation

`masci_tools.tools.greensfunction.intersite_shells_from_file(hdffile, reference_atom, show=False)`  
Construct the green's function pairs to calculate the Jij exchange constants for a given reference atom from a given greensf.hdf file

#### Parameters

- **hdffile** (`Any`) – filepath or file handle to a greensf.hdf file
- **reference\_atom** (`int`) – integer of the atom to calculate the Jij's for (corresponds to the i)
- **show** (`bool`) – if True the elements belonging to a shell are printed in a shell

**Return type** `Iterator[Tuple[float, GreensFunction, GreensFunction]]`

**Returns** flat iterator with distance and the two corresponding *GreensFunction* instances for each Jij calculation

`masci_tools.tools.greensfunction.listElements(hdffile, show=False)`  
Find the green's function elements contained in the given greens.hdf file

#### Parameters

- **hdffile** (`Any`) – filepath or file handle to a greensf.hdf file
- **show** (`bool`) – bool if True the found elements are printed in a table

**Return type** `List[GreensfElement]`

**Returns** list of *GreensfElement*

`masci_tools.tools.greensfunction.printElements(elements, index=None, mark=None)`  
Print the given list of *GreensfElement* in a nice table



### Parameters

- **elements** (`List[GreensfElement]`) – list of *GreensfElement* to be printed
- **index** (`Optional[List[int]]`) – optional list of indices to show instead of the default index in the list
- **mark** (`Optional[List[int]]`) – optional list of int with elements to emphasize with an arrow and color

**Return type** `None`

`masci_tools.tools.greensfunction.select_element_indices(elements, show=False, **selection_params)`

Select *GreensfElement* objects from a list based on constraints on their values

### Parameters

- **elements** (`List[GreensfElement]`) – list of *GreensfElement* to choose from
- **show** (`bool`) – bool if True the found elements will be printed

The Keyword arguments correspond to the names of the fields and their desired value

**Return type** `List[int]`

**Returns** list of the indices matching the criteria

`masci_tools.tools.greensfunction.select_elements(greensfunctions, show=False, **selection_params)`

Select *GreensFunction* objects from a list based on constraints on the values of their underlying *GreensfElement*

### Parameters

- **greensfunctions** (`List[GreensFunction]`) – list of *GreensFunction* to choose from
- **show** (`bool`) – bool if True the found elements will be printed

The Keyword arguments correspond to the names of the fields and their desired value

**Return type** `Iterator[GreensFunction]`

**Returns** iterator over the matching *GreensFunction*

`masci_tools.tools.greensfunction.select_elements_from_file(hdffile, show=False, **selection_params)`

Construct the green's function mathcing specified criteria from a given *greensf.hdf* file

### Parameters

- **hdffile** (`Any`) – file or file path to the *greensf.hdf* file
- **show** (`bool`) – bool if True the found elements will be printed

The Keyword arguments correspond to the names of the fields and their desired value

**Return type** `Iterator[GreensFunction]`

**Returns** iterator over the matching *GreensFunction*

This module collects functions for calculating properties with the greens functions calculated by Fleur. At the moment the following are implemented:

- Calculating Heisenberg  $J_0$  (spin stiffness) from onsite Green's functions
- Calculating Heisenberg  $J_{ij}$  exchange constants from intersite Green's functions
- Calculating the hybridization function from onsite Greens functions

```
masci_tools.tools.greensf_calculations.calculate_heisenberg_j0(greensfunction, onsite_delta,
                                                                show=False)
```

Calculate spin stiffness  $J_0$  for the given green's function using the formula

$$J_0 = \frac{1}{4\pi} \text{Im Tr}_L \int_{-\infty}^{E_F} dz \Delta (G^\uparrow(z) - G^\downarrow(z)) + \Delta^2 G^\uparrow(z) G^\downarrow(z)$$

#### Parameters

- **greensfunction** – *GreensFunction* to use for the calculation
- **onsite\_delta** – onsite exchange splitting to use for the calculation
- **show** – bool if True additional information about the used Greens functions is printed out

**Returns** the value of the spin stiffness in meV

```
masci_tools.tools.greensf_calculations.calculate_heisenberg_jij(hdffileORgreensfunctions,
                                                                reference_atom, onsite_delta,
                                                                show=False)
```

Calculate the Heisenberg exchange constants form Green's functions using the formula

$$J_{ij} = \frac{1}{4\pi} \text{Im Tr}_L \int_{-\infty}^{E_F} dz \Delta_i G_{ij}(z) \Delta_j G_{ji}(z)$$

#### Parameters

- **hdffileORgreensfunctions** – either pat/file-like object for the `greensf.hdf` file to use or list of *GreensFunction*
- **reference\_atom** – integer index of the atom to calculate the Jijs from
- **onsite\_delta** – List of floats containing the onsite exchange splitting for each atom type and l-channel
- **show** – bool if True additional information about the used Greens functions is printed out

**Returns** dict mapping the distances to all the calculated Jijs for that distance

```
masci_tools.tools.greensf_calculations.calculate_hybridization(greensfunction)
Calculate the hybridization function as
```

$$\Delta(z) = \frac{1}{2 * l + 1} \text{Tr} G^{-1}(z)$$

**Returns** numpy array of the hybridization function

## 6.1.3 IO helper functions and file parsers

### 6.1.3.1 KKR related IO

```
class masci_tools.io.kkr_params.kkrparams(**kwargs)
```

Class for creating and handling the parameter input for a KKR calculation Optional keyword arguments are passed to init and stored in values dictionary.

Example usage: `params = kkrparams(LMAX=3, BRAVAIS=array([[1,0,0], [0,1,0], [0,0,1]]))`

**Alternatively values can be set afterwards either individually with** `params.set_value('LMAX', 3)`

**or multiple keys at once with** `params.set_multiple_values(EMIN=-0.5, EMAX=1)`

Other useful functions

- print the description of a keyword: `params.get_description([key])` where `[key]` is a string for a keyword in `params.values`
- print a list of mandatory keywords: `params.get_all_mandatory()`
- print a list of keywords that are set including their value: `params.get_set_values()`

**change\_XC\_val\_kkrimp**(*val*)

Convert integer value of KKRhost KEXCOR input to KKRimp XC string input.

**fill\_keywords\_to\_inputfile**(*is\_voro\_calc=False, output='inputcard', no\_check=False, verbose=False*)

Fill new inputcard with keywords/values automatically check for input consistency (can be disabled by the `no_check` input) if `is_voro_calc==True` change mandatory list to match voronoi code, default is KKRcode

**classmethod get\_KKRcalc\_parameter\_defaults**(*silent=False*)

set defaults (defined in header of this file) and returns dict, `kkrparams_version`

**get\_all\_mandatory**()

Return a list of mandatory keys

**get\_description**(*key=None, search=None*)

Returns description of keyword 'key' If 'key' is None, print all descriptions of all available keywords If 'search' is not None, print all keys+descriptions where the search string is found

**get\_dict**(*group=None, subgroup=None*)

Returns values dictionary.

Prints values belonging to a certain group only if the 'group' argument is one of the following: 'lattice', 'chemistry', 'accuracy', 'external fields', 'scf cycle', 'other'

Additionally the subgroups argument allows to print only a subset of all keys in a certain group. The following subgroups are available.

- in 'lattice' group '2D mode', 'shape functions'
- in 'chemistry' group 'Atom types', 'Exchange-correlation', 'CPA mode', '2D mode'
- in 'accuracy' group 'Valence energy contour', 'Semicore energy contour', 'CPA mode', 'Screening clusters', 'Radial solver', 'Ewald summation', 'LLoyd'

**get\_missing\_keys**(*use\_aiida=False*)

Find list of mandatory keys that are not yet set

**get\_set\_values**()

Return a list of all keys/values that are set (i.e. not None)

**get\_type**(*key*)

Extract expected type of 'key' from format info

**get\_value**(*key*)

Gets value of keyword 'key'

**is\_mandatory**(*key*)

Returns mandatory flag (True/False) for keyword 'key'

**items**()

make `kkrparams.items()` work

**read\_keywords\_from\_inputcard**(*inputcard='inputcard', verbose=False*)

Read list of keywords from inputcard and extract values to keywords dict

**Example usage** `p = kkrparams(); p.read_keywords_from_inputcard('inputcard')`

**Note** converts '<RBLEFT>', '<RBRIGHT>', 'ZPERIODL', and 'ZPERIODR' automatically to Ang. units!

**remove\_value**(*key*)

Removes value of keyword 'key', i.e. resets to None

**set\_multiple\_values**(\*\**kwargs*)

Set multiple values (in example value1 and value2 of keywords 'key1' and 'key2') given as key1=value1, key2=value2

**set\_value**(*key*, *value*, *silent=False*)

Sets value of keyword 'key'

**classmethod split\_kkr\_options**(*valtxt*)

Split keywords after fixed length of 8 :param valtxt: list of strings or single string :returns: List of keywords of maximal length 8

**update\_to\_kkrimp**()

Update parameter settings to match kkrimp specification. Sets self.\_\_params\_type and calls \_update\_mandatory\_kkrimp()

**update\_to\_voronoi**()

Update parameter settings to match voronoi specification. Sets self.\_\_params\_type and calls \_update\_mandatory\_voronoi()

**masci\_tools.io.kkr\_read\_shapefun\_info.read\_shapefun**(*path='.'*)

Read vertices of shapefunctions with Zoom into shapefun of a single atom

**Author** Philipp Ruessmann

**Parameters** *path* – path where voronoi output is found (optional, defaults to '.')

**Returns** *pos* positions of the centers of the shapefunctions

**Returns out** dictionary of the vertices of the shapefunctions

Here I collect all functions needed to parse the output of a KKR calculation. These functions do not need aiida and are therefore separated from the actual parser file where `parse_kkr_outputfile` is called

**masci\_tools.io.parsers.kkrparser\_functions.check\_error\_category**(*err\_cat*, *err\_msg*, *out\_dict*)

Check if parser error of the non-critical category (*err\_cat* != 1) are actually consistent and may be discarded.

**Parameters**

- **err\_cat** – the error-category of the error message to be investigated
- **err\_msg** – the error-message
- **out\_dict** – the dict of results obtained from the parser function

**Returns** True/False if message is an error or warning

**masci\_tools.io.parsers.kkrparser\_functions.get\_kmeshinfo**(*outfile\_0init*, *outfile\_000*)

Extract kmesh info from output.0.txt and output.000.txt

**masci\_tools.io.parsers.kkrparser\_functions.get\_lattice\_vectors**(*outfile\_0init*)

read direct and reciprocal lattice vectors in internal units (useful for qdos generation)

**masci\_tools.io.parsers.kkrparser\_functions.get\_natom**(*outfile\_0init*)

extract NATYP value from output.0.txt

**masci\_tools.io.parsers.kkrparser\_functions.get\_noco\_rms**(*outfile*, *debug=False*)

Get average noco rms error

**masci\_tools.io.parsers.kkrparser\_functions.get\_nspin**(*outfile\_0init*)

extract NSPIN value from output.0.txt

---

```

masci_tools.io.parsers.kkrparser_functions.get_orbmom(outfile, natom)
    read orbmom info from outfile and return array (iteration, atom)=orbmom

masci_tools.io.parsers.kkrparser_functions.get_rms(outfile, outfile2, debug=False)
    Get rms error per atom (both values for charge and spin) and total (i.e. average) value

masci_tools.io.parsers.kkrparser_functions.get_single_particle_energies(outfile_000)
    extracts single particle energies from outfile_000 (output.000.txt) returns the valence contribution of the single
    particle energies

masci_tools.io.parsers.kkrparser_functions.get_spinmom_per_atom(outfile, natom,
                                                                nonco_out_file=None)
    Extract spin moment information from outfile and nonco_angles_out (if given)

masci_tools.io.parsers.kkrparser_functions.parse_array_float(outfile, searchstring, splitinfo,
                                                            replacepair=None, debug=False)
    Search for keyword searchstring in outfile and extract array of results
    Returns: array of results

masci_tools.io.parsers.kkrparser_functions.parse_kkr_outputfile(out_dict, outfile, outfile_0init,
                                                                outfile_000, timing_file,
                                                                potfile_out, nonco_out_file,
                                                                outfile_2='output.2.txt',
                                                                skip_readin=False,
                                                                debug=False)
    Parser method for the kkr outfile. It returns a dictionary with results

masci_tools.io.parsers.kkrparser_functions.use_BdG(outfile_0init)
    extract BdG run info from output.0.txt

masci_tools.io.parsers.kkrparser_functions.use_newsosol(outfile_0init)
    extract NEWSOSOL info from output.0.txt

Everything that is needed to parse the output of a voronoi calculation.

masci_tools.io.parsers.voroparser_functions.check_voronoi_output(potfile, outfile,
                                                                delta_emin_safety=0.1)
    Read output from voronoi code and create guess of energy contour

masci_tools.io.parsers.voroparser_functions.get_valence_min(outfile='out_voronoi')
    Construct minimum of energy contour (between valence band bottom and core states)

masci_tools.io.parsers.voroparser_functions.parse_voronoi_output(out_dict, outfile, potfile,
                                                                atominfo, radii, inputfile,
                                                                debug=False)
    Parse output of voronoi calculation and return (success, error_messages_list, out_dict)

Tools for the impurity calculation plugin and its workflows

class masci_tools.io.parsers.kkrimp_parser_functions.KkrimpParserFunctions
    Class of parser functions for KKRimp calculation

    Usage success, msg_list, out_dict = parse_kkrimp_outputfile().parse_kkrimp_outputfile(out_dict,
        files)

    parse_kkrimp_outputfile(out_dict, file_dict, debug=False)
    Main parser function for kkrimp, read information from files in file_dict and fills out_dict :param out_dict:
    dictionary that is filled with parsed output of the KKRimp calculation :param file_dict: dictionary of files
    that are parsed :returns: success (bool), msg_list(list of error/warning messages of parser), out_dict (filled
    dict of parsed output) :note: file_dict should contain the following keys

```

- ‘outfile’, the std\_out of the KKRimp calculation
- ‘out\_log’, the out\_log.000.txt file
- ‘out\_pot’, the output potential
- ‘out\_ensersp\_at’, the out\_energysp\_per\_atom\_eV file
- ‘out\_enertot\_at’, the out\_energytotal\_per\_atom\_eV file
- ‘out\_timing’, the timing file
- ‘kkrflex\_llyfac’, the file for the Lloyd factor
- ‘kkrflex\_angles’, the nonco\_angles file for the KKRimp calculation
- ‘out\_spinmoms’, the output spin moments file
- ‘out\_orbmoms’, the output orbital moments file

### 6.1.3.2 Fleur related IO

#### Input/Output Parser

Load both the outxml\_parser and inpxml\_parser

```
masci_tools.io.parsers.fleur.inpxml_parser(inpxmlfile, parser_info_out=None, strict=False,
                                           debug=False, base_url=None)
```

Parses the given inp.xml file to a python dictionary utilizing the schema defined by the version number to validate and corretly convert to the dictionary

##### Parameters

- **inpxmlfile** (`Union[_ElementTree, str, Path, bytes, PathLike, IO]`) – either path to the inp.xml file, opened file handle or a xml etree to be parsed
- **parser\_info\_out** (`Optional[Dict[str, Any]]`) – dict, with warnings, info, errors, ...
- **strict** (`bool`) – bool if True and no parser\_info\_out is provided any encountered error will immediately be raised

**Return type** `Dict[str, Any]`

**Returns** python dictionary with the parsed inp.xml

##### Raises

- **ValueError** – If the validation against the schema failed, or an irrecoverable error occurred during parsing
- **FileNotFoundError** – If no Schema file for the given version was found

```
masci_tools.io.parsers.fleur.outxml_parser(outxmlfile, parser_info_out=None, iteration_to_parse='last',
                                           minimal_mode=False, additional_tasks=None,
                                           optional_tasks=None, overwrite=False, append=False,
                                           list_return=False, strict=False, debug=False,
                                           ignore_validation=False, base_url=None)
```

Parses the out.xml file to a dictionary based on the version and the given tasks

##### Parameters

- **outxmlfile** (`Union[_ElementTree, str, Path, bytes, PathLike, IO]`) – either path to the out.xml file, opened file handle or a xml etree to be parsed

- **parser\_info\_out** (`Optional[Dict[str, Any]]`) – dict, with warnings, info, errors, ...
- **iteration\_to\_parse** (`Union[Literal['all', 'last', 'first'], int]`) – either str or int, (optional, default 'last') determines which iteration should be parsed. Accepted are 'all', 'first', 'last' or an index for the iteration
- **minimal\_mode** (`bool`) – bool, if True only total Energy, iteration number and distances are parsed
- **additional\_tasks** (`Optional[Dict[str, Dict[str, Any]]]`) – dict to define custom parsing tasks. For detailed explanation See [default\\_parse\\_tasks](#).
- **overwrite** (`bool`) – bool, if True and keys in additional\_tasks collide with defaults The defaults will be overwritten
- **append** (`bool`) – bool, if True and keys in additional\_tasks collide with defaults The inner tasks will be written into the dict. If inner keys collide they are overwritten
- **optional\_tasks** (`Optional[Iterable[str]]`) – Iterable of strings, defines additional tasks to perform. See [default\\_parse\\_tasks](#) for examples.
- **list\_return** (`bool`) – bool, if True one-item lists in the output dict are not converted to simple values
- **strict** (`bool`) – bool if True and no parser\_info\_out is provided any encountered error will immediately be raised
- **debug** (`bool`) – bool if True additional information is printed out in the logs
- **ignore\_validation** (`bool`) – bool, if True schema validation errors are only logged

**Return type** `Dict[str, Any]`

**Returns** python dictionary with the information parsed from the out.xml

**Raises**

- **ValueError** – If the validation against the schema failed, or an irrecoverable error occurred during parsing
- **FileNotFoundError** – If no Schema file for the given version was found
- **KeyError** – If an unknown task is encountered

## Inputgenerator related IO

This module contains functionality for writing input files for the input generator of fleur

`masci_tools.io.fleur_inpgen.conv_to_fortran(val, quote_strings=True)`

Convert values to fortran friendly strings

**Parameters**

- **val** – the value to be read and converted to a Fortran-friendly string.
- **quote\_strings** – bool, if True single quotes will be added to a string

`masci_tools.io.fleur_inpgen.get_input_data_text(key, val, value_only, mapping=None)`

Given a key and a value, return a string (possibly multiline for arrays) with the text to be added to the input file.

**Parameters**

- **key** – the flag name

- **val** – the flag value. If it is an array, a line for each element is produced, with variable indexing starting from 1. Each value is formatted using the `conv_to_fortran` function.
- **mapping** – Optional parameter, must be provided if `val` is a dictionary. It maps each key of the ‘val’ dictionary to the corresponding list index. For instance, if `key='magn'`, `val = {'Fe': 0.1, 'O': 0.2}` and `mapping = {'Fe': 2, 'O': 1}`, this function will return the two lines `magn(1) = 0.2` and `magn(2) = 0.1`. This parameter is ignored if ‘val’ is not a dictionary.

`masci_tools.io.fleur_inpgen.read_inpgen_file(file, convert_to_angstroem=True)`

Method which reads in an inpgen input file and parses the structure and name lists information.

#### Parameters

- **file** – path to the file to read or opened file handle
- **convert\_to\_angstroem** – bool if True the bravais matrix (and atom positions) are converted to angstroem

**Returns** tuple of bravais matrix, atom sites, periodic boundary conditions and parameters

`masci_tools.io.fleur_inpgen.write_inpgen_file(cell, atom_sites, kinds=None, return_contents=False, file='inpgen.in', pbc=(True, True, True), input_params=None, significant_figures_cell=9, significant_figures_positions=10, convert_from_angstroem=True)`

Write an input file for the fleur inputgenerator ‘inpgen’ from given inputs

#### Parameters

- **cell** – 3x3 arraylike. The bravais matrix of the structure, in Angstrom by default
- **atom\_sites** – either list of a dict containing the keys absolute ‘position’ in Angstrom (default) and ‘kind\_name’, i.e

```
[{'position': (0.0, 0.0, -1.0545708047819), 'kind_name': 'Fe123'},
 {'position': (1.4026317387183, 1.9836207751336, 0.0), 'kind_name':
 ↪ 'Pt'},
 {'position': (0.0, 0.0, 1.4026318234924), 'kind_name': 'Pt'}]
```

In this case the argument `kinds` is required. The other possibility is a list of tuples of the form of *AtomSiteProperties*

- **kinds** – a list of kind information containing the keys symbols, weights, mass, name i.e.

```
[{'symbols': ('Fe',), 'weights': (1.0,), 'mass': 55.845, 'name':
 ↪ 'Fe123'},
 {'symbols': ('Pt',), 'weights': (1.0,), 'mass': 195.084, 'name':
 ↪ 'Pt'}]
```

Required when `atom_sites` is a list of dicts

- **file** – Path or filehandle where the file should be written to. Defaults to ‘inpgen.in’ in the current folder.
- **pbc** – tuple of boolean length 3, optional, Periodic boundary conditions of the structure. Defaults to (True, True, True).
- **input\_params** – Optional dict containing further namelist which should be written to the file. Defaults to None.



- **significant\_figures\_cell** – int, how many decimal places should be written for the bravais matrix (default: 9)
- **significant\_figures\_positions** – int, how many decimal places should be written for the atom positions (default: 10)
- **convert\_from\_angstroem** – optional boolean, if True the positions and elements of the bravais matrix are converted to bohr from Angstroem

**Raises** **ValueError** – If some input is wrong or inconsistent.

**Returns** [list]: A report, list of strings, things which where logged within the process

Comments: This was extracted out of aiida-fleur for more general use, the datastructures stayed very close to what aiida provides (`to_raw()`), it may not yet be convenient for all usecases. I.e data so far has to be given in Angstrom and will be converted to fleur units. # This could be made optional

## Functions for modifying the input file

This module contains a class for organizing and grouping changes to a input file of fleur in a robust way.

Essentially a low-level version of the FleurinpModifier in `aiida_fleur`.

**class** `masci_tools.io.fleurxmlmodifier.FleurXMLModifier(validate_signatures=True)`

Class for grouping and organizing changes to a inp.xml file of fleur via the xml setting methods in `xml_setters_names` and `xml_setters_basic`

The basic usage is shown below

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fmode = FleurXMLModifier()

#Add changes by calling the methods on this class
#(names correspond to the setting methods in the xml_setters modules)
#They are not modifying a input file directly
#Instead all the tasks are collected and performed in one go

fmode.set_inpchanges({'Kmax': 4.0}) #Set Kmax to 4.0
fmode.shift_value({'Gmax': 5.0}) #Add 5 to the current value of Gmax

#Set the local orbital configuration on all iron atoms to '3s 3p'
fmode.set_species('all-Fe', {'lo': [{'n':3, 'l': 's', 'type': 'SCLO'},
                                   {'n':3, 'l': 'p', 'type': 'SCLO'}]})

#To undo the last change call undo
#fmode.undo()

#revert_all=True resets all added tasks
#fmode.undo(revert_all=True)

#To apply the changes to an input file use the modify_xmlfile method
new_xmltree = fmode.modify_xmlfile('/path/to/input/file/inp.xml')
```

**add\_number\_to\_attrib(\*args, \*\*kwargs)**

Appends a `add_number_to_attrib()` to the list of tasks that will be done on the xmltree.

**Parameters**

- **attributename** – the attribute name to change
- **add\_number** – number to add/multiply with the old attribute value
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **mode** – str (either *rel* or *abs*). *rel* multiplies the old value with *add\_number* *abs* adds the old value and *add\_number*
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

**Kwargs:**

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: *settable*, *settable\_contains*, *other*

**add\_number\_to\_first\_attrib(\*args, \*\*kwargs)**

Appends a [add\\_number\\_to\\_first\\_attrib\(\)](#) to the list of tasks that will be done on the xmltree.

**Parameters**

- **attributename** – the attribute name to change
- **add\_number** – number to add/multiply with the old attribute value
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **mode** – str (either *rel* or *abs*). *rel* multiplies the old value with *add\_number* *abs* adds the old value and *add\_number*

**Kwargs:**

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: *settable*, *settable\_contains*, *other*

**add\_task\_list(task\_list)**

Add a list of tasks to be added

**Parameters task\_list** – list of tuples first index is the name of the method second is defining the arguments by keyword in a dict

**classmethod apply\_modifications(xmltree, nmmp\_lines, modification\_tasks, validate\_changes=True)**

Applies given modifications to the fleurinp lxml tree. It also checks if a new lxml tree is validated against schema. Does not rise an error if inp.xml is not validated, simple prints a message about it.

**Parameters**

- **xmltree** – a lxml tree to be modified (IS MODIFIED INPLACE)
- **nmmp\_lines** – a n\_mmp\_mat file to be modified (IS MODIFIED INPLACE)
- **modification\_tasks** – a list of modification tuples

- **validate\_changes** – bool optional (default True), if True after all tasks are performed both the xmltree and nmmp\_lines are checked for consistency

**Returns** a modified lxml tree and a modified n\_mmp\_mat file

#### **changes()**

Prints out all changes currently registered on this instance

#### **create\_tag(\*args, \*\*kwargs)**

Appends a [create\\_tag\(\)](#) to the list of tasks that will be done on the xmltree.

##### **Parameters**

- **tag** – str of the tag to create or etree Element with the same name
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create\_parents** – bool optional (default False), if True and the given xpath has no results the the parent tags are created recursively
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

##### **Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

#### **delete\_att(\*args, \*\*kwargs)**

Appends a [delete\\_att\(\)](#) to the list of tasks that will be done on the xmltree.

##### **Parameters**

- **tag** – str of the attribute to delete
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the parent nodes to delete a attribute. By default all nodes are used.

##### **Kwargs:**

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other

#### **delete\_tag(\*args, \*\*kwargs)**

Appends a [delete\\_tag\(\)](#) to the list of tasks that will be done on the xmltree.

##### **Parameters**

- **tag** – str of the tag to delete
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the parent nodes to delete a tag. By default all nodes are used.

##### **Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**classmethod** `fromList(task_list, *args, **kwargs)`

Instantiate the FleurXMLModifier from a list of tasks to be added immediately

**Parameters** **task\_list** – list of tuples first index is the name of the method second is defining the arguments by keyword in a dict

Other arguments are passed on to the `__init__` method

**Returns** class with the task list instantiated

**get\_avail\_actions()**

Returns the allowed functions from FleurXMLModifier

**modify\_xmlfile**(*original\_inpxmlfile*, *original\_nmmp\_file=None*, *validate\_changes=True*)

Applies the registered modifications to a given inputfile

**Parameters**

- **original\_inpxmlfile** – either path to the inp.xml file, opened file handle or a xml etree to be parsed
- **original\_nmmp\_file** – path or list of str to a corresponding density matrix file

**Raises** **ValueError** – if the parsing of the input file

**Returns** a modified xmltree and if existent a modified density matrix file

**replace\_tag**(\*args, \*\*kwargs)

Appends a `replace_tag()` to the list of tasks that will be done on the xmltree.

**Parameters**

- **tag** – str of the tag to replace
- **newelement** – a new tag
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the parent nodes to replace a tag. By default all nodes are used.

**Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**rotate\_nmmpmat**(\*args, \*\*kwargs)

Appends a `rotate_nmmpmat()` to the list of tasks that will be done on the xmltree.

**Parameters**

- **species\_name** – string, name of the species you want to change
- **orbital** – integer, orbital quantum number of the LDA+U procedure to be modified
- **phi** – float, angle (radian), by which to rotate the density matrix
- **theta** – float, angle (radian), by which to rotate the density matrix

**set\_atomgroup**(\*args, \*\*kwargs)

Appends a `set_atomgroup()` to the list of tasks that will be done on the xmltree.

### Parameters

- **attributedict** – a python dict specifying what you want to change.
- **position** – position of an atom group to be changed. If equals to ‘all’, all species will be changed
- **species** – atom groups, corresponding to the given species will be changed
- **create** – bool, if species does not exist create it and all subtags?

**attributedict** is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a beta noco parameter it can be done via:

```
'attributedict': {'nocoParams': {'beta': val}}
```

**set\_atomgroup\_label(\*args, \*\*kwargs)**

Appends a [set\\_atomgroup\\_label\(\)](#) to the list of tasks that will be done on the xmltree.

### Parameters

- **atom\_label** – string, a label of the atom which specie will be changed. ‘all’ to change all the species
- **attributedict** – a python dict specifying what you want to change.
- **create** – bool, if species does not exist create it and all subtags?

**attributedict** is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a beta noco parameter it can be done via:

```
'attributedict': {'nocoParams': {'beta': val}}
```

**set\_attr\_value(\*args, \*\*kwargs)**

Appends a [set\\_attr\\_value\(\)](#) to the list of tasks that will be done on the xmltree.

### Parameters

- **attributename** – the attribute name to set
- **attribv** – value or list of values to set
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** – bool optional (default False), if True the tag is created if is missing

### Kwargs:

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other

**set\_complex\_tag(\*args, \*\*kwargs)**

Appends a [set\\_complex\\_tag\(\)](#) to the list of tasks that will be done on the xmltree.

### Parameters

- **tag\_name** – name of the tag to set
- **attributedict** – Keys in the dictionary correspond to names of tags and the values are the modifications to do on this tag (attributename, subdict with changes to the subtag, ...)
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create** – bool optional (default False), if True and the path, where the complex tag is set does not exist it is created

**Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**set\_first\_attrib\_value**(\*args, \*\*kwargs)

Appends a [set\\_first\\_attrib\\_value\(\)](#) to the list of tasks that will be done on the xmltree.

**Parameters**

- **attributename** – the attribute name to set
- **attribv** – value or list of values to set
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create** – bool optional (default False), if True the tag is created if is missing

**Kwargs:**

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other

**set\_first\_text**(\*args, \*\*kwargs)

Appends a [set\\_first\\_text\(\)](#) to the list of tasks that will be done on the xmltree.

**Parameters**

- **tag\_name** – str name of the tag, where the text should be set
- **text** – value or list of values to set
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create** – bool optional (default False), if True the tag is created if is missing

**Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**set\_inpchanges**(\*args, \*\*kwargs)

Appends a [set\\_inpchanges\(\)](#) to the list of tasks that will be done on the xmltree.

**Parameters**

- **change\_dict** – a dictionary with changes

- **path\_spec** – dict, with ggf. necessary further specifications for the path of the attribute

An example of change\_dict:

```
change_dict = {'itmax' : 1,
               'l_noco': True,
               'ctail': False,
               'l_ss': True}
```

**set\_kpath**(\*args, \*\*kwargs)

Appends a [set\\_kpath\(\)](#) to the list of tasks that will be done on the xmltree.

**Warning:** This method is only supported for input versions before the Max5 release

#### Parameters

- **kpath** – a dictionary with kpoint name as key and k point coordinate as value
- **count** – number of k-points
- **gamma** – bool that controls if the gamma-point should be included in the k-point mesh

**set\_kpointlist**(\*args, \*\*kwargs)

Appends a [set\\_kpointlist\(\)](#) to the list of tasks that will be done on the xmltree.

**Warning:** For input versions Max4 and older **all** keyword arguments are not valid (*name*, *kpoint\_type*, *special\_labels*, *switch* and *overwrite*)

#### Parameters

- **kpoints** – list or array containing the **relative** coordinates of the kpoints
- **weights** – list or array containing the weights of the kpoints
- **name** – str for the name of the list, if not given a default name is generated
- **kpoint\_type** – str specifying the type of the kPointList ('path', 'mesh', 'spex', 'tria', ...)
- **special\_labels** – dict mapping indices to labels. The labels will be inserted for the kpoints corresponding to the given index
- **switch** – bool, if True the kPointlist will be used by Fleur when starting the next calculation
- **overwrite** – bool, if True and a kPointlist with the given name already exists it will be overwritten

**set\_nkpts**(\*args, \*\*kwargs)

Appends a [set\\_nkpts\(\)](#) to the list of tasks that will be done on the xmltree.

**Warning:** This method is only supported for input versions before the Max5 release

#### Parameters

- **count** – number of k-points
- **gamma** – bool that controls if the gamma-point should be included in the k-point mesh

**set\_nmmpmat**(\*args, \*\*kwargs)

Appends a [set\\_nmmpmat\(\)](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **species\_name** – string, name of the species you want to change
- **orbital** – integer, orbital quantum number of the LDA+U procedure to be modified
- **spin** – integer, specifies which spin block should be modified
- **state\_occupations** – list, sets the diagonal elements of the density matrix and everything else to zero
- **denmat** – matrix, specify the density matrix explicitly
- **phi** – float, optional angle (radian), by which to rotate the density matrix before writing it
- **theta** – float, optional angle (radian), by which to rotate the density matrix before writing it

**set\_simple\_tag**(\*args, \*\*kwargs)

Appends a [set\\_simple\\_tag\(\)](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **tag\_name** – str name of the tag to modify/set
- **changes** – list of dicts or dict with the changes. Elements in list describe multiple tags. Keys in the dictionary correspond to {'attributename': attributevalue}
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create\_parents** – bool optional (default False), if True and the path, where the simple tags are set does not exist it is created

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**set\_species**(\*args, \*\*kwargs)

Appends a [set\\_species\(\)](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **species\_name** – string, name of the specie you want to change Can be name of the species, 'all' or 'all-<string>' (sets species with the string in the species name)
- **attributedict** – a python dict specifying what you want to change.
- **create** – bool, if species does not exist create it and all subtags?

**attributedict** is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a MT radius it can be done via:

```
attributedict = {'mtSphere' : {'radius' : 2.2}}
```

Another example:

```
'attributedict': {'special': {'socscale': 0.0}}
```



that switches SOC terms on a certain specie. `mtSphere`, `atomicCutoffs`, `energyParameters`, `lo`, `electronConfig`, `nocoParams`, `ldaU` and `special` keys are supported. To find possible keys of the inner dictionary please refer to the FLEUR documentation [flapw.de](http://flapw.de)

**set\_species\_label(\*args, \*\*kwargs)**

Appends a [`set\_species\_label\(\)`](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **atom\_label** – string, a label of the atom which specie will be changed. ‘all’ to change all the species
- **attributedict** – a python dict specifying what you want to change.

**set\_text(\*args, \*\*kwargs)**

Appends a [`set\_text\(\)`](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **tag\_name** – str name of the tag, where the text should be set
- **text** – value or list of values to set
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** – bool optional (default False), if True the tag is created if it is missing

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**shift\_value(\*args, \*\*kwargs)**

Appends a [`shift\_value\(\)`](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **change\_dict** – a python dictionary with the keys to shift and the shift values.
- **mode** – ‘abs’ if change given is absolute, ‘rel’ if relative
- **path\_spec** – dict, with ggf. necessary further specifications for the path of the attribute

An example of `change_dict`:

```
change_dict = {'itmax' : 1, 'dVac': -0.123}
```

**shift\_value\_species\_label(\*args, \*\*kwargs)**

Appends a [`shift\_value\_species\_label\(\)`](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **atom\_label** – string, a label of the atom which specie will be changed. ‘all’ if set up all species
- **attributename** – name of the attribute to change
- **value\_given** – value to add or to multiply by
- **mode** – ‘rel’ for multiplication or ‘abs’ for addition

**Kwargs if the attributename does not correspond to a unique path:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**switch\_kpointset**(\*args, \*\*kwargs)

Appends a [switch\\_kpointset\(\)](#) to the list of tasks that will be done on the xmltree.

**Warning:** This method is only supported for input versions after the Max5 release

**Parameters** **list\_name** – name of the kPoint set to use

**undo**(*revert\_all=False*)

Cancels the last change or all of them

**Parameters** **revert\_all** – set True if need to cancel all the changes, False if the last one.

**xml\_create\_tag**(\*args, \*\*kwargs)

Appends a [xml\\_create\\_tag\(\)](#) to the list of tasks that will be done on the xmltree.

**Parameters**

- **xpath** – a path where to place a new tag
- **element** – a tag name or etree Element to be created
- **place\_index** – defines the place where to put a created tag
- **tag\_order** – defines a tag order
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**xml\_delete\_att**(\*args, \*\*kwargs)

Appends a [xml\\_delete\\_att\(\)](#) to the list of tasks that will be done on the xmltree.

**Parameters**

- **xpath** – a path to the attribute to be deleted
- **attrib** – the name of an attribute
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**xml\_delete\_tag**(\*args, \*\*kwargs)

Appends a [xml\\_delete\\_tag\(\)](#) to the list of tasks that will be done on the xmltree.

**Parameters**

- **xpath** – a path to the tag to be deleted
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**xml\_replace\_tag**(\*args, \*\*kwargs)

Appends a [xml\\_replace\\_tag\(\)](#) to the list of tasks that will be done on the xmltree.

**Parameters**

- **xpath** – a path to the tag to be replaced
- **newelement** – a new tag

- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**xml\_set\_attrib\_value\_no\_create**(\*args, \*\*kwargs)

Appends a [xml\\_set\\_attrib\\_value\\_no\\_create\(\)](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **xpath** – a path where to set the attributes
- **attributename** – the attribute name to set
- **attribv** – value or list of values to set (if not str they will be converted with *str(value)*)
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

**xml\_set\_text\_no\_create**(\*args, \*\*kwargs)

Appends a [xml\\_set\\_text\\_no\\_create\(\)](#) to the list of tasks that will be done on the xmltree.

#### Parameters

- **xpath** – a path where to set the attributes
- **text** – value or list of values to set (if not str they will be converted with *str(value)*)
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

**class** masci\_tools.io.fleurxmlmodifier.**ModifierTask**(name, args, kwargs)

**args**

Alias for field number 1

**kwargs**

Alias for field number 2

**name**

Alias for field number 0

## Functions/Classes for loading/validating fleur XML files

**class** masci\_tools.io.parsers.fleur.fleur\_schema.schema\_dict.**SchemaDict**(\*args, xmlschema=None, \*\*kwargs)

Base class for schema dictionaries. Is locked on initialization with [freeze\(\)](#). Holds a reference to the xmlSchema for validating files.

Also provides interfaces for utility functions

**Parameters** **xmlschema** (Optional[XMLSchema]) – etree.XMLSchema object for validating files

All other arguments are passed on to [LockableDict](#)

**attrib\_xpath**(name, contains=None, not\_contains=None, exclude=None, tag\_name=None)

Tries to find a unique path from the schema\_dict based on the given name of the attribute and additional further specifications

#### Parameters

- **name** (str) – str, name of the attribute
- **root\_tag** – str, name of the tag from which the path should be relative

- **contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to be in the final path
- **not\_contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to NOT be in the final path
- **exclude** (`Optional[Iterable[str]]`) – list of str, here specific types of attributes can be excluded valid values are: `settable`, `settable_contains`, `other`
- **tag\_name** (`Optional[str]`) – str, if given this name will be used to find a path to a tag with the same name in `tag_xpath()`

**Return type** `str`

**Returns** str, xpath to the tag with the given attribute

**Raises**

- **NoPathFound** – If no path matching the criteria could be found
- **NoUniquePathFound** – If multiple paths matching the criteria are found

**classmethod** `clear_cache()`

Remove all stored entries in the schema dictionary cache

**Return type** `None`

**relative\_attr\_xpath**(*name*, *root\_tag*, *contains=None*, *not\_contains=None*, *exclude=None*, *tag\_name=None*)

Tries to find a unique relative path from the `schema_dict` based on the given name of the attribute name of the root, from which the path should be relative and additional further specifications

**Parameters**

- **schema\_dict** – dict, containing all the path information and more
- **name** (`str`) – str, name of the attribute
- **contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to be in the final path
- **not\_contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to NOT be in the final path
- **exclude** (`Optional[Iterable[str]]`) – list of str, here specific types of attributes can be excluded valid values are: `settable`, `settable_contains`, `other`
- **tag\_name** (`Optional[str]`) – str, if given this name will be used to find a path to a tag with the same name in `relative_tag_xpath()`

**Return type** `str`

**Returns** str, xpath for the given tag

**Raises**

- **NoPathFound** – If no path matching the criteria could be found
- **NoUniquePathFound** – If multiple paths matching the criteria are found

**relative\_tag\_xpath**(*name*, *root\_tag*, *contains=None*, *not\_contains=None*)

Tries to find a unique relative path from the `schema_dict` based on the given name of the tag name of the root, from which the path should be relative and additional further specifications

**Parameters**

- **name** (`str`) – str, name of the tag

- **root\_tag** (*str*) – str, name of the tag from which the path should be relative
- **contains** (*Union[str, Iterable[str], None]*) – str or list of str, this string has to be in the final path
- **not\_contains** (*Union[str, Iterable[str], None]*) – str or list of str, this string has to NOT be in the final path

**Return type** *str*

**Returns** str, xpath for the given tag

**Raises** **ValueError** – If no unique path could be found

**tag\_info**(*name, contains=None, not\_contains=None, parent=False, \*\*kwargs*)

Tries to find a unique path from the schema\_dict based on the given name of the tag and additional further specifications and returns the tag\_info entry for this tag

**Parameters**

- **schema\_dict** – dict, containing all the path information and more
- **name** (*str*) – str, name of the tag
- **contains** (*Union[str, Iterable[str], None]*) – str or list of str, this string has to be in the final path
- **not\_contains** (*Union[str, Iterable[str], None]*) – str or list of str, this string has to NOT be in the final path
- **parent** (*bool*) – bool, if True the tag\_info for the parent of the tag is returned

**Return type** *Dict[str, Any]*

**Returns** dict, tag\_info for the found xpath

**tag\_xpath**(*name, contains=None, not\_contains=None*)

Tries to find a unique path from the schema\_dict based on the given name of the tag and additional further specifications

**Parameters**

- **name** (*str*) – str, name of the tag
- **contains** (*Union[str, Iterable[str], None]*) – str or list of str, this string has to be in the final path
- **not\_contains** (*Union[str, Iterable[str], None]*) – str or list of str, this string has to NOT be in the final path

**Return type** *str*

**Returns** str, xpath for the given tag

**Raises**

- **NoPathFound** – If no path matching the criteria could be found
- **NoUniquePathFound** – If multiple paths matching the criteria are found

This module provides easy functions for loading a input/output xml file of fleur and providing a parsed xml etree together with its corresponding schema dict

**masci\_tools.io.io\_fleurxml.load\_inpxml**(*inpxmlfile, logger=None, base\_url=None, \*\*kwargs*)

Loads a inp.xml file for fleur together with its corresponding schema dictionary

**Parameters** `inpxmlfile` (`Union[_ElementTree, str, Path, bytes, PathLike, IO]`) – either path to the inpxml file, opened file handle or a xml etree to be parsed

**Return type** `Tuple[_ElementTree, InputSchemaDict]`

**Returns** parsed xmltree of the inpxmlfile and the schema dictionary for the corresponding input version

`masci_tools.io.io_fleurxml.load_outxml(outxmlfile, logger=None, base_url=None, **kwargs)`

Loads a out.xml file for fleur together with its corresponding schema dictionary

**Parameters** `outxmlfile` (`Union[_ElementTree, str, Path, bytes, PathLike, IO]`) – either path to the out.xml file, opened file handle or a xml etree to be parsed

**Return type** `Tuple[_ElementTree, OutputSchemaDict]`

**Returns** parsed xmltree of the outxmlfile and the schema dictionary for the corresponding output version

## Helper functions for the n\_mmp\_mat file

Simple IO routines for creating text for nmmp\_mat files

`masci_tools.io.io_nmmpmat.format_nmmpmat(denmat)`

Format a given 7x7 complex numpy array into the format for the n\_mmp\_mat file

Results in list of 14 strings. Every 2 lines correspond to one row in array Real and imaginary parts are formatted with 20.13f in alternating order

**Parameters** `denmat` (`ndarray`) – numpy array (7x7) and complex for formatting

**Raises** `ValueError` – If denmat has wrong shape or datatype

**Return type** `List[str]`

**Returns** list of str formatted in lines for the n\_mmp\_mat file

`masci_tools.io.io_nmmpmat.read_nmmpmat_block(nmmp_lines, block_index)`

Convert 14 line block of given nmmp\_lines into 7x7 complex numpy array

**Parameters**

- `nmmp_lines` – list of lines in the n\_mmp\_mat file
- `block_index` (`int`) – int specifying which 14 line block to convert

**Return type** `ndarray`

**Returns** 7x7 complex numpy array of the numbers in the given block

`masci_tools.io.io_nmmpmat.rotate_nmmpmat_block(denmat, orbital, phi=None, theta=None)`

Rotate the given 7x7 complex numpy array with the d-wigner matrix corresponding to the given orbital and angles

**Parameters**

- `denmat` (`ndarray`) – complex numpy array of shape 7x7
- `orbital` (`int`) – int of the orbital for the current block
- `phi` (`Optional[float]`) – float, angle (radian), by which to rotate the density matrix
- `theta` (`Optional[float]`) – float, angle (radian), by which to rotate the density matrix

**Return type** `ndarray`

**Returns** denmat rotated by the d-wigner matrix

`masci_tools.io.io_nmmpmat.write_nmmpmat(orbital, denmat, phi=None, theta=None)`

Generate list of str for n\_mmp\_mat file from given numpy array

#### Parameters

- **orbital** (`int`) – int of the orbital for the current block
- **denmat** (`ndarray`) – complex numpy array of shape  $(2*\text{orbital}+1 \times 2*\text{orbital}+1)$  with the wanted occupations
- **phi** (`Optional[float]`) – float, angle (radian), by which to rotate the density matrix
- **theta** (`Optional[float]`) – float, angle (radian), by which to rotate the density matrix

**Return type** `List[str]`

**Returns** list of str formatted in lines for the n\_mmp\_mat file

`masci_tools.io.io_nmmpmat.write_nmmpmat_from_orbitals(orbital, orbital_occupations, phi=None, theta=None)`

Generate list of str for n\_mmp\_mat file from orbital occupations

orbital occupations are provided in the following order (expressed as the spherical harmonics since it can be used for all orbitals):

- $Y_l^0$
- $1/\sqrt{2} (Y_l^{-1} + Y_l^1)$
- $i/\sqrt{2} (Y_l^{-1} - Y_l^1)$
- $1/\sqrt{2} (Y_l^{-2} + Y_l^2)$
- $i/\sqrt{2} (Y_l^{-2} - Y_l^2)$
- and so on ...

#### Parameters

- **orbital** (`int`) – int of the orbital for the current block
- **orbital\_occupations** (`List[float]`) – list like with length  $2*\text{orbital}+1$  with the occupations of the orbitals
- **phi** (`Optional[float]`) – float, angle (radian), by which to rotate the density matrix
- **theta** (`Optional[float]`) – float, angle (radian), by which to rotate the density matrix

**Return type** `List[str]`

**Returns** list of str formatted in lines for the n\_mmp\_mat file

`masci_tools.io.io_nmmpmat.write_nmmpmat_from_states(orbital, state_occupations, phi=None, theta=None)`

Generate list of str for n\_mmp\_mat file from diagonal occupations

#### Parameters

- **orbital** (`int`) – int of the orbital for the current block
- **state\_occupations** (`List[float]`) – list like with length  $2*\text{orbital}+1$  with the occupations of the diagonals
- **phi** (`Optional[float]`) – float, angle (radian), by which to rotate the density matrix

- **theta** (*Optional[float]*) – float, angle (radian), by which to rotate the density matrix

**Return type** `List[str]`

**Returns** list of str formatted in lines for the `n_mmp_mat` file

### 6.1.3.3 General HDF5 parser

This module contains a generic HDF5 reader

**class** `masci_tools.io.parsers.hdf5.reader.AttribTransformation(name, attrib_name, args, kwargs)`

**args**

Alias for field number 2

**attrib\_name**

Alias for field number 1

**kwargs**

Alias for field number 3

**name**

Alias for field number 0

**class** `masci_tools.io.parsers.hdf5.reader.HDF5Reader(file, move_to_memory=True)`

Class for reading in data from hdf5 files using a specified recipe

**Parameters**

- **file** – filepath to hdf file or opened file handle (mode ‘rb’)
- **move\_to\_memory** – bool if True after reading and transforming the data all leftover `h5py.Datasets` are moved into `np.arrays`

The recipe is passed to the `HDF5Reader.read()` method and consists of a dict specifying which attributes and datasets to read in and how to transform them

Each attribute/dataset entry corresponds to one entry point in the given `.hdf` file Available transformations can either be found in `transforms` or can be defined by the user with the `hdf5_transformation()` decorator

Basic Usage:

```
from masci_tools.io.parsers.hdf5 import HDF5Reader
import masci_tools.io.parsers.hdf5.recipes as recipes

#This example shows the usage for producing data from a bandstructure calculation
#in Fleur
with HDF5Reader('/path/to/hdf/banddos.hdf') as h5reader:
    data, attributes = h5reader.read(recipe=recipes.FleurBands)
print(data, attributes)
```

**read(recipe=None)**

Extracts datasets from HDF5 file, transforms them and puts all into a namedtuple.

**Parameters** **recipe** – dict with the format given in `recipes`

**Returns** two dicts with the datasets/attributes read in and transformed according to the recipe

**class** `masci_tools.io.parsers.hdf5.reader.Transformation(name, args, kwargs)`



**args**

Alias for field number 1

**kwargs**

Alias for field number 2

**name**

Alias for field number 0

This module defines commonly used recipes for the *HDF5Reader*

**Available are:**

- Recipe for bandstructure calculations with Fleur
- Recipes for almost all DOS calculation modes of Fleur

A Recipe is a python dictionary in a specific format.

A Template Example:

```
from masci_tools.io.parser.hdf5.readers import Transformation, AttribTransformation

RecipeExample = {
    'datasets': {
        'example_dataset': {
            'h5path': '/path/in/hdf/file',
            'transforms': [Transformation(name='get_first_element')]
        },
        'example_attrib_transform': {
            'h5path': '/other/path/in/hdf/file',
            'transforms': [AttribTransformation(name='multiply_by_attribute', attrib_
↪ name='example_attribute')]
        }
    },
    'attributes': {
        'example_attribute': {
            'h5path':
            '/path/in/hdf/file',
            'transforms':
            [Transformation(name='get_attribute', args=('attribName',)),
             Transformation(name='get_first_element')]
        }
    }
}
```

The Recipe consists of two sections 'datasets' and 'attributes'. All data from these two sections will be returned in separate python dictionaries by the *HDF5Reader* class

Each entry in those sections has to have a *h5path* entry, which will specify the dataset to initially read from the hdf file. Then each entry can define a entry *transforms* with a list of the namedtuples imported at the top of the code example. These corresponds to function calls to functions in *transforms* to transform the read in data

Entries in the *attributes* section are read and transformed first and can subsequently be used in transformations for the *datasets*. These correspond to the transforms created with the *AttribTransformation* namedtuple instead of *Transformation*.

`masci_tools.io.parsers.hdf5.recipes.bands_recipe_format(group, simple=False)`  
 Format for bandstructure calculations retrieving weights from the given group

### Parameters

- **group** – str of the group the weights should be taken from
- **simple** – bool, if True no additional weights are retrieved with the produced recipe

**Returns** dict of the recipe to retrieve a bandstructure calculation

`masci_tools.io.parsers.hdf5.recipes.dos_recipe_format(group)`

Format for density of states calculations retrieving the DOS from the given group

**Parameters** **group** – str of the group the DOS should be taken from

**Returns** dict of the recipe to retrieve a DOS calculation

`masci_tools.io.parsers.hdf5.recipes.get_fleur_bands_specific_weights(weight_name, group='Local')`

Recipe for bandstructure calculations only retrieving one additional weight besides the eigenvalues and kpath

### Parameters

- **weight\_name** – key or list of keys of the weight(s) to retrieve
- **group** – optional str (default Local) name of the group from where to take the weights

**Returns** dict of the recipe to retrieve a simple bandstructure plus the one specified weight

Collection of predefined transformations for the *HDF5Reader* class

All Transformation have to be able to handle (or fail gracefully with a clear error) for the following 3 cases:

1. The dataset is still a `h5py.Dataset` and might need to be transformed to a numpy array
2. The dataset is a numpy array
3. The dataset is a dict. This is needed to read arbitrary child dataset, where not all labels are known. Two options can be chosen apply the transformation to all keys in the dict or throw an error

**exception** `masci_tools.io.parsers.hdf5.transforms.HDF5TransformationError`

`masci_tools.io.parsers.hdf5.transforms.add_partial_sums(dataset, attribute_value, pattern_format, make_set=False, replace_entries=None)`

Add entries to the dataset dict (Only available for dict datasets) with sums over entries containing a given pattern formatted with a `attribute_value`

Used for example in the FleurBands recipe to calculate total atom weights with the `pattern_format` `'MT:{}'` and the atomtype as the `attribute_value`

### Parameters

- **dataset** – dataset to transform
- **attribute\_value** – value to multiply by (attribute value passed in from `_transform_dataset`)
- **pattern\_format** – callable returning a formatted string This will be called with every entry in the `attribute_value` list
- **replace\_entries** – list of str under which to enter the entries back

**Returns** dataset with new entries containing the sums over entries matching the given pattern

`masci_tools.io.parsers.hdf5.transforms.add_partial_sums_fixed(dataset, patterns, replace_entries=None)`

Add entries to the dataset dict (Only available for dict datasets) with sums over entries containing a given pattern

Used for example in the FleurBands recipe to calculate total atom weights with the patterns `['MT:1', 'MT:2', ...]`

#### Parameters

- **dataset** – dataset to transform
- **patterns** – list of str to sum entries over
- **replace\_entries** – list of str under which to enter the entries back

**Returns** dataset with new entries containing the sums over entries matching the given pattern

`masci_tools.io.parsers.hdf5.transforms.apply_lambda(dataset, lambda_func)`

**Applies a given lambda function to the dataset** This should be used with care. One possible example is converting to a boolean with lambda x: x==1

#### Parameters

- **dataset** – dataset to transform
- **lambda\_func** – lambda function to apply to the dataset

**Returns** return value of the lambda function

`masci_tools.io.parsers.hdf5.transforms.attributes(dataset)`

Extracts all attributes of the dataset

**Parameters** **dataset** – dataset to transform

**Returns** dict with all the set attributes on the dataset

`masci_tools.io.parsers.hdf5.transforms.calculate_norm(dataset, between_neighbours=False)`

Calculate norms on the given dataset. Calculates the norm of each row in the dataset

#### Parameters

- **dataset** – dataset to transform
- **between\_neighbours** – bool, if True the distance between subsequent entries in the dataset is calculated

**Returns** norms of the given dataset

`masci_tools.io.parsers.hdf5.transforms.convert_to_complex_array(dataset)`

Converts the given dataset of real numbers into dataset of complex numbers. This follows the convention of how complex numbers are normally written out by Fleur (last index 0 real part, last index 1 imag part)

**Parameters** **dataset** – dataset to transform

**Returns** dataset with complex values

`masci_tools.io.parsers.hdf5.transforms.convert_to_str(dataset, join=False)`

Converts the given dataset to a numpy array of type string

#### Parameters

- **dataset** – dataset to transform
- **join** – bool if True the result will be joined together

**Returns** numpy array of dtype str

`masci_tools.io.parsers.hdf5.transforms.cumulative_sum(dataset, beginning_zero=True)`

Calculate the cumulative sum of the dataset

**Parameters** **dataset** – dataset to transform

**Returns** cumulative sum of the dataset

`masci_tools.io.parsers.hdf5.transforms.flatten_array(dataset, order='C')`  
 Flattens the given dataset to one dimensional array. Copies the array !!

**Parameters**

- **dataset** – dataset to transform
- **order** – str { 'C', 'F', 'A', 'K' } flatten in column major or row-major order (see `numpy.flatten` documentation)

**Returns** flattened dataset

`masci_tools.io.parsers.hdf5.transforms.get_all_child_datasets(group, ignore=None, contains=None)`

Get all datasets contained in the given group

**Parameters**

- **group** – h5py object to extract from
- **ignore** – str or iterable of str (optional). These keys will be ignored
- **contains** – str or iterable of str (optional). This phrase has to be in the key

**Returns** a dict with the contained dataset entered with their names as keys

`masci_tools.io.parsers.hdf5.transforms.get_attribute(dataset, attribute_name)`  
 Extracts a specified attribute's value.

**Parameters**

- **dataset** – dataset to transform
- **attribute\_name** – str of the attribute to extract from the dataset

**Returns** value of the attribute on the dataset

`masci_tools.io.parsers.hdf5.transforms.get_first_element(dataset)`  
 Get the first element of the dataset.

**Parameters** **dataset** – dataset to transform

**Returns** first element of the dataset

`masci_tools.io.parsers.hdf5.transforms.get_name(dataset, full_path=False)`  
 Get the name of the dataset.

**Parameters**

- **dataset** – dataset to get the shape
- **full\_path** – bool, if True the full path to the dataset is returned

**Returns** name of the dataset

`masci_tools.io.parsers.hdf5.transforms.get_shape(dataset)`  
 Get the shape of the dataset.

**Parameters** **dataset** – dataset to get the shape

**Returns** shape of the dataset

`masci_tools.io.parsers.hdf5.transforms.hdf5_transformation(*, attribute_needed)`  
 Decorator for registering a function as a transformation functions on the [HDF5Reader](#) class

**Parameters** **attribute\_needed** – bool if True this function takes a previously processed attribute value and is therefore only available for the entries in datasets

`masci_tools.io.parsers.hdf5.transforms.index_dataset(dataset, index)`

Get the n-th element of the dataset.

**Parameters** **dataset** – dataset to transform

**Returns** first element of the dataset

`masci_tools.io.parsers.hdf5.transforms.merge_subgroup_datasets(group, ignore=None, contains=None, ignore_group=None, contains_group=None, stack_results=True, sort_key=None)`

Get all datasets contained in the given group

**Parameters**

- **group** – h5py object to extract from
- **ignore\_group** – str or iterable of str (optional). These keys will be ignored
- **contains\_group** – str or iterable of str (optional). This phrase has to be in the key
- **ignore** – str or iterable of str (optional). These keys of the datasets in the subgroup will be ignored
- **contains** – str or iterable of str (optional). This phrase has to be in the key of the datasets in the subgroup
- **stack\_results** – bool if True the resulting list of datasets will be used to construct one numpy array

**Returns** a dict with the contained dataset of the subgroups of the given group entered with their names as keys

`masci_tools.io.parsers.hdf5.transforms.move_to_memory(dataset)`

Moves the given dataset to memory, if it's not already there Creates numpy arrays for each dataset it finds

**Parameters** **dataset** – dataset to transform

**Returns** dataset with h5py.Datasets converted to numpy arrays

`masci_tools.io.parsers.hdf5.transforms.multiply_array(dataset, matrix, transpose=False)`

Multiply the given dataset with a matrix

**Parameters**

- **dataset** – dataset to multiply
- **matrix** – matrix to multiply by
- **transpose** – bool, if True the given matrix is transposed

**Returns** dataset multiplied with the given matrix

`masci_tools.io.parsers.hdf5.transforms.multiply_by_attribute(dataset, attribute_value, transpose=False)`

Multiply the given dataset with a previously parsed attribute, either scalar or matrix like

**Parameters**

- **dataset** – dataset to transform

- **attribute\_value** – value to multiply by (attribute value passed in from *\_transform\_dataset*)

**Only relevant for matrix multiplication:**

**param transpose** bool if True the Matrix order is transposed before multiplying

**Returns** dataset multiplied with the given attribute\_value

`masci_tools.io.parsers.hdf5.transforms.multiply_scalar(dataset, scalar_value)`

Multiply the given dataset with a scalar\_value

**Parameters**

- **dataset** – dataset to transform
- **scalar\_value** – value to multiply the dataset by

**Returns** the dataset multiplied by the scalar if it is a dict all entries are multiplied

`masci_tools.io.parsers.hdf5.transforms.periodic_elements(dataset)`

**Converts the given dataset (int or list of ints)** To the atomic symbols corresponding to the atomic number

**Parameters** **dataset** – dataset to transform

**Returns** str or array of str with the atomic elements

`masci_tools.io.parsers.hdf5.transforms.repeat_array(dataset, n_repeats)`

Use numpy.repeat to repeat each element in array n-times

**Parameters**

- **dataset** – dataset to transform
- **n\_repeats** – int, time to repeat each element

**Returns** dataset with elements repeated n-times

`masci_tools.io.parsers.hdf5.transforms.repeat_array_by_attribute(dataset, attribute_value)`

Use numpy.repeat to repeat each element in array n-times (given by attribute\_value)

**Parameters**

- **dataset** – dataset to transform
- **attribute\_shape** – int, time to repeat the elements in the given array

**Returns** dataset with elements repeated n-times

`masci_tools.io.parsers.hdf5.transforms.shift_by_attribute(dataset, attribute_value,  
negative=False)`

Shift the dataset by the given value of the attribute

**Parameters**

- **dataset** – dataset to transform
- **attribute\_value** – value to shift the dataset by
- **negative** – bool, if True the scalar\_value will be subtracted

**Returns** the dataset shifted by the scalar if it is a dict all entries are shifted

`masci_tools.io.parsers.hdf5.transforms.shift_dataset(dataset, scalar_value, negative=False)`

Shift the dataset by the given scalar\_value

**Parameters**

- **dataset** – dataset to transform
- **scalar\_value** – value to shift the dataset by
- **negative** – bool, if True the scalar\_value will be subtracted

**Returns** the dataset shifted by the scalar if it is a dict all entries are shifted

`masci_tools.io.parsers.hdf5.transforms.slice_dataset(dataset, slice_arg)`

Slice the dataset with the given slice argument.

**Parameters**

- **dataset** – dataset to transform
- **slice\_arg** – slice to apply to the dataset

**Returns** first element of the dataset

`masci_tools.io.parsers.hdf5.transforms.split_array(dataset, suffixes=None, name=None)`

Split the arrays in a dataset into multiple entries by their first index

If the dataset is a dict the entries will be split up. If the dataset is not a dict a dict is created with the dataset entered under *name* and this will be split up

**Parameters**

- **dataset** – dataset to transform
- **suffix** – Optional list of str to use for suffixes for the split up entries. by default it is the value of the first index of the original array
- **name** – str for the case of the dataset not being a dict. Key for the entry in the new dict for the original dataset. The returned dataset will only contain the split up entries
- **dataset** – dict with the entries split up

`masci_tools.io.parsers.hdf5.transforms.stack_datasets(dataset, axis=0, sort_key=None)`

Stack the entries in the given dict dataset along the given axis

**Parameters**

- **dataset** – dict dataset to transform
- **axis** – int along which axis should be stacked

**Returns** the array resulting from stacking all entries in the dictionary

`masci_tools.io.parsers.hdf5.transforms.sum_over_dict_entries(dataset, overwrite_dict=False, entries=None, dict_entry='sum', entry_format=None)`

Sum the datasets contained in the given dict dataset

**Parameters**

- **dataset** – dataset to transform
- **overwrite\_dict** – bool if True, the result will overwrite the dictionary if False it is entered under *sum* in the dict

**Returns** dataset with summed entries

`masci_tools.io.parsers.hdf5.transforms.tile_array(dataset, n_repeats)`

Use `numpy.tile` to repeat array n-times

**Parameters**

- **dataset** – dataset to transform
- **attribute\_shape** – int, time sto repeat the given array

**Returns** dataset repeated n-times

`masci_tools.io.parsers.hdf5.transforms.tile_array_by_attribute(dataset, attribute_value)`

Use `numpy.tile` to repeat array n-times (given by `attribute_value`)

**Parameters**

- **dataset** – dataset to transform
- **attribute\_shape** – int, time sto repeat the given array

**Returns** dataset repeated n-times

### 6.1.3.4 Definition of default parsing tasks for fleur out.xml

This module contains the dictionary with all defined tasks for the `outxml_parser`. The entries in the `TASK_DEFINITION` dict specify how to parse specific attributes tags.

This needs to be maintained if the specifications do not work for a new schema version because of changed attribute names for example.

Each entry in the `TASK_DEFINITION` dict can contain a series of keys, which by default correspond to the keys in the output dictionary

The following keys are expected in each entry:

- param parse\_type** str, defines which methods to use when extracting the information
- param path\_spec** dict with all the arguments that should be passed to `tag_xpath` or `attrib_xpath` to get the correct path
- param subdict** str, if present the parsed values are put into this key in the output dictionary
- param overwrite\_last** bool, if True no list is inserted and each entry overwrites the last

For the allAttribs `parse_type` there are more keys that can appear:

- param base\_value** str, optional. If given the attribute with this name will be inserted into the key from the `task_definition` all other keys are formatted as `{task_key}_{attribute_name}`
- param ignore** list of str, these attributes will be ignored
- param overwrite** list of str, these attributes will not create a list and overwrite any value that might be there
- param flat** bool, if False the dict parsed from the tag is inserted as a dict into the correspondin key if True the values will be extracted and put into the output dictionary with the format `{task_key}_{attribute_name}`

Each task entry can have additional keys to specify, when to perform the task. These are denoted with underscores in their names and are all optional:

- param \_general** bool, default False. If True the parsing is not performed for each iteration on the iteration node but beforehand and on the root node



**param \_modes** list of tuples, sets conditions for the keys in fleur\_modes to perform the task  
 .e.g. [(‘jspins’, 2), (‘soc’, True)] means only perform this task for a magnetic soc calculation

**param \_minimal** bool, default False, denotes task to perform when minimal\_mode=True is passed to the parser

**param \_special** bool, default False, If true these tasks are not added by default and need to be added manually

**param \_conversions** list of str, gives the names of functions in fleur\_outxml\_conversions to perform after parsing

**The following keys are special at the moment:**

- ‘fleur\_modes’ specifies how to identify the type of the calculation (e.g. SOC, magnetic, lda+u) this is used to determine, whether additional things should be parsed

Following is the current specification of tasks

```

1  __working_out_versions__ = {'0.34'}
2  __base_version__ = '0.34'
3
4  TASKS_DEFINITION = {
5      #-----Definitions for general info from outfile (start, endtime, number_
6      ↪ iterations)-----
7      'general_out_info': {
8          '_general': True,
9          '_minimal': True,
10         '_conversions': ['calculate_walltime'],
11         'creator_name': {
12             'parse_type': 'attrib',
13             'path_spec': {
14                 'name': 'version',
15                 'not_contains': 'git'
16             }
17         },
18         'creator_target_architecture': {
19             'parse_type': 'text',
20             'path_spec': {
21                 'name': 'targetComputerArchitectures'
22             }
23         },
24         'output_file_version': {
25             'parse_type': 'attrib',
26             'path_spec': {
27                 'name': 'fleurOutputVersion'
28             }
29         },
30         'number_of_iterations': {
31             'parse_type': 'numberNodes',
32             'path_spec': {
33                 'name': 'iteration'
34             }
35         },
36         'number_of_atoms': {
37             'parse_type': 'attrib',

```

(continues on next page)

(continued from previous page)

```

37         'path_spec': {
38             'name': 'nat'
39         }
40     },
41     'number_of_atom_types': {
42         'parse_type': 'attrib',
43         'path_spec': {
44             'name': 'ntype'
45         }
46     },
47     'number_of_kpoints': {
48         'parse_type': 'attrib',
49         'path_spec': {
50             'name': 'count',
51             'contains': 'numericalParameters'
52         }
53     },
54     'start_date': {
55         'parse_type': 'allAttribs',
56         'path_spec': {
57             'name': 'startDateAndTime'
58         },
59         'ignore': ['zone'],
60         'flat': False,
61     },
62     'end_date': {
63         'parse_type': 'allAttribs',
64         'path_spec': {
65             'name': 'endDateAndTime'
66         },
67         'ignore': ['zone'],
68         'flat': False,
69     }
70 },
71 -----Definitions for general info from input section of outfile (kmax, symmetries,
72 ..)-----
73 'general_inp_info': {
74     '_general': True,
75     '_minimal': True,
76     'title': {
77         'parse_type': 'text',
78         'path_spec': {
79             'name': 'comment'
80         }
81     },
82     'kmax': {
83         'parse_type': 'attrib',
84         'path_spec': {
85             'name': 'Kmax'
86         }
87     },
88     'gmax': {

```

(continues on next page)

(continued from previous page)

```

88         'parse_type': 'attrib',
89         'path_spec': {
90             'name': 'Gmax'
91         }
92     },
93     'number_of_spin_components': {
94         'parse_type': 'attrib',
95         'path_spec': {
96             'name': 'jspins'
97         }
98     },
99     'number_of_symmetries': {
100         'parse_type': 'numberNodes',
101         'path_spec': {
102             'name': 'symOp'
103         }
104     },
105     'number_of_species': {
106         'parse_type': 'numberNodes',
107         'path_spec': {
108             'name': 'species'
109         }
110     },
111     'film': {
112         'parse_type': 'exists',
113         'path_spec': {
114             'name': 'filmPos'
115         }
116     },
117 },
118 #-----Defintions for lda+u info from input section (species, ldau tags)-----
119 'ldau_info': {
120     '_general': True,
121     '_modes': [('ldau', True)],
122     '_conversions': ['convert_ldau_definitions'],
123     'parsed_ldau': {
124         'parse_type': 'allAttribs',
125         'path_spec': {
126             'name': 'ldaU',
127             'contains': 'species'
128         },
129         'subdict': 'ldau_info',
130         'flat': False,
131         'only_required': True
132     },
133     'ldau_species': {
134         'parse_type': 'parentAttribs',
135         'path_spec': {
136             'name': 'ldaU',
137             'contains': 'species'
138         },
139         'subdict': 'ldau_info',

```

(continues on next page)

(continued from previous page)

```

140         'flat': False,
141         'only_required': True
142     }
143 },
144 #-----Defintions for relaxation info from input section (bravais matrix, atompos)
145 #-----for Bulk and film
146 'bulk_relax_info': {
147     '_general': True,
148     '_modes': [('relax', True), ('film', False)],
149     '_conversions': ['convert_relax_info'],
150     'lat_row1': {
151         'parse_type': 'text',
152         'path_spec': {
153             'name': 'row-1',
154             'contains': 'bulkLattice/bravais'
155         }
156     },
157     'lat_row2': {
158         'parse_type': 'text',
159         'path_spec': {
160             'name': 'row-2',
161             'contains': 'bulkLattice/bravais'
162         }
163     },
164     'lat_row3': {
165         'parse_type': 'text',
166         'path_spec': {
167             'name': 'row-3',
168             'contains': 'bulkLattice/bravais'
169         }
170     },
171     'atom_positions': {
172         'parse_type': 'text',
173         'path_spec': {
174             'name': 'relPos'
175         }
176     },
177     'position_species': {
178         'parse_type': 'parentAttribs',
179         'path_spec': {
180             'name': 'relPos'
181         },
182         'flat': False,
183         'only_required': True
184     },
185     'element_species': {
186         'parse_type': 'allAttribs',
187         'path_spec': {
188             'name': 'species'
189         },
190         'flat': False,
191         'ignore': ['vcaAddCharge', 'magField']

```

(continues on next page)

(continued from previous page)

```

192     },
193 },
194 'film_relax_info': {
195     '_general': True,
196     '_modes': [('relax', True), ('film', True)],
197     '_conversions': ['convert_relax_info'],
198     'lat_row1': {
199         'parse_type': 'text',
200         'path_spec': {
201             'name': 'row-1',
202             'contains': 'filmLattice/bravais'
203         }
204     },
205     'lat_row2': {
206         'parse_type': 'text',
207         'path_spec': {
208             'name': 'row-2',
209             'contains': 'filmLattice/bravais'
210         }
211     },
212     'lat_row3': {
213         'parse_type': 'text',
214         'path_spec': {
215             'name': 'row-3',
216             'contains': 'filmLattice/bravais'
217         }
218     },
219     'atom_positions': {
220         'parse_type': 'text',
221         'path_spec': {
222             'name': 'filmPos'
223         }
224     },
225     'position_species': {
226         'parse_type': 'parentAttribs',
227         'path_spec': {
228             'name': 'filmPos'
229         },
230         'flat': False,
231         'only_required': True
232     },
233     'element_species': {
234         'parse_type': 'allAttribs',
235         'path_spec': {
236             'name': 'species'
237         },
238         'flat': False,
239         'ignore': ['vcaAddCharge', 'magField']
240     },
241 },
242 #---General iteration tasks
243 # iteration number

```

(continues on next page)

(continued from previous page)

```

244 # total energy (only total or also contributions, also lda+u correction)
245 # distances (nonmagnetic and magnetic, lda+u density matrix)
246 # charges (total, interstitial, mt sphere)
247 # fermi energy and bandgap
248 # magnetic moments
249 # orbital magnetic moments
250 # forces
251 'iteration_number': {
252     '_minimal': True,
253     'number_of_iterations_total': {
254         'parse_type': 'attrib',
255         'path_spec': {
256             'name': 'overallNumber'
257         },
258         'overwrite_last': True,
259     }
260 },
261 'total_energy': {
262     '_minimal': True,
263     '_conversions': ['convert_total_energy'],
264     'energy_hartree': {
265         'parse_type': 'singleValue',
266         'path_spec': {
267             'name': 'totalEnergy'
268         }
269     },
270 },
271 'distances': {
272     '_minimal': True,
273     'density_convergence': {
274         'parse_type': 'attrib',
275         'path_spec': {
276             'name': 'distance',
277             'tag_name': 'chargeDensity'
278         }
279     },
280     'density_convergence_units': {
281         'parse_type': 'attrib',
282         'path_spec': {
283             'name': 'units',
284             'tag_name': 'densityConvergence',
285         },
286         'overwrite_last': True,
287     }
288 },
289 'magnetic_distances': {
290     '_minimal': True,
291     '_modes': [('jspin', 2)],
292     'overall_density_convergence': {
293         'parse_type': 'attrib',
294         'path_spec': {
295             'name': 'distance',

```

(continues on next page)

(continued from previous page)

```

296         'tag_name': 'overallChargeDensity'
297     },
298 },
299 'spin_density_convergence': {
300     'parse_type': 'attrib',
301     'path_spec': {
302         'name': 'distance',
303         'tag_name': 'spinDensity'
304     }
305 },
306 },
307 'total_energy_contributions': {
308     'sum_of_eigenvalues': {
309         'parse_type': 'singleValue',
310         'path_spec': {
311             'name': 'sumOfEigenvalues'
312         },
313         'only_required': True
314     },
315     'energy_core_electrons': {
316         'parse_type': 'singleValue',
317         'path_spec': {
318             'name': 'coreElectrons',
319             'contains': 'sumOfEigenvalues'
320         },
321         'only_required': True
322     },
323     'energy_valence_electrons': {
324         'parse_type': 'singleValue',
325         'path_spec': {
326             'name': 'valenceElectrons'
327         },
328         'only_required': True
329     },
330     'charge_den_xc_den_integral': {
331         'parse_type': 'singleValue',
332         'path_spec': {
333             'name': 'chargeDenXCDenIntegral'
334         },
335         'only_required': True
336     },
337 },
338 'ldau_energy_correction': {
339     '_modes': [('ldau', True)],
340     'ldau_energy_correction': {
341         'parse_type': 'singleValue',
342         'path_spec': {
343             'name': 'dftUCorrection'
344         },
345         'subdict': 'ldau_info',
346         'only_required': True
347     },

```

(continues on next page)

(continued from previous page)

```

348 },
349 'nummp_distances': {
350     '_minimal': True,
351     '_modes': [('ldau', True)],
352     'density_matrix_distance': {
353         'parse_type': 'attrib',
354         'path_spec': {
355             'name': 'distance',
356             'contains': 'ldaUDensityMatrixConvergence'
357         },
358         'subdict': 'ldau_info'
359     },
360 },
361 'fermi_energy': {
362     'fermi_energy': {
363         'parse_type': 'singleValue',
364         'path_spec': {
365             'name': 'FermiEnergy'
366         },
367     },
368 },
369 'bandgap': {
370     '_modes': [('bz_integration', 'hist')],
371     'bandgap': {
372         'parse_type': 'singleValue',
373         'path_spec': {
374             'name': 'bandgap'
375         },
376     },
377 },
378 'magnetic_moments': {
379     '_modes': [('jspin', 2)],
380     'magnetic_moments': {
381         'parse_type': 'allAttribs',
382         'path_spec': {
383             'name': 'magneticMoment'
384         },
385         'base_value': 'moment',
386         'ignore': ['atomType']
387     },
388 },
389 'orbital_magnetic_moments': {
390     '_modes': [('jspin', 2), ('soc', True)],
391     'orbital_magnetic_moments': {
392         'parse_type': 'allAttribs',
393         'path_spec': {
394             'name': 'orbMagMoment'
395         },
396         'base_value': 'moment',
397         'ignore': ['atomType']
398     },
399 },

```

(continues on next page)



(continued from previous page)

```

400 'forces': {
401     '_minimal': True,
402     '_modes': [('relax', True)],
403     '_conversions': ['convert_forces'],
404     'force_units': {
405         'parse_type': 'attrib',
406         'path_spec': {
407             'name': 'units',
408             'tag_name': 'totalForcesOnRepresentativeAtoms'
409         },
410         'overwrite_last': True
411     },
412     'parsed_forces': {
413         'parse_type': 'allAttribs',
414         'path_spec': {
415             'name': 'forceTotal'
416         },
417         'flat': False,
418         'only_required': True
419     }
420 },
421 'charges': {
422     '_conversions': ['calculate_total_magnetic_moment'],
423     'spin_dependent_charge': {
424         'parse_type': 'allAttribs',
425         'path_spec': {
426             'name': 'spinDependentCharge',
427             'contains': 'allElectronCharges',
428             'not_contains': 'fixed'
429         },
430         'only_required': True
431     },
432     'total_charge': {
433         'parse_type': 'singleValue',
434         'path_spec': {
435             'name': 'totalCharge',
436             'contains': 'allElectronCharges',
437             'not_contains': 'fixed'
438         },
439         'only_required': True
440     }
441 },
442 #-----Tasks for forcetheorem Calculations
443 # DMI, JIJ, MAE, SSDISP
444 'forcetheorem_dmi': {
445     '_special': True,
446     'dmi_force': {
447         'parse_type': 'allAttribs',
448         'path_spec': {
449             'name': 'Entry',
450             'contains': 'DMI'
451         }

```

(continues on next page)

(continued from previous page)

```

452     },
453     'dmi_force_qs': {
454         'parse_type': 'attrib',
455         'path_spec': {
456             'name': 'qpoints',
457             'contains': 'Forcetheorem_DMI'
458         }
459     },
460     'dmi_force_angles': {
461         'parse_type': 'attrib',
462         'path_spec': {
463             'name': 'Angles',
464             'contains': 'Forcetheorem_DMI'
465         }
466     },
467     'dmi_force_units': {
468         'parse_type': 'attrib',
469         'path_spec': {
470             'name': 'units',
471             'contains': 'Forcetheorem_DMI'
472         }
473     }
474 },
475 'forcetheorem_ssdisp': {
476     '_special': True,
477     'spst_force': {
478         'parse_type': 'allAttribs',
479         'path_spec': {
480             'name': 'Entry',
481             'contains': 'SSDISP'
482         }
483     },
484     'spst_force_qs': {
485         'parse_type': 'attrib',
486         'path_spec': {
487             'name': 'qvectors',
488             'contains': 'Forcetheorem_SSDISP'
489         }
490     },
491     'spst_force_units': {
492         'parse_type': 'attrib',
493         'path_spec': {
494             'name': 'units',
495             'contains': 'Forcetheorem_SSDISP'
496         }
497     }
498 },
499 'forcetheorem_mae': {
500     '_special': True,
501     'mae_force': {
502         'parse_type': 'allAttribs',
503         'path_spec': {

```

(continues on next page)

(continued from previous page)

```

504         'name': 'Angle',
505         'contains': 'MAE'
506     },
507 },
508     'mae_force_units': {
509         'parse_type': 'attrib',
510         'path_spec': {
511             'name': 'units',
512             'contains': 'Forcetheorem_MAE'
513         }
514     },
515 },
516     'forcetheorem_jij': {
517         '_special': True,
518         'jij_force': {
519             'parse_type': 'allAttribs',
520             'path_spec': {
521                 'name': 'Config',
522                 'contains': 'JIJ'
523             }
524         },
525         'jij_force_units': {
526             'parse_type': 'attrib',
527             'path_spec': {
528                 'name': 'units',
529                 'contains': 'Forcetheorem_JIJ'
530             }
531         },
532     },
533     'torques': {
534         '_minimum_version': '0.35', #Typo torque/torque before
535         '_optional': True,
536         'torque_x': {
537             'parse_type': 'attrib',
538             'path_spec': {
539                 'name': 'sigma_x',
540                 'contains': 'noncollinearTorque'
541             }
542         },
543         'torque_y': {
544             'parse_type': 'attrib',
545             'path_spec': {
546                 'name': 'sigma_y',
547                 'contains': 'noncollinearTorque'
548             }
549         },
550     },
551     'noco_angles': {
552         '_general': True,
553         '_optional': True,
554         'noco_alpha': {
555             'parse_type': 'attrib',

```

(continues on next page)

(continued from previous page)

```

556         'path_spec': {
557             'name': 'alpha',
558             'tag_name': 'nocoParams',
559             'contains': 'Group'
560         }
561     },
562     'noco_beta': {
563         'parse_type': 'attrib',
564         'path_spec': {
565             'name': 'beta',
566             'tag_name': 'nocoParams',
567             'contains': 'Group'
568         }
569     }
570 },
571 'corelevels': {
572     '_optional': True,
573     'corestates': {
574         'parse_type': 'allAttribs',
575         'path_spec': {
576             'name': 'coreStates'
577         },
578         'subtags': True,
579         'flat': False
580     }
581 }
582 }
```

In this module migration functions for the task definitions are collected

`masci_tools.io.parsers.fleur.task_migrations.migrate_033_to_031(definition_dict)`

Migrate definitions for MaX5 release to MaX4 release

**Changes:**

- LDA+U density matrix distance output did not exist

**Return type** `Dict[str, Dict[str, Any]]`

`masci_tools.io.parsers.fleur.task_migrations.migrate_034_to_033(definition_dict)`

Migrate definitions for MaX5 bugfix release to MaX5 release

**Changes:**

- forcetheorem units attribute did not exist (get from 'sumValenceSingleParticleEnergies')

**Return type** `Dict[str, Dict[str, Any]]`

## 6.1.4 Commandline interface (CLI)

### 6.1.4.1 masci\_tools

CLI for the *masci-tools* library.

```
masci_tools [OPTIONS] COMMAND [ARGS]...
```

#### Options

**-v, --version**  
Show the version and exit.

### fleur-schema

Commands related to the Fleur XML Schemas

```
masci_tools fleur-schema [OPTIONS] COMMAND [ARGS]...
```

#### add

Adds a new xml schema file to the folder in *masci\_tools/io/parsers/fleur/fleur\_schema* corresponding to its version number

```
masci_tools fleur-schema add [OPTIONS] SCHEMA_FILE
```

#### Options

**--overwrite**  
Overwrite any existing schema-file

**--test-xml-file <test\_xml\_file>**  
Example xmlfile for this schema version to test the file parser against

#### Arguments

**SCHEMA\_FILE**  
Required argument

#### validate-input

Validate the given inp.xml file against the Fleur schema stored for the version of the input

```
masci_tools fleur-schema validate-input [OPTIONS] XML_FILE
```

## Arguments

### **XML\_FILE**

Required argument

## **validate-output**

Validate the given out.xml file against the Fleur schema stored for the version of the output

```
masci_tools fleur-schema validate-output [OPTIONS] XML_FILE
```

## Arguments

### **XML\_FILE**

Required argument

## **parse**

Commands for parsing information from KKR/Fleur files

```
masci_tools parse [OPTIONS] COMMAND [ARGS]...
```

## **all-attrs**

Parse all attributes of the specified tag from the given xml file

```
masci_tools parse all-attrs [OPTIONS] XML_FILE
```

## Options

**-n, --name** <name>

**-c, --contains** <contains>

**-nc, --not-contains** <not\_contains>

**--subtags**

**--text**

## Arguments

### **XML\_FILE**

Required argument

## attrib

Parse the specified attribute from the given xml file

```
maschi_tools parse attrib [OPTIONS] XML_FILE
```

### Options

**-n, --name** <name>

**-c, --contains** <contains>

**-nc, --not-contains** <not\_contains>

**-t, --tag** <tag>

### Arguments

**XML\_FILE**

Required argument

## cell

Parse the unit cell definition fo the given xml file

```
maschi_tools parse cell [OPTIONS] XML_FILE
```

### Arguments

**XML\_FILE**

Required argument

## constants

Parse the mathematical constants used in the given xml-file

```
maschi_tools parse constants [OPTIONS] XML_FILE
```

### Arguments

**XML\_FILE**

Required argument

## fleur-modes

Parse the Fleur modes of the given xml file

```
masci_tools parse fleur-modes [OPTIONS] XML_FILE
```

## Arguments

**XML\_FILE**

Required argument

## inp-file

Parse the Fleur inp.xml into a python dictionary

```
masci_tools parse inp-file [OPTIONS] XML_FILE
```

## Arguments

**XML\_FILE**

Required argument

## kpoints

Parse the used kpoints from the given xml-file

```
masci_tools parse kpoints [OPTIONS] XML_FILE
```

## Arguments

**XML\_FILE**

Required argument

## nkpts

Extract the number of kpoints used in the given xml file

```
masci_tools parse nkpts [OPTIONS] XML_FILE
```



## Arguments

### **XML\_FILE**

Required argument

## number-nodes

Return how often the specified tag occurs in the given xml file

```
maschi_tools parse number-nodes [OPTIONS] XML_FILE
```

## Options

**-n, --name** <name>

**-c, --contains** <contains>

**-nc, --not-contains** <not\_contains>

## Arguments

### **XML\_FILE**

Required argument

## out-file

Parse the Fleur out.xml into a python dictionary

```
maschi_tools parse out-file [OPTIONS] XML_FILE
```

## Arguments

### **XML\_FILE**

Required argument

## parameters

Parse the calculation parameters of the given xml file

```
maschi_tools parse parameters [OPTIONS] XML_FILE
```

## Arguments

### **XML\_FILE**

Required argument

## parent-attrs

Parse all attributes of the parent of the specified tag from the given xml file

```
masci_tools parse parent-attrs [OPTIONS] XML_FILE
```

## Options

**-n, --name** <name>

**-c, --contains** <contains>

**-nc, --not-contains** <not\_contains>

## Arguments

### **XML\_FILE**

Required argument

## relaxation

Parse the relaxation information for the given xml file

```
masci_tools parse relaxation [OPTIONS] XML_FILE
```

## Arguments

### **XML\_FILE**

Required argument

## structure

Parse the structure information in the given Fleur xml file

```
masci_tools parse structure [OPTIONS] XML_FILE
```

## Arguments

### **XML\_FILE**

Required argument

## **symmetry**

Parse the symmetry information for the given xml file

```
maschi_tools parse symmetry [OPTIONS] XML_FILE
```

## Arguments

### **XML\_FILE**

Required argument

## **tag-exists**

Return whether the specified tag exists in the given xml file

```
maschi_tools parse tag-exists [OPTIONS] XML_FILE
```

## Options

**-n, --name** <name>

**-c, --contains** <contains>

**-nc, --not-contains** <not\_contains>

## Arguments

### **XML\_FILE**

Required argument

## **text**

Parse the text of the specified tag from the given xml file

```
maschi_tools parse text [OPTIONS] XML_FILE
```

## Options

**-n, --name** <name>  
**-c, --contains** <contains>  
**-nc, --not-contains** <not\_contains>

## Arguments

**XML\_FILE**  
Required argument

## plot

Commands for visualizing data

```
masci_tools plot [OPTIONS] COMMAND [ARGS]...
```

## fleur-bands

Plot bandstructures from the banddos.hdf file from Fleur

```
masci_tools plot fleur-bands [OPTIONS] BANDDOS_FILE
```

## Options

**-w, --weight** <weight>  
**--backend** <backend>  
    **Options** matplotlib | mpl | bokeh  
**--save**  
**--show**  
**-r, --recipe** <recipe>  
    **Options** FleurBands | FleurOrbcompBands | FleurjDOSBands | FleurSimpleBands | FleurMCD-Bands

## Arguments

**BANDDOS\_FILE**  
Required argument

## fleur-dos

Plot density of states from the banddos.hdf file from Fleur

```
masci_tools plot fleur-dos [OPTIONS] BANDDOS_FILE
```

### Options

**--total** <total>

**--interstitial** <interstitial>

**--atoms** <atoms>

**--l\_resolved** <l\_resolved>

**--backend** <backend>

**Options** matplotlib | mpl | bokeh

**--save**

**--show**

**-r, --recipe** <recipe>

**Options** FleurDOS | FleurORBCOMP | FleurJDOS | FleurMCD

### Arguments

**BANDDOS\_FILE**

Required argument

## 6.1.5 Utility Functions/Classes

### 6.1.5.1 Custom Datatypes

This module defines subclasses of UserDict and UserList to be able to prevent unintended modifications

`masci_tools.util.lockable_containers.LockContainer(lock_object)`

Contextmanager for temporarily locking a lockable object. Object is unfrozen when exiting with block

**Parameters** `lock_object` (`Union[LockableList, LockableDict]`) – lockable container (not yet frozen)

**Return type** `Generator`

**class** `masci_tools.util.lockable_containers.LockableDict(*args, recursive=True, **kwargs)`

Subclass of UserDict, which can prevent modifications to itself. Raises *RuntimeError* if modification is attempted.

Use `LockableDict.freeze()` to enforce. `LockableDict.get_unlocked()` returns a copy of the locked object with builtin lists and dicts

**Parameters** `recursive` (`bool`) – bool if True (default) all subitems (lists or dicts) are converted into their lockable counterparts

All other args or kwargs will be passed on to initialize the *UserDict*

**IMPORTANT NOTE:** This is not a direct subclass of dict. So `isinstance(a, dict)` will be False if a is an `LockableDict`

**freeze()**

Freezes the object. This prevents further modifications

**Return type** `None`

**get\_unlocked()**

Get copy of object with builtin lists and dicts

**Return type** `Dict[~S, ~T]`

**property locked:** `bool`

Returns whether the object is locked

**Return type** `bool`

**class** `maschi_tools.util.lockable_containers.LockableList(*args, recursive=True, **kwargs)`

Subclass of `UserList`, which can prevent modifications to itself. Raises `RuntimeError` if modification is attempted.

Use `LockableList.freeze()` to enforce. `LockableList.get_unlocked()` returns a copy of the locked object with builtin lists and dicts

**Parameters** `recursive (bool)` – bool if True (default) all subitems (lists or dicts) are converted into their lockable counterparts

All other args or kwargs will be passed on to initialize the `UserList`

**IMPORTANT NOTE:** This is not a direct subclass of list. So `isinstance(a, list)` will be False if a is an `LockableList`

**append(item)**

`S.append(value)` – append value to the end of the sequence

**Return type** `None`

**clear()**

Clear the list

**Return type** `None`

**extend(other)**

`S.extend(iterable)` – extend sequence by appending elements from the iterable

**Return type** `None`

**freeze()**

Freezes the object. This prevents further modifications

**Return type** `None`

**get\_unlocked()**

Get copy of object with builtin lists and dicts

**Return type** `List[~T]`

**insert(i, item)**

`S.insert(index, value)` – insert value before index

**Return type** `None`

**property locked:** `bool`

Returns whether the object is locked

**Return type** `bool`

**pop**(*i*=- 1)

return the value at index *i* (default last) and remove it from list

**Return type** `~T`

**remove**(*item*)

S.remove(value) – remove first occurrence of value. Raise ValueError if the value is not present.

**Return type** `None`

**reverse**()

S.reverse() – reverse *IN PLACE*

**Return type** `None`

This module defines a small helper class to make case insensitive dictionary lookups available naturally

**class** `masci_tools.util.case_insensitive_dict.CaseInsensitiveDict(*args, upper=False, **kwargs)`  
Dict with case insensitive lookup. Used in Schema dicts to make finding paths for tags and attributes easier. Does not preserve the case of the inserted key. Does not support case insensitive lookups in nested dicts Subclass of `masci_tools.util.lockable_containers.LockableDict`. So can be frozen via the `freeze()` method

**Parameters** **upper** (`bool`) – bool if True the method `upper()` will be used instead of `lower()` to normalize keys

All other args or kwargs will be passed on to initialize the `UserDict`

**IMPORTANT NOTE:** This is not a direct subclass of dict. So `isinstance(a, dict)` will be False if *a* is an `CaseInsensitiveDict`

**class** `masci_tools.util.case_insensitive_dict.CaseInsensitiveFrozenSet(iterable=None, upper=False)`

Frozenset (i.e. immutable set) with case insensitive membership tests. Used in Schema dicts in `tag_info` entries to make flexible classification easy Preserves the case of the entered keys (`original_case()` returns the case of the first encounter)

**Parameters** **iterable** (`Optional[Iterable[~T]]`) – iterable only containing str

**Note:** There might be subtle differences to expected behaviour with the methods `__radd__`, `__ror__`, and so on

**difference**(*\*others*)

Return the difference of two or more sets as a new set.

(i.e. all elements that are in this set but not the others.)

**Return type** `CaseInsensitiveFrozenSet`

**intersection**(*\*others*)

Return the intersection of two sets as a new set.

(i.e. all elements that are in both sets.)

**Return type** `CaseInsensitiveFrozenSet`

**isdisjoint**(*other*)

Return True if two sets have a null intersection.

**Return type** `bool`

**issubset**(*other*)

Report whether another set contains this set.

**Return type** `bool`

**issuperset**(*other*)

Report whether this set contains another set.

**Return type** `bool`

**symmetric\_difference**(*other*)

Return the symmetric difference of two sets as a new set.

(i.e. all elements that are in exactly one of the sets.)

**Return type** `CaseInsensitiveFrozenSet`

**union**(\**others*)

Return the union of sets as a new set.

(i.e. all elements that are in either set.)

**Return type** `CaseInsensitiveFrozenSet`

### 6.1.5.2 Common XML utility

Common functions for parsing input/output files or XMLschemas from FLEUR

`maschi_tools.util.xml.common_functions.abs_to_rel_xpath(xpath, new_root)`

Convert a given xpath to be relative from a tag appearing in the original xpath.

**Parameters**

- **xpath** (`str`) – str of the xpath to convert
- **new\_root** (`str`) – str of the tag from which the new xpath should be relative

**Return type** `str`

**Returns** str of the relative xpath

`maschi_tools.util.xml.common_functions.check_complex_xpath(node, base_xpath, complex_xpath)`

Check that the given complex xpath produces a subset of the results for the simple xpath

**Parameters**

- **node** – root node of an etree or an etree
- **base\_xpath** – str of the xpath without complex syntax
- **complex\_xpath** – str of the xpath to check

**Raises** `ValueError` – If the complex\_xpath does not produce a subset of the results of the base\_xpath

`maschi_tools.util.xml.common_functions.clear_xml(tree)`

Removes comments and executes xinclude tags of an xml tree.

**Parameters** **tree** (`_ElementTree`) – an xml-tree which will be processed

**Return type** `Tuple[_ElementTree, Set[str]]`

**Returns** `cleared_tree`, an xmltree without comments and with replaced xinclude tags

`maschi_tools.util.xml.common_functions.eval_xpath(node, xpath, logger=None, list_return=False, namespaces=None)`

Tries to evaluate an xpath expression. If it fails it logs it. If a absolute path is given (starting with '/') and the tag of the node does not match the root. It will try to find the tag in the path and convert it into a relative path

**Parameters**

- **node** – root node of an etree



- **xpath** – xpath expression (relative, or absolute)
- **logger** – logger object for logging warnings, errors, if not provided all errors will be raised
- **list\_return** – if True, the returned quantity is always a list even if only one element is in it
- **namespaces** – dict, passed to namespaces argument in xpath call

**Returns** text, attribute or a node list

`masci_tools.util.xml.common_functions.get_xml_attribute(node, attributename, logger=None)`

Get an attribute value from a node.

#### Parameters

- **node** (`_Element`) – a node from etree
- **attributename** (`str`) – a string with the attribute name.
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised

**Return type** `Optional[str]`

**Returns** either attributevalue, or None

`masci_tools.util.xml.common_functions.reverse_xinclude(xmltree, schema_dict, included_tags, **kwargs)`

Split the xmltree back up according to the given included tags. The original xmltree will be returned with the corresponding xinclude tags and the included trees are returned in a dict mapping the inserted filename to the extracted tree

Tags for which no known filename is known are returned under unknown-1.xml, ... The following tags have known filenames:

- *relaxation*: relax.xml
- *kPointLists*: kpts.xml
- *symmetryOperations*: sym.xml
- *atomSpecies*: species.xml
- *atomGroups*: atoms.xml

Additional mappings can be given in the keyword arguments

#### Parameters

- **xmltree** (`_ElementTree`) – an xml-tree which will be processed
- **schema\_dict** (`SchemaDict`) – Schema dictionary containing all the necessary information
- **included\_tags** (`Iterable[str]`) – Iterable of str, containing the names of the tags to be excluded

**Return type** `Tuple[_ElementTree, Dict[Union[PathLike, str], _ElementTree]]`

**Returns** xmltree with the inserted xinclude tags and a dict mapping the filenames to the excluded trees

**Raises** `ValueError` – if the tag can not be found in the given xmltree

`masci_tools.util.xml.common_functions.split_off_attrib(xpath)`

Splits off attribute of the given xpath (part after @)

**Parameters** **xpath** (`str`) – str of the xpath to split up

**Return type** `Tuple[str, str]`

`masci_tools.util.xml.common_functions.split_off_tag(xpath)`

Splits off the last part of the given xpath

**Parameters** `xpath` (`str`) – str of the xpath to split up

**Return type** `Tuple[str, str]`

`masci_tools.util.xml.common_functions.validate_xml(xmltree, schema, error_header='File does not validate')`

Checks a given xmltree against a schema and produces a nice error message with all the validation errors collected

**Parameters**

- `xmltree` (`_ElementTree`) – xmltree of the file to validate
- `schema` (`XMLSchema`) – etree.XMLSchema to validate against
- `error_header` (`str`) – str to lead a evtl error message with

**Raises** `etree.DocumentInvalid` if the schema does not validate

**Return type** `None`

Common functions for converting types to and from XML files

`masci_tools.util.xml.converters.convert_fleur_electronconfig(econfig_element)`

Convert electronConfig tag to eConfig string

**Return type** `str`

`masci_tools.util.xml.converters.convert_fleur_lo(loelements)`

Converts lo xml elements from the inp.xml file into a lo string for the inpgen

**Return type** `str`

`masci_tools.util.xml.converters.convert_from_fortran_bool(stringbool)`

Converts a string in this case ('T', 'F', or 't', 'f') to True or False

**Parameters** `stringbool` (`Union[str, bool]`) – a string ('t', 'f', 'F', 'T')

**Return type** `bool`

**Returns** boolean (either True or False)

`masci_tools.util.xml.converters.convert_from_xml(xmlstring, schema_dict, name, text=False, constants=None, logger=None, list_return=False)`

Tries to convert a given string to the types specified in the schema\_dict. First succeeded conversion will be returned

If no logger is given and a attribute cannot be converted an error is raised

**Parameters**

- `stringattribute` – str, Attribute to convert.
- `schema_dict` (`SchemaDict`) – Schema dictionary containing all the information
- `name` (`str`) – name of the attribute or element
- `text` (`bool`) – bool, decides whether to take the definitions for text or attributes
- `constants` (`Optional[Dict[str, float]]`) – dict, of constants defined in fleur input
- `logger` (`Optional[Logger]`) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails

- **list\_return** (*bool*) – if True, the returned quantity is always a list even if only one element is in it

**Return type** `Tuple[Union[int, float, bool, str, List[Union[int, float, bool, str]]], List[Union[int, float, bool, str, List[Union[int, float, bool, str]]]], bool]`

**Returns** The converted value of the first successful conversion

`masci_tools.util.xml.converters.convert_from_xml_explicit(xmlstring, definitions, constants=None, logger=None, list_return=False)`

Tries to convert a given string to the types given in definitions. First successful conversion will be returned

If no logger is given and an attribute cannot be converted an error is raised

#### Parameters

- **stringattribute** – str, Attribute to convert.
- **definitions** (`List[AttributeType]`) – list of `AttributeType` definitions
- **constants** (`Optional[Dict[str, float]]`) – dict, of constants defined in fleur input
- **logger** (`Optional[Logger]`) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise an error is raised, when the first conversion fails
- **list\_return** (*bool*) – if True, the returned quantity is always a list even if only one element is in it

**Return type** `Tuple[Union[int, float, bool, str, List[Union[int, float, bool, str]]], List[Union[int, float, bool, str, List[Union[int, float, bool, str]]]], bool]`

**Returns** The converted value of the first successful conversion

`masci_tools.util.xml.converters.convert_from_xml_single_values(xmlstring, possible_types, constants=None, logger=None)`

Tries to convert a given string attribute to the types given in possible\_types. First successful conversion will be returned

If no logger is given and an attribute cannot be converted an error is raised

#### Parameters

- **stringattribute** – str, Attribute to convert.
- **possible\_types** (`Tuple[Union[Literal['int'], Literal['switch'], Literal['string'], Literal['float'], Literal['float_expression']], ...]`) – list of str What types it will try to convert to
- **constants** (`Optional[Dict[str, float]]`) – dict, of constants defined in fleur input
- **logger** (`Optional[Logger]`) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise an error is raised, when the first conversion fails
- **list\_return** – if True, the returned quantity is always a list even if only one element is in it

**Return type** `Tuple[List[Union[int, float, bool, str]], bool]`

**Returns** The converted value of the first successful conversion

`masci_tools.util.xml.converters.convert_str_version_number(version_str)`

Convert the version number as an integer for easy comparisons

**Parameters** **version\_str** (*str*) – str of the version number, e.g. '0.33'

**Return type** `Tuple[int, int]`

**Returns** tuple of ints representing the version str

`masci_tools.util.xml.converters.convert_to_fortran_bool(boolean)`

Converts a Boolean as string to the format defined in the input

**Parameters** `boolean` (`Union[bool, str]`) – either a boolean or a string ('True', 'False', 'F', 'T')

**Return type** `Union[Literal['T'], Literal['F']]`

**Returns** a string (either 't' or 'f')

`masci_tools.util.xml.converters.convert_to_xml(value, schema_dict, name, text=False, logger=None, list_return=False)`

Tries to convert a given string to the types specified in the `schema_dict`. First succeeded conversion will be returned

If no logger is given and a attribute cannot be converted an error is raised

**Parameters**

- **stringattribute** – str, Attribute to convert.
- **schema\_dict** (`SchemaDict`) – Schema dictionary containing all the information
- **name** (`str`) – name of the attribute or element
- **text** (`bool`) – bool, decides whether to take the definitions for text or attributes
- **constants** – dict, of constants defined in fleur input
- **logger** (`Optional[Logger]`) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list\_return** (`bool`) – if True, the returned quantity is always a list even if only one element is in it

**Return type** `Tuple[Union[str, List[str]], bool]`

**Returns** The converted value of the first successful conversion

`masci_tools.util.xml.converters.convert_to_xml_explicit(value, definitions, logger=None, float_format='.10', list_return=False)`

Tries to convert a given list of values to str for a xml file based on the definitions (length and type). First succeeded conversion will be returned

**Parameters**

- **textvalue** – value to convert
- **definitions** (`List[AttributeType]`) – list of `AttributeType` definitions
- **logger** (`Optional[Logger]`) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list\_return** (`bool`) – if True, the returned quantity is always a list even if only one element is in it

**Return type** `Tuple[Union[str, List[str]], bool]`

**Returns** The converted value of the first succesful conversion

```
masci_tools.util.xml.converters.convert_to_xml_single_values(value, possible_types, logger=None,
                                                             float_format='.10')
```

Tries to convert a given attribute value to a string for a xml file according to the types given in possible\_types. First succeeded conversion will be returned

#### Parameters

- **value** (`Union[Any, Iterable[Any]]`) – value to convert.
- **possible\_types** (`Tuple[Union[Literal['int'], Literal['switch'], Literal['string'], Literal['float'], Literal['float_expression']], ...]`) – list of str What types it will try to convert from
- **logger** (`Optional[Logger]`) – logger object for logging warnings if given the errors are logged and the list is returned with the unconverted values otherwise a error is raised, when the first conversion fails
- **list\_return** – if True, the returned quantity is always a list even if only one element is in it

**Return type** `Tuple[List[str], bool]`

**Returns** The converted str of the value of the first succesful conversion

### 6.1.5.3 XML Setter functions

Functions for modifying the xml input file of Fleur utilizing the schema dict and as little knowledge of the concrete xpaths as possible

```
masci_tools.util.xml.xml_setters_names.add_number_to_attrib(xmltree, schema_dict, attributename,
                                                            add_number, complex_xpath=None,
                                                            mode='abs', occurrences=None,
                                                            **kwargs)
```

Adds a given number to the attribute value in a xmltree specified by the name of the attribute and optional further specification If there are no nodes under the specified xpath an error is raised

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **attributename** – the attribute name to change
- **add\_number** – number to add/multiply with the old attribute value
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **mode** – str (either *rel* or *abs*). *rel* multiplies the old value with *add\_number* *abs* adds the old value and *add\_number*
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

#### Kwargs:

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: set-table, settable\_contains, other

**Returns** xmltree with shifted attribute

```
masci_tools.util.xml.xml_setters_names.add_number_to_first_attr(xmltree, schema_dict,
                                                                attributename, add_number,
                                                                complex_xpath=None,
                                                                mode='abs', **kwargs)
```

Adds a given number to the first occurrence of an attribute value in a xmltree specified by the name of the attribute and optional further specification If there are no nodes under the specified xpath an error is raised

**Parameters**

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **attributename** – the attribute name to change
- **add\_number** – number to add/multiply with the old attribute value
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **mode** – str (either *rel* or *abs*). *rel* multiplies the old value with *add\_number* *abs* adds the old value and *add\_number*

**Kwargs:**

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other

**Returns** xmltree with shifted attribute

```
masci_tools.util.xml.xml_setters_names.create_tag(xmltree, schema_dict, tag, complex_xpath=None,
                                                  create_parents=False, occurrences=None,
                                                  **kwargs)
```

This method creates a tag with a uniquely identified xpath under the nodes of its parent. If there are no nodes evaluated the subtags can be created with *create\_parents=True*

The tag is always inserted in the correct place if a order is enforced by the schema

**Parameters**

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **tag** – str of the tag to create or etree Element with the same name to insert
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create\_parents** – bool optional (default False), if True and the given xpath has no results the the parent tags are created recursively
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Returns** xmltree with created tags

```
masci_tools.util.xml.xml_setters_names.delete_att(xmltree, schema_dict, attrib_name,
                                                    complex_xpath=None, occurrences=None,
                                                    **kwargs)
```

This method deletes a attribute with a uniquely identified xpath.

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **tag** – str of the attribute to delete
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the parent nodes to delete a attribute. By default all nodes are used.

#### Kwargs:

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: set-table, settable\_contains, other

**Returns** xmltree with deleted attributes

```
masci_tools.util.xml.xml_setters_names.delete_tag(xmltree, schema_dict, tag_name,
                                                    complex_xpath=None, occurrences=None,
                                                    **kwargs)
```

This method deletes a tag with a uniquely identified xpath.

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **tag** – str of the tag to delete
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the parent nodes to delete a tag. By default all nodes are used.

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Returns** xmltree with deleted tags

```
masci_tools.util.xml.xml_setters_names.replace_tag(xmltree, schema_dict, tag_name, newelement,
                                                    complex_xpath=None, occurrences=None,
                                                    **kwargs)
```

This method deletes a tag with a uniquely identified xpath.

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **tag** – str of the tag to replace
- **newelement** – etree Element to replace the tag
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the parent nodes to replace a tag. By default all nodes are used.

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Returns** xmltree with replaced tags

```
masci_tools.util.xml.xml_setters_names.set_atomgroup(xmltree, schema_dict, attributedict,
                                                      position=None, species=None, create=False)
```

Method to set parameters of an atom group of the fleur inp.xml file.

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – xml etree of the inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **attributedict** (`Dict[str, Any]`) – a python dict specifying what you want to change.
- **position** (`Union[int, Literal['all'], None]`) – position of an atom group to be changed. If equals to 'all', all species will be changed
- **species** (`Optional[str]`) – atom groups, corresponding to the given species will be changed
- **create** (`bool`) – bool, if species does not exist create it and all subtags?

**Return type** `Union[_Element, _ElementTree]`

**Returns** xml etree of the new inp.xml

**attributedict** is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a beta noco parameter it can be done via:

```
'attributedict': {'nocoParams': {'beta': val}}
```

```
masci_tools.util.xml.xml_setters_names.set_atomgroup_label(xmltree, schema_dict, atom_label,
                                                           attributedict, create=False)
```

This method calls `set_atomgroup()` method for a certain atom species that corresponds to an atom with a given label.

#### Parameters



- **xmldata** (`Union[_Element, _ElementTree]`) – xml etree of the inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **atom\_label** (`str`) – string, a label of the atom which specie will be changed. ‘all’ to change all the species
- **attributedict** (`Dict[str, Any]`) – a python dict specifying what you want to change.
- **create** (`bool`) – bool, if species does not exist create it and all subtags?

**Return type** `Union[_Element, _ElementTree]`

**Returns** xml etree of the new inp.xml

**attributedict** is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a beta noco parameter it can be done via:

```
'attributedict': {'nocoParams': {'beta': val}}
```

```
masci_tools.util.xml.xml_setters_names.set_attr_value(xmldata, schema_dict, attributename,
                                                         attribv, complex_xpath=None,
                                                         occurrences=None, create=False,
                                                         **kwargs)
```

Sets an attribute in a xmldata to a given value, specified by its name and further specifications. If there are no nodes under the specified xpath a tag can be created with `create=True`. The attribute values are converted automatically according to the types of the attribute with `convert_to_xml()` if they are not `str` already.

#### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **attributename** – the attribute name to set
- **attribv** – value or list of values to set
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** – bool optional (default False), if True the tag is created if is missing

#### Kwargs:

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other

**Returns** xmldata with set attribute

```
masci_tools.util.xml.xml_setters_names.set_complex_tag(xmldata, schema_dict, tag_name, changes,
                                                         complex_xpath=None, create=False,
                                                         **kwargs)
```

Function to correctly set tags/attributes for a given tag. Goes through the attributedict and decides based on the schema\_dict, how the corresponding key has to be handled. The tag is specified via its name and evtl. further specification

Supports:

- attributes
- tags with text only
- simple tags, i.e. only attributes (can be optional single/multiple)
- complex tags, will recursively create/modify them

#### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **tag\_name** – name of the tag to set
- **attributedict** – Keys in the dictionary correspond to names of tags and the values are the modifications to do on this tag (attributename, subdict with changes to the subtag, ...)
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create** – bool optional (default False), if True and the path, where the complex tag is set does not exist it is created

Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Returns** xmldata with changes to the complex tag

```
masci_tools.util.xml.xml_setters_names.set_first_attrib_value(xmldata, schema_dict,
                                                                attributename, attribv,
                                                                complex_xpath=None,
                                                                create=False, **kwargs)
```

Sets the first occurrence of an attribute in a xmldata to a given value, specified by its name and further specifications. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are converted automatically according to the types of the attribute with [convert\\_to\\_xml\(\)](#) if they are not *str* already.

#### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **attributename** – the attribute name to set
- **attribv** – value or list of values to set
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create** – bool optional (default False), if True the tag is created if is missing

Kwargs:

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other

**Returns** xmltree with set attribute

```
masci_tools.util.xml.xml_setters_names.set_first_text(xmltree, schema_dict, attributename, attribv,
                                                       complex_xpath=None, create=False,
                                                       **kwargs)
```

Sets the text the first occurrence of a tag in a xmltree to a given value, specified by the name of the tag and further specifications. By default the text will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with *convert\_to\_xml()* if they are not *str* already.

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **tag\_name** – str name of the tag, where the text should be set
- **text** – value or list of values to set
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create** – bool optional (default False), if True the tag is created if is missing

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Returns** xmltree with set text

```
masci_tools.util.xml.xml_setters_names.set_inpchanges(xmltree, schema_dict, change_dict,
                                                       path_spec=None)
```

This method sets all the attribute and texts provided in the change\_dict.

The first occurrence of the attribute/tag is set

#### Parameters

- **xmltree** (*Union[\_Element, \_ElementTree]*) – xml tree that represents inp.xml
- **schema\_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **path\_spec** (*Optional[Dict[str, Any]]*) – dict, with ggf. necessary further specifications for the path of the attribute

**Params change\_dict** dictionary {attrib\_name : value} with all the wanted changes.

An example of change\_dict:

```
change_dict = {'itmax' : 1,
               'l_noco': True,
               'ctail': False,
               'l_ss': True}
```

**Return type** *Union[\_Element, \_ElementTree]*

**Returns** an xmltree of the inp.xml file with changes.

`masci_tools.util.xml.xml_setters_names.set_kpath(xmltree, schema_dict, kpath, count, gamma=False)`  
 Sets a k-path directly into inp.xml as a alternative kpoint set with purpose ‘bands’

**Warning:** This method is only supported for input versions before the Max5 release

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – xml tree that represents inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **kpath** (`Dict[str, Iterable[float]]`) – a dictionary with kpoint name as key and k point coordinate as value
- **count** (`int`) – number of k-points
- **gamma** (`bool`) – bool that controls if the gamma-point should be included in the k-point mesh

**Return type** `Union[_Element, _ElementTree]`

**Returns** an xmltree of the inp.xml file with changes.

`masci_tools.util.xml.xml_setters_names.set_kpath_max4(xmltree, schema_dict, kpath, count, gamma=False)`

Sets a k-path directly into inp.xml as a alternative kpoint set with purpose ‘bands’

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – xml tree that represents inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **kpath** (`Dict[str, Iterable[float]]`) – a dictionary with kpoint name as key and k point coordinate as value
- **count** (`int`) – number of k-points
- **gamma** (`bool`) – bool that controls if the gamma-point should be included in the k-point mesh

**Return type** `Union[_Element, _ElementTree]`

**Returns** an xmltree of the inp.xml file with changes.

`masci_tools.util.xml.xml_setters_names.set_kpointlist(xmltree, schema_dict, kpoints, weights, name=None, kpoint_type='path', special_labels=None, switch=False, overwrite=False)`

Explicitely create a kPointList from the given kpoints and weights. This routine will add the specified kPointList with the given name.

**Warning:** For input versions Max4 and older **all** keyword arguments are not valid (*name*, *kpoint\_type*, *special\_labels*, *switch* and *overwrite*)

#### Parameters

- **xmldata** (`Union[_Element, _ElementTree]`) – xml tree that represents inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **kpoints** (`Iterable[Iterable[float]]`) – list or array containing the **relative** coordinates of the kpoints
- **weights** (`Iterable[float]`) – list or array containing the weights of the kpoints
- **name** (`Optional[str]`) – str for the name of the list, if not given a default name is generated
- **kpoint\_type** (`Literal['path', 'mesh', 'tria', 'tria-bulk', 'spex-mesh']`) – str specifying the type of the kPointList ('path', 'mesh', 'spex', 'tria', ...)
- **special\_labels** (`Optional[Dict[int, str]]`) – dict mapping indices to labels. The labels will be inserted for the kpoints corresponding to the given index
- **switch** (`bool`) – bool, if True the kPointlist will be used by Fleur when starting the next calculation
- **overwrite** (`bool`) – bool, if True and a kPointlist with the given name already exists it will be overwritten

**Return type** `Union[_Element, _ElementTree]`

**Returns** an xmldata of the inp.xml file with changes.

`masci_tools.util.xml.xml_setters_names.set_kpointlist_max4(xmldata, schema_dict, kpoints, weights)`  
 Explicitly create a kPointList from the given kpoints and weights. This routine is specific to input versions Max4 and before and will replace any existing kPointCount, kPointMesh, ... with the specified kPointList

#### Parameters

- **xmldata** (`Union[_Element, _ElementTree]`) – xml tree that represents inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **kpoints** (`Iterable[Iterable[float]]`) – list or array containing the **relative** coordinates of the kpoints
- **weights** (`Iterable[float]`) – list or array containing the weights of the kpoints

**Return type** `Union[_Element, _ElementTree]`

**Returns** an xmldata of the inp.xml file with changes.

`masci_tools.util.xml.xml_setters_names.set_nkpts(xmldata, schema_dict, count, gamma=False)`  
 Sets a k-point mesh directly into inp.xml

**Warning:** This method is only supported for input versions before the Max5 release

#### Parameters

- **xmldata** (`Union[_Element, _ElementTree]`) – xml tree that represents inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **count** (`int`) – number of k-points
- **gamma** (`bool`) – bool that controls if the gamma-point should be included in the k-point mesh

**Return type** `Union[_Element, _ElementTree]`

**Returns** an xmltree of the inp.xml file with changes.

`masci_tools.util.xml.xml_setters_names.set_nkpts_max4(xmltree, schema_dict, count, gamma=False)`  
 Sets a k-point mesh directly into inp.xml specific for inputs of version Max4

**Parameters**

- **xmltree** (`Union[_Element, _ElementTree]`) – xml tree that represents inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **count** (`int`) – number of k-points
- **gamma** (`bool`) – bool that controls if the gamma-point should be included in the k-point mesh

**Return type** `Union[_Element, _ElementTree]`

**Returns** an xmltree of the inp.xml file with changes.

`masci_tools.util.xml.xml_setters_names.set_simple_tag(xmltree, schema_dict, tag_name, changes, complex_xpath=None, create_parents=False, **kwargs)`

Sets one or multiple *simple* tag(s) in an xmltree. A simple tag can only hold attributes and has no subtags. The tag is specified by its name and further specification. If the tag can occur multiple times all existing tags are DELETED and new ones are written. If the tag only occurs once it will automatically be created if its missing.

**Parameters**

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **tag\_name** – str name of the tag to modify/set
- **changes** – list of dicts or dict with the changes. Elements in list describe multiple tags. Keys in the dictionary correspond to {‘attributename’: attributevalue}
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **create\_parents** – bool optional (default False), if True and the path, where the simple tags are set does not exist it is created

**Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Returns** xmltree with set simple tags

`masci_tools.util.xml.xml_setters_names.set_species(xmltree, schema_dict, species_name, attributedict, create=False)`

Method to set parameters of a species tag of the fleur inp.xml file.

**Parameters**

- **xmltree** (`Union[_Element, _ElementTree]`) – xml etree of the inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input

- **species\_name** (`str`) – string, name of the specie you want to change Can be name of the species, 'all' or 'all-<string>' (sets species with the string in the species name)
- **attributedict** (`Dict[str, Any]`) – a python dict specifying what you want to change.
- **create** (`bool`) – bool, if species does not exist create it and all subtags?

**Raises** `ValueError` – if species name is non existent in inp.xml and should not be created. also if other given tags are garbage. (errors from `eval_xpath()` methods)

**Return xmlltree** xml etree of the new inp.xml

**attributedict** is a python dictionary containing dictionaries that specify attributes to be set inside the certain specie. For example, if one wants to set a MT radius it can be done via:

```
attributedict = {'mtSphere' : {'radius' : 2.2}}
```

Another example:

```
'attributedict': {'special': {'socscale': 0.0}}
```

that switches SOC terms on a certain specie. `mtSphere`, `atomicCutoffs`, `energyParameters`, `lo`, `electronConfig`, `nocoParams`, `ldaU` and `special` keys are supported. To find possible keys of the inner dictionary please refer to the FLEUR documentation [flapw.de](http://flapw.de)

**Return type** `Union[_Element, _ElementTree]`

```
masci_tools.util.xml.xml_setters_names.set_species_label(xmltree, schema_dict, atom_label,
                                                         attributedict, create=False)
```

This method calls `set_species()` method for a certain atom species that corresponds to an atom with a given label

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – xml etree of the inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **atom\_label** (`str`) – string, a label of the atom which specie will be changed. 'all' to change all the species
- **attributedict** (`Dict[str, Any]`) – a python dict specifying what you want to change.
- **create** (`bool`) – bool, if species does not exist create it and all subtags?

**Return type** `Union[_Element, _ElementTree]`

**Returns** xml etree of the new inp.xml

```
masci_tools.util.xml.xml_setters_names.set_text(xmltree, schema_dict, tag_name, text,
                                                complex_xpath=None, occurrences=None,
                                                create=False, **kwargs)
```

Sets the text on tags in a xmlltree to a given value, specified by the name of the tag and further specifications. By default the text will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with `create=True`. The text values are converted automatically according to the types with `convert_to_xml()` if they are not `str` already.

#### Parameters

- **xmltree** – an xmlltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input

- **tag\_name** – str name of the tag, where the text should be set
- **text** – value or list of values to set
- **complex\_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** – bool optional (default False), if True the tag is created if is missing

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Returns** xmltree with set text

```
masci_tools.util.xml.xml_setters_names.shift_value(xmltree, schema_dict, change_dict, mode='abs',
                                                    path_spec=None)
```

Shifts numerical values of attributes directly in the inp.xml file.

The first occurrence of the attribute is shifted

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – xml tree that represents inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **change\_dict** (`Dict[str, Any]`) – a python dictionary with the keys to shift and the shift values.
- **mode** (`Literal['abs', 'rel']`) – ‘abs’ if change given is absolute, ‘rel’ if relative
- **path\_spec** (`Optional[Dict[str, Any]]`) – dict, with ggf. necessary further specifications for the path of the attribute

**Return type** `Union[_Element, _ElementTree]`

**Returns** a xml tree with shifted values

An example of change\_dict:

```
change_dict = {'itmax' : 1, 'dVac': -0.123}
```

```
masci_tools.util.xml.xml_setters_names.shift_value_species_label(xmltree, schema_dict,
                                                                    atom_label, attributename,
                                                                    value_given, mode='abs',
                                                                    **kwargs)
```

Shifts the value of an attribute on a species by label if atom\_label contains ‘all’ then applies to all species

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – xml etree of the inp.xml
- **schema\_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **atom\_label** (`str`) – string, a label of the atom which specie will be changed. ‘all’ if set up all species
- **attributename** (`str`) – name of the attribute to change



- **value\_given** (*Any*) – value to add or to multiply by
- **mode** (*Literal*['abs', 'rel']) – 'rel' for multiplication or 'abs' for addition

**Kwargs if the attributename does not correspond to a unique path:**

**param contains** *str*, this string has to be in the final path

**param not\_contains** *str*, this string has to NOT be in the final path

**Return type** *Union*[\_Element, \_ElementTree]

**Returns** xml etree of the new inp.xml

`masci_tools.util.xml.xml_setters_names.switch_kpointset(xmltree, schema_dict, list_name)`  
Switch the used k-point set

**Warning:** This method is only supported for input versions after the Max5 release

#### Parameters

- **xmltree** (*Union*[\_Element, \_ElementTree]) – xml tree that represents inp.xml
- **schema\_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **list\_name** (*str*) – name of the kPoint set to use

**Return type** *Union*[\_Element, \_ElementTree]

**Returns** an xmltree of the inp.xml file with changes.

`masci_tools.util.xml.xml_setters_names.switch_kpointset_max4(xmltree, schema_dict, list_name)`  
Sets a k-point mesh directly into inp.xml specific for inputs of version Max4

**Warning:** This method is only supported for input versions after the Max5 release

#### Parameters

- **xmltree** (*Union*[\_Element, \_ElementTree]) – xml tree that represents inp.xml
- **schema\_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **list\_name** (*str*) – name of the kPoint set to use

**Return type** *Union*[\_Element, \_ElementTree]

**Returns** an xmltree of the inp.xml file with changes.

This module contains useful methods for initializing or modifying a n\_mmp\_mat file for LDA+U

`masci_tools.util.xml.xml_setters_nmmpmat.rotate_nmmpmat(xmltree, nmmp_lines, schema_dict, species_name, orbital, phi, theta)`

Rotate the density matrix with the given angles phi and theta

#### Parameters

- **xmltree** (*Union*[\_Element, \_ElementTree]) – an xmltree that represents inp.xml

- **nmmplines** (`List[str]`) – list of lines in the `n_mmp_mat` file
- **schema\_dict** (`SchemaDict`) – `InputSchemaDict` containing all information about the structure of the input
- **species\_name** (`str`) – string, name of the species you want to change
- **orbital** (`int`) – integer, orbital quantum number of the LDA+U procedure to be modified
- **phi** (`float`) – float, angle (radian), by which to rotate the density matrix
- **theta** (`float`) – float, angle (radian), by which to rotate the density matrix

#### Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

**Return type** `List[str]`

**Returns** list with modified `nmmplines`

```
masci_tools.util.xml.xml_setters_nmmmpmat.set_nmmmpmat(xmltree, nmmplines, schema_dict,
                                                         species_name, orbital, spin,
                                                         state_occupations=None,
                                                         orbital_occupations=None, denmat=None,
                                                         phi=None, theta=None)
```

Routine sets the block in the `n_mmp_mat` file specified by `species_name`, `orbital` and `spin` to the desired density matrix

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an `xmltree` that represents `inp.xml`
- **nmmplines** (`List[str]`) – list of lines in the `n_mmp_mat` file
- **schema\_dict** (`SchemaDict`) – `InputSchemaDict` containing all information about the structure of the input
- **species\_name** (`str`) – string, name of the species you want to change
- **orbital** (`int`) – integer, orbital quantum number of the LDA+U procedure to be modified
- **spin** (`int`) – integer, specifies which spin block should be modified
- **state\_occupations** (`Optional[List[float]]`) – list, sets the diagonal elements of the density matrix and everything else to zero
- **denmat** (`Optional[ndarray]`) – matrix, specify the density matrix explicitly
- **phi** (`Optional[float]`) – float, optional angle (radian), by which to rotate the density matrix before writing it
- **theta** (`Optional[float]`) – float, optional angle (radian), by which to rotate the density matrix before writing it

#### Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

**Return type** `List[str]`

**Returns** list with modified `nmmplines`

```
masci_tools.util.xml.xml_setters_nmmpmat.validate_nmmpmat(xmltree, nmmplines, schema_dict)
```

Checks that the given nmmplines is valid with the given xmltree

Checks that the number of blocks is as expected from the inp.xml and each block does not contain non-zero elements outside their size given by the orbital quantum number in the inp.xml. Additionally the occupations, i.e. diagonal elements are checked that they are in between 0 and the maximum possible occupation

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **nmmplines** (`List[str]`) – list of lines in the n\_mmp\_mat file

**Raises** `ValueError` – if any of the above checks are violated.

**Return type** `None`

Functions for modifying the xml input file of Fleur with explicit xpath arguments These can still use the schema dict for finding information about the xpath

```
masci_tools.util.xml.xml_setters_xpaths.eval_xpath_create(xmltree, schema_dict, xpath, base_xpath,
                                                         create_parents=False,
                                                         occurrences=None, list_return=False)
```

Evaluates and xpath and creates tag if the result is empty

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to place a new tag
- **base\_xpath** – path where to place a new tag without complex syntax ([ ] conditions and so on)
- **create\_parents** – bool optional (default False), if True also the parents of the tag are created if they are missing
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag if the tag is missing. By default all nodes are used.
- **list\_return** – if True, the returned quantity is always a list even if only one element is in it

**Returns** list of nodes from the result of the xpath expression

```
masci_tools.util.xml.xml_setters_xpaths.xml_add_number_to_attrib(xmltree, schema_dict, xpath,
                                                                base_xpath, attributename,
                                                                add_number, mode='abs',
                                                                occurrences=None)
```

Adds a given number to the attribute value in a xmltree. By default the attribute will be shifted on all nodes returned for the specified xpath. If there are no nodes under the specified xpath an error is raised

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to set the attributes
- **base\_xpath** – path where to place a new tag without complex syntax ([ ] conditions and so on)
- **attributename** – the attribute name to change

- **add\_number** – number to add/multiply with the old attribute value
- **mode** – str (either *rel* or *abs*). *rel* multiplies the old value with *add\_number* *abs* adds the old value and *add\_number*
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

#### Raises

- **ValueError** – If the attribute is unknown or cannot be float or int
- **ValueError** – If the evaluation of the old values failed
- **ValueError** – If a float result is written to a integer attribute

**Returns** xmltree with shifted attribute

```
masci_tools.util.xml.xml_setters_xpaths.xml_add_number_to_first_attrib(xmltree, schema_dict,
                                                                    xpath, base_xpath,
                                                                    attributename,
                                                                    add_number,
                                                                    mode='abs')
```

Adds a given number to the first occurrence of a attribute value in a xmltree. If there are no nodes under the specified xpath an error is raised

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to set the attributes
- **base\_xpath** – path where to place a new tag without complex syntax ([ ] conditions and so on)
- **attributename** – the attribute name to change
- **add\_number** – number to add/multiply with the old attribute value
- **mode** – str (either *rel* or *abs*). *rel* multiplies the old value with *add\_number* *abs* adds the old value and *add\_number*

#### Raises

- **ValueError** – If the attribute is unknown or cannot be float or int
- **ValueError** – If the evaluation of the old values failed
- **ValueError** – If a float result is written to a integer attribute

**Returns** xmltree with shifted attribute

```
masci_tools.util.xml.xml_setters_xpaths.xml_create_tag_schema_dict(xmltree, schema_dict, xpath,
                                                                    base_xpath, element,
                                                                    create_parents=False,
                                                                    occurrences=None)
```

This method evaluates an xpath expression and creates a tag in a xmltree under the returned nodes. If there are no nodes evaluated the subtags can be created with *create\_parents=True*

The tag is always inserted in the correct place if a order is enforced by the schema

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input

- **xpath** – a path where to place a new tag
- **base\_xpath** – path where to place a new tag without complex syntax ([] conditions and so on)
- **element** – a tag name or etree Element to be created
- **create\_parents** – bool optional (default False), if True and the given xpath has no results the the parent tags are created recursively
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**Raises** **ValueError** – If the nodes are missing and *create\_parents=False*

**Returns** xmltree with created tags

```
masci_tools.util.xml.xml_setters_xpaths.xml_set_attrib_value(xmltree, schema_dict, xpath,
                                                             base_xpath, attributename, attribv,
                                                             occurrences=None, create=False)
```

Sets an attribute in a xmltree to a given value. By default the attribute will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are converted automatically according to the types of the attribute with *convert\_to\_xml()* if they are not *str* already.

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to set the attributes
- **base\_xpath** – path where to place a new tag without complex syntax ([] conditions and so on)
- **attributename** – the attribute name to set
- **attribv** – value or list of values to set
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** – bool optional (default False), if True the tag is created if is missing

#### Raises

- **ValueError** – If the conversion to string failed
- **ValueError** – If the tag is missing and *create=False*
- **ValueError** – If the attributename is not allowed on the base\_xpath

**Returns** xmltree with set attribute

```
masci_tools.util.xml.xml_setters_xpaths.xml_set_complex_tag(xmltree, schema_dict, xpath,
                                                             base_xpath, attributedict,
                                                             create=False)
```

Recursive Function to correctly set tags/attributes for a given tag. Goes through the attributedict and decides based on the schema\_dict, how the corresponding key has to be handled.

Supports:

- attributes
- tags with text only
- simple tags, i.e. only attributes (can be optional single/multiple)

- complex tags, will recursively create/modify them

#### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to set the attributes
- **base\_xpath** – path where to place a new tag without complex syntax ([]) conditions and so on)
- **tag\_name** – name of the tag to set
- **attributedict** – Keys in the dictionary correspond to names of tags and the values are the modifications to do on this tag (attributename, subdict with changes to the subtag, ...)
- **create** – bool optional (default False), if True and the path, where the complex tag is set does not exist it is created

**Returns** xmldata with changes to the complex tag

```
masci_tools.util.xml.xml_setters_xpaths.xml_set_first_attr_value(xmldata, schema_dict, xpath,
                                                                    base_xpath, attributename,
                                                                    attribv, create=False)
```

Sets the first occurrence attribute in a xmldata to a given value. If there are no nodes under the specified xpath a tag can be created with *create=True*. The attribute values are converted automatically according to the types of the attribute with [convert\\_to\\_xml\(\)](#) if they are not *str* already.

#### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to set the attribute
- **base\_xpath** – path where to place a new tag without complex syntax ([]) conditions and so on)
- **attributename** – the attribute name to set
- **attribv** – value or list of values to set
- **create** – bool optional (default False), if True the tag is created if is missing

#### Raises

- **ValueError** – If the conversion to string failed
- **ValueError** – If the tag is missing and *create=False*
- **ValueError** – If the attributename is not allowed on the base\_xpath

**Returns** xmldata with set attribute

```
masci_tools.util.xml.xml_setters_xpaths.xml_set_first_text(xmldata, schema_dict, xpath,
                                                            base_xpath, text, create=False)
```

Sets the text on the first occurrence of a tag in a xmldata to a given value. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with [convert\\_to\\_xml\(\)](#) if they are not *str* already.

#### Parameters

- **xmldata** – an xmldata that represents inp.xml

- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to set the text
- **base\_xpath** – path where to place a new tag without complex syntax ([] conditions and so on)
- **text** – value or list of values to set
- **create** – bool optional (default False), if True the tag is created if is missing

#### Raises

- **ValueError** – If the conversion to string failed
- **ValueError** – If the tag is missing and *create=False*

**Returns** xmltree with set text

```
masci_tools.util.xml.xml_setters_xpaths.xml_set_simple_tag(xmltree, schema_dict, xpath,
                                                            base_xpath, tag_name, changes,
                                                            create_parents=False)
```

Sets one or multiple *simple* tag(s) in an xmltree. A simple tag can only hold attributes and has no subtags. If the tag can occur multiple times all existing tags are DELETED and new ones are written. If the tag only occurs once it will automatically be created if its missing.

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to set the attributes
- **base\_xpath** – path where to place a new tag without complex syntax ([] conditions and so on)
- **tag\_name** – name of the tag to set
- **changes** – list of dicts or dict with the changes. Elements in list describe multiple tags. Keys in the dictionary correspond to {‘attributename’: attributevalue}
- **create\_parents** – bool optional (default False), if True and the path, where the simple tags are set does not exist it is created

**Returns** xmltree with set simple tags

```
masci_tools.util.xml.xml_setters_xpaths.xml_set_text(xmltree, schema_dict, xpath, base_xpath, text,
                                                       occurrences=None, create=False)
```

Sets the text on tags in a xmltree to a given value. By default the text will be set on all nodes returned for the specified xpath. If there are no nodes under the specified xpath a tag can be created with *create=True*. The text values are converted automatically according to the types with [convert\\_to\\_xml\(\)](#) if they are not *str* already.

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **schema\_dict** – InputSchemaDict containing all information about the structure of the input
- **xpath** – a path where to set the text
- **base\_xpath** – path where to place a new tag without complex syntax ([] conditions and so on)
- **text** – value or list of values to set
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

- **create** – bool optional (default False), if True the tag is created if is missing

#### Raises

- **ValueError** – If the conversion to string failed
- **ValueError** – If the tag is missing and *create=False*

**Returns** xmltree with set text

Basic functions for modifying the xml input file of Fleur. These functions DO NOT have the ability to create missing tags on the fly. This functionality is added on top in [xml\\_setters\\_xpaths](#) since we need the schema dictionary to do these operations robustly

```
maschi_tools.util.xml.xml_setters_basic.xml_create_tag(xmltree, xpath, element, place_index=None,
                                                       tag_order=None, occurrences=None,
                                                       correct_order=True, several=True)
```

This method evaluates an xpath expression and creates a tag in a xmltree under the returned nodes. If there are no nodes under the specified xpath an error is raised.

The tag is appended by default, but can be inserted at a certain index (*place\_index*) or can be inserted according to a given order of tags

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **xpath** – a path where to place a new tag
- **element** – a tag name or etree Element to be created
- **place\_index** – defines the place where to put a created tag
- **tag\_order** – defines a tag order
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.
- **correct\_order** – bool, if True (default) and a tag\_order is given, that does not correspond to the given order in the xmltree (only order wrong no unknown tags) it will be corrected and a warning is given This is necessary for some edge cases of the xml schemas of fleur
- **several** – bool, if True multiple tags of the given name are allowed

**Raises** **ValueError** – If the insertion failed in any way (tag\_order does not match, failed to insert, ...)

**Returns** xmltree with created tags

```
maschi_tools.util.xml.xml_setters_basic.xml_delete_att(xmltree, xpath, attrib, occurrences=None)
```

Deletes an xml attribute in an xmletree.

#### Parameters

- **xmltree** – an xmltree that represents inp.xml
- **xpath** – a path to the attribute to be deleted
- **attrib** – the name of an attribute
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**Returns** xmltree with deleted attribute

```
maschi_tools.util.xml.xml_setters_basic.xml_delete_tag(xmltree, xpath, occurrences=None)
```

Deletes a xml tag in an xmletree.



### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **xpath** – a path to the tag to be deleted
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**Returns** xmldata with deleted tag

`masci_tools.util.xml.xml_setters_basic.xml_replace_tag(xmldata, xpath, newelement, occurrences=None)`

replaces xml tags by another tag on an xmldata in place

### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **xpath** – a path to the tag to be replaced
- **newelement** – a new tag
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

**Returns** xmldata with replaced tag

`masci_tools.util.xml.xml_setters_basic.xml_set_attr_value_no_create(xmldata, xpath, attributename, attribv, occurrences=None)`

Sets an attribute in a xmldata to a given value. By default the attribute will be set on all nodes returned for the specified xpath.

### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **xpath** – a path where to set the attributes
- **attributename** – the attribute name to set
- **attribv** – value or list of values to set (if not str they will be converted with `str(value)`)
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

**Raises** `ValueError` – If the lengths of attribv or occurrences do not match number of nodes

**Returns** xmldata with set attribute

`masci_tools.util.xml.xml_setters_basic.xml_set_text_no_create(xmldata, xpath, text, occurrences=None)`

Sets the text of a tag in a xmldata to a given value. By default the text will be set on all nodes returned for the specified xpath.

### Parameters

- **xmldata** – an xmldata that represents inp.xml
- **xpath** – a path where to set the text
- **text** – value or list of values to set (if not str they will be converted with `str(value)`)
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

**Raises** `ValueError` – If the lengths of text or occurrences do not match number of nodes

**Returns** xmldata with set text

#### 6.1.5.4 XML Getter functions

This module provides functions to extract distinct parts of the fleur xml files for easy versioning and reuse

`masci_tools.util.xml.xml_getters.get_cell(xmltree, schema_dict, logger=None, convert_to_angstroem=True)`

Get the Bravais matrix from the given fleur xml file. In addition a list determining in, which directions there are periodic boundary conditions in the system.

**Warning:** Only the explicit definition of the Bravais matrix is supported. Old inputs containing the *latnam* definitions are not supported

##### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors
- **convert\_to\_angstroem** (`bool`) – bool if True the bravais matrix is converted to angstroem

**Return type** `Tuple[ndarray, Tuple[bool, bool, bool]]`

**Returns** numpy array of the bravais matrix and list of boolean values for periodic boundary conditions

`masci_tools.util.xml.xml_getters.get_fleur_modes(xmltree, schema_dict, logger=None)`

Determine the calculation modes of fleur for the given xml file. Calculation modes are things that change the produced files or output in the out.xml files

##### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

**Return type** `Dict[str, Any]`

**Returns** dictionary with all the extracted calculation modes

The following modes are inspected:

- *jspin*: How many spins are considered in the calculation
- *noco*: Is the calculation non-collinear?
- *soc*: Is spin-orbit coupling included?
- *relax*: Is the calculation a structure relaxation?
- *gw*: Special mode for GW/Spex calculations
- *force\_theorem*: Is a Force theorem calculation performed?
- *film*: Is the structure a film system
- *ldau*: Is LDA+U included?
- *dos*: Is it a density of states calculation?

- *band*: Is it a bandstructure calculation?
- *bz\_integration*: How is the integration over the Brillouin-Zone performed?

```
masci_tools.util.xml.xml_getters.get_kpoints_data(xmltree, schema_dict, name=None, index=None,
                                                  only_used=False, logger=None,
                                                  convert_to_angstroem=True)
```

Get the kpoint sets defined in the given fleur xml file.

**Warning:** For file versions before Max5 the name argument is not valid

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **name** (`Optional[str]`) – str, optional, if given only the kpoint set with the given name is returned
- **index** (`Optional[int]`) – int, optional, if given only the kpoint set with the given index is returned
- **only\_used** (`bool`) – bool if True only the kpoint list used in the calculation is returned
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors
- **convert\_to\_angstroem** (`bool`) – bool if True the bravais matrix is converted to angstroem

**Return type** `Tuple[Union[List[List[float]], Dict[str, List[List[float]]], Union[List[float], Dict[str, List[float]]], ndarray, Tuple[bool, bool, bool]]`

**Returns** tuple containing the kpoint information

The tuple contains the following entries:

1. **kpoints** dict or list (list if there is only one kpoint set), containing the coordinates of the kpoints
2. **weights** dict or list (list if there is only one kpoint set), containing the weights of the kpoints
3. **cell** numpy array, bravais matrix of the given system
4. **pbc** list of booleans, determines in which directions periodic boundary conditions are applicable

```
masci_tools.util.xml.xml_getters.get_kpoints_data_max4(xmltree, schema_dict, logger=None,
                                                         convert_to_angstroem=True,
                                                         only_used=False)
```

Get the kpoint sets defined in the given fleur xml file.

**Note:** This function is specific to file version before and including the Max4 release of fleur

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

- **convert\_to\_angstroem** (*bool*) – bool if True the bravais matrix is converted to angstroem
- **only\_used** (*bool*) – (Has no effect for Max4) bool if True only the kpoint list used in the calculation is returned

**Return type** `Tuple[List[List[float]], List[float], ndarray, Tuple[bool, bool, bool]]`

**Returns** tuple containing the kpoint information

The tuple contains the following entries:

1. **kpoints** list containing the coordinates of the kpoints
2. **weights** list containing the weights of the kpoints
3. **cell** numpy array, bravais matrix of the given system
4. **pb** list of booleans, determines in which directions periodic boundary conditions are applicable

`masci_tools.util.xml.xml_getters.get_nkpts(xmltree, schema_dict, logger=None)`

Get the number of kpoints that will be used in the calculation specified in the given fleur XML file.

**Warning:** For file versions before Max5 only kPointList or kPointCount tags will work. However, for kPointCount there is no real guarantee that for every occasion it will correspond to the number of kpoints. So a warning is written out

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

**Return type** `int`

**Returns** int with the number of kpoints

`masci_tools.util.xml.xml_getters.get_nkpts_max4(xmltree, schema_dict, logger=None)`

Get the number of kpoints that will be used in the calculation specified in the given fleur XML file. Version specific for Max4 versions or older

**Warning:** For file versions before Max5 only kPointList or kPointCount tags will work. However, for kPointCount there is no real guarantee that for every occasion it will correspond to the number of kpoints. So a warning is written out

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

**Return type** `int`

**Returns** int with the number of kpoints

```
masci_tools.util.xml.xml_getters.get_parameter_data(xmltree, schema_dict, inpgen_ready=True,
                                                    write_ids=True, extract_econfig=False,
                                                    logger=None)
```

This routine returns a python dictionary produced from the inp.xml file, which contains all the parameters needed to setup a new inp.xml from an inpgen input file to produce the same input (for parameters that the inpgen can control)

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **inpgen\_ready** (`bool`) – Bool, return a dict which can be inputted into inpgen while setting atoms
- **write\_ids** (`bool`) – Bool, if True the atom ids are added to the atom namelists
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

**Return type** `Dict[str, Any]`

**Returns** dict, which will lead to the same inp.xml (in case if other defaults, which can not be controlled by input for inpgen, were changed)

```
masci_tools.util.xml.xml_getters.get_relaxation_information(xmltree, schema_dict, logger=None)
```

Get the relaxation information from the given fleur XML file. This includes the current displacements, energy and posforce evolution

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

**Return type** `Dict[str, Any]`

**Returns** dict with the relaxation information

**Raises** **ValueError** – If no relaxation section is included in the xml tree

```
masci_tools.util.xml.xml_getters.get_relaxation_information_pre029(xmltree, schema_dict,
                                                                    logger=None)
```

Get the relaxation information from the given fleur XML file. This includes the current displacements, energy and posforce evolution

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

**Return type** `None`

**Returns** dict with the relaxation information

**Raises** **ValueError** – If no relaxation section is included in the xml tree

```
masci_tools.util.xml.xml_getters.get_structure_data(xmltree, schema_dict, include_relaxations=True,
                                                    site_namedtuple=False,
                                                    convert_to_angstroem=True,
                                                    normalize_kind_name=True, logger=None)
```

Get the structure defined in the given fleur xml file.

**Warning:** Only the explicit definition of the Bravais matrix is supported. Old inputs containing the *latnam* definitions are not supported

**Warning:** In versions 0.5.0 or later the output of the atom sites was restructured to be more interoperable with other IO functions (e.g. `write_inpgen_file()`) The new format returns a list of `AtomSiteProperties` instead of the list of tuples (position, symbol)

For better compatibility this output is not default in 0.5.0 but instead is enabled by `site_namedtuple=True` and a DeprecationWarning is given when this argument is False.

---

**Note:** In versions 0.5.0 or later the returned atom positions correspond to the relaxed structure if a `relaxation` section is present in the xmltree

---

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **include\_relaxations** (`bool`) – bool if True and a relaxation section is included the resulting positions correspond to the relaxed structure
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors
- **convert\_to\_angstroem** (`bool`) – bool if True the bravais matrix is converted to angstroem

**Return type** `Tuple[List[AtomSiteProperties], ndarray, Tuple[bool, bool, bool]]`

**Returns** tuple containing the structure information

The tuple contains the following entries:

1. **atom\_data** list of (named)tuples containing the absolute positions and symbols of the atoms
2. **cell** numpy array, bravais matrix of the given system
3. **pbc** list of booleans, determines in which directions periodic boundary conditions are applicable

```
masci_tools.util.xml.xml_getters.get_symmetry_information(xmltree, schema_dict, logger=None)
```

Get the symmetry information from the given fleur XML file. This includes the rotation matrices and shifts defined in the `symmetryOperations` tag.

---

**Note:** Only the explicit definition of the used symmetry operations in the xml file is supported.

---

#### Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file

- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – schema dictionary corresponding to the file version of the xmltree
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

**Return type** `Tuple[List[ndarray], List[ndarray]]`

**Returns** tuple of the rotations and their respective shifts

**Raises** **ValueError** – If no symmetryOperations section is included in the xml tree

### 6.1.5.5 Basic IO helper functions

Here commonly used functions that do not need aiida-stuff (i.e. can be tested without a database) are collected.

**class** `masci_tools.io.common_functions.AtomSiteProperties`(*position, symbol, kind*)

**kind**

Alias for field number 2

**position**

Alias for field number 0

**symbol**

Alias for field number 1

`masci_tools.io.common_functions.abs_to_rel`(*vector, cell*)

Converts a position vector in absolute coordinates to relative coordinates.

**Parameters**

- **vector** – list or np.array of length 3, vector to be converted
- **cell** – Bravais matrix of a crystal 3x3 Array, List of list or np.array

**Returns** list of length 3 of scaled vector, or False if vector was not length 3

`masci_tools.io.common_functions.abs_to_rel_f`(*vector, cell, pbc*)

Converts a position vector in absolute coordinates to relative coordinates for a film system.

**Parameters**

- **vector** – list or np.array of length 3, vector to be converted
- **cell** – Bravais matrix of a crystal 3x3 Array, List of list or np.array
- **pbc** – Boundary conditions, List or Tuple of 3 Boolean

**Returns** list of length 3 of scaled vector, or False if vector was not length 3

`masci_tools.io.common_functions.angles_to_vec`(*magnitude, theta, phi*)

convert (magnitude, theta, phi) to (x,y,z)

theta/phi need to be in radians!

Input can be single number, list of numpy.ndarray data Returns x,y,z vector

`masci_tools.io.common_functions.camel_to_snake`(*name*)

Converts camelCase to snake\_case variable names Used in the Fleur parser to convert attribute names from the xml files

`masci_tools.io.common_functions.convert_to_fortran`(*val, quote\_strings=True*)

**Parameters** **val** – the value to be read and converted to a Fortran-friendly string.

`masci_tools.io.common_functions.convert_to_fortran_string(string)`

converts some parameter strings to the format for the inpgen :param string: some string :returns: string in right format (extra “” if not already present)

`masci_tools.io.common_functions.convert_to_pystd(value)`

Recursively convert numpy datatypes to standard python, needed by aiida-core.

**Usage:** `converted = convert_to_pystd(to_convert)`

where `to_convert` can be a dict, array, list, or single valued variable

`masci_tools.io.common_functions.fac(n)`

Returns the factorial of n

`masci_tools.io.common_functions.filter_out_empty_dict_entries(dict_to_filter)`

Filter out entries in a given dict that correspond to empty values. At the moment this is empty lists, dicts and None

**Parameters** `dict_to_filter` – dict to filter

**Returns** dict without empty entries

`masci_tools.io.common_functions.find_symmetry_relation(from_pos, to_pos, rotations, shifts, cell, relative_pos=False, film=False)`

Find symmetry relation between the given vectors. This functions assumes that a symmetry relation exists otherwise an error is raised

**Parameters**

- **from\_pos** – vector to rotate
- **to\_pos** – vector to rotate to
- **rotations** – list of np.arrays with the given symmetry rotations
- **shifts** – list of np.arrays with the given shifts for the symmetry operations
- **cell** – Bravais matrix of a crystal 3x3 Array, List of list or np.array
- **relative\_pos** – bool if True the given vectors are assumed to be in internal coordinates
- **film** – bool if True the vectors are assumed to be film coordinates

**Returns** tuple of rotation and shift mapping `from_pos` to `to_pos`

**Raises** `ValueError` – If no symmetry relation is found

`masci_tools.io.common_functions.get_corestates_from_potential(potfile='potential')`

Read core states from potential file

`masci_tools.io.common_functions.get_ef_from_potfile(potfile)`

extract fermi energy from potfile

`masci_tools.io.common_functions.get_highest_core_state(nstates, energies, lmoments)`

Find highest lying core state from list of core states, needed to find and check energy contour

`masci_tools.io.common_functions.get_wigner_matrix(l, phi, theta)`

Produces the wigner rotation matrix for the density matrix

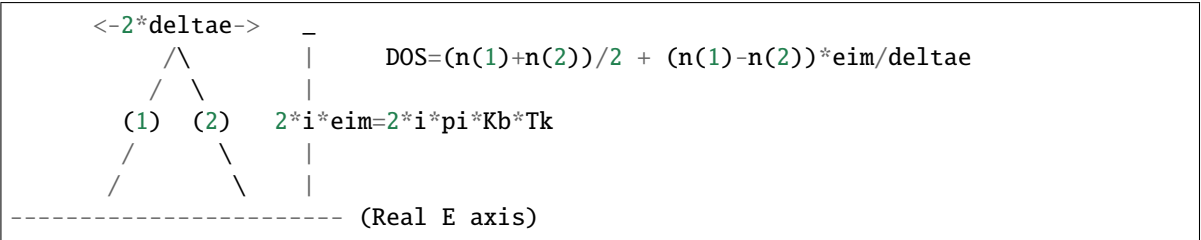
**Parameters**

- **l** – int, orbital quantum number
- **phi** – float, angle (radian) corresponds to euler angle alpha
- **theta** – float, angle (radian) corresponds to euler angle beta



`masci_tools.io.common_functions.interpolate_dos(dosfile, return_original=False)`  
interpolation function copied from complexdos3 fortran code

Principle of DOS here: Two-point contour integration for DOS in the middle of the two points. The input DOS and energy must be complex. Parameter `deltae` should be of the order of magnitude of `eim`:



**Parameters** `input` – either absolute path of ‘complex.dos’ file or file handle to it

**Returns** `E_Fermi`, numpy array of interpolated dos

**Note** output units are in Ry!

`masci_tools.io.common_functions.is_sequence(arg)`  
Checks if `arg` is a sequence

**Return type** `bool`

`masci_tools.io.common_functions.open_general(filename_or_handle, iomode=None)`  
Open a file directly from a path or use a file handle if that is given. Also take care of closed files by reopening them. This is intended to be used like this:

```
f = open_general(outfile)
with f: # make sure the file is properly closed
    txt = f.readlines()
```

`masci_tools.io.common_functions.rel_to_abs(vector, cell)`  
Converts a position vector in internal coordinates to absolute coordinates in Angstrom.

**Parameters**

- **vector** – list or np.array of length 3, vector to be converted
- **cell** – Bravais matrix of a crystal 3x3 Array, List of list or np.array

**Returns** list of length 3 of scaled vector, or False if vector was not length 3

`masci_tools.io.common_functions.rel_to_abs_f(vector, cell)`  
Converts a position vector in internal coordinates to absolute coordinates in Angstrom for a film structure (2D).

`masci_tools.io.common_functions.skipHeader(seq, n)`  
Iterate over a sequence skipping the first `n` elements

**Args:** `seq` (iterable): Iterable sequence `n` (int): Number of Elements to skip in the beginning of the sequence

**Yields:** item: Elements in `seq` after the first `n` elements

`masci_tools.io.common_functions.vec_to_angles(vec)`  
converts vector (x,y,z) to (magnitude, theta, phi)

Small utility functions for inspecting hdf files and converting the complete file structure into a python dictionary

`masci_tools.io.hdf5_util.h5dump(file, group='/')`  
Shows the overall filestructure of an hdf file Goes through all groups and subgroups and prints the attributes or the shape and datatype of the datasets

**Parameters** `filepath` – path to the hdf file

`masci_tools.io.hdf5_util.hdfList(name, obj)`

Print the name of the current object (indented to create a nice tree structure)

Also prints attribute values and dataset shapes and datatypes

`masci_tools.io.hdf5_util.read_groups(hdfdata, flatten=False)`

Recursive function to read a hdf datastructure and extract the datasets and attributes

**Parameters**

- **hdfdata** – current hdf group to process
- **flatten** – bool, if True the dictionary will be flattened (does not check for lost information)

**Returns** two dictionaries, one with the datasets the other with the attributes in the file

`masci_tools.io.hdf5_util.read_hdf_simple(file, flatten=False)`

Reads in an hdf file and returns its context in a nested dictionary

**Parameters**

- **filepath** – path or filehandle to the hdf file
- **flatten** – bool, if True the dictionary will be flattened (does not check for lost information)

**Returns** two dictionaries, one with the datasets the other with the attributes in the file

**Non unique group attribute or dataset names will be overwritten in the return dict**

#### 6.1.5.6 Logging Utility

This module defines useful utility for logging related functionality

**class** `masci_tools.util.logging_util.DictHandler(log_dict, ignore_unknown_levels=False, **kwargs)`

Custom Handler for the logging module inserting logging messages into a given dictionary.

Messages are grouped into list under the names of the error categories. Keyword arguments can be used to modify the keys for the different levels

**emit(record)**

Emit a record.

**class** `masci_tools.util.logging_util.OutParserLogAdapter(logger, extra)`

This adapter expects the passed in dict-like object to have a ‘iteration’ key, whose value is prepended as [Iteration i] to the message

**process(msg, kwargs)**

Process the logging message and keyword arguments passed in to a logging call to insert contextual information. You can either manipulate the message itself, the keyword args or both. Return the message and kwargs modified (or not) to suit your needs.

Normally, you’ll only need to override this one method in a LoggerAdapter subclass for your specific needs.

### 6.1.5.7 Fleur parser utility

This module contains helper functions for extracting information easily from the `schema_dicts` defined for the Fleur input/output

Also provides convenient functions to use just a attribute name for extracting the attribute from the right place in the given etree

`masci_tools.util.schema_dict_util.attrib_exists(node, schema_dict, name, logger=None, **kwargs)`

Evaluates whether the attribute exists in the xmltree based on the given name and additional further specifications with the available type information

#### Parameters

- **node** (`Union[_Element, _ElementTree]`) – etree Element, on which to execute the xpath evaluations
- **schema\_dict** (`SchemaDict`) – dict, containing all the path information and more
- **name** (`str`) – str, name of the tag
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised

#### Kwargs:

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other

#### Return type `bool`

**Returns** bool, True if any tag with the attribute exists

`masci_tools.util.schema_dict_util.eval_simple_xpath(node, schema_dict, name, logger=None, **kwargs)`

Evaluates a simple xpath expression of the tag in the xmltree based on the given name and additional further specifications with the available type information

#### Parameters

- **node** – etree Element, on which to execute the xpath evaluations
- **schema\_dict** – dict, containing all the path information and more
- **name** – str, name of the tag
- **logger** – logger object for logging warnings, errors, if not provided all errors will be raised

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param list\_return** bool, if True a list is always returned

**Returns** etree Elements obtained via the simple xpath expression

`masci_tools.util.schema_dict_util.evaluate_attribute(node, schema_dict, name, constants=None, logger=None, **kwargs)`

Evaluates the value of the attribute based on the given name and additional further specifications with the available type information

#### Parameters

- **node** (`Union[_Element, _ElementTree]`) – etree Element, on which to execute the xpath evaluations
- **schema\_dict** (`SchemaDict`) – dict, containing all the path information and more
- **name** (`str`) – str, name of the attribute
- **constants** (`Optional[Dict[str, float]]`) – dict, contains the defined constants
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised

#### Kwargs:

**param tag\_name** str, name of the tag where the attribute should be parsed

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param exclude** list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other

**param list\_return** if True, the returned quantity is always a list even if only one element is in it

**param optional** bool, if True and no logger given none or an empty list is returned

**Return type** Any

**Returns** list or single value, converted in `convert_xml_attribute`

`masci_tools.util.schema_dict_util.evaluate_parent_tag(node, schema_dict, name, constants=None, logger=None, **kwargs)`

Evaluates all attributes of the parent tag based on the given name and additional further specifications with the available type information

#### Parameters

- **node** (`Union[_Element, _ElementTree]`) – etree Element, on which to execute the xpath evaluations
- **schema\_dict** (`SchemaDict`) – dict, containing all the path information and more
- **name** (`str`) – str, name of the tag
- **constants** (`Optional[Dict[str, float]]`) – dict, contains the defined constants
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param only\_required** bool (optional, default False), if True only required attributes are parsed

**param ignore** list of str (optional), attributes not to parse

**param list\_return** if True, the returned quantity is always a list even if only one element is in it

**param strict\_missing\_error** if True, and no logger is given an error is raised if any attribute is not found

**Return type** Any

**Returns** dict, with attribute values converted via `convert_xml_attribute`

```
masci_tools.util.schema_dict_util.evaluate_single_value_tag(node, schema_dict, name,
                                                            constants=None, logger=None,
                                                            **kwargs)
```

Evaluates the value and unit attribute of the tag based on the given name and additional further specifications with the available type information

#### Parameters

- **node** (`Union[_Element, _ElementTree]`) – etree Element, on which to execute the xpath evaluations
- **schema\_dict** (`SchemaDict`) – dict, containing all the path information and more
- **name** (`str`) – str, name of the tag
- **constants** (`Optional[Dict[str, float]]`) – dict, contains the defined constants
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param only\_required** bool (optional, default False), if True only required attributes are parsed

**param ignore** list of str (optional), attributes not to parse

**param list\_return** if True, the returned quantity is always a list even if only one element is in it

**param strict\_missing\_error** if True, and no logger is given an error is raised if any attribute is not found

**Return type** Any

**Returns** value and unit, both converted in `convert_xml_attribute`

```
masci_tools.util.schema_dict_util.evaluate_tag(node, schema_dict, name, constants=None,
                                                logger=None, subtags=False, text=True, **kwargs)
```

Evaluates all attributes of the tag based on the given name and additional further specifications with the available type information

#### Parameters

- **node** (`Union[_Element, _ElementTree]`) – etree Element, on which to execute the xpath evaluations
- **schema\_dict** (`SchemaDict`) – dict, containing all the path information and more
- **name** (`str`) – str, name of the tag

- **constants** (`Optional[Dict[str, float]]`) – dict, contains the defined constants
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised
- **subtags** (`bool`) – optional bool, if True the subtags of the given tag are evaluated
- **text** (`bool`) – optional bool, if True the text of the tag is also parsed

**Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param only\_required** bool (optional, default False), if True only required attributes are parsed

**param ignore** list of str (optional), attributes not to parse

**param list\_return** if True, the returned quantity is always a list even if only one element is in it

**param strict\_missing\_error** if True, and no logger is given an error is raised if any attribute is not found

**Return type** Any

**Returns** dict, with attribute values converted via `convert_xml_attribute`

`masci_tools.util.schema_dict_util.evaluate_text(node, schema_dict, name, constants=None, logger=None, **kwargs)`

Evaluates the text of the tag based on the given name and additional further specifications with the available type information

**Parameters**

- **node** (`Union[_Element, _ElementTree]`) – etree Element, on which to execute the xpath evaluations
- **schema\_dict** (`SchemaDict`) – dict, containing all the path information and more
- **name** (`str`) – str, name of the tag
- **constants** (`Optional[Dict[str, float]]`) – dict, contains the defined constants
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised

**Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**param list\_return** if True, the returned quantity is always a list even if only one element is in it

**param optional** bool, if True and no logger given none or an empty list is returned

**Return type** Any

**Returns** list or single value, converted in `convert_xml_text`

```
masci_tools.util.schema_dict_util.get_attrib_xpath(schema_dict, name, contains=None,
                                                    not_contains=None, exclude=None,
                                                    tag_name=None)
```

DEPRECATED

Tries to find a unique path from the schema\_dict based on the given name of the attribute and additional further specifications

#### Parameters

- **schema\_dict** – dict, containing all the path information and more
- **name** – str, name of the attribute
- **root\_tag** – str, name of the tag from which the path should be relative
- **contains** – str or list of str, this string has to be in the final path
- **not\_contains** – str or list of str, this string has to NOT be in the final path
- **exclude** – list of str, here specific types of attributes can be excluded valid values are: set-table, settable\_contains, other
- **tag\_name** – str, if given this name will be used to find a path to a tag with the same name in [get\\_tag\\_xpath\(\)](#)

**Returns** str, xpath to the tag with the given attribute

**Raises** [ValueError](#) – If no unique path could be found

```
masci_tools.util.schema_dict_util.get_number_of_nodes(node, schema_dict, name, logger=None,
                                                       **kwargs)
```

Evaluates the number of occurrences of the tag in the xmltree based on the given name and additional further specifications with the available type information

#### Parameters

- **node** ([Union\[\\_Element, \\_ElementTree\]](#)) – etree Element, on which to execute the xpath evaluations
- **schema\_dict** ([SchemaDict](#)) – dict, containing all the path information and more
- **name** ([str](#)) – str, name of the tag
- **logger** ([Optional\[Logger\]](#)) – logger object for logging warnings, errors, if not provided all errors will be raised

#### Kwargs:

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Return type** [int](#)

**Returns** bool, True if any nodes with the path exist

```
masci_tools.util.schema_dict_util.get_relative_attrib_xpath(schema_dict, name, root_tag,
                                                            contains=None, not_contains=None,
                                                            exclude=None, tag_name=None)
```

DEPRECATED

Tries to find a unique relative path from the schema\_dict based on the given name of the attribute name of the root, from which the path should be relative and additional further specifications

#### Parameters

- **schema\_dict** – dict, containing all the path information and more
- **name** – str, name of the attribute
- **contains** – str or list of str, this string has to be in the final path
- **not\_contains** – str or list of str, this string has to NOT be in the final path
- **exclude** – list of str, here specific types of attributes can be excluded valid values are: settable, settable\_contains, other
- **tag\_name** – str, if given this name will be used to find a path to a tag with the same name in `get_relative_tag_xpath()`

**Returns** str, xpath for the given tag

**Raises** **ValueError** – If no unique path could be found

```
masci_tools.util.schema_dict_util.get_relative_tag_xpath(schema_dict, name, root_tag,
                                                         contains=None, not_contains=None)
```

DEPRECATED

Tries to find a unique relative path from the schema\_dict based on the given name of the tag name of the root, from which the path should be relative and additional further specifications

#### Parameters

- **schema\_dict** – dict, containing all the path information and more
- **name** – str, name of the tag
- **root\_tag** – str, name of the tag from which the path should be relative
- **contains** – str or list of str, this string has to be in the final path
- **not\_contains** – str or list of str, this string has to NOT be in the final path

**Returns** str, xpath for the given tag

**Raises** **ValueError** – If no unique path could be found

```
masci_tools.util.schema_dict_util.get_tag_info(schema_dict, name, contains=None,
                                                not_contains=None, path_return=True,
                                                convert_to_builtin=False, multiple_paths=False,
                                                parent=False)
```

DEPRECATED

Tries to find a unique path from the schema\_dict based on the given name of the tag and additional further specifications and returns the tag\_info entry for this tag

#### Parameters

- **schema\_dict** – dict, containing all the path information and more
- **name** – str, name of the tag
- **contains** – str or list of str, this string has to be in the final path
- **not\_contains** – str or list of str, this string has to NOT be in the final path
- **path\_return** – bool, if True the found path will be returned alongside the tag\_info
- **convert\_to\_builtin** – bool, if True the CaseInsensitiveFrozenSets are converted to normal sets with the right case of the attributes



- **multiple\_paths** – bool, if True multiple paths are allowed to match as long as they have the same tag\_info
- **parent** – bool, if True the tag\_info for the parent of the tag is returned

**Returns** dict, tag\_info for the found xpath

**Returns** str, xpath to the tag if *path\_return=True*

`masci_tools.util.schema_dict_util.get_tag_xpath(schema_dict, name, contains=None, not_contains=None)`

DEPRECATED

Tries to find a unique path from the schema\_dict based on the given name of the tag and additional further specifications

**Parameters**

- **schema\_dict** – dict, containing all the path information and more
- **name** – str, name of the tag
- **contains** – str or list of str, this string has to be in the final path
- **not\_contains** – str or list of str, this string has to NOT be in the final path

**Returns** str, xpath for the given tag

**Raises** **ValueError** – If no unique path could be found

`masci_tools.util.schema_dict_util.read_constants(root, schema_dict, logger=None)`

Reads in the constants defined in the inp.xml and returns them combined with the predefined constants from fleur as a dictionary

**Parameters**

- **root** (`Union[_Element, _ElementTree]`) – root of the etree of the inp.xml file
- **schema\_dict** (`SchemaDict`) – schema\_dictionary of the version of the file to read (inp.xml or out.xml)
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors

**Return type** `Dict[str, float]`

**Returns** a python dictionary with all defined constants

`masci_tools.util.schema_dict_util.tag_exists(node, schema_dict, name, logger=None, **kwargs)`

Evaluates whether the tag exists in the xmltree based on the given name and additional further specifications with the available type information

**Parameters**

- **node** (`Union[_Element, _ElementTree]`) – etree Element, on which to execute the xpath evaluations
- **schema\_dict** (`SchemaDict`) – dict, containing all the path information and more
- **name** (`str`) – str, name of the tag
- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised

**Kwargs:**

**param contains** str, this string has to be in the final path

**param not\_contains** str, this string has to NOT be in the final path

**Return type** bool

**Returns** bool, True if any nodes with the path exist

This module contains the functions necessary to parse mathematical expressions with predefined constants given in the inp.xml file of Fleur

**exception** masci\_tools.util.fleur\_calculate\_expression.MissingConstant

**masci\_tools.util.fleur\_calculate\_expression.calculate\_expression**(*expression*, *constants=None*)  
 Recursively evaluates the given expression string with the given defined constants

**Parameters**

- **expression** (Union[str, float, int]) – str containing the expression to be parsed
- **constants** (Optional[Dict[str, float]]) – dict with all defined constants (predefined in the Fleur code or defined in the inp.xml)

**Return type** float

**Returns** float value of the given expression string

**masci\_tools.util.fleur\_calculate\_expression.calculate\_expression\_partial**(*expression*,  
*constants=None*, *prevCommand=None*)

Recursively evaluates the given expression string with the given defined constants and returns the unevaluated part of the expression

**Parameters**

- **expression** (Union[str, float, int]) – str containing the expression to be parsed
- **constants** (Optional[Dict[str, float]]) – dict with all defined constants (predefined in the Fleur code or defined in the inp.xml)
- **prevCommand** (Optional[str]) – str, which gives the command before the beginning of the current block if it is given the calculation is stopped, when a command is encountered, which should be executed after prevCommand (order of operations)

**Return type** Tuple[float, str]

**Returns** float value of the given expression string

**masci\_tools.util.fleur\_calculate\_expression.evaluate\_bracket**(*expression*, *constants*)  
 Evaluates the bracket opened at the start of the expression

**Parameters**

- **expression** (str) – expression to be parsed
- **constants** (Dict[str, float]) – dict with defined constants

**Return type** Tuple[Union[float, int], str]

**Returns** value of the expression inside the brackets and remaining string of the expression after the corresponding closed bracket

**masci\_tools.util.fleur\_calculate\_expression.get\_first\_number**(*expression*)

Reads the number in the beginning of the expression string. This number can begin with a sign +-, a number or the decimal point

**Parameters** **expression** (str) – str of the expression

**Return type** `Tuple[float, str]`

**Returns** float value of the number in the beginning and the string of the remaining expression

`masci_tools.util.fleur_calculate_expression.get_first_string(expression)`

Reads the letter string in the beginning of the expression string.

**Parameters** `expression (str)` – str of the expression

**Return type** `Tuple[str, str]`

**Returns** letter string in the beginning and the string of the remaining expression

This module contains a class which organizes the known parsing tasks for outxml files and provides functionality for adding custom tasks easily

**class** `masci_tools.util.parse_tasks.ParseTasks(version, task_file=None, validate_defaults=False)`

Representation of all known parsing tasks for the out.xml file

When set up it will initialize the known default tasks and check if they work for the given output version

Accessing definition of task example

```
from masci_tools.io.parsers.fleur import ParseTasks

p = ParseTasks('0.33')
totE_definition = p.tasks['total_energy']
```

**add\_task(task\_name, task\_definition, append=False, overwrite=False)**

Add a new task definition to the tasks dictionary

Will first check if the definition has all the required keys

**Parameters**

- **task\_name (str)** – str, key in the tasks dict
- **task\_definition (Dict[str, Any])** – dict with the defined tasks
- **overwrite (bool)** – bool (optional), if True and the key is present in the dictionary it will be overwritten with the new definition
- **append (bool)** – bool (optional), if True and the key is present in the dictionary the new definitions will be inserted into this dictionary (inner keys WILL BE OVERWRITTEN). Additionally if an inner key is overwritten with an empty dict the inner key will be removed

The following keys are expected in each entry of the `task_definition` dictionary:

**param parse\_type** str, defines which methods to use when extracting the information

**param path\_spec** dict with all the arguments that should be passed to `tag_xpath` or `attrib_xpath` to get the correct path

**param subdict** str, if present the parsed values are put into this key in the output dictionary

**param overwrite\_last** bool, if True no list is inserted and each entry overwrites the last

For the `allAttribs` `parse_type` there are more keys that can appear:

**param base\_value** str, optional. If given the attribute with this name will be inserted into the key from the `task_definition` all other keys are formatted as `{task_key}_{attribute_name}`

**param ignore** list of str, these attributes will be ignored

**param overwrite** list of str, these attributes will not create a list and overwrite any value that might be there

**param flat** bool, if False the dict parsed from the tag is inserted as a dict into the correspondin key if True the values will be extracted and put into the output dictionary with the format {task\_key}\_{attribute\_name}

**Return type** `None`

**property all\_attribs\_function:** `Set[str]`

Return the registered parse functions for parsing multipl attributes

**Return type** `Set[str]`

**property conversion\_functions:** `Dict[str, Callable]`

Return the registered conversion functions

**Return type** `Dict[str, Callable]`

**determine\_tasks**(*fleurmodes*, *optional\_tasks=None*, *minimal=False*)

Determine, which tasks to perform based on the fleur\_modes

**Parameters**

- **fleurmodes** (`Dict[str, Any]`) – dict with the calculation modes
- **minimal** (`bool`) – bool, whether to only perform minimal tasks

**Return type** `None`

**property general\_tasks:** `List[str]`

Tasks to perform for the root node

**Return type** `List[str]`

**property iteration\_tasks:** `List[str]`

Tasks to perform for each iteration

**Return type** `List[str]`

**property migrations:** `Dict[str, Dict[str, Union[Literal['compatible'], Callable]]]`

Return the registered migrations

**Return type** `Dict[str, Dict[str, Union[Literal['compatible'], Callable]]]`

**property optional\_tasks:** `Set[str]`

Return a set of the available optional defined tasks

**Return type** `Set[str]`

**property parse\_functions:** `Dict[str, Callable]`

Return the registered parse functions

**Return type** `Dict[str, Callable]`

**perform\_task**(*task\_name*, *node*, *out\_dict*, *schema\_dict*, *constants*, *logger=None*, *use\_lists=True*)

Evaluates the task given in the tasks\_definition dict

**Parameters**

- **task\_name** (`str`) – str, specifies the task to perform
- **node** (`Union[_Element, _ElementTree]`) – etree.Element, the xpath expressions are evaluated from this node
- **out\_dict** (`Dict`) – dict, output will be put in this dictionary
- **schema\_dict** (`Union[InputSchemaDict, OutputSchemaDict]`) – dict, here all paths and attributes are stored according to the outputschema

- **constants** (`Dict[str, float]`) – dict with all the defined mathematical constants
- **logger** (`Union[Logger, LoggerAdapter, None]`) – logger object for logging warnings, errors
- **root\_tag** – str, this string will be appended in front of any xpath before it is evaluated
- **use\_lists** (`bool`) – bool, if True lists are created for each key if not otherwise specified

**Return type** `Dict`

**show\_available\_tasks**(*show\_definitions=False*)

Print all currently available task keys. If *show\_definitions* is True also the corresponding definitions will be printed

**Return type** `None`

`masci_tools.util.parse_tasks.find_migration(start, target, migrations)`

Tries to find a migration path from the start to the target version via the defined migration functions

**Parameters**

- **start** (`str`) – str of the starting version
- **target** (`str`) – str of the target version
- **migrations** (`Dict[str, Dict[str, Union[Literall[‘compatible’], Callable]]]`) – dict of funcs registered via the `register_migration_function` decorator

**Return type** `Optional[List[Callable]]`

**Returns** list of migration functions to be called to go from start to target

This module defines decorators for the `ParseTasks` class to make extending/modifying the parser more convenient

**Up till now 3 decorators are defined:**

- ``register_migration`` marks a function of making backwards incompatible changes to the parsing tasks
- ``register_parsing_function`` gives a mapping between available parsing functions and the keywords in the parsing tasks
- ``conversion_function`` makes the decorated function available to be called easily after a certain parsing task has occurred

`masci_tools.util.parse_tasks_decorators.conversion_function(func)`

Marks a function as a conversion function, which can be called after performing a parsing task. The function can be specified via the `_conversions` control key in the task definitions.

**A conversion function has to have the following arguments:**

**param** `out_dict` dict with the previously parsed information

**param** `parser_info_out` dict, with warnings, info, errors, ...

and return only the modified output dict

**Return type** `Callable`

`masci_tools.util.parse_tasks_decorators.register_migration(base_version, target_version)`

Decorator to add migration for task definition dictionary to the `ParseTasks` class The function should only take the dict of task definitions as an argument

**Parameters**

- **base\_version** (`str`) – str of the version, from which the migration starts

- **target\_version** (`Union[str, List[str]]`) – str or list of str with the versions that work after the migration has been performed

**Return type** `Callable`

`masci_tools.util.parse_tasks_decorators.register_parsing_function(parse_type_name, all_attribs_keys=False)`

Decorator to add parse type for task definition dictionary.

**Parameters**

- **parse\_type\_name** (`str`) – str, the function can be selected in task definitions via this string
- **all\_attribs\_keys** (`bool`) – bool, if True the arguments for parsing multiple attributes are valid

**The decorated function has to have the following arguments:**

**param node** etree Element, on which to execute the xpath evaluations

**param schema\_dict** dict, containing all the path information and more

**param name** str, name of the tag/attribute

**param parser\_info\_out** dict, with warnings, info, errors, ...

**param kwargs** here all other keyword arguments are collected

**Return type** `Callable`

This module contains custom conversion functions for the outxml\_parser, which cannot be handled by the standard parsing framework

`masci_tools.io.parsers.fleur.outxml_conversions.calculate_total_magnetic_moment(out_dict, logger)`

Calculate the the total magnetic moment per cell

**Parameters** `out_dict` (`Dict[str, Any]`) – dict with the already parsed information

**Return type** `Dict[str, Any]`

`masci_tools.io.parsers.fleur.outxml_conversions.calculate_walltime(out_dict, logger)`

Calculate the walltime from start and end time

**Parameters**

- **out\_dict** (`Dict[str, Any]`) – dict with the already parsed information
- **logger** (`Logger`) – logger object for logging warnings, errors, if not provided all errors will be raised

**Return type** `Dict[str, Any]`

`masci_tools.io.parsers.fleur.outxml_conversions.convert_forces(out_dict, logger)`

Convert the parsed forces from a iteration

**Parameters** `out_dict` (`Dict[str, Any]`) – dict with the already parsed information

**Return type** `Dict[str, Any]`

`masci_tools.io.parsers.fleur.outxml_conversions.convert_ldau_definitions(out_dict, logger)`

Convert the parsed information from LDA+U into a more readable dict

ldau\_info has keys for each species with LDA+U (`{species_name}/{atom_number}`) and this in turn contains a dict with the LDA+U definition for the given orbital (spdf)

**Parameters** `out_dict` (`Dict[str, Any]`) – dict with the already parsed information

**Return type** `Dict[str, Any]`

`maschi_tools.io.parsers.fleur.outxml_conversions.convert_relax_info(out_dict, logger)`  
Convert the general relaxation information

**Parameters** `out_dict` (`Dict[str, Any]`) – dict with the already parsed information

**Return type** `Dict[str, Any]`

`maschi_tools.io.parsers.fleur.outxml_conversions.convert_total_energy(out_dict, logger)`  
Convert total energy to eV

**Return type** `Dict[str, Any]`

Functions for expanding/splitting or converting electron configuration strings

`maschi_tools.util.econfig.convert_fleur_config_to_econfig(fleurconf_str, keep_spin=False)`  
‘[Kr] (4d3/2) (4d5/2) (4f5/2) (4f7/2)’ -> ‘[Kr] 4d10 4f14’, or ‘[Kr] 4d3/2 4d5/2 4f5/2 4f7/2’

# for now only use for coreconfig, it will fill all orbitals, since it has no information on the filling.

**Parameters**

- **fleurconf\_str** (`str`) – string of the electron config like it is read from the inp.xml
- **keep\_spin** (`bool`) – bool if True the spin indices will be kept in the converted string

**Return type** `str`

**Returns** string of the electron config to be used in the inpgen

`maschi_tools.util.econfig.get_coreconfig(element, full=False)`  
returns the econfiguration as a string of an element.

**Parameters**

- **element** (`Union[str, int]`) – element string
- **full** (`bool`) – a bool if True the econfig without [He]... is returned

**Return type** `Optional[str]`

**Returns** coreconfig string

`maschi_tools.util.econfig.get_econfig(element, full=False)`  
returns the econfiguration as a string of an element.

**Parameters**

- **element** (`Union[str, int]`) – element string
- **full** (`bool`) – a bool if True the econfig without [He]... is returned

**Return type** `Optional[str]`

**Returns** a econfig string

`maschi_tools.util.econfig.rek_econ(econfigstr)`  
recursive routine to return a full econfig ‘[Xe] 4f14 | 5d10 6s2 6p4’ -> ‘1s 2s ... 4f14 | 5d10 6s2 6p4’

**Parameters** **econfigstr** (`str`) – electron config string to expand

**Return type** `Optional[str]`

**Returns** expanded econfig string

## 6.1.6 Basic Fleur Schema parser functions

Load all fleur schema related functions

**class** `masci_tools.io.parsers.fleur.fleur_schema.AttributeType`(*base\_type*, *length*)

**base\_type:** `str`

Alias for field number 0

**length:** `Optional[Union[int, Literal['unbounded']]]`

Alias for field number 1

**class** `masci_tools.io.parsers.fleur.fleur_schema.InputSchemaDict`(\*args, *xmlschema=None*, \*\*kwargs)

This class contains information parsed from the FleurInputSchema.xsd

The keys contain the following information:

**inp\_version** Version string of the input schema represented in this object

**tag\_paths** simple xpath expressions to all valid tag names Multiple paths or ambiguous tag names are parsed as a list

**\_basic\_types** Parsed definitions of all simple Types with their respective base type (int, float, ...) and evtl. length restrictions (Only used in the schema construction itself)

**attrib\_types** All possible base types for all valid attributes. If multiple are possible a list, with 'string' always last (if possible)

**simple\_elements** All elements with simple types and their type definition with the additional attributes

**unique\_attribs** All attributes and their paths, which occur only once and have a unique path

**unique\_path\_attribs** All attributes and their paths, which have a unique path but occur in multiple places

**other\_attribs** All attributes and their paths, which are not in 'unique\_attribs' or 'unique\_path\_attribs'

**omitt\_contained\_tags** All tags, which only contain a list of one other tag

**tag\_info** For each tag (path), the valid attributes and tags (optional, several, order, simple, text)

**classmethod** `fromPath`(*path*)

load the FleurInputSchema dict for the specified FleurInputSchema file

**Parameters** **path** (`PathLike`) – path to the input schema file

**Return type** `InputSchemaDict`

**Returns** InputSchemaDict object with the information for the provided file

**classmethod** `fromVersion`(*version*, *logger=None*, *no\_cache=False*)

load the FleurInputSchema dict for the specified version

**Parameters**

- **version** (`str`) – str with the desired version, e.g. '0.33'
- **logger** (`Optional[Logger]`) – logger object for warnings, errors and information, ...

**Return type** `InputSchemaDict`



**Returns** InputSchemaDict object with the information for the provided version

**property inp\_version:** Tuple[int, int]

Returns the input version as an integer for comparisons (> or <)

**Return type** Tuple[int, int]

**exception** masci\_tools.io.parsers.fleur.fleur\_schema.NoPathFound

**exception** masci\_tools.io.parsers.fleur.fleur\_schema.NoUniquePathFound

**class** masci\_tools.io.parsers.fleur.fleur\_schema.OutputSchemaDict(\*args, xmlschema=None, \*\*kwargs)

This object contains information parsed from the FleurOutputSchema.xsd

The keys contain the following information:

**out\_version** Version string of the output schema represented in this class

**input\_tag** Name of the element containing the fleur input

**tag\_paths** simple xpath expressions to all valid tag names not in an iteration Multiple paths or ambiguous tag names are parsed as a list

**iteration\_tag\_paths** simple relative xpath expressions to all valid tag names inside an iteration. Multiple paths or ambiguous tag names are parsed as a list

**\_basic\_types** Parsed definitions of all simple Types with their respective base type (int, float, ...) and evtl. length restrictions (Only used in the schema construction itself)

**\_input\_basic\_types** Part of the parsed definitions of all simple Types with their respective base type (int, float, ...) and evtl. length restrictions from the input schema (Only used in the schema construction itself)

**attrib\_types** All possible base types for all valid attributes. If multiple are possible a list, with 'string' always last (if possible)

**simple\_elements** All elements with simple types and their type definition with the additional attributes

**unique\_attribs** All attributes and their paths, which occur only once and have a unique path outside of an iteration

**unique\_path\_attribs** All attributes and their paths, which have a unique path but occur in multiple places outside of an iteration

**other\_attribs** All attributes and their paths, which are not in 'unique\_attribs' or 'unique\_path\_attribs' outside of an iteration

**iteration\_unique\_attribs** All attributes and their relative paths, which occur only once and have a unique path inside of an iteration

**iteration\_unique\_path\_attribs** All attributes and their relative paths, which have a unique path but occur in multiple places inside of an iteration

**iteration\_other\_attribs** All attributes and their relative paths, which are not in 'unique\_attribs' or 'unique\_path\_attribs' inside of an iteration

**omitt\_contained\_tags** All tags, which only contain a list of one other tag

**tag\_info** For each tag outside of an iteration (path), the valid attributes and tags (optional, several, order, simple, text)

**iteration\_tag\_info** For each tag inside of an iteration (relative path), the valid attributes and tags (optional, several, order, simple, text)

**classmethod** `fromPath(path, inp_path=None, inpschema_dict=None)`

load the FleurOutputSchema dict for the specified paths

**Parameters**

- **path** (`PathLike`) – path to the FleurOutputSchema file
- **inp\_path** (`Optional[PathLike]`) – path to the FleurInputSchema file (defaults to same folder as path)

**Return type** `OutputSchemaDict`

**Returns** OutputSchemaDict object with the information for the provided files

**classmethod** `fromVersion(version, inp_version=None, logger=None, no_cache=False)`

load the FleurOutputSchema dict for the specified version

**Parameters**

- **version** (`str`) – str with the desired version, e.g. ‘0.33’
- **inp\_version** (`Optional[str]`) – str with the desired input version, e.g. ‘0.33’ (defaults to version)
- **logger** (`Optional[Logger]`) – logger object for warnings, errors and information, ...

**Return type** `OutputSchemaDict`

**Returns** OutputSchemaDict object with the information for the provided versions

**property** `inp_version: Tuple[int, int]`

Returns the input version as an integer for comparisons (> or <)

**Return type** `Tuple[int, int]`

**property** `out_version: Tuple[int, int]`

Returns the output version as an integer for comparisons (> or <)

**Return type** `Tuple[int, int]`

**class** `masci_tools.io.parsers.fleur.fleur_schema.SchemaDict(*args, xmlschema=None, **kwargs)`

Base class for schema dictionaries. Is locked on initialization with `freeze()`. Holds a reference to the xmlSchema for validating files.

Also provides interfaces for utility functions

**Parameters** `xmlschema` (`Optional[XMLSchema]`) – etree.XMLSchema object for validating files

All other arguments are passed on to `LockableDict`

**attrib\_xpath**(`name, contains=None, not_contains=None, exclude=None, tag_name=None`)

Tries to find a unique path from the schema\_dict based on the given name of the attribute and additional further specifications

**Parameters**

- **name** (`str`) – str, name of the attribute
- **root\_tag** – str, name of the tag from which the path should be relative
- **contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to be in the final path
- **not\_contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to NOT be in the final path

- **exclude** (*Optional*[*Iterable*[*str*]]) – list of str, here specific types of attributes can be excluded valid values are: *settable*, *settable\_contains*, *other*
- **tag\_name** (*Optional*[*str*]) – str, if given this name will be used to find a path to a tag with the same name in *tag\_xpath()*

**Return type** *str*

**Returns** str, xpath to the tag with the given attribute

**Raises**

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

**classmethod** *clear\_cache()*

Remove all stored entries in the schema dictionary cache

**Return type** *None*

**relative\_attr\_xpath**(*name*, *root\_tag*, *contains=None*, *not\_contains=None*, *exclude=None*, *tag\_name=None*)

Tries to find a unique relative path from the schema\_dict based on the given name of the attribute name of the root, from which the path should be relative and additional further specifications

**Parameters**

- **schema\_dict** – dict, containing all the path information and more
- **name** (*str*) – str, name of the attribute
- **contains** (*Union*[*str*, *Iterable*[*str*], *None*]) – str or list of str, this string has to be in the final path
- **not\_contains** (*Union*[*str*, *Iterable*[*str*], *None*]) – str or list of str, this string has to NOT be in the final path
- **exclude** (*Optional*[*Iterable*[*str*]]) – list of str, here specific types of attributes can be excluded valid values are: *settable*, *settable\_contains*, *other*
- **tag\_name** (*Optional*[*str*]) – str, if given this name will be used to find a path to a tag with the same name in *relative\_tag\_xpath()*

**Return type** *str*

**Returns** str, xpath for the given tag

**Raises**

- *NoPathFound* – If no path matching the criteria could be found
- *NoUniquePathFound* – If multiple paths matching the criteria are found

**relative\_tag\_xpath**(*name*, *root\_tag*, *contains=None*, *not\_contains=None*)

Tries to find a unique relative path from the schema\_dict based on the given name of the tag name of the root, from which the path should be relative and additional further specifications

**Parameters**

- **name** (*str*) – str, name of the tag
- **root\_tag** (*str*) – str, name of the tag from which the path should be relative
- **contains** (*Union*[*str*, *Iterable*[*str*], *None*]) – str or list of str, this string has to be in the final path

- **not\_contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to NOT be in the final path

**Return type** `str`

**Returns** str, xpath for the given tag

**Raises** `ValueError` – If no unique path could be found

**tag\_info**(*name*, *contains=None*, *not\_contains=None*, *parent=False*, *\*\*kwargs*)

Tries to find a unique path from the `schema_dict` based on the given name of the tag and additional further specifications and returns the `tag_info` entry for this tag

**Parameters**

- **schema\_dict** – dict, containing all the path information and more
- **name** (`str`) – str, name of the tag
- **contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to be in the final path
- **not\_contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to NOT be in the final path
- **parent** (`bool`) – bool, if True the `tag_info` for the parent of the tag is returned

**Return type** `Dict[str, Any]`

**Returns** dict, `tag_info` for the found xpath

**tag\_xpath**(*name*, *contains=None*, *not\_contains=None*)

Tries to find a unique path from the `schema_dict` based on the given name of the tag and additional further specifications

**Parameters**

- **name** (`str`) – str, name of the tag
- **contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to be in the final path
- **not\_contains** (`Union[str, Iterable[str], None]`) – str or list of str, this string has to NOT be in the final path

**Return type** `str`

**Returns** str, xpath for the given tag

**Raises**

- **NoPathFound** – If no path matching the criteria could be found
- **NoUniquePathFound** – If multiple paths matching the criteria are found

`masci_tools.io.parsers.fleur.fleur_schema.schema_dict_version_dispatch(output_schema=False)`

Decorator for creating variations of functions based on the inp/out version of the `schema_dict`. All functions here need to have the signature:

```
def f(node, schema_dict, *args, **kwargs):
    pass
```

So `schema_dict` is the second positional argument

Inspired by `singledispatch` in the `functools` module

**Return type** `Callable`

functions to extract information about the fleur schema input or output

**class** `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.AttributeType`(*base\_type*,  
*length*)

**base\_type:** `str`

Alias for field number 0

**length:** `Optional[Union[int, Literal['unbounded']]]`

Alias for field number 1

**class** `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.TagInfo`(\*args,  
\*\*kwargs)

Dict representing the entries for the tag information.

`masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.extract_attribute_types`(*xmlschema*,  
*namespaces*,  
\*\*kwargs)

Determine the required type of all attributes

#### Parameters

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `CaseInsensitiveDict[str, List[AttributeType]]`

**Returns** possible types of the attributes in a dictionary, if multiple types are possible a list is inserted for the tag

`masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.extract_text_types`(*xmlschema*,  
*namespaces*,  
\*\*kwargs)

Determine the required type of all elements with text

#### Parameters

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `CaseInsensitiveDict[str, List[AttributeType]]`

**Returns** possible types of the attributes in a dictionary, if multiple types are possible a list is inserted for the tag

`masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_basic_types`(*xmlschema*,  
*namespaces*,  
\*\*kwargs)

find all types, which can be traced back directly to a base\_type

#### Parameters

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `Dict[str, List[AttributeType]]`

**Returns** dictionary with type names and their corresponding type\_definition meaning a dictionary with possible base types and evtl. length restriction

```
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_input_tag(xmlschema,
                                                                                       names-
                                                                                       paces,
                                                                                       **kwargs)
```

Returns the tag for the input type element of the outxmlschema

**Parameters**

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `str`

**Returns** name of the element with the type 'FleurInputType'

```
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_omittable_tags(xmlschema,
                                                                                          names-
                                                                                          paces,
                                                                                          **kwargs)
```

find tags with no attributes and, which are only used to mask a list of one other possible tag (e.g. atomSpecies)

**Parameters**

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `List[str]`

**Returns** list of tags, containing only a sequence of one allowed tag

```
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_other_attribs(xmlschema,
                                                                                          names-
                                                                                          paces,
                                                                                          **kwargs)
```

Determine all other attributes not contained in settable or settable\_contains

**Parameters**

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `CaseInsensitiveDict[str, List[str]]`

**Returns** dictionary with all attributes and the corresponding list of paths to the tag

```
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_root_tag(xmlschema,
                                                                                      names-
                                                                                      paces,
                                                                                      **kwargs)
```

Returns the tag for the root element of the xmlschema

**Parameters**

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `str`

**Returns** name of the single element defined in the first level of the schema

```
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_tag_info(xmlschema,
                                                                                       names-
                                                                                       paces,
                                                                                       **kwargs)
```

#### Get all important information about the tags

- allowed attributes
- contained tags (simple (only attributes), optional (with default values), several, order, text tags)

##### Parameters

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `Dict[str, TagInfo]`

**Returns** dictionary with the tag information

```
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_tag_paths(xmlschema,
                                                                                       names-
                                                                                       paces,
                                                                                       **kwargs)
```

Determine simple xpaths to all possible tags

##### Parameters

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `CaseInsensitiveDict[str, Union[List[str], str]]`

**Returns** possible paths of all tags in a dictionary, if multiple paths are possible a list is inserted for the tag

```
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_text_tags(xmlschema,
                                                                                       names-
                                                                                       paces,
                                                                                       **kwargs)
```

find all elements, who can contain text

##### Parameters

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `CaseInsensitiveFrozenSet[str]`

**Returns** dictionary with tags and their corresponding type\_definition meaning a dicationary with possible base types and evtl. length restriction

```
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_unique_attribs(xmlschema,
                                                                                          names-
                                                                                          paces,
                                                                                          **kwargs)
```

Determine all attributes, which can be set through set\_inpchanges in aiida\_fleur Meaning ONE possible path and no tags in the path with maxOccurs!=1

##### Parameters

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `CaseInsensitiveDict[str, str]`

**Returns** dictionary with all settable attributes and the corresponding path to the tag

`masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.get_unique_path_attribs(xmlschema, namespaces, **kwargs)`

Determine all attributes, with multiple possible path that do have at least one path with all contained tags max-Occurs!=1

**Parameters**

- **xmlschema** (`_ElementTree`) – xmltree representing the schema
- **namespaces** (`Dict[str, str]`) – dictionary with the defined namespaces

**Return type** `CaseInsensitiveDict[str, List[str]]`

**Returns** dictionary with all attributes and the corresponding list of paths to the tag

`masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions.type_order(type_def)`  
Key function for sorting the type definitions to avoid conflicts

Sorted by base\_type first (bool before int, string at the end) and then by length in ascending order (unbounded last)

**Parameters** **type\_def** (`AttributeType`) – definition to be sorted

**Return type** `Tuple[int, float]`

## 6.1.7 Defined constants

Here we collect physical constants which are used throughout the code. That way we ensure consistency.

---

**Note:** For masci-tools versions after v0.4.6, the constants values used in the KKR functions for conversion between Angstrom and Bohr radius, and electron Volt and Rydberg, have been replaced by the NIST values by default. Prior to that, two different versions had been in use. If you need to work with KKR calculations / aiida-kkr workchains performed with these constants versions, you can switch to these older KKR constants versions by setting the environment variable `MASCI_TOOLS_USE_OLD_CONSTANTS` prior to loading masci-tools. During interpreter runtime, the chosen version cannot be switched.

- For KKR constants versions used starting from masci-tools v0.4.7, more specifically starting from commit 66953f8, Apr 28, 2021, do not set `MASCI_TOOLS_USE_OLD_CONSTANTS`.
  - For KKR constants versions used in masci-tools v0.4.0-dev7 - v0.4.6, more specifically starting from commit c171563, Feb 16, 2021, to prior to commit 66953f8, Apr 28, 2021, set `MASCI_TOOLS_USE_OLD_CONSTANTS` to 'interim'.
  - For KKR constants versions used prior to masci-tools v0.4.0-dev7, more specifically prior to commit c171563, Feb 16, 2021, set `MASCI_TOOLS_USE_OLD_CONSTANTS` to 'old' or 'True'.
- 

```
1 .. literalinclude:: ../../../../masci_tools/util/constants.py
2   :language: python
3   :lines: 28-
```

(continues on next page)



(continued from previous page)

```

4      :linenos:
5
6      """
7      import numpy as np
8      import os
9
10     # NIST https://physics.nist.gov/cgi-bin/cuu/Value?hrev
11     HTR_TO_EV = 27.211386245988  #(53)
12     RY_TO_EV = 13.605693122994  #(26)
13     BOHR_A = 0.5291772108
14     HTR_TO_KELVIN = 315_775.02480407
15     #Scipy bohr 5.29177210903e-11 m
16     #Scipy htr 27.211386245988 eV
17     # NIST BOHR 0.529177210903 #(80)
18     #https://physics.nist.gov/cgi-bin/cuu/Value?bohrrada0
19
20     #KKR constants versions. See module docstring for details.
21     _MASCI_TOOLS_USE_OLD_CONSTANTS = os.environ.get('MASCI_TOOLS_USE_OLD_CONSTANTS', None)
22     if _MASCI_TOOLS_USE_OLD_CONSTANTS and _MASCI_TOOLS_USE_OLD_CONSTANTS.lower() in ['old',
23     ↪ 'true']:
24         ANG_BOHR_KKR = 1.8897261254578281
25         RY_TO_EV_KKR = 13.605693009
26     elif _MASCI_TOOLS_USE_OLD_CONSTANTS and _MASCI_TOOLS_USE_OLD_CONSTANTS.lower() in [
27     ↪ 'interim']:
28         ANG_BOHR_KKR = 1.8897261249935897
29         RY_TO_EV_KKR = RY_TO_EV
30     else:
31         #Set the constants to the NIST values
32         ANG_BOHR_KKR = 1.8897261246257702
33         RY_TO_EV_KKR = RY_TO_EV
34
35     #Fleur
36     #htr_eV  = 27.21138602
37     #bohr=0.5291772108
38     #bohrtocm=0.529177e-8
39     #pymatgen uses scipy.constants
40     #ase: Bohr 0.5291772105638411
41     #Hartree 27.211386024367243
42     #Rydberg 13.605693012183622
43     #1/Bohr
44     #1.8897261258369282
45     #aiida-core units:
46     #bohr_to_ang = 0.52917720859
47
48     #Predefined constants in the Fleur Code (These are accepted in the inp.xml)
49     FLEUR_DEFINED_CONSTANTS = {
50         'Pi': np.pi,
51         'Deg': 2 * np.pi / 360.0,
52         'Ang': 1.8897261247728981,
53         'nm': 18.897261247728981,
54         'pm': 0.018897261247728981,
55         'Bohr': 1.0,

```

(continues on next page)

(continued from previous page)

```

54     'Htr': 1.0,
55     'eV': 1.0 / HTR_TO_EV,
56     'Ry': 0.5
57 }
58
59 PERIODIC_TABLE_ELEMENTS = {
60     0: { # This is for empty spheres etc.
61         'mass': 1.000000,
62         'name': 'Unknown',
63         'symbol': 'X'
64     },
65     1: {
66         'mass': 1.00794,
67         'name': 'Hydrogen',
68         'symbol': 'H',
69         'econfig': '1s1',
70         'fleur_default_econfig': '| 1s1',
71         'lo': '',
72         'rmt': 0.65,
73         'lmax': '',
74         'jri': 981,
75         'soc': False,
76         'mag': False
77     },
78     2: {
79         'mass': 4.002602,
80         'name': 'Helium',
81         'symbol': 'He',
82         'econfig': '1s2',
83         'fleur_default_econfig': '| 1s2',
84         'lo': '',
85         'rmt': 1.2,
86         'lmax': '',
87         'jri': 981
88     },
89     3: {
90         'mass': 6.941,
91         'name': 'Lithium',
92         'symbol': 'Li',
93         'econfig': '1s2 | 2s1',
94         'fleur_default_econfig': '1s2 | 2s1',
95         'lo': '',
96         'rmt': 2.13,
97         'lmax': '',
98         'jri': 981
99     },
100    4: {
101        'mass': 9.012182,
102        'name': 'Beryllium',
103        'symbol': 'Be',
104        'econfig': '1s2 | 2s2',
105        'fleur_default_econfig': '1s2 | 2s2',

```

(continues on next page)

(continued from previous page)

```

106     'lo': '',
107     'rmt': 1.87,
108     'lmax': '',
109     'jri': 981
110 },
111 5: {
112     'mass': 10.811,
113     'name': 'Boron',
114     'symbol': 'B',
115     'econfig': '1s2 | 2s2 2p1',
116     'fleur_default_econfig': '1s2 | 2s2 2p1',
117     'lo': '',
118     'rmt': 1.4,
119     'lmax': '',
120     'jri': 981
121 },
122 6: {
123     'mass': 12.0107,
124     'name': 'Carbon',
125     'symbol': 'C',
126     'econfig': '[He] 2s2 | 2p2',
127     'fleur_default_econfig': '[He] 2s2 | 2p2',
128     'lo': '',
129     'rmt': 1.2,
130     'lmax': '',
131     'jri': 981
132 },
133 7: {
134     'mass': 14.0067,
135     'name': 'Nitrogen',
136     'symbol': 'N',
137     'econfig': '[He] 2s2 | 2p3',
138     'fleur_default_econfig': '[He] 2s2 | 2p3',
139     'lo': '',
140     'rmt': 1.0,
141     'lmax': '',
142     'jri': 981
143 },
144 8: {
145     'mass': 15.9994,
146     'name': 'Oxygen',
147     'symbol': 'O',
148     'econfig': '[He] 2s2 | 2p4',
149     'fleur_default_econfig': '[He] 2s2 | 2p4',
150     'lo': '',
151     'rmt': 1.1,
152     'lmax': '',
153     'jri': 981
154 },
155 9: {
156     'mass': 18.9984032,
157     'name': 'Fluorine',

```

(continues on next page)

(continued from previous page)

```

158     'symbol': 'F',
159     'econfig': '[He] 2s2 | 2p5',
160     'fleur_default_econfig': '[He] 2s2 | 2p5',
161     'lo': '',
162     'rmt': 1.2,
163     'lmax': '',
164     'jri': 981
165 },
166 10: {
167     'mass': 20.1797,
168     'name': 'Neon',
169     'symbol': 'Ne',
170     'econfig': '[He] 2s2 | 2p6',
171     'fleur_default_econfig': '[He] 2s2 | 2p6',
172     'lo': '',
173     'rmt': 2.1,
174     'lmax': '',
175     'jri': 981
176 },
177 11: {
178     'mass': 22.98977,
179     'name': 'Sodium',
180     'symbol': 'Na',
181     'econfig': '[He] 2s2 | 2p6 3s1',
182     'fleur_default_econfig': '[He] | 2s2 2p6 3s1',
183     'lo': '2s 2p',
184     'rmt': 2.1,
185     'lmax': '',
186     'jri': 981
187 },
188 12: {
189     'mass': 24.305,
190     'name': 'Magnesium',
191     'symbol': 'Mg',
192     'econfig': '[He] 2s2 | 2p6 3s2',
193     'fleur_default_econfig': '[He] 2s2 | 2p6 3s2',
194     'lo': '2p',
195     'rmt': 2.3,
196     'lmax': '',
197     'jri': 981
198 },
199 13: {
200     'mass': 26.981538,
201     'name': 'Aluminium',
202     'symbol': 'Al',
203     'econfig': '[He] 2s2 2p6 | 3s2 3p1',
204     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p1',
205     'lo': '',
206     'rmt': 2.1,
207     'lmax': '',
208     'jri': 981
209 },

```

(continues on next page)

(continued from previous page)

```

210 14: {
211     'mass': 28.0855,
212     'name': 'Silicon',
213     'symbol': 'Si',
214     'econfig': '[He] 2s2 2p6 | 3s2 3p2',
215     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p2',
216     'lo': '',
217     'rmt': 2.0,
218     'lmax': '',
219     'jri': 981
220 },
221 15: {
222     'mass': 30.973761,
223     'name': 'Phosphorus',
224     'symbol': 'P',
225     'econfig': '[He] 2s2 2p6 | 3s2 3p3',
226     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p3',
227     'lo': '',
228     'rmt': 1.9,
229     'lmax': '',
230     'jri': 981
231 },
232 16: {
233     'mass': 32.065,
234     'name': 'Sulfur',
235     'symbol': 'S',
236     'econfig': '[He] 2s2 2p6 | 3s2 3p4',
237     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p4',
238     'lo': '',
239     'rmt': 1.7,
240     'lmax': '',
241     'jri': 981
242 },
243 17: {
244     'mass': 35.453,
245     'name': 'Chlorine',
246     'symbol': 'Cl',
247     'econfig': '[He] 2s2 2p6 | 3s2 3p5',
248     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p5',
249     'lo': '',
250     'rmt': 1.7,
251     'lmax': '',
252     'jri': 981
253 },
254 18: {
255     'mass': 39.948,
256     'name': 'Argon',
257     'symbol': 'Ar',
258     'econfig': '[He] 2s2 2p6 | 3s2 3p6',
259     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p6',
260     'lo': '',
261     'rmt': 1.8,

```

(continues on next page)

(continued from previous page)

```
262     'lmax': '',
263     'jri': 981
264 },
265 19: {
266     'mass': 39.0983,
267     'name': 'Potassium',
268     'symbol': 'K',
269     'econfig': '[Ne] 3s2 | 3p6 4s1',
270     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s1',
271     'lo': '3s 3p',
272     'rmt': 2.0,
273     'lmax': '',
274     'jri': 981
275 },
276 20: {
277     'mass': 40.078,
278     'name': 'Calcium',
279     'symbol': 'Ca',
280     'econfig': '[Ne] 3s2 | 3p6 4s2',
281     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2',
282     'lo': '3s 3p',
283     'rmt': 2.3,
284     'lmax': '',
285     'jri': 981
286 },
287 21: {
288     'mass': 44.955912,
289     'name': 'Scandium',
290     'symbol': 'Sc',
291     'econfig': '[Ne] 3s2 3p6 | 4s2 3d1',
292     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d1',
293     'lo': '3s 3p',
294     'rmt': 2.2,
295     'lmax': '',
296     'jri': 981
297 },
298 22: {
299     'mass': 47.867,
300     'name': 'Titanium',
301     'symbol': 'Ti',
302     'econfig': '[Ne] | 3s2 3p6 4s2 3d2',
303     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d2',
304     'lo': '3s 3p',
305     'rmt': 2.1,
306     'lmax': '',
307     'jri': 981
308 },
309 23: {
310     'mass': 50.9415,
311     'name': 'Vanadium',
312     'symbol': 'V',
313     'econfig': '[Ne] 3s2 3p6 | 4s2 3d3',
```

(continues on next page)

(continued from previous page)

```

314     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d3',
315     'lo': '3s 3p',
316     'rmt': 1.9,
317     'lmax': '',
318     'jri': 981
319 },
320 24: {
321     'mass': 51.9961,
322     'name': 'Chromium',
323     'symbol': 'Cr',
324     'econfig': '[Ne] 3s2 3p6 | 4s1 3d5',
325     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s1 3d5',
326     'lo': '3s 3p',
327     'rmt': 1.8,
328     'lmax': '',
329     'jri': 981
330 },
331 25: {
332     'mass': 54.938045,
333     'name': 'Manganese',
334     'symbol': 'Mn',
335     'econfig': '[Ne] 3s2 3p6 | 4s2 3d5',
336     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d5',
337     'lo': '3s 3p',
338     'rmt': 2.0,
339     'lmax': '',
340     'jri': 981
341 },
342 26: {
343     'mass': 55.845,
344     'name': 'Iron',
345     'symbol': 'Fe',
346     'econfig': '[Ne] 3s2 3p6 | 4s2 3d6',
347     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d6',
348     'lo': '3s 3p',
349     'rmt': 2.00,
350     'lmax': '',
351     'jri': 981
352 },
353 27: {
354     'mass': 58.933195,
355     'name': 'Cobalt',
356     'symbol': 'Co',
357     'econfig': '[Ne] 3s2 3p6 | 4s2 3d7',
358     'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s2 3d7',
359     'lo': '3p',
360     'rmt': 1.9,
361     'lmax': '',
362     'jri': 981
363 },
364 28: {
365     'mass': 58.6934,

```

(continues on next page)

(continued from previous page)

```

366     'name': 'Nickel',
367     'symbol': 'Ni',
368     'econfig': '[Ne] 3s2 3p6 | 4s2 3d8',
369     'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s2 3d8',
370     'lo': '3p',
371     'rmt': 1.9,
372     'lmax': '',
373     'jri': 981
374 },
375 29: {
376     'mass': 63.546,
377     'name': 'Copper',
378     'symbol': 'Cu',
379     'econfig': '[Ne] 3s2 3p6 | 4s1 3d10',
380     'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s1 3d10',
381     'lo': '3p',
382     'rmt': 2.1,
383     'lmax': '',
384     'jri': 981
385 },
386 30: {
387     'mass': 65.38,
388     'name': 'Zinc',
389     'symbol': 'Zn',
390     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2',
391     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2',
392     'lo': '3d',
393     'rmt': 2.2,
394     'lmax': '',
395     'jri': 981
396 },
397 31: {
398     'mass': 69.723,
399     'name': 'Gallium',
400     'symbol': 'Ga',
401     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p1',
402     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p1',
403     'lo': '3d',
404     'rmt': 2.1,
405     'lmax': '',
406     'jri': 981
407 },
408 32: {
409     'mass': 72.64,
410     'name': 'Germanium',
411     'symbol': 'Ge',
412     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p2',
413     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p2',
414     'lo': '3d',
415     'rmt': 2.1,
416     'lmax': '',
417     'jri': 981

```

(continues on next page)



(continued from previous page)

```

418 },
419 33: {
420     'mass': 74.9216,
421     'name': 'Arsenic',
422     'symbol': 'As',
423     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p3',
424     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p3',
425     'lo': '3d',
426     'rmt': 2.0,
427     'lmax': '',
428     'jri': 981
429 },
430 34: {
431     'mass': 78.96,
432     'name': 'Selenium',
433     'symbol': 'Se',
434     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p4',
435     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p4',
436     'lo': '3d',
437     'rmt': 2.0,
438     'lmax': '',
439     'jri': 981
440 },
441 35: {
442     'mass': 79.904,
443     'name': 'Bromine',
444     'symbol': 'Br',
445     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p5',
446     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p5',
447     'lo': '3d',
448     'rmt': 2.1,
449     'lmax': '',
450     'jri': 981
451 },
452 36: {
453     'mass': 83.798,
454     'name': 'Krypton',
455     'symbol': 'Kr',
456     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p6',
457     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p6',
458     'lo': '3d',
459     'rmt': 2.2,
460     'lmax': '',
461     'jri': 981
462 },
463 37: {
464     'mass': 85.4678,
465     'name': 'Rubidium',
466     'symbol': 'Rb',
467     'econfig': '[Ar] 3d10 4s2 | 4p6 5s1',
468     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1',
469     'lo': '4s 4p',

```

(continues on next page)

(continued from previous page)

```

470     'rmt': 2.4,
471     'lmax': '',
472     'jri': 981
473 },
474 38: {
475     'mass': 87.62,
476     'name': 'Strontium',
477     'symbol': 'Sr',
478     'econfig': '[Ar] 3d10 4s2 | 4p6 5s2',
479     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2',
480     'lo': '4s 4p',
481     'rmt': 2.4,
482     'lmax': '',
483     'jri': 981
484 },
485 39: {
486     'mass': 88.90585,
487     'name': 'Yttrium',
488     'symbol': 'Y',
489     'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d1',
490     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d1',
491     'lo': '4s 4p',
492     'rmt': 2.4,
493     'lmax': '',
494     'jri': 981
495 },
496 40: {
497     'mass': 91.224,
498     'name': 'Zirconium',
499     'symbol': 'Zr',
500     'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d2',
501     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d2',
502     'lo': '4s 4p',
503     'rmt': 2.3,
504     'lmax': '',
505     'jri': 981
506 },
507 41: {
508     'mass': 92.90638,
509     'name': 'Niobium',
510     'symbol': 'Nb',
511     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d4',
512     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d4',
513     'lo': '4s 4p',
514     'rmt': 2.1,
515     'lmax': '',
516     'jri': 981
517 },
518 42: {
519     'mass': 95.96,
520     'name': 'Molybdenum',
521     'symbol': 'Mo',

```

(continues on next page)

(continued from previous page)

```

522     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d5',
523     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d5',
524     'lo': '4s 4p',
525     'rmt': 2.0,
526     'lmax': '',
527     'jri': 981
528 },
529 43: {
530     'mass': 98.0,
531     'name': 'Technetium',
532     'symbol': 'Tc',
533     'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d5',
534     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d5',
535     'lo': '4s 4p',
536     'rmt': 2.1,
537     'lmax': '',
538     'jri': 981
539 },
540 44: {
541     'mass': 101.07,
542     'name': 'Ruthenium',
543     'symbol': 'Ru',
544     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d7',
545     'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 5s1 4d7',
546     'lo': '4p',
547     'rmt': 2.1,
548     'lmax': '',
549     'jri': 981
550 },
551 45: {
552     'mass': 102.9055,
553     'name': 'Rhodium',
554     'symbol': 'Rh',
555     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d8',
556     'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 5s1 4d8',
557     'lo': '4p',
558     'rmt': 2.1,
559     'lmax': '',
560     'jri': 981
561 },
562 46: {
563     'mass': 106.42,
564     'name': 'Palladium',
565     'symbol': 'Pd',
566     'econfig': '[Ar] 4s2 3d10 4p6 | 4d10',
567     'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 4d10',
568     'lo': '4p',
569     'rmt': 2.1,
570     'lmax': '',
571     'jri': 981
572 },
573 47: {

```

(continues on next page)

(continued from previous page)

```

574     'mass': 107.8682,
575     'name': 'Silver',
576     'symbol': 'Ag',
577     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d10',
578     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d10',
579     'lo': '4s 4p',
580     'rmt': 2.3,
581     'lmax': '',
582     'jri': 981
583 },
584 48: {
585     'mass': 112.411,
586     'name': 'Cadmium',
587     'symbol': 'Cd',
588     'econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2',
589     'fleur_default_econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2',
590     'lo': '4d',
591     'rmt': 2.4,
592     'lmax': '',
593     'jri': 981
594 },
595 49: {
596     'mass': 114.818,
597     'name': 'Indium',
598     'symbol': 'In',
599     'econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2 5p1',
600     'fleur_default_econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2 5p1',
601     'lo': '4d',
602     'rmt': 2.2,
603     'lmax': '',
604     'jri': 981
605 },
606 50: {
607     'mass': 118.71,
608     'name': 'Tin',
609     'symbol': 'Sn',
610     'econfig': '[Kr] 4d10 | 5s2 5p2',
611     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p2',
612     'lo': '4d',
613     'rmt': 2.3,
614     'lmax': '',
615     'jri': 981
616 },
617 51: {
618     'mass': 121.76,
619     'name': 'Antimony',
620     'symbol': 'Sb',
621     'econfig': '[Kr] 4d10 | 5s2 5p3',
622     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p3',
623     'lo': '4d',
624     'rmt': 2.3,
625     'lmax': '',

```

(continues on next page)

(continued from previous page)

```

626     'jri': 981
627 },
628 52: {
629     'mass': 127.6,
630     'name': 'Tellurium',
631     'symbol': 'Te',
632     'econfig': '[Kr] 4d10 | 5s2 5p4',
633     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p4',
634     'lo': '4d',
635     'rmt': 2.3,
636     'lmax': '',
637     'jri': 981
638 },
639 53: {
640     'mass': 126.90447,
641     'name': 'Iodine',
642     'symbol': 'I',
643     'econfig': '[Kr] 4d10 | 5s2 5p5',
644     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p5',
645     'lo': '4d',
646     'rmt': 2.2,
647     'lmax': '',
648     'jri': 981
649 },
650 54: {
651     'mass': 131.293,
652     'name': 'Xenon',
653     'symbol': 'Xe',
654     'econfig': '[Kr] 4d10 | 5s2 5p6',
655     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p6',
656     'lo': '4d',
657     'rmt': 2.2,
658     'lmax': '',
659     'jri': 981
660 },
661 55: {
662     'mass': 132.9054519,
663     'name': 'Caesium',
664     'symbol': 'Cs',
665     'econfig': '[Kr] 4d10 5s2 | 5p6 6s1',
666     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s1',
667     'lo': '5s 5p',
668     'rmt': 2.4,
669     'lmax': '',
670     'jri': 981
671 },
672 56: {
673     'mass': 137.327,
674     'name': 'Barium',
675     'symbol': 'Ba',
676     'econfig': '[Kr] 4d10 5s2 | 5p6 6s2',
677     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2',

```

(continues on next page)

(continued from previous page)

```

678     'lo': '5s 5p',
679     'rmt': 2.4,
680     'lmax': '',
681     'jri': 981
682 },
683 57: {
684     'mass': 138.90547,
685     'name': 'Lanthanum',
686     'symbol': 'La',
687     'econfig': '[Kr] 4d10 5s2 | 5p6 6s2 5d1',
688     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 5d1',
689     'lo': '5s 5p',
690     'rmt': 2.2,
691     'lmax': '',
692     'jri': 981
693 },
694 58: {
695     'mass': 140.116,
696     'name': 'Cerium',
697     'symbol': 'Ce',
698     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f1 5d1',
699     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f1 5d1',
700     'lo': '5s 5p',
701     'rmt': 2.2,
702     'lmax': '',
703     'jri': 981
704 },
705 59: {
706     'mass': 140.90765,
707     'name': 'Praseodymium',
708     'symbol': 'Pr',
709     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f3',
710     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f3',
711     'lo': '5s 5p',
712     'rmt': 2.4,
713     'lmax': '',
714     'jri': 981
715 },
716 60: {
717     'mass': 144.242,
718     'name': 'Neodymium',
719     'symbol': 'Nd',
720     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f4',
721     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f4',
722     'lo': '5s 5p',
723     'rmt': 2.1,
724     'lmax': '',
725     'jri': 981
726 },
727 61: {
728     'mass': 145.0,
729     'name': 'Promethium',

```

(continues on next page)

(continued from previous page)

```

730     'symbol': 'Pm',
731     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f5',
732     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f5',
733     'lo': '5s 5p',
734     'rmt': 2.4,
735     'lmax': '',
736     'jri': 981
737 },
738 62: {
739     'mass': 150.36,
740     'name': 'Samarium',
741     'symbol': 'Sm',
742     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f6',
743     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f6',
744     'lo': '5s 5p',
745     'rmt': 2.1,
746     'lmax': '',
747     'jri': 981
748 },
749 63: {
750     'mass': 151.964,
751     'name': 'Europium',
752     'symbol': 'Eu',
753     'econfig': '[Kr] 4d10 | 4f7 5s2 5p6 6s2',
754     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 4f7 6s2',
755     'lo': '5s 5p',
756     'rmt': 2.4,
757     'lmax': '',
758     'jri': 981
759 },
760 64: {
761     'mass': 157.25,
762     'name': 'Gadolinium',
763     'symbol': 'Gd',
764     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f7 5d1',
765     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f7 5d1',
766     'lo': '5s 5p',
767     'rmt': 2.2,
768     'lmax': '',
769     'jri': 981
770 },
771 65: {
772     'mass': 158.92535,
773     'name': 'Terbium',
774     'symbol': 'Tb',
775     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f9',
776     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f8 5d1',
777     'lo': '5s 5p',
778     'rmt': 2.1,
779     'lmax': '',
780     'jri': 981
781 },

```

(continues on next page)

(continued from previous page)

```

782 66: {
783     'mass': 162.5,
784     'name': 'Dysprosium',
785     'symbol': 'Dy',
786     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f10',
787     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f9 5d1',
788     'lo': '5s 5p',
789     'rmt': 2.4,
790     'lmax': '',
791     'jri': 981
792 },
793 67: {
794     'mass': 164.93032,
795     'name': 'Holmium',
796     'symbol': 'Ho',
797     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f11',
798     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f10 5d1',
799     'lo': '5s 5p',
800     'rmt': 2.4,
801     'lmax': '',
802     'jri': 981
803 },
804 68: {
805     'mass': 167.259,
806     'name': 'Erbium',
807     'symbol': 'Er',
808     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f12',
809     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f11 5d1',
810     'lo': '5s 5p',
811     'rmt': 2.5,
812     'lmax': '',
813     'jri': 981
814 },
815 69: {
816     'mass': 168.93421,
817     'name': 'Thulium',
818     'symbol': 'Tm',
819     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f13',
820     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f12 5d1',
821     'lo': '5s 5p',
822     'rmt': 2.4,
823     'lmax': '',
824     'jri': 981
825 },
826 70: {
827     'mass': 173.054,
828     'name': 'Ytterbium',
829     'symbol': 'Yb',
830     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f14',
831     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f13 5d1',
832     'lo': '5s 5p',
833     'rmt': 2.6,

```

(continues on next page)



(continued from previous page)

```

834     'lmax': '',
835     'jri': 981
836 },
837 71: {
838     'mass': 174.9668,
839     'name': 'Lutetium',
840     'symbol': 'Lu',
841     'econfig': '[Kr] 4d10 | 4f14 5s2 5p6 5d1 6s2',
842     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 4f14 6s2 5d1',
843     'lo': '5s 5p',
844     'rmt': 2.5,
845     'lmax': '',
846     'jri': 981
847 },
848 72: {
849     'mass': 178.49,
850     'name': 'Hafnium',
851     'symbol': 'Hf',
852     'econfig': '[Kr] 4d10 | 4f14 5s2 5p6 5d2 6s2',
853     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d2',
854     'lo': '5s 5p',
855     'rmt': 2.3,
856     'lmax': '',
857     'jri': 981
858 },
859 73: {
860     'mass': 180.94788,
861     'name': 'Tantalum',
862     'symbol': 'Ta',
863     'econfig': '[Kr] 4d10 4f14 | 5s2 5p6 5d3 6s2',
864     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d3',
865     'lo': '5s 5p',
866     'rmt': 2.2,
867     'lmax': '',
868     'jri': 981
869 },
870 74: {
871     'mass': 183.84,
872     'name': 'Tungsten',
873     'symbol': 'W',
874     'econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d4',
875     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d4',
876     'lo': '5s 5p',
877     'rmt': 2.1,
878     'lmax': '',
879     'jri': 981
880 },
881 75: {
882     'mass': 186.207,
883     'name': 'Rhenium',
884     'symbol': 'Re',
885     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d5',

```

(continues on next page)

(continued from previous page)

```

886     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d5',
887     'lo': '5s 5p',
888     'rmt': 2.1,
889     'lmax': '',
890     'jri': 981
891 },
892 76: {
893     'mass': 190.23,
894     'name': 'Osmium',
895     'symbol': 'Os',
896     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d6',
897     'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d6',
898     'lo': '5p',
899     'rmt': 2.1,
900     'lmax': '',
901     'jri': 981
902 },
903 77: {
904     'mass': 192.217,
905     'name': 'Iridium',
906     'symbol': 'Ir',
907     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d7',
908     'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d7',
909     'lo': '5p',
910     'rmt': 2.1,
911     'lmax': '',
912     'jri': 981
913 },
914 78: {
915     'mass': 195.084,
916     'name': 'Platinum',
917     'symbol': 'Pt',
918     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d8',
919     'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d8',
920     'lo': '5p',
921     'rmt': 2.1,
922     'lmax': '',
923     'jri': 981
924 },
925 79: {
926     'mass': 196.966569,
927     'name': 'Gold',
928     'symbol': 'Au',
929     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d9',
930     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d9',
931     'lo': '5s 5p',
932     'rmt': 2.2,
933     'lmax': '',
934     'jri': 981
935 },
936 80: {
937     'mass': 200.59,

```

(continues on next page)

(continued from previous page)

```

938     'name': 'Mercury',
939     'symbol': 'Hg',
940     'econfig': '[Kr] 5s2 4d10 4f14 | 5p6 5d10 6s2',
941     'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 5p6 | 5d10 6s2',
942     'lo': '5d',
943     'rmt': 2.4,
944     'lmax': '',
945     'jri': 981
946 },
947 81: {
948     'mass': 204.3833,
949     'name': 'Thallium',
950     'symbol': 'Tl',
951     'econfig': '[Xe] 4f14 | 5d10 6s2 6p1',
952     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p1',
953     'lo': '5d',
954     'rmt': 2.4,
955     'lmax': '',
956     'jri': 981
957 },
958 82: {
959     'mass': 207.2,
960     'name': 'Lead',
961     'symbol': 'Pb',
962     'econfig': '[Xe] 4f14 | 5d10 6s2 6p2',
963     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p2',
964     'lo': '5d',
965     'rmt': 2.4,
966     'lmax': '',
967     'jri': 981
968 },
969 83: {
970     'mass': 208.9804,
971     'name': 'Bismuth',
972     'symbol': 'Bi',
973     'econfig': '[Xe] 4f14 | 5d10 6s2 6p3',
974     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p3',
975     'lo': '5d',
976     'rmt': 2.4,
977     'lmax': '',
978     'jri': 981
979 },
980 84: {
981     'mass': 209.0,
982     'name': 'Polonium',
983     'symbol': 'Po',
984     'econfig': '[Xe] 4f14 | 5d10 6s2 6p4',
985     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p4',
986     'lo': '5d',
987     'rmt': 2.2,
988     'lmax': '',
989     'jri': 981

```

(continues on next page)

(continued from previous page)

```

990 },
991 85: {
992     'mass': 210.0,
993     'name': 'Astatine',
994     'symbol': 'At',
995     'econfig': '[Xe] 4f14 | 5d10 6s2 6p5',
996     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p5',
997     'lo': '5d',
998     'rmt': 2.2,
999     'lmax': '',
1000     'jri': 981
1001 },
1002 86: {
1003     'mass': 222.0,
1004     'name': 'Radon',
1005     'symbol': 'Rn',
1006     'econfig': '[Xe] 4f14 | 5d10 6s2 6p6',
1007     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p6',
1008     'lo': '5d',
1009     'rmt': 2.2,
1010     'lmax': '',
1011     'jri': 981
1012 }, # TODO: after wards not righth
1013 87: {
1014     'mass': 223.0,
1015     'name': 'Francium',
1016     'symbol': 'Fr',
1017     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s1',
1018     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s1',
1019     'lo': '6s 6p',
1020     'rmt': 2.2,
1021     'lmax': '',
1022     'jri': 981
1023 },
1024 88: {
1025     'mass': 226.0,
1026     'name': 'Radium',
1027     'symbol': 'Ra',
1028     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2',
1029     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2',
1030     'lo': '6s 6p',
1031     'rmt': 2.2,
1032     'lmax': '',
1033     'jri': 981
1034 },
1035 89: {
1036     'mass': 227.0,
1037     'name': 'Actinium',
1038     'symbol': 'Ac',
1039     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 6d1',
1040     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1',
1041     'lo': '6s 6p',

```

(continues on next page)

(continued from previous page)

```

1042     'rmt': 2.2,
1043     'lmax': '',
1044     'jri': 981
1045 },
1046 90: {
1047     'mass': 232.03806,
1048     'name': 'Thorium',
1049     'symbol': 'Th',
1050     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 6d1 5f1',
1051     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1 5f1',
1052     'lo': '6s 6p',
1053     'rmt': 2.2,
1054     'lmax': '',
1055     'jri': 981
1056 },
1057 91: {
1058     'mass': 231.03588,
1059     'name': 'Protactinium',
1060     'symbol': 'Pa',
1061     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 6d1 5f2',
1062     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1 5f2',
1063     'lo': '6s 6p',
1064     'rmt': 2.2,
1065     'lmax': '',
1066     'jri': 981
1067 },
1068 92: {
1069     'mass': 238.02891,
1070     'name': 'Uranium',
1071     'symbol': 'U',
1072     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f4',
1073     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f4',
1074     'lo': '6s 6p',
1075     'rmt': 2.3,
1076     'lmax': '',
1077     'jri': 981
1078 },
1079 93: {
1080     'mass': 237.0,
1081     'name': 'Neptunium',
1082     'symbol': 'Np',
1083     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f5',
1084     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f5',
1085     'lo': '6s 6p',
1086     'rmt': 2.1,
1087     'lmax': '',
1088     'jri': 981
1089 },
1090 94: {
1091     'mass': 244.0,
1092     'name': 'Plutonium',
1093     'symbol': 'Pu',

```

(continues on next page)

(continued from previous page)

```

1094     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f6',
1095     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f6',
1096     'lo': '6s 6p',
1097     'rmt': 2.2,
1098     'lmax': '',
1099     'jri': 981
1100 },
1101 95: {
1102     'mass': 243.0,
1103     'name': 'Americium',
1104     'symbol': 'Am',
1105     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f7',
1106     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f7',
1107     'lo': '6s 6p',
1108     'rmt': 2.4,
1109     'lmax': '',
1110     'jri': 981
1111 },
1112 96: {
1113     'mass': 247.0,
1114     'name': 'Curium',
1115     'symbol': 'Cm',
1116     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f8',
1117     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f8',
1118     'lo': '6s 6p',
1119     'rmt': 2.4,
1120     'lmax': '',
1121     'jri': 981
1122 },
1123 97: {
1124     'mass': 247.0,
1125     'name': 'Berkelium',
1126     'symbol': 'Bk',
1127     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f9',
1128     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f9',
1129     'lo': '6s 6p',
1130     'rmt': 2.4,
1131     'lmax': '',
1132     'jri': 981
1133 },
1134 98: {
1135     'mass': 251.0,
1136     'name': 'Californium',
1137     'symbol': 'Cf',
1138     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f10',
1139     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f10',
1140     'lo': '6s 6p',
1141     'rmt': 2.4,
1142     'lmax': '',
1143     'jri': 981
1144 },
1145 99: {

```

(continues on next page)

(continued from previous page)

```

1146     'mass': 252.0,
1147     'name': 'Einsteinium',
1148     'symbol': 'Es',
1149     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f11',
1150     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f11',
1151     'lo': '6s 6p',
1152     'rmt': 2.4,
1153     'lmax': '',
1154     'jri': 981
1155 },
1156 100: {
1157     'mass': 257.0,
1158     'name': 'Fermium',
1159     'symbol': 'Fm',
1160     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f12',
1161     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f12',
1162     'lo': '6s 6p',
1163     'rmt': 2.4,
1164     'lmax': '',
1165     'jri': 981
1166 },
1167 101: {
1168     'mass': 258.0,
1169     'name': 'Mendelevium',
1170     'symbol': 'Md',
1171     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f13',
1172     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f13',
1173     'lo': '6s 6p',
1174     'rmt': 2.4,
1175     'lmax': '',
1176     'jri': 981
1177 },
1178 102: {
1179     'mass': 259.0,
1180     'name': 'Nobelium',
1181     'symbol': 'No',
1182     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14',
1183     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f14',
1184     'lo': '6s 6p',
1185     'rmt': 2.4,
1186     'lmax': '',
1187     'jri': 981
1188 },
1189 103: {
1190     'mass': 262.0,
1191     'name': 'Lawrencium',
1192     'symbol': 'Lr',
1193     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d1',
1194     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f14 6d1',
1195     'lo': '6s 6p 5f',
1196     'rmt': 2.4,
1197     'lmax': '',

```

(continues on next page)

(continued from previous page)

```

1198     'jri': 981
1199 },
1200 104: {
1201     'mass': 267.0,
1202     'name': 'Rutherfordium',
1203     'symbol': 'Rf',
1204     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d2',
1205     'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d2',
1206     'lo': '6p 5f',
1207     'rmt': 2.4,
1208     'lmax': '',
1209     'jri': 981
1210 },
1211 105: {
1212     'mass': 268.0,
1213     'name': 'Dubnium',
1214     'symbol': 'Db',
1215     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d3',
1216     'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 5f14 6d3',
1217     'lo': '6p 5f',
1218     'rmt': 2.4,
1219     'lmax': '',
1220     'jri': 981
1221 },
1222 106: {
1223     'mass': 271.0,
1224     'name': 'Seaborgium',
1225     'symbol': 'Sg',
1226     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d4',
1227     'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 5f14 6d4',
1228     'lo': '6p 5f',
1229     'rmt': 2.4,
1230     'lmax': '',
1231     'jri': 981
1232 },
1233 107: {
1234     'mass': 272.0,
1235     'name': 'Bohrium',
1236     'symbol': 'Bh',
1237     'econfig': '[Rn] 7s2 5f14 | 6d5',
1238     'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 6p6 5f14 | 7s2 6d5',
1239     'lo': '',
1240     'rmt': 2.4,
1241     'lmax': '',
1242     'jri': 981
1243 },
1244 108: {
1245     'mass': 270.0,
1246     'name': 'Hassium',
1247     'symbol': 'Hs',
1248     'econfig': '[Rn] 7s2 5f14 | 6d6',
1249     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d6',

```

(continues on next page)



(continued from previous page)

```

1250     'lo': '',
1251     'rmt': 2.4,
1252     'lmax': '',
1253     'jri': 981
1254 },
1255 109: {
1256     'mass': 276.0,
1257     'name': 'Meitnerium',
1258     'symbol': 'Mt',
1259     'econfig': '[Rn] 7s2 5f14 | 6d7',
1260     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d7',
1261     'lo': '',
1262     'rmt': 2.4,
1263     'lmax': '',
1264     'jri': 981
1265 },
1266 110: {
1267     'mass': 281.0,
1268     'name': 'Darmstadtium',
1269     'symbol': 'Ds',
1270     'econfig': '[Rn] 7s2 5f14 | 6d8',
1271     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d8',
1272     'lo': '',
1273     'rmt': 2.4,
1274     'lmax': '',
1275     'jri': 981
1276 },
1277 111: {
1278     'mass': 280.0,
1279     'name': 'Roentgenium',
1280     'symbol': 'Rg',
1281     'econfig': '[Rn] 7s2 5f14 | 6d9',
1282     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d9',
1283     'lo': '',
1284     'rmt': 2.4,
1285     'lmax': '',
1286     'jri': 981
1287 },
1288 112: {
1289     'mass': 285.0,
1290     'name': 'Copernicium',
1291     'symbol': 'Cn',
1292     'econfig': '[Rn] 7s2 5f14 | 6d10',
1293     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d10',
1294     'lo': '6d',
1295     'rmt': 2.4,
1296     'lmax': '',
1297     'jri': 981
1298 },
1299 113: {
1300     'mass': 285.0,
1301     'name': 'Nihomium',

```

(continues on next page)

(continued from previous page)

```

1302     'symbol': 'Nh',
1303     'econfig': '[Rn] 7s2 5f14 | 6d10 7p1',
1304     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p1',
1305     'lo': '6d',
1306     'rmt': 2.4,
1307     'lmax': '',
1308     'jri': 981
1309 },
1310 114: {
1311     'mass': 289.0,
1312     'name': 'Flerovium',
1313     'symbol': 'Fl',
1314     'econfig': '[Rn] 7s2 5f14 | 6d10 7p2',
1315     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p2',
1316     'lo': '6d',
1317     'rmt': 2.4,
1318     'lmax': '',
1319     'jri': 981
1320 },
1321 115: {
1322     'mass': 0.0,
1323     'name': 'Moscovium',
1324     'symbol': 'Mc',
1325     'econfig': '[Rn] 7s2 5f14 | 6d10 7p3',
1326     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p3',
1327     'lo': '6d',
1328     'rmt': 2.4,
1329     'lmax': '',
1330     'jri': 981
1331 },
1332 116: {
1333     'mass': 293.0,
1334     'name': 'Livermorium',
1335     'symbol': 'Lv',
1336     'econfig': '[Rn] 7s2 5f14 | 6d10 7p4',
1337     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p4',
1338     'lo': '6d',
1339     'rmt': 2.4,
1340     'lmax': '',
1341     'jri': 981
1342 },
1343 117: {
1344     'mass': 0.0,
1345     'name': 'Tennessine',
1346     'symbol': 'Ts',
1347     'econfig': '[Rn] 7s2 5f14 | 6d10 7p5',
1348     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p5',
1349     'lo': '6d',
1350     'rmt': 2.4,
1351     'lmax': '',
1352     'jri': 981
1353 },

```

(continues on next page)

(continued from previous page)

```
1354 118: {  
1355     'mass': 0.0,  
1356     'name': 'Oganesson',  
1357     'symbol': 'Og',  
1358     'econfig': '[Rn] 7s2 5f14 | 6d10 7p6',  
1359     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p6',  
1360     'lo': '6d',  
1361     'rmt': 2.4,  
1362     'lmax': '',  
1363     'jri': 981  
1364 }  
1365 }
```



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### m

`masci_tools.io.common_functions`, 195  
`masci_tools.io.fleur_inpgen`, 115  
`masci_tools.io.fleurxmlmodifier`, 117  
`masci_tools.io.hdf5_util`, 197  
`masci_tools.io.io_fleurxml`, 129  
`masci_tools.io.io_nmmpmat`, 130  
`masci_tools.io.kkr_params`, 110  
`masci_tools.io.kkr_read_shapefun_info`, 112  
`masci_tools.io.parsers.fleur`, 114  
`masci_tools.io.parsers.fleur.default_parse_tasks`,  
140  
`masci_tools.io.parsers.fleur.fleur_schema`,  
212  
`masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`,  
217  
`masci_tools.io.parsers.fleur.outxml_conversions`,  
210  
`masci_tools.io.parsers.fleur.task_migrations`,  
152  
`masci_tools.io.parsers.hdf5.reader`, 132  
`masci_tools.io.parsers.hdf5.recipes`, 133  
`masci_tools.io.parsers.hdf5.transforms`, 134  
`masci_tools.io.parsers.kkrimp_parser_functions`,  
113  
`masci_tools.io.parsers.kkrparser_functions`,  
112  
`masci_tools.io.parsers.voroparser_functions`,  
113  
`masci_tools.tools.cf_calculation`, 103  
`masci_tools.tools.greensf_calculations`, 109  
`masci_tools.tools.greensfunction`, 105  
`masci_tools.util.case_insensitive_dict`, 163  
`masci_tools.util.constants`, 220  
`masci_tools.util.econfig`, 211  
`masci_tools.util.fleur_calculate_expression`,  
206  
`masci_tools.util.lockable_containers`, 161  
`masci_tools.util.logging_util`, 198  
`masci_tools.util.parse_tasks`, 207  
`masci_tools.util.parse_tasks_decorators`, 209  
`masci_tools.util.schema_dict_util`, 199  
`masci_tools.util.xml.common_functions`, 164  
`masci_tools.util.xml.converters`, 166  
`masci_tools.util.xml.xml_getters`, 190  
`masci_tools.util.xml.xml_setters_basic`, 188  
`masci_tools.util.xml.xml_setters_names`, 169  
`masci_tools.util.xml.xml_setters_nmmpmat`, 181  
`masci_tools.util.xml.xml_setters_xpaths`, 183  
`masci_tools.vis.bokeh_plots`, 94  
`masci_tools.vis.bokeh_plotter`, 91  
`masci_tools.vis.common`, 69  
`masci_tools.vis.data`, 64  
`masci_tools.vis.fleur`, 57  
`masci_tools.vis.kkr_plot_bandstruc_qdos`, 59  
`masci_tools.vis.kkr_plot_dos`, 59  
`masci_tools.vis.kkr_plot_FS_qdos`, 59  
`masci_tools.vis.kkr_plot_shapefun`, 59  
`masci_tools.vis.matplotlib_plotter`, 72  
`masci_tools.vis.parameters`, 60  
`masci_tools.vis.plot_methods`, 76





## Symbols

```
--atoms <atoms>
    masci_tools-plot-fleur-dos command line
    option, 161
--backend <backend>
    masci_tools-plot-fleur-bands command
    line option, 160
    masci_tools-plot-fleur-dos command line
    option, 161
--contains <contains>
    masci_tools-parse-all-attrs command
    line option, 154
    masci_tools-parse-attr command line
    option, 155
    masci_tools-parse-number-nodes command
    line option, 157
    masci_tools-parse-parent-attrs
    command line option, 158
    masci_tools-parse-tag-exists command
    line option, 159
    masci_tools-parse-text command line
    option, 160
--interstitial <interstitial>
    masci_tools-plot-fleur-dos command line
    option, 161
--l_resolved <l_resolved>
    masci_tools-plot-fleur-dos command line
    option, 161
--name <name>
    masci_tools-parse-all-attrs command
    line option, 154
    masci_tools-parse-attr command line
    option, 155
    masci_tools-parse-number-nodes command
    line option, 157
    masci_tools-parse-parent-attrs
    command line option, 158
    masci_tools-parse-tag-exists command
    line option, 159
    masci_tools-parse-text command line
    option, 160
--not-contains <not_contains>
    masci_tools-parse-all-attrs command
    line option, 154
    masci_tools-parse-attr command line
    option, 155
    masci_tools-parse-number-nodes command
    line option, 157
    masci_tools-parse-parent-attrs
    command line option, 158
    masci_tools-parse-tag-exists command
    line option, 159
    masci_tools-parse-text command line
    option, 160
--overwrite
    masci_tools-fleur-schema-add command
    line option, 153
--recipe <recipe>
    masci_tools-plot-fleur-bands command
    line option, 160
    masci_tools-plot-fleur-dos command line
    option, 161
--save
    masci_tools-plot-fleur-bands command
    line option, 160
    masci_tools-plot-fleur-dos command line
    option, 161
--show
    masci_tools-plot-fleur-bands command
    line option, 160
    masci_tools-plot-fleur-dos command line
    option, 161
--subtags
    masci_tools-parse-all-attrs command
    line option, 154
--tag <tag>
    masci_tools-parse-attr command line
    option, 155
--test-xml-file <test_xml_file>
    masci_tools-fleur-schema-add command
    line option, 153
--text
    masci_tools-parse-all-attrs command
    line option, 154
```

```
--total <total>
    masci_tools-plot-fleur-dos command line
    option, 161
--version
    masci_tools command line option, 153
--weight <weight>
    masci_tools-plot-fleur-bands command
    line option, 160
-c
    masci_tools-parse-all-attrs command
    line option, 154
    masci_tools-parse-attr command line
    option, 155
    masci_tools-parse-number-nodes command
    line option, 157
    masci_tools-parse-parent-attrs
    command line option, 158
    masci_tools-parse-tag-exists command
    line option, 159
    masci_tools-parse-text command line
    option, 160
-n
    masci_tools-parse-all-attrs command
    line option, 154
    masci_tools-parse-attr command line
    option, 155
    masci_tools-parse-number-nodes command
    line option, 157
    masci_tools-parse-parent-attrs
    command line option, 158
    masci_tools-parse-tag-exists command
    line option, 159
    masci_tools-parse-text command line
    option, 160
-nc
    masci_tools-parse-all-attrs command
    line option, 154
    masci_tools-parse-attr command line
    option, 155
    masci_tools-parse-number-nodes command
    line option, 157
    masci_tools-parse-parent-attrs
    command line option, 158
    masci_tools-parse-tag-exists command
    line option, 159
    masci_tools-parse-text command line
    option, 160
-r
    masci_tools-plot-fleur-bands command
    line option, 160
    masci_tools-plot-fleur-dos command line
    option, 161
-t
    masci_tools-parse-attr command line
```

```
option, 155
-v
    masci_tools command line option, 153
-w
    masci_tools-plot-fleur-bands command
    line option, 160
```

## A

```
abs_to_rel() (in module
    masci_tools.io.common_functions), 195
abs_to_rel_f() (in module
    masci_tools.io.common_functions), 195
abs_to_rel_xpath() (in module
    masci_tools.util.xml.common_functions),
    164
add_data_key() (masci_tools.vis.data.PlotData
    method), 64
add_number_to_attr() (in module
    masci_tools.util.xml.xml_setters_names),
    169
add_number_to_attr()
    (masci_tools.io.fleurxmlmodifier.FleurXMLModifier
    method), 117
add_number_to_first_attr() (in module
    masci_tools.util.xml.xml_setters_names),
    170
add_number_to_first_attr()
    (masci_tools.io.fleurxmlmodifier.FleurXMLModifier
    method), 118
add_parameter() (masci_tools.vis.parameters.Plotter
    method), 61
add_partial_sums() (in module
    masci_tools.io.parsers.hdf5.transforms),
    134
add_partial_sums_fixed() (in module
    masci_tools.io.parsers.hdf5.transforms),
    134
add_task() (masci_tools.util.parse_tasks.ParseTasks
    method), 207
add_task_list() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier
    method), 118
add_tooltips() (masci_tools.vis.bokeh_plotter.BokehPlotter
    method), 93
all_attrs_function
    (masci_tools.util.parse_tasks.ParseTasks
    property), 208
angles_to_vec() (in module
    masci_tools.io.common_functions), 195
append() (masci_tools.util.lockable_containers.LockableList
    method), 162
apply() (masci_tools.vis.data.PlotData method), 65
apply_lambda() (in module
    masci_tools.io.parsers.hdf5.transforms),
    135
```

[apply\\_modifications\(\)](#) (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* class method), 118  
[args](#) (*masci\_tools.io.fleurxmlmodifier.ModifierTask* attribute), 127  
[args](#) (*masci\_tools.io.parsers.hdf5.reader.AttribTransformation* attribute), 132  
[args](#) (*masci\_tools.io.parsers.hdf5.reader.Transformation* attribute), 132  
[asymmetric\\_lorentz\(\)](#) (in module *masci\_tools.vis.plot\_methods*), 77  
[asymmetric\\_lorentz\\_gauss\\_conv\(\)](#) (in module *masci\_tools.vis.plot\_methods*), 77  
[asymmetric\\_lorentz\\_gauss\\_sum\(\)](#) (in module *masci\_tools.vis.plot\_methods*), 77  
[atomDiff](#) (*masci\_tools.tools.greensfunction.GreensfElement* attribute), 107  
[AtomSiteProperties](#) (class in *masci\_tools.io.common\_functions*), 195  
[atomType](#) (*masci\_tools.tools.greensfunction.GreensfElement* attribute), 107  
[atomTypep](#) (*masci\_tools.tools.greensfunction.GreensfElement* attribute), 107  
[attrib\\_exists\(\)](#) (in module *masci\_tools.util.schema\_dict\_util*), 199  
[attrib\\_name](#) (*masci\_tools.io.parsers.hdf5.reader.AttribTransformation* attribute), 132  
[attrib\\_xpath\(\)](#) (*masci\_tools.io.parsers.fleur.fleur\_schema.SchemaDictExpressionDict* method), 127  
[attrib\\_xpath\(\)](#) (*masci\_tools.io.parsers.fleur.fleur\_schema.SchemaDict* method), 214  
[AttribTransformation](#) (class in *masci\_tools.io.parsers.hdf5.reader*), 132  
[attributes\(\)](#) (in module *masci\_tools.io.parsers.hdf5.transforms*), 135  
[AttributeType](#) (class in *masci\_tools.io.parsers.fleur.fleur\_schema*), 212  
[AttributeType](#) (class in *masci\_tools.io.parsers.fleur.fleur\_schema.fleur\_schema\_parser.functions*), 217  
**B**  
[BANDDOS\\_FILE](#)  
     *masci\_tools-plot-fleur-bands* command line option, 160  
     *masci\_tools-plot-fleur-dos* command line option, 161  
[bands\(\)](#) (in module *masci\_tools.vis.common*), 69  
[bands\\_recipe\\_format\(\)](#) (in module *masci\_tools.io.parsers.hdf5.recipes*), 133  
[barchart\(\)](#) (in module *masci\_tools.vis.plot\_methods*), 77  
[base\\_type](#) (*masci\_tools.io.parsers.fleur.fleur\_schema.AttributeType* attribute), 212  
[base\\_type](#) (*masci\_tools.io.parsers.fleur.fleur\_schema.fleur\_schema\_parser.attribute*), 217  
[bokeh\\_bands\(\)](#) (in module *masci\_tools.vis.bokeh\_plots*), 94  
[bokeh\\_dos\(\)](#) (in module *masci\_tools.vis.bokeh\_plots*), 95  
[bokeh\\_line\(\)](#) (in module *masci\_tools.vis.bokeh\_plots*), 96  
[bokeh\\_multi\\_scatter\(\)](#) (in module *masci\_tools.vis.bokeh\_plots*), 96  
[bokeh\\_scatter\(\)](#) (in module *masci\_tools.vis.bokeh\_plots*), 97  
[bokeh\\_spectral\\_function\(\)](#) (in module *masci\_tools.vis.bokeh\_plots*), 97  
[bokeh\\_spinpol\\_bands\(\)](#) (in module *masci\_tools.vis.bokeh\_plots*), 98  
[bokeh\\_spinpol\\_dos\(\)](#) (in module *masci\_tools.vis.bokeh\_plots*), 98  
[BokehPlotter](#) (class in *masci\_tools.vis.bokeh\_plotter*), 91  
**C**  
[calculate\\_expression\(\)](#) (in module *masci\_tools.util.fleur\_calculate\_expression*), 206  
[calculate\\_expression\\_partial\(\)](#) (in module *masci\_tools.util.fleur\_calculate\_expression*), 206  
[calculate\\_heisenberg\\_j0\(\)](#) (in module *masci\_tools.tools.greensf\_calculations*), 109  
[calculate\\_heisenberg\\_jij\(\)](#) (in module *masci\_tools.tools.greensf\_calculations*), 110  
[calculate\\_hybridization\(\)](#) (in module *masci\_tools.tools.greensf\_calculations*), 110  
[calculate\\_norm\(\)](#) (in module *masci\_tools.io.parsers.hdf5.transforms*), 135  
[calculate\\_total\\_magnetic\\_moment\(\)](#) (in module *masci\_tools.io.parsers.fleur.outxml\_conversions*), 210  
[calculate\\_walltime\(\)](#) (in module *masci\_tools.io.parsers.fleur.outxml\_conversions*), 210  
[camel\\_to\\_snake\(\)](#) (in module *masci\_tools.io.common\_functions*), 195  
[CaseInsensitiveDict](#) (class in *masci\_tools.util.case\_insensitive\_dict*), 163  
[CaseInsensitiveFrozenSet](#) (class in *masci\_tools.util.case\_insensitive\_dict*), 163  
[CDF\\_voigt\\_profile\(\)](#) (in module *masci\_tools.vis.plot\_methods*), 77

CFCalculation	(class in <i>masci_tools.tools.cf_calculation</i> ), 103	convert_forces()	(in module <i>masci_tools.io.parsers.fleur.outxml_conversions</i> ), 210
CFCoefficient	(class in <i>masci_tools.tools.cf_calculation</i> ), 104	convert_from_fortran_bool()	(in module <i>masci_tools.util.xml.converters</i> ), 166
change_XC_val_kkrimp()	( <i>masci_tools.io.kkr_params.kkrparams</i> method), 111	convert_from_xml()	(in module <i>masci_tools.util.xml.converters</i> ), 166
change_zoom()	(in module <i>masci_tools.vis.kkr_plot_shapefun</i> ), 59	convert_from_xml_explicit()	(in module <i>masci_tools.util.xml.converters</i> ), 167
changes()	( <i>masci_tools.io.fleurxmlmodifier.FleurXMLModifier</i> method), 119	convert_from_xml_single_values()	(in module <i>masci_tools.util.xml.converters</i> ), 167
check_complex_xpath()	(in module <i>masci_tools.util.xml.common_functions</i> ), 164	convert_ldau_definitions()	(in module <i>masci_tools.io.parsers.fleur.outxml_conversions</i> ), 210
check_error_category()	(in module <i>masci_tools.io.parsers.kkrparser_functions</i> ), 112	convert_relax_info()	(in module <i>masci_tools.io.parsers.fleur.outxml_conversions</i> ), 211
check_voronoi_output()	(in module <i>masci_tools.io.parsers.voroparser_functions</i> ), 113	convert_str_version_number()	(in module <i>masci_tools.util.xml.converters</i> ), 167
clear()	( <i>masci_tools.util.lockable_containers.LockableList</i> method), 162	convert_to_complete_list()	( <i>masci_tools.vis.parameters.Plotter</i> static method), 62
clear_cache()	( <i>masci_tools.io.parsers.fleur.fleur_schema.SchemaDict</i> class method), 128	convert_to_complex_array()	(in module <i>masci_tools.io.parsers.hdf5.transforms</i> ), 135
clear_cache()	( <i>masci_tools.io.parsers.fleur.fleur_schema.SchemaDict</i> class method), 215	convert_to_fortran()	(in module <i>masci_tools.io.common_functions</i> ), 195
clear_xml()	(in module <i>masci_tools.util.xml.common_functions</i> ), 164	convert_to_fortran_bool()	(in module <i>masci_tools.util.xml.converters</i> ), 168
colormesh_plot()	(in module <i>masci_tools.vis.plot_methods</i> ), 78	convert_to_fortran_string()	(in module <i>masci_tools.io.common_functions</i> ), 196
colors	(class in <i>masci_tools.tools.greensfunction</i> ), 107	convert_to_pystd()	(in module <i>masci_tools.io.common_functions</i> ), 196
ColumnDataSourceWrapper	(class in <i>masci_tools.vis.data</i> ), 64	convert_to_str()	(in module <i>masci_tools.io.parsers.hdf5.transforms</i> ), 135
construct_corelevel_spectrum()	(in module <i>masci_tools.vis.plot_methods</i> ), 78	convert_to_xml()	(in module <i>masci_tools.util.xml.converters</i> ), 168
contour	( <i>masci_tools.tools.greensfunction.GreensfElement</i> attribute), 107	convert_to_xml_explicit()	(in module <i>masci_tools.util.xml.converters</i> ), 168
conv_to_fortran()	(in module <i>masci_tools.io.fleur_inpgen</i> ), 115	convert_to_xml_single_values()	(in module <i>masci_tools.util.xml.converters</i> ), 168
convention	( <i>masci_tools.tools.cf_calculation.CFCoefficient</i> attribute), 104	convert_total_energy()	(in module <i>masci_tools.io.parsers.fleur.outxml_conversions</i> ), 211
conversion_function()	(in module <i>masci_tools.util.parse_tasks_decorators</i> ), 209	copy_data()	( <i>masci_tools.vis.data.PlotData</i> method), 65
conversion_functions	( <i>masci_tools.util.parse_tasks.ParseTasks</i> property), 208	create_tag()	(in module <i>masci_tools.util.xml.xml_setters_names</i> ), 170
convert_fleur_config_to_econfig()	(in module <i>masci_tools.util.econfig</i> ), 211	create_tag()	( <i>masci_tools.io.fleurxmlmodifier.FleurXMLModifier</i> method), 119
convert_fleur_electronconfig()	(in module <i>masci_tools.util.xml.converters</i> ), 166	cumulative_sum()	(in module <i>masci_tools.io.parsers.hdf5.transforms</i> ),
convert_fleur_lo()	(in module <i>masci_tools.util.xml.converters</i> ), 166		

135

## D

`data_keys` (*masci\_tools.vis.data.PlotData* property), 65

`default()` (*masci\_tools.vis.common.PlotBackend* static method), 69

`default_histogram()` (in module *masci\_tools.vis.plot\_methods*), 78

`delete_att()` (in module *masci\_tools.util.xml.xml\_setters\_names*), 171

`delete_att()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 119

`delete_tag()` (in module *masci\_tools.util.xml.xml\_setters\_names*), 171

`delete_tag()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 119

`determine_tasks()` (*masci\_tools.util.parse\_tasks.ParseTasks* method), 208

`dict_of_lists_to_list_of_dicts()` (*masci\_tools.vis.parameters.Plotter* static method), 62

`DictHandler` (class in *masci\_tools.util.logging\_util*), 198

`difference()` (*masci\_tools.util.case\_insensitive\_dict.CaseInsensitiveDict* method), 163

`dispersionplot()` (in module *masci\_tools.vis.kkr\_plot\_bandstruc\_qdos*), 59

`distinct_datasets()` (*masci\_tools.vis.data.PlotData* method), 65

`doniach_sunjic()` (in module *masci\_tools.vis.plot\_methods*), 78

`dos()` (in module *masci\_tools.vis.common*), 70

`dos_recipe_format()` (in module *masci\_tools.io.parsers.hdf5.recipes*), 134

`dosplot()` (in module *masci\_tools.vis.kkr\_plot\_dos*), 59

`draw_lines()` (*masci\_tools.vis.matplotlib\_plotter.MatplotlibPlotter* method), 75

`draw_straight_lines()` (*masci\_tools.vis.bokeh\_plotter.BokehPlotter* method), 93

## E

`emit()` (*masci\_tools.util.logging\_util.DictHandler* method), 198

`energy_dependence()` (*masci\_tools.tools.greensfunction.GreensFunction* method), 106

`ensure_plotter_consistency()` (in module *masci\_tools.vis.parameters*), 63

`eval_simple_xpath()` (in module *masci\_tools.util.schema\_dict\_util*), 199

`eval_xpath()` (in module *masci\_tools.util.xml.common\_functions*), 164

`eval_xpath_create()` (in module *masci\_tools.util.xml.xml\_setters\_xpaths*), 183

`evaluate_attribute()` (in module *masci\_tools.util.schema\_dict\_util*), 199

`evaluate_bracket()` (in module *masci\_tools.util.fleur\_calculate\_expression*), 206

`evaluate_parent_tag()` (in module *masci\_tools.util.schema\_dict\_util*), 200

`evaluate_single_value_tag()` (in module *masci\_tools.util.schema\_dict\_util*), 201

`evaluate_tag()` (in module *masci\_tools.util.schema\_dict\_util*), 201

`evaluate_text()` (in module *masci\_tools.util.schema\_dict\_util*), 202

`expand_parameters()` (*masci\_tools.vis.parameters.Plotter* method), 62

`extend()` (*masci\_tools.util.lockable\_containers.LockableList* method), 162

`extract_attribute_types()` (in module *masci\_tools.io.parsers.fleur.fleur\_schema.fleur\_schema\_parser\_fleur*), 217

`extract_text_types()` (in module *masci\_tools.io.parsers.fleur.fleur\_schema.fleur\_schema\_parser\_fleur*), 217

## F

`fac()` (in module *masci\_tools.io.common\_functions*), 196

`fill_keywords_to_inputfile()` (*masci\_tools.io.kkr\_params.kkrparams* method), 111

`filter_out_empty_dict_entries()` (in module *masci\_tools.io.common\_functions*), 196

`find_migration()` (in module *masci\_tools.util.parse\_tasks*), 209

`find_symmetry_relation()` (in module *masci\_tools.io.common\_functions*), 196

`flatten_array()` (in module *masci\_tools.io.parsers.hdf5.transforms*), 136

`FleurXMLModifier` (class in *masci\_tools.io.fleurxmlmodifier*), 117

`format_nmmpmat()` (in module *masci\_tools.io.io\_nmmpmat*), 130

`freeze()` (*masci\_tools.util.lockable\_containers.LockableDict* method), 162

`freeze()` (*masci\_tools.util.lockable\_containers.LockableList* method), 162



`from_str()` (*masci\_tools.vis.common.PlotBackend* static method), 69  
`fromFile()` (*masci\_tools.tools.greensfunction.GreensFunction* class method), 106  
`fromList()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* class method), 120  
`fromPath()` (*masci\_tools.io.parsers.fleur.fleur\_schema.InputSchemaDict* class method), 212  
`fromPath()` (*masci\_tools.io.parsers.fleur.fleur\_schema.OutputSchemaDict* class method), 213  
`fromVersion()` (*masci\_tools.io.parsers.fleur.fleur\_schema.InputSchemaDict* class method), 212  
`fromVersion()` (*masci\_tools.io.parsers.fleur.fleur\_schema.OutputSchemaDict* class method), 214  
`FSqdos2D()` (in module *masci\_tools.vis.kkr\_plot\_FS\_qdos*), 59

**G**

`gauss_one()` (in module *masci\_tools.vis.plot\_methods*), 78  
`gaussian()` (in module *masci\_tools.vis.plot\_methods*), 78  
`general_tasks` (*masci\_tools.util.parse\_tasks.ParseTasks* property), 208  
`get_all_child_datasets()` (in module *masci\_tools.io.parsers.hdf5.transforms*), 136  
`get_all_mandatory()` (*masci\_tools.io.kkr\_params.kkrparams* method), 111  
`get_attrib_xpath()` (in module *masci\_tools.util.schema\_dict\_util*), 202  
`get_attribute()` (in module *masci\_tools.io.parsers.hdf5.transforms*), 136  
`get_avail_actions()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 120  
`get_basic_types()` (in module *masci\_tools.io.parsers.fleur.fleur\_schema.fleur\_schema\_parser\_functions*), 217  
`get_bokeh_help()` (in module *masci\_tools.vis.bokeh\_plots*), 99  
`get_cell()` (in module *masci\_tools.util.xml.xml\_getters*), 190  
`get_coefficient()` (*masci\_tools.tools.greensfunction.GreensFunction* method), 106  
`get_coreconfig()` (in module *masci\_tools.util.econfig*), 211  
`get_corestates_from_potential()` (in module *masci\_tools.io.common\_functions*), 196  
`get_description()` (*masci\_tools.io.kkr\_params.kkrparams* method), 111  
`get_description()` (*masci\_tools.vis.parameters.Plotter* method), 62  
`get_dict()` (*masci\_tools.io.kkr\_params.kkrparams* method), 111  
`get_dict()` (*masci\_tools.vis.parameters.Plotter* method), 62  
`get_econfig()` (in module *masci\_tools.util.econfig*), 211  
`get_from_potfile()` (in module *masci\_tools.io.common\_functions*), 196  
`get_hdf5_element()` (in module *masci\_tools.io.parsers.hdf5.transforms*), 216  
`get_schemaDict` (*masci\_tools.io.parsers.hdf5.transforms* method), 216  
`get_first_number()` (in module *masci\_tools.util.fleur\_calculate\_expression*), 206  
`get_first_string()` (in module *masci\_tools.util.fleur\_calculate\_expression*), 207  
`get_fleur_bands_specific_weights()` (in module *masci\_tools.io.parsers.hdf5.recipes*), 134  
`get_fleur_modes()` (in module *masci\_tools.util.xml.xml\_getters*), 190  
`get_function_result()` (*masci\_tools.vis.data.PlotData* method), 65  
`get_help()` (in module *masci\_tools.vis.common*), 70  
`get_highest_core_state()` (in module *masci\_tools.io.common\_functions*), 196  
`get_input_data_text()` (in module *masci\_tools.io.fleur\_inpgen*), 115  
`get_input_tag()` (in module *masci\_tools.io.parsers.fleur.fleur\_schema.fleur\_schema\_parser\_functions*), 218  
`get_keys()` (*masci\_tools.vis.data.PlotData* method), 65  
`get_KKRcalc_parameter_defaults()` (*masci\_tools.io.kkr\_params.kkrparams* class method), 111  
`get_kmeshinfo()` (in module *masci\_tools.io.parsers.kkrparser\_functions*), 112  
`get_kpoints_data()` (in module *masci\_tools.util.xml.xml\_getters*), 191  
`get_kpoints_data_max4()` (in module *masci\_tools.util.xml.xml\_getters*), 191  
`get_lattice_vectors()` (in module *masci\_tools.io.parsers.kkrparser\_functions*), 112  
`get_mask()` (*masci\_tools.vis.data.PlotData* method), 65  
`get_missing_keys()` (*masci\_tools.io.kkr\_params.kkrparams* method), 111  
`get_mpl_help()` (in module *masci\_tools.vis.plot\_methods*), 78  
`get_multiple_kwargs()`

`(masci_tools.vis.parameters.Plotter method), 62`  
`get_name()` (in module `masci_tools.io.parsers.hdf5.transforms`), 136  
`get_natom()` (in module `masci_tools.io.parsers.kkrparser_functions`), 112  
`get_nkpts()` (in module `masci_tools.util.xml.xml_getters`), 192  
`get_nkpts_max4()` (in module `masci_tools.util.xml.xml_getters`), 192  
`get_noco_rms()` (in module `masci_tools.io.parsers.kkrparser_functions`), 112  
`get_nspin()` (in module `masci_tools.io.parsers.kkrparser_functions`), 112  
`get_number_of_nodes()` (in module `masci_tools.util.schema_dict_util`), 203  
`get_omittable_tags()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`), 218  
`get_orbmom()` (in module `masci_tools.io.parsers.kkrparser_functions`), 112  
`get_other_attribs()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`), 218  
`get_parameter_data()` (in module `masci_tools.util.xml.xml_getters`), 192  
`get_plotter()` (in module `masci_tools.vis.common`), 70  
`get_relative_attrib_xpath()` (in module `masci_tools.util.schema_dict_util`), 203  
`get_relative_tag_xpath()` (in module `masci_tools.util.schema_dict_util`), 204  
`get_relaxation_information()` (in module `masci_tools.util.xml.xml_getters`), 193  
`get_relaxation_information_pre029()` (in module `masci_tools.util.xml.xml_getters`), 193  
`get_rms()` (in module `masci_tools.io.parsers.kkrparser_functions`), 113  
`get_root_tag()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`), 218  
`get_set_values()` (`masci_tools.io.kkr_params.kkrparams GreensFunction` method), 111  
`get_shape()` (in module `masci_tools.io.parsers.hdf5.transforms`), 136  
`get_single_particle_energies()` (in module `masci_tools.io.parsers.kkrparser_functions`), 113  
`get_spinmom_per_atom()` (in module `masci_tools.io.parsers.kkrparser_functions`), 113  
`get_structure_data()` (in module `masci_tools.util.xml.xml_getters`), 193  
`get_symmetry_information()` (in module `masci_tools.util.xml.xml_getters`), 194  
`get_tag_info()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`), 218  
`get_tag_info()` (in module `masci_tools.util.schema_dict_util`), 204  
`get_tag_paths()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`), 219  
`get_tag_xpath()` (in module `masci_tools.util.schema_dict_util`), 205  
`get_text_tags()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`), 219  
`get_type()` (`masci_tools.io.kkr_params.kkrparams GreensFunction` method), 111  
`get_unique_attribs()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`), 219  
`get_unique_path_attribs()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_functions`), 220  
`get_unlocked()` (`masci_tools.util.lockable_containers.LockableDict` method), 162  
`get_unlocked()` (`masci_tools.util.lockable_containers.LockableList` method), 162  
`get_valence_min()` (in module `masci_tools.io.parsers.voroparser_functions`), 113  
`get_value()` (`masci_tools.io.kkr_params.kkrparams` method), 111  
`get_values()` (`masci_tools.vis.data.PlotData` method), 66  
`get_wigner_matrix()` (in module `masci_tools.io.common_functions`), 196  
`get_xml_attribute()` (in module `masci_tools.util.xml.common_functions`), 165  
`GreensElement` (class in `masci_tools.tools.greensfunction`), 107  
`GreensFunction` (class in `masci_tools.tools.greensfunction`), 105  
`group_data()` (`masci_tools.vis.data.PlotData` method), 66

## H

`h5dump()` (in module `masci_tools.io.hdf5_util`), 197  
`hdf5_transformation()` (in module

`masci_tools.io.parsers.hdf5.transforms`),  
136  
HDF5Reader (class in `masci_tools.io.parsers.hdf5.reader`),  
132  
HDF5TransformationError, 134  
hdfList() (in module `masci_tools.io.hdf5_util`), 198  
histogram() (in module `masci_tools.vis.plot_methods`),  
79  
hyp2f2() (in module `masci_tools.vis.plot_methods`), 79

## I

index\_dataset() (in module  
`masci_tools.io.parsers.hdf5.transforms`),  
137  
inp\_version(`masci_tools.io.parsers.fleur.fleur_schema.InputSchemaDict`  
property), 213  
inp\_version(`masci_tools.io.parsers.fleur.fleur_schema.OutputSchemaDict`  
property), 214  
InputSchemaDict (class in  
`masci_tools.io.parsers.fleur.fleur_schema`),  
212  
inpxml\_parser() (in module  
`masci_tools.io.parsers.fleur`), 114  
insert() (`masci_tools.util.lockable_containers.LockableList`  
method), 162  
interpolate\_dos() (in module  
`masci_tools.io.common_functions`), 196  
intersection() (`masci_tools.util.case_insensitive_dict.CaseInsensitiveDict`  
method), 163  
intersite\_shell\_indices() (in module  
`masci_tools.tools.greensfunction`), 108  
intersite\_shells() (in module  
`masci_tools.tools.greensfunction`), 108  
intersite\_shells\_from\_file() (in module  
`masci_tools.tools.greensfunction`), 108  
is\_general() (`masci_tools.vis.parameters.Plotter`  
method), 62  
is\_mandatory() (`masci_tools.io.kkr_params.kkrparams`  
method), 111  
is\_sequence() (in module  
`masci_tools.io.common_functions`), 197  
isdisjoint() (`masci_tools.util.case_insensitive_dict.CaseInsensitiveDict`  
method), 163  
issubset() (`masci_tools.util.case_insensitive_dict.CaseInsensitiveDict`  
method), 163  
issuperset() (`masci_tools.util.case_insensitive_dict.CaseInsensitiveDict`  
method), 163  
items() (`masci_tools.io.kkr_params.kkrparams`  
method), 111  
items() (`masci_tools.vis.data.PlotData` method), 66  
iteration\_tasks(`masci_tools.util.parse_tasks.ParseTasks`  
property), 208

## K

keys() (`masci_tools.vis.data.PlotData` method), 66  
kind(`masci_tools.io.common_functions.AtomSiteProperties`  
attribute), 195  
KkrimpParserFunctions (class in  
`masci_tools.io.parsers.kkrimp_parser_functions`),  
113  
kkrparams (class in `masci_tools.io.kkr_params`), 110  
kresolved(`masci_tools.tools.greensfunction.GreensfElement`  
attribute), 107  
kwargs (`masci_tools.io.fleurxmlmodifier.ModifierTask`  
attribute), 127  
kwargs(`masci_tools.io.parsers.hdf5.reader.AttribTransformation`  
attribute), 132  
kwargs(`masci_tools.io.parsers.hdf5.reader.Transformation`  
attribute), 133

## L

1 (`masci_tools.tools.cf_calculation.CFCoefficient` at-  
tribute), 104  
1 (`masci_tools.tools.greensfunction.GreensfElement` at-  
tribute), 107  
length(`masci_tools.io.parsers.fleur.fleur_schema.AttributeType`  
attribute), 212  
length(`masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser_fu`  
attribute), 217  
line() (in module `masci_tools.vis.common`), 70  
listElements() (in module  
`masci_tools.tools.greensfunction`), 108  
load\_bokeh\_defaults() (in module  
`masci_tools.vis.bokeh_plots`), 99  
load\_defaults() (in module `masci_tools.vis.common`),  
70  
load\_defaults() (`masci_tools.vis.parameters.Plotter`  
method), 63  
load\_inpxml() (in module `masci_tools.io.io_fleurxml`),  
129  
load\_mpl\_defaults() (in module  
`masci_tools.vis.plot_methods`), 79  
load\_outxml() (in module `masci_tools.io.io_fleurxml`),  
130  
LockableDict (class in  
`masci_tools.util.lockable_containers`), 161  
LockableList (class in  
`masci_tools.util.lockable_containers`), 162  
LockableList() (in module  
`masci_tools.util.lockable_containers`), 161  
locked(`masci_tools.util.lockable_containers.LockableDict`  
property), 162  
locked(`masci_tools.util.lockable_containers.LockableList`  
property), 162  
lorentzian() (in module  
`masci_tools.vis.plot_methods`), 79



lorentzian\_one() (in module `masci_tools.vis.plot_methods`), 79  
 lp (`masci_tools.tools.greensfunction.GreensfElement` attribute), 107

## M

m (`masci_tools.tools.cf_calculation.CFCoefficient` attribute), 104  
 masci\_tools command line option  
   --version, 153  
   -v, 153  
 masci\_tools.io.common\_functions  
   module, 195  
 masci\_tools.io.fleur\_inpgen  
   module, 115  
 masci\_tools.io.fleurxmlmodifier  
   module, 117  
 masci\_tools.io.hdf5\_util  
   module, 197  
 masci\_tools.io.io\_fleurxml  
   module, 129  
 masci\_tools.io.io\_nmmpmat  
   module, 130  
 masci\_tools.io.kkr\_params  
   module, 110  
 masci\_tools.io.kkr\_read\_shapefun\_info  
   module, 112  
 masci\_tools.io.parsers.fleur  
   module, 114  
 masci\_tools.io.parsers.fleur.default\_parse\_tasks  
   module, 140  
 masci\_tools.io.parsers.fleur.fleur\_schema  
   module, 212  
 masci\_tools.io.parsers.fleur.fleur\_schema.fleur\_schema\_parser\_functions  
   module, 217  
 masci\_tools.io.parsers.fleur.outxml\_conversions  
   module, 210  
 masci\_tools.io.parsers.fleur.task\_migrations  
   module, 152  
 masci\_tools.io.parsers.hdf5.reader  
   module, 132  
 masci\_tools.io.parsers.hdf5.recipes  
   module, 133  
 masci\_tools.io.parsers.hdf5.transforms  
   module, 134  
 masci\_tools.io.parsers.kkrimp\_parser\_functions  
   module, 113  
 masci\_tools.io.parsers.kkrparser\_functions  
   module, 112  
 masci\_tools.io.parsers.voroparser\_functions  
   module, 113  
 masci\_tools.tools.cf\_calculation  
   module, 103  
 masci\_tools.tools.greensf\_calculations  
   module, 109  
 masci\_tools.tools.greensfunction  
   module, 105  
 masci\_tools.util.case\_insensitive\_dict  
   module, 163  
 masci\_tools.util.constants  
   module, 220  
 masci\_tools.util.econfig  
   module, 211  
 masci\_tools.util.fleur\_calculate\_expression  
   module, 206  
 masci\_tools.util.lockable\_containers  
   module, 161  
 masci\_tools.util.logging\_util  
   module, 198  
 masci\_tools.util.parse\_tasks  
   module, 207  
 masci\_tools.util.parse\_tasks\_decorators  
   module, 209  
 masci\_tools.util.schema\_dict\_util  
   module, 199  
 masci\_tools.util.xml.common\_functions  
   module, 164  
 masci\_tools.util.xml.converters  
   module, 166  
 masci\_tools.util.xml.xml\_getters  
   module, 190  
 masci\_tools.util.xml.xml\_setters\_basic  
   module, 188  
 masci\_tools.util.xml.xml\_setters\_names  
   module, 169  
 masci\_tools.util.xml.xml\_setters\_nmmpmat  
   module, 181  
 masci\_tools.util.xml.xml\_setters\_xpaths  
   module, 183  
 masci\_tools.vis.bokeh\_plots  
   module, 94  
 masci\_tools.vis.bokeh\_plotter  
   module, 91  
 masci\_tools.vis.common  
   module, 69  
 masci\_tools.vis.data  
   module, 64  
 masci\_tools.vis.fleur  
   module, 57  
 masci\_tools.vis.kkr\_plot\_bandstruc\_qdos  
   module, 59  
 masci\_tools.vis.kkr\_plot\_dos  
   module, 59  
 masci\_tools.vis.kkr\_plot\_FS\_qdos  
   module, 59  
 masci\_tools.vis.kkr\_plot\_shapefun  
   module, 59  
 masci\_tools.vis.matplotlib\_plotter

- module, 72
- masci\_tools.vis.parameters
  - module, 60
- masci\_tools.vis.plot\_methods
  - module, 76
- masci\_tools-fleur-schema-add command line
  - option
    - overwrite, 153
    - test-xml-file <test\_xml\_file>, 153
    - SCHEMA\_FILE, 153
- masci\_tools-fleur-schema-validate-input
  - command line option
    - XML\_FILE, 154
- masci\_tools-fleur-schema-validate-output
  - command line option
    - XML\_FILE, 154
- masci\_tools-parse-all-attrs command line
  - option
    - contains <contains>, 154
    - name <name>, 154
    - not-contains <not\_contains>, 154
    - subtags, 154
    - text, 154
    - c, 154
    - n, 154
    - nc, 154
    - XML\_FILE, 154
- masci\_tools-parse-attr command line
  - option
    - contains <contains>, 155
    - name <name>, 155
    - not-contains <not\_contains>, 155
    - tag <tag>, 155
    - c, 155
    - n, 155
    - nc, 155
    - t, 155
    - XML\_FILE, 155
- masci\_tools-parse-cell command line option
  - XML\_FILE, 155
- masci\_tools-parse-constants command line
  - option
    - XML\_FILE, 155
- masci\_tools-parse-fleur-modes command line
  - option
    - XML\_FILE, 156
- masci\_tools-parse-inp-file command line
  - option
    - XML\_FILE, 156
- masci\_tools-parse-kpoints command line
  - option
    - XML\_FILE, 156
- masci\_tools-parse-nkpts command line option
  - XML\_FILE, 157

- masci\_tools-parse-number-nodes command line
  - option
    - contains <contains>, 157
    - name <name>, 157
    - not-contains <not\_contains>, 157
    - c, 157
    - n, 157
    - nc, 157
    - XML\_FILE, 157
- masci\_tools-parse-out-file command line
  - option
    - XML\_FILE, 157
- masci\_tools-parse-parameters command line
  - option
    - XML\_FILE, 158
- masci\_tools-parse-parent-attrs command
  - line option
    - contains <contains>, 158
    - name <name>, 158
    - not-contains <not\_contains>, 158
    - c, 158
    - n, 158
    - nc, 158
    - XML\_FILE, 158
- masci\_tools-parse-relaxation command line
  - option
    - XML\_FILE, 158
- masci\_tools-parse-structure command line
  - option
    - XML\_FILE, 159
- masci\_tools-parse-symmetry command line
  - option
    - XML\_FILE, 159
- masci\_tools-parse-tag-exists command line
  - option
    - contains <contains>, 159
    - name <name>, 159
    - not-contains <not\_contains>, 159
    - c, 159
    - n, 159
    - nc, 159
    - XML\_FILE, 159
- masci\_tools-parse-text command line option
  - contains <contains>, 160
  - name <name>, 160
  - not-contains <not\_contains>, 160
  - c, 160
  - n, 160
  - nc, 160
  - XML\_FILE, 160
- masci\_tools-plot-fleur-bands command line
  - option
    - backend <backend>, 160
    - recipe <recipe>, 160

```
--save, 160
--show, 160
--weight <weight>, 160
-r, 160
-w, 160
BANDDOS_FILE, 160
masci_tools-plot-fleur-dos command line
    option
--atoms <atoms>, 161
--backend <backend>, 161
--interstitial <interstitial>, 161
--l_resolved <l_resolved>, 161
--recipe <recipe>, 161
--save, 161
--show, 161
--total <total>, 161
-r, 161
BANDDOS_FILE, 161
mask_data() (masci_tools.vis.data.PlotData method),
    66
MatplotlibPlotter (class in
    masci_tools.vis.matplotlib_plotter), 72
max() (masci_tools.vis.data.PlotData method), 66
merge_subgroup_datasets() (in module
    masci_tools.io.parsers.hdf5.transforms),
    137
migrate_033_to_031() (in module
    masci_tools.io.parsers.fleur.task_migrations),
    152
migrate_034_to_033() (in module
    masci_tools.io.parsers.fleur.task_migrations),
    152
migrations (masci_tools.util.parse_tasks.ParseTasks
    property), 208
min() (masci_tools.vis.data.PlotData method), 67
MissingConstant, 206
ModifierTask (class in
    masci_tools.io.fleurxmlmodifier), 127
modify_xmlfile() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier
    method), 120
module
    masci_tools.io.common_functions, 195
    masci_tools.io.fleur_inpgen, 115
    masci_tools.io.fleurxmlmodifier, 117
    masci_tools.io.hdf5_util, 197
    masci_tools.io.io_fleurxml, 129
    masci_tools.io.io_nmmpmat, 130
    masci_tools.io.kkr_params, 110
    masci_tools.io.kkr_read_shapefun_info,
        112
    masci_tools.io.parsers.fleur, 114
    masci_tools.io.parsers.fleur.default_parse_tasks,
        140
```

```
masci_tools.io.parsers.fleur.fleur_schema,
    212
masci_tools.io.parsers.fleur.fleur_schema.fleur_schema,
    217
masci_tools.io.parsers.fleur.outxml_conversions,
    210
masci_tools.io.parsers.fleur.task_migrations,
    152
masci_tools.io.parsers.hdf5.reader, 132
masci_tools.io.parsers.hdf5.recipes, 133
masci_tools.io.parsers.hdf5.transforms,
    134
masci_tools.io.parsers.kkrimp_parser_functions,
    113
masci_tools.io.parsers.kkrparser_functions,
    112
masci_tools.io.parsers.voroparser_functions,
    113
masci_tools.tools.cf_calculation, 103
masci_tools.tools.greensf_calculations,
    109
masci_tools.tools.greensfunction, 105
masci_tools.util.case_insensitive_dict,
    163
masci_tools.util.constants, 220
masci_tools.util.econfig, 211
masci_tools.util.fleur_calculate_expression,
    206
masci_tools.util.lockable_containers, 161
masci_tools.util.logging_util, 198
masci_tools.util.parse_tasks, 207
masci_tools.util.parse_tasks_decorators,
    209
masci_tools.util.schema_dict_util, 199
masci_tools.util.xml.common_functions,
    164
masci_tools.util.xml.converters, 166
masci_tools.util.xml.xml_getters, 190
masci_tools.util.xml.xml_setters_basic,
    188
masci_tools.util.xml.xml_setters_names,
    169
masci_tools.util.xml.xml_setters_nmmpmat,
    181
masci_tools.util.xml.xml_setters_xpaths,
    183
masci_tools.vis.bokeh_plots, 94
masci_tools.vis.bokeh_plotter, 91
masci_tools.vis.common, 69
masci_tools.vis.data, 64
masci_tools.vis.fleur, 57
masci_tools.vis.kkr_plot_bandstruc_qdos,
    59
masci_tools.vis.kkr_plot_dos, 59
```

[masci\\_tools.vis.kkr\\_plot\\_FS\\_qdos](#), 59  
[masci\\_tools.vis.kkr\\_plot\\_shapefun](#), 59  
[masci\\_tools.vis.matplotlib\\_plotter](#), 72  
[masci\\_tools.vis.parameters](#), 60  
[masci\\_tools.vis.plot\\_methods](#), 76  
[move\\_to\\_memory\(\)](#) (in module [masci\\_tools.io.parsers.hdf5.transforms](#)), 137  
[multi\\_scatter\\_plot\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 79  
[multiaxis\\_scatterplot\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 80  
[multiple\\_scatterplots\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 81  
[multiplot\\_moved\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 81  
[multiply\\_array\(\)](#) (in module [masci\\_tools.io.parsers.hdf5.transforms](#)), 137  
[multiply\\_by\\_attribute\(\)](#) (in module [masci\\_tools.io.parsers.hdf5.transforms](#)), 137  
[multiply\\_scalar\(\)](#) (in module [masci\\_tools.io.parsers.hdf5.transforms](#)), 138

## N

[name](#) ([masci\\_tools.io.fleurxmlmodifier.ModifierTask](#) attribute), 127  
[name](#) ([masci\\_tools.io.parsers.hdf5.reader.AttribTransformation](#) attribute), 132  
[name](#) ([masci\\_tools.io.parsers.hdf5.reader.Transformation](#) attribute), 133  
[NestedPlotParameters\(\)](#) (in module [masci\\_tools.vis.parameters](#)), 60  
[nLO](#) ([masci\\_tools.tools.greensfunction.GreensfElement](#) attribute), 107  
[NoPathFound](#), 213  
[normalize\\_list\\_or\\_array\(\)](#) (in module [masci\\_tools.vis.data](#)), 68  
[NoUniquePathFound](#), 213  
[nspins](#) ([masci\\_tools.tools.greensfunction.GreensFunction](#) property), 106  
[num\\_plots](#) ([masci\\_tools.vis.parameters.Plotter](#) property), 63

## O

[onsite](#) ([masci\\_tools.tools.greensfunction.GreensfElement](#) attribute), 107  
[open\\_general\(\)](#) (in module [masci\\_tools.io.common\\_functions](#)), 197  
[optional\\_tasks](#) ([masci\\_tools.util.parse\\_tasks.ParseTasks](#) property), 208

[out\\_version](#) ([masci\\_tools.io.parsers.fleur.fleur\\_schema.OutputSchemaDict](#) property), 214  
[OutParserLogAdapter](#) (class in [masci\\_tools.util.logging\\_util](#)), 198  
[OutputSchemaDict](#) (class in [masci\\_tools.io.parsers.fleur.fleur\\_schema](#)), 213  
[outxml\\_parser\(\)](#) (in module [masci\\_tools.io.parsers.fleur](#)), 114

## P

[parse\\_array\\_float\(\)](#) (in module [masci\\_tools.io.parsers.kkrparser\\_functions](#)), 113  
[parse\\_functions](#) ([masci\\_tools.util.parse\\_tasks.ParseTasks](#) property), 208  
[parse\\_kkr\\_outputfile\(\)](#) (in module [masci\\_tools.io.parsers.kkrparser\\_functions](#)), 113  
[parse\\_kkrimp\\_outputfile\(\)](#) ([masci\\_tools.io.parsers.kkrimp\\_parser\\_functions.KkrimpParserFunctions](#) method), 113  
[parse\\_voronoi\\_output\(\)](#) (in module [masci\\_tools.io.parsers.voroparser\\_functions](#)), 113  
[ParseTasks](#) (class in [masci\\_tools.util.parse\\_tasks](#)), 207  
[PDF](#) (class in [masci\\_tools.vis.plot\\_methods](#)), 77  
[perform\\_task\(\)](#) ([masci\\_tools.util.parse\\_tasks.ParseTasks](#) method), 208  
[performIntegration\(\)](#) ([masci\\_tools.tools.cf\\_calculation.CFCalculation](#) method), 103  
[periodic\\_elements\(\)](#) (in module [masci\\_tools.io.parsers.hdf5.transforms](#)), 138  
[periodic\\_table\\_plot\(\)](#) (in module [masci\\_tools.vis.bokeh\\_plots](#)), 99  
[plot\\_bands\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 82  
[plot\\_bands\\_and\\_dos\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 82  
[plot\\_certain\\_bands\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 83  
[plot\\_colortable\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 83  
[plot\\_convergence\(\)](#) (in module [masci\\_tools.vis.bokeh\\_plots](#)), 100  
[plot\\_convergence\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 83  
[plot\\_convergence\\_results\(\)](#) (in module [masci\\_tools.vis.bokeh\\_plots](#)), 101  
[plot\\_convergence\\_results\(\)](#) (in module [masci\\_tools.vis.plot\\_methods](#)), 84

- `plot_convergence_results_m()` (in module `masci_tools.vis.bokeh_plots`), 101  
`plot_convergence_results_m()` (in module `masci_tools.vis.plot_methods`), 84  
`plot_convex_hull2d()` (in module `masci_tools.vis.bokeh_plots`), 102  
`plot_convex_hull2d()` (in module `masci_tools.vis.plot_methods`), 84  
`plot_corelevel_spectra()` (in module `masci_tools.vis.plot_methods`), 85  
`plot_corelevels()` (in module `masci_tools.vis.plot_methods`), 85  
`plot_crystal_field_calculation()` (in module `masci_tools.tools.cf_calculation`), 105  
`plot_crystal_field_potential()` (in module `masci_tools.tools.cf_calculation`), 105  
`plot_dos()` (in module `masci_tools.vis.plot_methods`), 85  
`plot_fleur_bands()` (in module `masci_tools.vis.fleur`), 57  
`plot_fleur_bands_characterize()` (in module `masci_tools.vis.fleur`), 57  
`plot_fleur_dos()` (in module `masci_tools.vis.fleur`), 58  
`plot_kwargs()` (`masci_tools.vis.bokeh_plotter.BokehPlotter` method), 93  
`plot_kwargs()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotter` method), 75  
`plot_lattice_constant()` (in module `masci_tools.vis.bokeh_plots`), 102  
`plot_lattice_constant()` (in module `masci_tools.vis.plot_methods`), 86  
`plot_one_element_corelv()` (in module `masci_tools.vis.plot_methods`), 86  
`plot_relaxation_results()` (in module `masci_tools.vis.plot_methods`), 86  
`plot_residuen()` (in module `masci_tools.vis.plot_methods`), 86  
`plot_shapefun()` (in module `masci_tools.vis.kkr_plot_shapefun`), 60  
`plot_spectral_function()` (in module `masci_tools.vis.plot_methods`), 87  
`plot_spinpol_bands()` (in module `masci_tools.vis.plot_methods`), 88  
`plot_spinpol_dos()` (in module `masci_tools.vis.plot_methods`), 88  
`PlotBackend` (class in `masci_tools.vis.common`), 69  
`PlotData` (class in `masci_tools.vis.data`), 64  
`PlotDataIterator` (class in `masci_tools.vis.data`), 67  
`Plotter` (class in `masci_tools.vis.parameters`), 60  
`pop()` (`masci_tools.util.lockable_containers.LockableList` method), 163  
`position` (`masci_tools.io.common_functions.AtomSiteProperties` attribute), 195  
`prefactor()` (`masci_tools.tools.cf_calculation.CFCalculation` method), 103  
`prepare_figure()` (`masci_tools.vis.bokeh_plotter.BokehPlotter` method), 94  
`prepare_plot()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotter` method), 75  
`printElements()` (in module `masci_tools.tools.greensfunction`), 108  
`process()` (`masci_tools.util.logging_util.OutParserLogAdapter` method), 198  
`process_data_arguments()` (in module `masci_tools.vis.data`), 68  
`pseudo_voigt_profile()` (in module `masci_tools.vis.plot_methods`), 89
- ## R
- `read()` (`masci_tools.io.parsers.hdf5.reader.HDF5Reader` method), 132  
`read_constants()` (in module `masci_tools.util.schema_dict_util`), 205  
`read_groups()` (in module `masci_tools.io.hdf5_util`), 198  
`read_hdf_simple()` (in module `masci_tools.io.hdf5_util`), 198  
`read_inpgen_file()` (in module `masci_tools.io.fleur_inpgen`), 116  
`read_keywords_from_inputcard()` (`masci_tools.io.kkr_params.kkrparams` method), 111  
`read_nmmpmat_block()` (in module `masci_tools.io.io_nmmpmat`), 130  
`read_shapefun()` (in module `masci_tools.io.kkr_read_shapefun_info`), 112  
`readCDN()` (`masci_tools.tools.cf_calculation.CFCalculation` method), 104  
`readPot()` (`masci_tools.tools.cf_calculation.CFCalculation` method), 104  
`register_migration()` (in module `masci_tools.util.parse_tasks_decorators`), 209  
`register_parsing_function()` (in module `masci_tools.util.parse_tasks_decorators`), 210  
`rek_econ()` (in module `masci_tools.util.econfig`), 211  
`rel_to_abs()` (in module `masci_tools.io.common_functions`), 197  
`rel_to_abs_f()` (in module `masci_tools.io.common_functions`), 197  
`relative_attrib_xpath()` (`masci_tools.io.parsers.fleur.fleur_schema.schema_dict.SchemaDict` method), 128  
`relative_attrib_xpath()` (`masci_tools.io.parsers.fleur.fleur_schema.SchemaDict` method), 215



relative\_tag\_xpath() (in module *masci\_tools.io.parsers.fleur.fleur\_schema.schema\_dict.SchemaDict*), 128

relative\_tag\_xpath() (in module *masci\_tools.io.parsers.fleur.fleur\_schema.schema\_dict.SchemaDict*), 215

remove() (in module *masci\_tools.util.lockable\_containers.LockableList*), 163

remove\_added\_parameters() (in module *masci\_tools.vis.parameters.Plotter*), 63

remove\_value() (in module *masci\_tools.io.kkr\_params.kkrparams*), 111

repeat\_array() (in module *masci\_tools.io.parsers.hdf5.transforms*), 138

repeat\_array\_by\_attribute() (in module *masci\_tools.io.parsers.hdf5.transforms*), 138

replace\_tag() (in module *masci\_tools.util.xml.xml\_setters\_names*), 171

replace\_tag() (in module *masci\_tools.io.fleurxmlmodifier.FleurXMLModifier*), 120

reset\_bokeh\_plot\_defaults() (in module *masci\_tools.vis.bokeh\_plots*), 102

reset\_defaults() (in module *masci\_tools.vis.common*), 71

reset\_defaults() (in module *masci\_tools.vis.parameters.Plotter*), 63

reset\_mpl\_plot\_defaults() (in module *masci\_tools.vis.plot\_methods*), 89

reset\_parameters() (in module *masci\_tools.vis.parameters.Plotter*), 63

reverse() (in module *masci\_tools.util.lockable\_containers.LockableList*), 163

reverse\_xinclude() (in module *masci\_tools.util.xml.common\_functions*), 165

rotate\_nmmpmat() (in module *masci\_tools.util.xml.xml\_setters\_nmmpmat*), 181

rotate\_nmmpmat() (in module *masci\_tools.io.fleurxmlmodifier.FleurXMLModifier*), 120

rotate\_nmmpmat\_block() (in module *masci\_tools.io.io\_nmmpmat*), 130

save\_mpl\_defaults() (in module *masci\_tools.vis.plot\_methods*), 89

save\_plot() (in module *masci\_tools.vis.bokeh\_plotter.BokehPlotter*), 94

save\_plot() (in module *masci\_tools.vis.matplotlib\_plotter.MatplotlibPlotter*), 76

scatter() (in module *masci\_tools.vis.common*), 71

schema\_dict\_version\_dispatch() (in module *masci\_tools.io.parsers.fleur.fleur\_schema*), 216

SCHEMA\_FILE

masci-tools-fleur-schema-add command line option, 153

SchemaDict (class in module *masci\_tools.io.parsers.fleur.fleur\_schema*), 214

SchemaDict (class in module *masci\_tools.io.parsers.fleur.fleur\_schema.schema\_dict*), 127

select\_element\_indices() (in module *masci\_tools.tools.greensfunction*), 109

select\_elements() (in module *masci\_tools.tools.greensfunction*), 109

select\_elements\_from\_file() (in module *masci\_tools.tools.greensfunction*), 109

set\_atomgroup() (in module *masci\_tools.util.xml.xml\_setters\_names*), 172

set\_atomgroup() (in module *masci\_tools.io.fleurxmlmodifier.FleurXMLModifier*), 120

set\_atomgroup\_label() (in module *masci\_tools.util.xml.xml\_setters\_names*), 172

set\_atomgroup\_label() (in module *masci\_tools.io.fleurxmlmodifier.FleurXMLModifier*), 121

set\_attrib\_value() (in module *masci\_tools.util.xml.xml\_setters\_names*), 173

set\_attrib\_value() (in module *masci\_tools.io.fleurxmlmodifier.FleurXMLModifier*), 121

set\_bokeh\_plot\_defaults() (in module *masci\_tools.vis.bokeh\_plots*), 103

set\_color\_palette\_by\_num\_plots() (in module *masci\_tools.vis.bokeh\_plotter.BokehPlotter*), 94

set\_complex\_tag() (in module *masci\_tools.util.xml.xml\_setters\_names*), 173

set\_complex\_tag() (in module *masci\_tools.io.fleurxmlmodifier.FleurXMLModifier*), 121

set\_default\_backend() (in module *masci\_tools.vis.common*), 71

set\_defaults() (in module *masci\_tools.vis.common*), 71

set\_defaults() (in module *masci\_tools.vis.parameters.Plotter*), 63

## S

`set_first_attrib_value()` (in module `masci_tools.util.xml.xml_setters_names`),  
174  
`set_first_attrib_value()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 122  
`set_first_text()` (in module `masci_tools.util.xml.xml_setters_names`),  
175  
`set_first_text()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 122  
`set_inpchanges()` (in module `masci_tools.util.xml.xml_setters_names`),  
175  
`set_inpchanges()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 122  
`set_kpath()` (in module `masci_tools.util.xml.xml_setters_names`),  
176  
`set_kpath()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 123  
`set_kpath_max4()` (in module `masci_tools.util.xml.xml_setters_names`),  
176  
`set_kpointlist()` (in module `masci_tools.util.xml.xml_setters_names`),  
176  
`set_kpointlist()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 123  
`set_kpointlist_max4()` (in module `masci_tools.util.xml.xml_setters_names`),  
177  
`set_legend()` (`masci_tools.vis.bokeh_plotter.BokehPlotters` method), 94  
`set_limits()` (`masci_tools.vis.bokeh_plotter.BokehPlotters` method), 94  
`set_limits()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotters` method), 76  
`set_mpl_plot_defaults()` (in module `masci_tools.vis.plot_methods`), 89  
`set_multiple_values()` (`masci_tools.io.kkr_params.kkrparams` method), 112  
`set_nkpts()` (in module `masci_tools.util.xml.xml_setters_names`),  
177  
`set_nkpts()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 123  
`set_nkpts_max4()` (in module `masci_tools.util.xml.xml_setters_names`),  
178  
`set_nmmpmat()` (in module `masci_tools.util.xml.xml_setters_nmmpmat`),  
182  
`set_nmmpmat()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 123  
`set_parameters()` (`masci_tools.vis.parameters.Plotter` method), 63  
`set_scale()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotters` method), 76  
`set_simple_tag()` (in module `masci_tools.util.xml.xml_setters_names`),  
178  
`set_simple_tag()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 124  
`set_single_default()` (`masci_tools.vis.parameters.Plotter` method),  
63  
`set_species()` (in module `masci_tools.util.xml.xml_setters_names`),  
178  
`set_species()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 124  
`set_species_label()` (in module `masci_tools.util.xml.xml_setters_names`),  
179  
`set_species_label()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 125  
`set_text()` (in module `masci_tools.util.xml.xml_setters_names`),  
179  
`set_text()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 125  
`set_value()` (`masci_tools.io.kkr_params.kkrparams` method), 112  
`shift_by_attribute()` (in module `masci_tools.io.parsers.hdf5.transforms`),  
138  
`shift_data()` (`masci_tools.vis.data.PlotData` method),  
67  
`shift_dataset()` (in module `masci_tools.io.parsers.hdf5.transforms`),  
138  
`shift_value()` (in module `masci_tools.util.xml.xml_setters_names`),  
180  
`shift_value()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 125  
`shift_value_species_label()` (in module `masci_tools.util.xml.xml_setters_names`),  
180  
`shift_value_species_label()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 125  
`show_available_tasks()` (`masci_tools.util.parse_tasks.ParseTasks` method), 209

`show_bokeh_plot_defaults()` (in module `masci_tools.vis.bokeh_plots`), 103  
`show_colorbar()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotter` method), 76  
`show_defaults()` (in module `masci_tools.vis.common`), 71  
`show_legend()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotter` method), 76  
`show_mpl_plot_defaults()` (in module `masci_tools.vis.plot_methods`), 90  
`single_plot` (`masci_tools.vis.parameters.Plotter` property), 63  
`single_scatterplot()` (in module `masci_tools.vis.plot_methods`), 90  
`skipHeader()` (in module `masci_tools.io.common_functions`), 197  
`slice_dataset()` (in module `masci_tools.io.parsers.hdf5.transforms`), 139  
`sort_data()` (`masci_tools.vis.data.PlotData` method), 67  
`sphavg` (`masci_tools.tools.greensfunction.GreensfElement` attribute), 107  
`spin_down` (`masci_tools.tools.cf_calculation.CFCoefficient` attribute), 104  
`spin_up` (`masci_tools.tools.cf_calculation.CFCoefficient` attribute), 104  
`spinpol_bands()` (in module `masci_tools.vis.common`), 71  
`spinpol_dos()` (in module `masci_tools.vis.common`), 72  
`split_array()` (in module `masci_tools.io.parsers.hdf5.transforms`), 139  
`split_kkr_options()` (`masci_tools.io.kkr_params.kkrparams` class method), 112  
`split_off_attrib()` (in module `masci_tools.util.xml.common_functions`), 165  
`split_off_tag()` (in module `masci_tools.util.xml.common_functions`), 166  
`stack_datasets()` (in module `masci_tools.io.parsers.hdf5.transforms`), 139  
`sum_over_dict_entries()` (in module `masci_tools.io.parsers.hdf5.transforms`), 139  
`sum_weights_over_atoms()` (in module `masci_tools.vis.fleur`), 58  
`surface_plot()` (in module `masci_tools.vis.plot_methods`), 90  
`switch_kpointset()` (in module `masci_tools.util.xml.xml_setters_names`), 181  
`switch_kpointset()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 126  
`switch_kpointset_max4()` (in module `masci_tools.util.xml.xml_setters_names`), 181  
`Symbol` (`masci_tools.io.common_functions.AtomSiteProperties` attribute), 195  
`symmetric_difference()` (`masci_tools.util.case_insensitive_dict.CaseInsensitiveFrozenSet` method), 164  

## T

`tag_exists()` (in module `masci_tools.util.schema_dict_util`), 205  
`tag_info()` (`masci_tools.io.parsers.fleur.fleur_schema.schema_dict.SchemaDict` method), 129  
`tag_info()` (`masci_tools.io.parsers.fleur.fleur_schema.SchemaDict` method), 216  
`tag_xpath()` (`masci_tools.io.parsers.fleur.fleur_schema.schema_dict.SchemaDict` method), 129  
`tag_xpath()` (`masci_tools.io.parsers.fleur.fleur_schema.SchemaDict` method), 216  
`TagInfo` (class in `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser`), 217  
`tile_array()` (in module `masci_tools.io.parsers.hdf5.transforms`), 139  
`tile_array_by_attribute()` (in module `masci_tools.io.parsers.hdf5.transforms`), 140  
`to_m_index()` (`masci_tools.tools.greensfunction.GreensFunction` static method), 106  
`to_spin_indices()` (`masci_tools.tools.greensfunction.GreensFunction` static method), 107  
`trace_energy_dependence()` (`masci_tools.tools.greensfunction.GreensFunction` method), 107  
`Transformation` (class in `masci_tools.io.parsers.hdf5.reader`), 132  
`truncate_colormap()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotter` static method), 76  
`type_order()` (in module `masci_tools.io.parsers.fleur.fleur_schema.fleur_schema_parser`), 220  

## U

`undo()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 126  
`union()` (`masci_tools.util.case_insensitive_dict.CaseInsensitiveFrozenSet` method), 164  
`unit` (`masci_tools.tools.cf_calculation.CFCoefficient` attribute), 104



`update_to_kkrimp()` (*masci\_tools.io.kkr\_params.kkrparams* module method), 112  
`update_to_voronoi()` (*masci\_tools.io.kkr\_params.kkrparams* module method), 112  
`use_BdG()` (*masci\_tools.io.parsers.kkrparser\_functions*), 113  
`use_newsosol()` (*masci\_tools.io.parsers.kkrparser\_functions*), 113  
**V**  
`validate_nmmpmat()` (*masci\_tools.util.xml.xml\_setters\_nmmpmat*), 182  
`validate_xml()` (*masci\_tools.util.xml.common\_functions*), 166  
`values()` (*masci\_tools.vis.data.PlotData* method), 67  
`vec_to_angles()` (*masci\_tools.io.common\_functions*), 197  
`voigt_profile()` (*masci\_tools.vis.plot\_methods*), 91  
**W**  
`waterfall_plot()` (*masci\_tools.vis.plot\_methods*), 91  
`write_inpgen_file()` (*masci\_tools.io.fleur\_inpgen*), 116  
`write_nmmpmat()` (*masci\_tools.io.io\_nmmpmat*), 131  
`write_nmmpmat_from_orbitals()` (*masci\_tools.io.io\_nmmpmat*), 131  
`write_nmmpmat_from_states()` (*masci\_tools.io.io\_nmmpmat*), 131  
**X**  
`xml_add_number_to_attrb()` (*masci\_tools.util.xml.xml\_setters\_xpaths*), 183  
`xml_add_number_to_first_attrb()` (*masci\_tools.util.xml.xml\_setters\_xpaths*), 184  
`xml_create_tag()` (*masci\_tools.util.xml.xml\_setters\_basic*), 188  
`xml_create_tag()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* module method), 126  
`xml_create_tag_schema_dict()` (*masci\_tools.util.xml.xml\_setters\_xpaths*), 184  
`xml_delete_att()` (*masci\_tools.util.xml.xml\_setters\_basic*), 188  
`xml_delete_att()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* module method), 126  
`xml_delete_tag()` (*masci\_tools.util.xml.xml\_setters\_basic*), 188  
`xml_delete_tag()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* module method), 126  
**XML\_FILE**  
`masci_tools-fleur-schema-validate-input` command line option, 154  
`masci_tools-fleur-schema-validate-output` command line option, 154  
`masci_tools-parse-all-attrs` command line option, 154  
`masci_tools-parse-attrb` command line option, 155  
`masci_tools-parse-cell` command line option, 155  
`masci_tools-parse-constants` command line option, 155  
`masci_tools-parse-fleur-modes` command line option, 156  
`masci_tools-parse-inp-file` command line option, 156  
`masci_tools-parse-kpoints` command line option, 156  
`masci_tools-parse-nkpts` command line option, 157  
`masci_tools-parse-number-nodes` command line option, 157  
`masci_tools-parse-out-file` command line option, 157  
`masci_tools-parse-parameters` command line option, 158  
`masci_tools-parse-parent-attrs` command line option, 158  
`masci_tools-parse-relaxation` command line option, 158  
`masci_tools-parse-structure` command line option, 159  
`masci_tools-parse-symmetry` command line option, 159  
`masci_tools-parse-tag-exists` command line option, 159  
`masci_tools-parse-text` command line option, 160  
`xml_replace_tag()` (*masci\_tools.util.xml.xml\_setters\_basic*), 189  
`xml_replace_tag()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* module method), 126  
`xml_set_attrb_value()` (*masci\_tools.util.xml.xml\_setters\_xpaths*), 185  
`xml_set_attrb_value_no_create()` (*masci\_tools.util.xml.xml\_setters\_basic*), 189  
`xml_set_attrb_value_no_create()` (*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier* module method), 126

*method*), [127](#)

`xml_set_complex_tag()` (*in module*  
*masci\_tools.util.xml.xml\_setters\_xpaths*),  
[185](#)

`xml_set_first_attrib_value()` (*in module*  
*masci\_tools.util.xml.xml\_setters\_xpaths*),  
[186](#)

`xml_set_first_text()` (*in module*  
*masci\_tools.util.xml.xml\_setters\_xpaths*),  
[186](#)

`xml_set_simple_tag()` (*in module*  
*masci\_tools.util.xml.xml\_setters\_xpaths*),  
[187](#)

`xml_set_text()` (*in module*  
*masci\_tools.util.xml.xml\_setters\_xpaths*),  
[187](#)

`xml_set_text_no_create()` (*in module*  
*masci\_tools.util.xml.xml\_setters\_basic*), [189](#)

`xml_set_text_no_create()`  
(*masci\_tools.io.fleurxmlmodifier.FleurXMLModifier*  
*method*), [127](#)

## Z

`zoom_in()` (*in module*  
*masci\_tools.vis.kkr\_plot\_shapefun*), [60](#)