
Masci-tools Documentation

Release 0.15.0

The JuDFT team

May 12, 2023

CONTENTS

1	Requirements to use this code:	3
2	Installation Instructions:	5
3	Acknowledgments:	7
4	User's Guide	9
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	13
4.1.2.3	User Methods	13
4.1.2.4	Modifying the density matrix for LDA+U calculations	15
4.1.3	Usage Examples	16
4.1.3.1	add_number_to_attrib	16
4.1.3.2	add_number_to_first_attrib	18
4.1.3.3	clone_species	21
4.1.3.4	create_tag	22
4.1.3.5	delete_att	25
4.1.3.6	delete_tag	28
4.1.3.7	replace_tag	30
4.1.3.8	set_atomgroup	33
4.1.3.9	set_atomgroup_label	34
4.1.3.10	set_attrib_value	36
4.1.3.11	set_first_attrib_value	38
4.1.3.12	set_first_text	40
4.1.3.13	set_inpchanges	43
4.1.3.14	set_kpath	44
4.1.3.15	set_kpointlist	45
4.1.3.16	set_kpointmesh	46
4.1.3.17	set_kpointpath	47
4.1.3.18	set_nkpts	48
4.1.3.19	set_simple_tag	50
4.1.3.20	set_species	52
4.1.3.21	set_species_label	55

4.1.3.22	set_text	56
4.1.3.23	set_xcfunctional	59
4.1.3.24	shift_value	60
4.1.3.25	shift_value_species_label	62
4.1.3.26	switch_species	63
4.1.4	General HDF5 file reader	66
4.1.4.1	Basic Usage	66
4.1.4.2	Structure of recipes for the reader.HDF5Reader	67
4.1.5	Plotting Fleur DOS/bandstructures	72
4.1.5.1	Reading banddos.hdf files	72
4.1.5.2	Bandstructures	73
4.1.5.3	Density of States	81
4.1.6	Plotting KKR DOS/bandstructures	91
4.1.6.1	Density of states	91
4.1.6.2	Bandstructure	91
4.1.7	General Plotting routines	96
4.1.7.1	Available Routines	96
4.1.7.2	Providing Data	97
4.1.7.3	Customizing Plots	98
5	Developer's Guide	109
5.1	Developers Guide	109
5.1.1	Test suite of maschi-tools	109
5.1.1.1	Installation	109
5.1.1.2	Running the tests	109
5.1.1.3	Regenerating test files	109
5.1.2	Updating or adapting the Fleur Parsers	109
5.1.2.1	Adding/modifying a Fleur Schema:	110
5.1.2.2	Adapting the outxml_parser:	110
5.1.2.3	Migrating the parsing tasks	111
5.1.3	Using the Plotter class	112
5.1.3.1	Description	112
5.1.3.2	Writing a plotting function	112
5.1.4	Using the PlotData class	135
5.1.4.1	Description	135
5.1.4.2	Initializing PlotData without a mapping	136
5.1.4.3	Available routines on PlotData	137
6	Reference	139
6.1	Reference	139
6.1.1	Changelog	139
6.1.1.1	latest	139
6.1.1.2	v.0.15.0	139
6.1.1.3	v.0.14.0	140
6.1.1.4	v.0.13.0	140
6.1.1.5	v.0.12.0	141
6.1.1.6	v.0.11.3	142
6.1.1.7	v.0.11.2	142
6.1.1.8	v.0.11.1	142
6.1.1.9	v.0.11.0	143
6.1.1.10	v0.10.1	144
6.1.1.11	v.0.10.0	144
6.1.1.12	v.0.9.1	146
6.1.1.13	v.0.9.0	146

6.1.1.14	v0.8.0	147
6.1.1.15	v0.7.2	149
6.1.1.16	v0.7.1	149
6.1.1.17	v0.7.0	150
6.1.1.18	v0.6.2	152
6.1.1.19	v0.6.1	152
6.1.1.20	v0.6.0	152
6.1.1.21	v0.5.0	153
6.1.1.22	v0.4.10	154
6.1.1.23	v0.4.9	154
6.1.1.24	v0.4.8	155
6.1.1.25	v0.4.7	155
6.1.1.26	v0.4.6	156
6.1.1.27	v0.4.5	156
6.1.1.28	v0.4.4	157
6.1.1.29	v0.4.3	157
6.1.1.30	v0.4.2	158
6.1.1.31	v0.4.1	158
6.1.1.32	v0.4.0	158
6.1.2	Source code documentation	159
6.1.2.1	Visualisation and Plotting	159
6.1.2.2	Calculation tools	210
6.1.2.3	IO helper functions and file parsers	223
6.1.2.4	Commandline interface (CLI)	308
6.1.2.5	Utility Functions/Classes	320
6.1.2.6	Basic Fleur Schema parser functions	405
6.1.2.7	Defined constants	418
6.1.2.8	Extensions to Sphinx	444
7	Indices and tables	445
	Python Module Index	447
	Index	449

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`
- `numpy`
- `matplotlib`
- `bokeh` (optional)
- `seaborn`
- `ase`
- `mendeleeev`
- `click`
- `click-completion`
- `PyYAML`
- `tabulate`
- `deepdiff`
- `humanfriendly`
- `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 `0.35`. The following table shows the version compatibility of the input parser.

File Version	Compatible
0.27 - 0.36	Yes
0.37	Fallback to 0.36

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 *Updating or adapting the Fleur Parsers*.

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	0.29 version is assumed (no XML validation)
0.30 - 0.31	Yes (no XML validation)
0.32	No (Does not exist for any release version of fleur)
0.33	Yes (no XML validation)
0.34 - 0.36	Yes
0.37 -	Fallback to 0.36

Using File handles

Both the `inpxml_parser()` and `outxml_parser()` can also be used with file handles like shown below.

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

with open('/path/to/random/inp.xml', 'rb') as file:
    input_dict = inpxml_parser(file)
```

Notice that the file has to be opened in binary mode.

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_parameterdata()`: Get the information about the calculation parameters needed to reproduce a calculation starting from the `inp`gen
- `get_structuredata()`: Get the structure from the xml file (atom positions + unit cell)
- `get_kpointsdata()`: 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.fleur_xml 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.fleur_xml 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')

```

To manage the context of these functions the *FleurXMLContext()* is available to write the same code as above more concisely.

```

from masci_tools.io.fleur_xml import load_inpxml, FleurXMLContext
xmltree, schema_dict = load_inpxml('/path/to/inp.xml')

with FleurXMLContext(xmltree, schema_dict) as root:
    spins = root.attribute('jspins')
    noco = root.attribute('l_noco', default=False)

    #Not nesting the context we need to specify which elements are meant
    mt_radii = root.attribute('radius', contains='species')

    #Nesting using find (the first element is return)
    with root.find('atomspecies') as all_species:
        mt_radii = all_species.attribute('radius')

    #Nesting using iter (each iteration returns a new context for the next element)
    mt_radii = []
    for species in root.iter('species'):
        mt_radii.append(species.attribute('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 XML tree.

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 XML tree 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

- `FleurXMLModifier.modify_xmlfile()`: Applies the registered changes to a given `inp.xml` (and optional `n_mmp_mat` file)
- `FleurXMLModifier.changes()`: Displays the current list of changes.
- `FleurXMLModifier.undo()`: Removes the last task or all tasks from the list of changes.

Modification methods

Simple changes

set_inpchanges Change the value of multiple text or attribute values at once.

Show Examples

shift_value Shift or multiply the value of multiple text or attribute values at once.

Show Examples

Modifying atom species

set_species Change parameters or add new tags in atomic species elements. For example changing the MT radius or adding DFT+U.

Show Examples

set_species_label Change parameters or add new tags in atomic species of a specific atom. The atom is identified by its label.

Show Examples

clone_species Duplicate a given species element with a different name.

Show Examples

Modifying atom groups

set_atomgroup Change parameters or add new elements in `atomGroup` elements, i.e. the elements containing the symmetry equivalent atoms.

Show Examples

set_atomgroup_label Change parameters or add new elements in `atomGroup` elements, i.e. the elements containing the symmetry equivalent atoms. The group to modify is identified by a given label of an atom.

Show Examples

switch_species Change the species of a given atom group.

Show Examples

switch_species_label Change the species of a atom group containing a given atom.

Show Examples

Modifying kpoint sets

switch_kpointset Switch the used kpoint set MaX 5.0 or newer

Show Examples

set_kpointlist Create a k-point list from a list of coordinates and weights.

Show Examples

set_nkpts Set the number of kpts (`kpointCount`) MaX 4.0 or older

Show Examples

set_kpath Set a explicit path for bandstructure MaX 4.0 or older

Show Examples

Setting generic XML elements and attributes

set_attrib_value Set arbitrary XML attribute values

Show Examples

set_first_attrib_value Set arbitrary XML attribute values for the first occurrence of the given attribute in the XML tree

Show Examples

set_text Set the text of arbitrary XML elements

Show Examples

set_first_text Set the text of arbitrary XML elements for the first occurrence of the given element in the XML tree

Show Examples

add_number_to_attrib Add to or multiply the values of arbitrary XML attributes

Show Examples

add_number_to_first_attrib Add to or multiply the values of arbitrary XML attributes for the first occurrence of the given attribute in the XML tree

Show Examples

set_simple_tag Create or change arbitrary simple XML elements, i.e. elements without child elements

[Show Examples](#)

set_complex_tag Create or change an arbitrary complex XML element, i.e. an element with child elements.

[Show Examples](#)

Manipulating the DFT+U density matrix

set_nmmpmat Initialize the `n_mmp_mat` file with a given density matrix

[Show Examples](#)

rotate_nmmpmat Rotate one or multiple block(s) of the `n_mmp_mat` with euler angles

[Show Examples](#)

align_nmmpmat_to_sqa Rotate one or multiple blocks of the `n_mmp_mat` with euler angles to align with the spin-quantization axis specified.

[Show Examples](#)

Using explicit XPath expressions for modifications

These routines should be used with a lot of care, since they have much less checks than all other setting functions

xml_create_tag Create a XML element with the given name as a child of the results of the XPath expression

[Show Examples](#)

xml_delete_tag Delete the results of the XPath expression.

[Show Examples](#)

xml_delete_att Delete a give XML attribute from the results of the XPath expression.

[Show Examples](#)

xml_replace_tag Replace the results of the XPath expression with a given XML element.

[Show Examples](#)

xml_set_attrib_value_no_create Set XML attribute values on the results of the XPath expression.

[Show Examples](#)

xml_set_text_no_create Set text on the results of the XPath expression.

[Show Examples](#)

4.1.2.4 Modifying the density matrix for LDA+U calculations

The above mentioned `FleurXMLModifier.set_nmmpmat()`, `FleurXMLModifier.rotate_nmmpmat()` and `FleurXMLModifier.align_nmmpmat_to_sqa()` 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 `FleurXMLModifier.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()
# Add LDA+U procedure
fm.set_species('Nd-1', {'ldaU':{'l': 3, 'U': 6.76, 'J': 0.76, 'l_amf': 'F'}})
# 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, since no n_mmp_mat file is provided
fm.set_nmmpmat('Nd-1', orbital=3, spin=1, state_occupations=[1,1,1,1,0,0,0])
new_xmltree, add_files = fm.modify_xmlfile('/path/to/original/inp.xml')
print(add_files['n_mmp_mat'])
```

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 Usage Examples

4.1.3.1 add_number_to_attrib

Go to the API reference

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Added filters*

Simple Usage

Add/Multiply values for multiple occurrences in the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f7d1c3d8500>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
```

(continues on next page)

(continued from previous page)

```
fm.add_number_to_attrrib('itmax', 10)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f7cfc1a5480>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f7d1d487f80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.add_number_to_attrrib('radius', 1.05, mode='rel')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The attribute radius has multiple possible paths with the current
↪ specification.
contains: None, not_contains: None, exclude: []
The following are possible:
- /fleurInput/atomGroups/atomGroup/mtSphere/@radius
- /fleurInput/atomSpecies/species/mtSphere/@radius
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f7cfc106f80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
```

(continues on next page)

(continued from previous page)

```
fm.add_number_to_attr('radius', 1.05, mode='rel', contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f7cfc11b500>
```

Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f7d1d484f80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.add_number_to_attr('spinUp',
                    -1,
                    contains='species',
                    filters={
                        'species': {
                            'atomicNumber': {'>': 30}
                        }
                    })
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f7cfc105940>
```

4.1.3.2 add_number_to_first_attr

Go to the API reference

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Added filters*

Simple Usage

Add/Multiply value for the first occurrences of an attribute in the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa020618080>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.add_number_to_first_attrib('itmax', 10)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa010231240>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa0101a2e40>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.add_number_to_first_attrib('radius', 1.05, mode='rel')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The attribute radius has multiple possible paths with the current_
↪ specification.
contains: None, not_contains: None, exclude: []
The following are possible:
- /fleurInput/atomGroups/atomGroup/mtSphere/@radius
- /fleurInput/atomSpecies/species/mtSphere/@radius
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa020528380>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.add_number_to_first_attrub('radius', 1.05, mode='rel', contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f9ff7c93ec0>
```

Added filters

The filters argument allows to be more specific

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa0118201c0>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.add_number_to_first_attrub('spinUp',
                              -1,
                              contains='species',
                              filters={
                                  'species': {
                                      'atomicNumber': {'>': 30}
                                  }})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa010195600>
```

4.1.3.3 clone_species

Go to the API reference

Examples

- *Simple Usage*
- *Modifying the cloned species*

Simple Usage

Clone (and possibly change) a species in the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmldata, schema_dict = load_inpxml('../files/inp.xml')
xmldata
```

```
<lxml.etree._ElementTree at 0x7fc7f46c4240>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.clone_species('Fe-1', 'Fe-clone-1')
new_xmldata, additional_files = fm.modify_xmlfile(xmldata, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fc7d0a212c0>
```

Modifying the cloned species

The changes dictionary is passed on to set_species to modify the cloned species

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmldata, schema_dict = load_inpxml('../files/inp.xml')
xmldata
```

```
<lxml.etree._ElementTree at 0x7fc7f5fc0c80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.clone_species('Fe-1', 'Fe-clone-1', {'mtSphere' : {'radius' : 2.7}})
new_xmldata, additional_files = fm.modify_xmlfile(xmldata, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fc7d0993140>
```

4.1.3.4 create_tag

Go to the API reference

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Nested creation*
- *Creation of pre-created XML element*
- *Added filters*

Simple Usage

Create a tag with a given name in the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmldata, schema_dict = load_inpxml('../files/inp.xml')
xmldata
```

```
<lxml.etree._ElementTree at 0x7fa62c2abcc0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.create_tag('mtnocoparams')
new_xmldata, additional_files = fm.modify_xmlfile(xmldata, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa601c93b80>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa6240770c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.create_tag('lo')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The tag lo has multiple possible paths with the current specification.
contains: None, not_contains: None
The following are possible:
- /fleurInput/atomGroups/atomGroup/lo
- /fleurInput/atomSpecies/species/lo
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa601d76bc0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.create_tag('lo', contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa601a52000>
```

Nested creation

With `create_parents=True` evtl. missing parent tags are also created

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa601d76840>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.create_tag('diagElements',
              create_parents=True,
              contains='species',
              not_contains='torque')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa62d3b5040>
```

Creation of pre-created XML element

Passing a XML element instead of a name will insert this element at the correct position

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa601c929c0>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.create_tag(etree.Element('lo', n='5', l='0', type='SCLO'),
              contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa624033440>
```

Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa6240fbf80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.create_tag(etree.Element('lo', n='5', l='0', type='SCLO'),
              contains='species',
              filters={
                  'species': {
                      'name': {'contains': 'Fe'}}
              })
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa6240fbf40>
```

4.1.3.5 delete_att

Go to the API reference

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Added filters*

Simple Usage

Delete an attribute with a given name from the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f77b831ff00>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.delete_att('mixparam')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f7794de5600>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f7794df0180>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.delete_att('alpha')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The attribute alpha has multiple possible paths with the current_
↪ specification.
contains: None, not_contains: None, exclude: []
The following are possible:
- /fleurInput/calculationSetup/scfLoop/@alpha
- /fleurInput/atomGroups/atomGroup/absPos/@alpha
- /fleurInput/atomGroups/atomGroup/filmPos/@alpha
- /fleurInput/atomGroups/atomGroup/nocoParams/@alpha
- /fleurInput/atomGroups/atomGroup/relPos/@alpha
```

(continues on next page)

(continued from previous page)

```
- /fleurInput/atomSpecies/species/nocoParams/@alpha
- /fleurInput/calculationSetup/greensFunction/contourSemicircle/@alpha
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f7794e922c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.delete_att('alpha', contains='nocoParams', not_contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f77a2634080>
```

Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f77a260f500>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.delete_att('alpha',
              contains='nocoParams',
              not_contains='species',
              filters={
                  'atomGroup': {
                      'species': {'not-contains': 'Fe'}
                  }
              })
```

(continues on next page)

(continued from previous page)

```
    })
    new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f7794e73d80>
```

4.1.3.6 delete_tag

[Go to the API reference](#)

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Added filters*

Simple Usage

Delete a tag with a given name from the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f14eb535280>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.delete_tag('expertModes')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f14d81211c0>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f14d8079780>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.delete_tag('lo')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The tag lo has multiple possible paths with the current specification.
contains: None, not_contains: None
The following are possible:
- /fleurInput/atomGroups/atomGroup/lo
- /fleurInput/atomSpecies/species/lo
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f14d8086600>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.delete_tag('lo', contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f14d8139040>
```

Added filters

The filters argument allows to be more specific

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f14d8123880>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.delete_tag('lo',
              contains='species',
              filters={
                  'species': {
                      'name': {'contains': 'Fe'}}
              })
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f14d8121340>
```

4.1.3.7 replace_tag

Go to the API reference

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Added filters*

Simple Usage

Replace a tag with a given name with a different XML element

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa7c857fd80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.replace_tag('expertmodes', etree.Element('greensFunction'))
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa7a8129640>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa7c857f080>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.replace_tag('lo', etree.Element('ldaU', l='3'))
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The tag lo has multiple possible paths with the current specification.
contains: None, not_contains: None
The following are possible:
- /fleurInput/atomGroups/atomGroup/lo
- /fleurInput/atomSpecies/species/lo
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa7c823f780>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.replace_tag('lo', etree.Element('ldaU', l='3'), contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa7a8155580>
```

Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fa7a812b780>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.replace_tag('lo',
               etree.Element('ldaU', l='3'),
               contains='species',
               filters={
                   'species': {
                       'atomicNumber': {'>': 30}
                   }
               })
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fa7ccf3f900>
```

4.1.3.8 set_atomgroup

Go to the API reference

Examples

- *Simple Usage*
- *Specifying the number of the atomgroup*
- *Modifying all species*

Simple Usage

Set the attributes of a atom group, specified by species name, position, etc.

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fae9c4e0580>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, species='Fe-1')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fae72ff5500>
```

Specifying the number of the atomgroup

Providing the position argument will modify the n-th atomgroup in the inp.xml file. (Indexing starts at 1)

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fae72f63880>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
```

(continues on next page)

(continued from previous page)

```
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, position=1)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fae72f6c080>
```

Modifying all species

Providing 'all' to either position or ``species`` will modify all atomgroups

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fae73028440>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, species='all')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fae72f62e80>
```

4.1.3.9 set_atomgroup_label

Go to the [API reference](#)

Examples

- *Simple Usage*
- *Modifying all species*

Simple Usage

Set the attributes of a atom group, given by an atom label occuring in it's definition

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f35c053f940>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_atomgroup_label('222', {'nocoParams': {'alpha': 1.57}})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f35b0208740>
```

Modifying all species

Providing 'all' as the first argument applies the changes to all species

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f35b0227d40>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_atomgroup_label('all', {'nocoParams': {'alpha': 1.57}})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f35c2e53a80>
```

4.1.3.10 set_attrib_value

Go to the API reference

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Added filters*

Simple Usage

Set the value of an attribute possibly occurring multiple times

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmldata, schema_dict = load_inpxml('../files/inp.xml')
xmldata
```

```
<lxml.etree._ElementTree at 0x7f214c463980>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_attrib_value('itmax', 180)
new_xmldata, additional_files = fm.modify_xmlfile(xmldata, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f213c135440>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmldata, schema_dict = load_inpxml('../files/inp.xml')
xmldata
```

```
<lxml.etree._ElementTree at 0x7f213d7209c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_attr_value('radius', 1.05)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

NoUniquePathFound: The attribute radius has multiple possible paths with the current specification.
 ↳ specification.
contains: None, **not_contains:** None, **exclude:** []
 The following are possible:

- /fleurInput/atomGroups/atomGroup/mtSphere/@radius
- /fleurInput/atomSpecies/species/mtSphere/@radius

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f2123bb2f40>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_attr_value('radius', 1.05, contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f213c168080>
```

Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f2123bc8180>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_attr_value('spinUp',
                  1,
                  contains='species',
                  filters={
                      'species': {
                          'atomicNumber': {'>': 30}
                      }})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f214f56c980>
```

4.1.3.11 set_first_attr_value

Go to the API reference

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Added filters*

Simple Usage

Set the value of the first occurrence of an attribute

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7effc4463e80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_first_attr_value('itmax', 180)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7effb4125240>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7effc6d1ffc0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_first_attrib_value('radius', 1.05)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The attribute radius has multiple possible paths with the current
↪ specification.
contains: None, not_contains: None, exclude: []
The following are possible:
- /fleurInput/atomGroups/atomGroup/mtSphere/@radius
- /fleurInput/atomSpecies/species/mtSphere/@radius
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7effb416fc80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_first_attrib_value('radius', 1.05, contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7effb4156d80>
```

Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7effc4462a80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_first_attrib_value('spinUp',
                           1,
                           contains='species',
                           filters={
                               'species': {
                                   'atomicNumber': {'>': 30}
                               }})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7effc445bc00>
```

4.1.3.12 set_first_text

Go to the [API reference](#)

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Added filters*

Simple Usage

Set the value of text of the first occurrence of a tag in the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fd4443dc140>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_first_text('kpoint', [1,2,3])
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fd42c599c40>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fd447ca2f80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_first_text('valenceConfig', '(1s1/2)')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fd42c5d0b40>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_first_text('valenceConfig', '(1s1/2)', contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fd42c5badc0>
```

Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fd42c563a80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_first_text('valenceConfig',
                  '(1s1/2)',
                  contains='species',
                  filters={
                      'species': {
                          './lo/@n': {'>': 4}
                      })
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fd4490b3a80>
```


4.1.3.13 set_inpchanges

Go to the API reference

Examples

- *Simple Usage*
- *Attribute selection not unique*
- *Attribute selection*

Simple Usage

Set the value of attributes/text occurring once in the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fbfc07c3d80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_inpchanges({'itmax' : 1, 'kmax': 4.3})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fbfa4909280>
```

Attribute selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fbfca01c400>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
```

(continues on next page)

(continued from previous page)

```
fm.set_inpchanges({'itmax' : 1, 'theta': 1.57})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The attribute theta has multiple possible paths with the current
↪ specification.
contains: None, not_contains: None, exclude: ['other']
The following are possible:
- /fleurInput/calculationSetup/soc/@theta
- /fleurInput/forceTheorem/DMI/@theta
- /fleurInput/forceTheorem/MAE/@theta
```

Attribute selection

Selection can be done by adding conditions on what the XPath should(n't) contain in the `path_spec` argument

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fbfca0104c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_inpchanges({'itmax' : 1, 'theta': 1.57},
                  path_spec={'theta': {'contains': 'soc'}})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fbfca010c00>
```

4.1.3.14 set_kpath

Go to the [API reference](#)

Examples

- *Simple Usage*

Simple Usage

Sets a k-path directly as an alternative kpoint set with purpose ‘bands’

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp_max4.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fc5846bfd40>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_kpath({'Point': [0, 0, 0], 'Point2': [1, 1, 1]}, 200)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fc56c651b00>
```

4.1.3.15 set_kpointlist

Go to the API reference

Examples

- *Simple Usage*
- *MaX 4 compatibility*

Simple Usage

Create a k-point list with the given points and weights

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f4a48343d00>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_kpointlist([[0,0,0],[0.5,0.5,0.5]],
```

(continues on next page)

(continued from previous page)

```
[1,1], switch=True)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f4a24e75980>
```

MaX 4 compatibility

For input files before the MaX 5 release the previous kpoint list will always be overwritten and the `switch` argument has no effect

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp_max4.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f4a4c3fe080>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_kpointlist([[0,0,0],[0.5,0.5,0.5]],
                  [1,1])
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f4a24de2e00>
```

4.1.3.16 set_kpointmesh

Go to the API reference

Examples

- *Simple Usage*

Simple Usage

Create a kpoint mesh using spglib

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fd10c6c7700>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_kpointmesh([4, 4, 1], switch=True)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fd0e92088c0>
```

4.1.3.17 set_kpointpath

Go to the API reference

Examples

- *Simple Usage*
- *Specifying custom points*

Simple Usage

Create a kpoint list for a bandstructure calculation (using ASE kpath generation)

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f96a468bf80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_kpointpath(nkpts=50, switch=True)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f9684238c80>
```

Specifying custom points

By default the path is chosen by ASE. By providing path this can be modified

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f96a42482c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_kpointpath(path='CA',
                  nkpts=25,
                  special_points={
                      'C': [0, 0, 0],
                      'A': [0, 0, 0.5]
                  })
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f96754dfa40>
```

4.1.3.18 set_nkpts

Go to the API reference

Examples

- *Simple Usage*
- *Include Gamma point*

Simple Usage

Sets a k-point mesh directly specific for inputs of version Max4

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp_max4.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f1b702959c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_nkpts(200)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f1b7037e700>
```

Include Gamma point

Sets a k-point mesh directly specific for inputs of version Max4

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp_max4.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f1b45ca2a80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_nkpts(200, gamma=True)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f1b53735300>
```

4.1.3.19 set_simple_tag

Go to the API reference

Examples

- *Simple Usage*
- *Tag selection not unique*
- *Tag selection*
- *Nested creation*
- *Added filters*

Simple Usage

Sets one or multiple simple tag(s) (no subtags/text) in an xmltree.

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f21c457fb80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_simple_tag('soc', {'theta': 0.1, 'phi': 0.2, 'l_soc': True})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f21a10b99c0>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f21c868ccc0>
```



```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_simple_tag('lo', [{'l': 0, 'n': 6, 'type': 'SCL0'},
                        {'l': 1, 'n': 6, 'type': 'SCL0'}])
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

NoUniquePathFound: The tag lo has multiple possible paths with the current specification.
contains: None, **not_contains:** None
 The following are possible:

- /fleurInput/atomGroups/atomGroup/lo
- /fleurInput/atomSpecies/species/lo

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f21c9a67640>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_simple_tag('lo', [{'l': 0, 'n': 6, 'type': 'SCL0'},
                        {'l': 1, 'n': 6, 'type': 'SCL0'}],
                  contains='species')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f21c4115ac0>
```

Nested creation

With `create_parents=True` if the parents of the tag are missing they are created

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f21ae826400>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_simple_tag('realAxis', {'ne': 1300, 'ellow': -1.0, 'elup': 1.0}, create_
↳parents=True)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f21a0d60500>
```

Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f21c457ecc0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_simple_tag('lo', [{'l': 0, 'n': 6, 'type': 'SCLO'},
                        {'l': 1, 'n': 6, 'type': 'SCLO'}],
                  contains='species',
                  filters={
                      'species': {
                          'atomicNumber': {'>': 30}
                      }
                  })
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f21a0d5d580>
```

4.1.3.20 set_species

Go to the API reference

Examples

- *Simple Usage*
- *Adding child elements*

- *Modifying all species*
- *Modifying a subset of species*

Simple Usage

Set the attributes of a species, specified by name position, etc.

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fd65412f940>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_species('Fe-1', {'mtSphere' : {'radius' : 2.7}})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fd63cb71080>
```

Adding child elements

The changes dictionary is not limited to setting attributes. Child elements can also be created. Notice here that if multiple elements of a given name are allowed, all previously existing elements are deleted and replaced with the ones specified in changes.

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7fd65412ee80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_species('Fe-1', {'ldaU' : {'l' : 3, 'U': 4.0, 'J': 0.5, 'l_amf': True},
                        'lo': [{'l': 0, 'n': 6, 'type': 'SCL0'},
                               {'l': 1, 'n': 6, 'type': 'SCL0'}]})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fd6581f1340>
```

Modifying all species

Providing 'all' as the first argument applies the changes to all species

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmldata, schema_dict = load_inpxml('../files/inp.xml')
xmldata
```

```
<lxml.etree._ElementTree at 0x7fd63cb71a00>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_species('all', {'mtSphere' : {'radius' : 2.7}})
new_xmldata, additional_files = fm.modify_xmlfile(xmldata, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fd63e3dd300>
```

Modifying a subset of species

Providing 'all-<search string>' as the first argument applies the changes to all species which contain the search string in its name

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmldata, schema_dict = load_inpxml('../files/inp.xml')
xmldata
```

```
<lxml.etree._ElementTree at 0x7fd63cb36cc0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_species('all-Pt', {'mtSphere' : {'radius' : 2.7},
                          'lo': {'l': 0, 'n': 6, 'type': 'SCL0'}},)
new_xmldata, additional_files = fm.modify_xmlfile(xmldata, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7fd63c797c40>
```

4.1.3.21 set_species_label

Go to the API reference

Examples

- *Simple Usage*
- *Adding child elements*
- *Modifying all species*

Simple Usage

Set the attributes of a species, given by an atom label occurring in it's atom type

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f21e84576c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_species_label('222', {'mtSphere' : {'radius' : 2.7}})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f21c4fbd200>
```

Adding child elements

The changes dictionary is not limited to setting attributes Child elements can also be created. Notice here that if multiple elements of a given name are allowed all previously existing elements are deleted and replaced with the ones specified in changes

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<xml.etree.ElementTree at 0x7f21c4c0cc80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_species_label('222', {'ldaU' : {'l' : 3, 'U': 4.0, 'J': 0.5, 'l_amf': True},
                             'lo': [{'l': 0, 'n': 6, 'type': 'SCLO'},
                                    {'l': 1, 'n': 6, 'type': 'SCLO'}]})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<xml.etree.ElementTree at 0x7f21d2735f80>
```

Modifying all species

Providing 'all' as the first argument applies the changes to all species

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<xml.etree.ElementTree at 0x7f21c4f17a40>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_species_label('all', {'mtSphere' : {'radius' : 2.7}})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<xml.etree.ElementTree at 0x7f21ec564080>
```

4.1.3.22 set_text

Go to the [API reference](#)

Examples

- *Simple Usage*
- *Setting one value*
- *Tag selection not unique*
- *Tag selection*

- *Added filters*

Simple Usage

Set the value of text of tags in the XML tree possibly occurring mutiple times

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f069012c2c0>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_text('kpoint', [[1,2,3], [4,5,6]])
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f066c4f1680>
```

Setting one value

If only one value for the text is given all elements are set ot the same value in this case only one list of three coordinates is set

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f0691a44d80>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_text('kpoint', [4,4,4])
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f06919e8400>
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f0691b5f240>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_text('q', '1 1 1')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
NoUniquePathFound: The tag q has multiple possible paths with the current specification.
contains: None, not_contains: None
The following are possible:
- /fleurInput/forceTheorem/DMI/qVectors/q
- /fleurInput/forceTheorem/Jij/qVectors/q
- /fleurInput/forceTheorem/spinSpiralDispersion/q
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f066fbd66c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_text('q', '1 1 1', contains='spinSpiral')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f066fd91580>
```


Added filters

The filters argument allows to be more specific

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f066c52d280>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_text('valenceConfig',
            '(1s1/2)',
            contains='species',
            filters={
                'species': {
                    './lo/@n': {'>': 4}
                }
            })
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f066c540d40>
```

4.1.3.23 set_xcfunctional

Go to the API reference

Examples

- *Simple Usage*
- *Setting a LibXC functional*

Simple Usage

Set the Exchange Correlation potential tag

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f1404387700>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_xcfunctional('pbe')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f13e0135140>
```

Setting a LibXC functional

Set the Exchange Correlation potential tag

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f14074392c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.set_xcfunctional({'exchange': 'lda_x', 'correlation': 'lda_c_xalpha'}, libxc=True)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f13e0177940>
```

4.1.3.24 shift_value

Go to the [API reference](#)

Examples

- *Simple Usage*
- *Relative shift*

Simple Usage

Add/Multiply values of attributes occurring once in the XML tree

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f5ca423bc80>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.shift_value({'itmax' : 5, 'mindistance': 1})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f5c874f8580>
```

Relative shift

Passing mode="rel" or mode="relative" will multiply the initial attribute values with the provided number instead of adding them

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f5ca423b340>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.shift_value({'itmax' : 2, 'mindistance': 3}, mode="relative")
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f5c799de940>
```

4.1.3.25 shift_value_species_label

Go to the API reference

Examples

- *Simple Usage*
- *Relative shift*
- *Modifying all species*

Simple Usage

Add/Multiply numeric attributes in a or multiple atom species

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f70981ff8c0>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.shift_value_species_label('222', 'radius', 1)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f706dd3d000>
```

Relative shift

Passing mode="rel" or mode="relative" will multiply the initial attribute value with the provided number instead of adding them

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f706dd3e080>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier
```

(continues on next page)

(continued from previous page)

```
fm = FleurXMLModifier()
fm.shift_value_species_label('222', 'radius', 1.2, mode="relative")
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f707b4f8600>
```

Modifying all species

Providing 'all' as the first argument applies the changes to all atomgroups

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f706dd3d080>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.shift_value_species_label('all', 'lmax', 2)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f706dcb1200>
```

4.1.3.26 switch_species

Go to the API reference

Examples

- *Simple Usage*
- *Specifying the number of the atomgroup*
- *Create a clone of the old species*
- *Modifying all species*

Simple Usage

Change the species of an atom group, identified by species name, position, etc.

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f5bf18dbc00>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.switch_species('Pt-1', species='Fe-1')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f5bf0241400>
```

Specifying the number of the atomgroup

Providing the position argument will modify the n-th atomgroup in the inp.xml file. (Indexing starts at 1)

```
from maschi_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f5bf1494600>
```

```
from maschi_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.switch_species('Pt-1', position=1)
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f5bf028bb00>
```

Create a clone of the old species

Specifying a non-existent new species name and `clone=True` will create a clone of the old species with that name. Additionally the `changes` argument can then be used to apply changes to the new clone

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f5bf18db100>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.switch_species('Fe-clone-2', species='Fe-1', clone=True, changes={'mtSphere' : {
    ↪ 'radius' : 2.7}})
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f5bdd7aaf40>
```

Modifying all species

Providing 'all' to either position or ``species`` will modify the species of all atomgroups

```
from masci_tools.io.fleur_xml import load_inpxml
from lxml import etree

xmltree, schema_dict = load_inpxml('../files/inp.xml')
xmltree
```

```
<lxml.etree._ElementTree at 0x7f5bf12f0280>
```

```
from masci_tools.io.fleurxmlmodifier import FleurXMLModifier

fm = FleurXMLModifier()
fm.switch_species('Fe-1', species='all')
new_xmltree, additional_files = fm.modify_xmlfile(xmltree, validate_changes=False)
```

```
<IPython.core.display.HTML object>
```

```
<lxml.etree._ElementTree at 0x7f5bf028a980>
```

4.1.4 General HDF5 file reader

Fleur uses the HDF5 library for output files containing large datasets. The masci-tools library provides the `reader.HDF5Reader` class to extract and transform information from these files. The h5py library is used to get information from .hdf files.

4.1.4.1 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 masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_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 `reader.HDF5Reader.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 `masci_tools.io.parsers.hdf5.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 `reader.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.

4.1.4.2 Structure of recipes for the reader.HDF5Reader

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

```

1 def bands_recipe_format(group: Literal['Local', 'jDOS', 'Orbcomp', 'MCD'], simple: bool,
2   ↪= False) -> HDF5Recipe:
3     """
4     Format for bandstructure calculations retrieving weights from the given group
5
6     :param group: str of the group the weights should be taken from
7     :param simple: bool, if True no additional weights are retrieved with the produced
8     ↪recipe
9
10    :returns: dict of the recipe to retrieve a bandstructure calculation
11    """
12
13    if group == 'Local':
14        atom_prefix = 'MT:'
15        weight_atom_terminator = '[spdf]'
16    elif group == 'jDOS':
17        atom_prefix = 'jDOS:'
18        weight_atom_terminator = '[spdf]'
19    elif group == 'Orbcomp':
20        atom_prefix = 'ORB:'
21        weight_atom_terminator = ','
22    elif group == 'MCD':
23        atom_prefix = 'At'
24        weight_atom_terminator = 'NC'
25    else:
26        raise ValueError(f'Unknown group: {group}')
27
28    recipe = HDF5Recipe({
29        'datasets': {
30            'eigenvalues': {
31                'h5path':
32                f'/{group}/BS/eigenvalues',
33                'transforms': [
34                    AttribTransformation(name='shift_by_attribute',
35                                       attrib_name='fermi_energy',
36                                       kwargs={
37                                           'negative': True,
38                                       }),
39                    Transformation(name='multiply_scalar', args=(HTR_TO_EV,)),
40                    Transformation(name='split_array', kwargs={
41                        'suffixes': ['up', 'down'],
42                        'name': 'eigenvalues'
43                    }),
44                    Transformation(name='flatten_array')
45                ],
46            },
47            'unpack_dict':

```

(continues on next page)

(continued from previous page)

```

45     True
46 },
47     'kpath': {
48         'h5path':
49         '/kpts/coordinates',
50         'transforms': [
51             AttribTransformation(name='multiply_by_attribute',
52                                 attrib_name='reciprocal_cell',
53                                 kwargs={'transpose': True}),
54             Transformation(name='calculate_norm', kwargs={'between_neighbours':
↪ True}),
55             Transformation(name='cumulative_sum'),
56             AttribTransformation(name='repeat_array_by_attribute', attrib_name=
↪ 'nbands'),
57         ]
58     },
59 },
60     'attributes': {
61         'group_name': {
62             'h5path': f'/{group}',
63             'transforms': [
64                 Transformation(name='get_name'),
65             ],
66         },
67         'kpoints': {
68             'h5path': '/kpts/coordinates',
69         },
70         'nkpts': {
71             'h5path': '/Local/BS/eigenvalues',
72             'transforms': [Transformation(name='get_shape'),
73                             Transformation(name='index_dataset', args=(1,))]
74         },
75         'nbands': {
76             'h5path': '/Local/BS/eigenvalues',
77             'transforms': [Transformation(name='get_shape'),
78                             Transformation(name='index_dataset', args=(2,))]
79         },
80         'atoms_elements': {
81             'h5path': '/atoms/atomicNumbers',
82             'description': 'Atomic numbers',
83             'transforms': [Transformation(name='periodic_elements')]
84         },
85         'n_types': {
86             'h5path':
87             '/atoms',
88             'description':
89             'Number of atom types',
90             'transforms':
91             [Transformation(name='get_attribute', args=('nTypes',)),
92              Transformation(name='get_first_element')]
93         },
94         'atoms_position': {

```

(continues on next page)

(continued from previous page)

```

95         'h5path': '/atoms/positions',
96         'description': 'Atom coordinates per atom',
97     },
98     'atoms_groups': {
99         'h5path': '/atoms/equivAtomsGroup'
100     },
101     'reciprocal_cell': {
102         'h5path': '/cell/reciprocalCell'
103     },
104     'bravais_matrix': {
105         'h5path': '/cell/bravaisMatrix',
106         'description': 'Coordinate transformation internal to physical for atoms
→ ',
107         'transforms': [Transformation(name='multiply_scalar', args=(BOHR_A,))]
108     },
109     'special_kpoint_indices': {
110         'h5path': '/kpts/specialPointIndices',
111         'transforms': [Transformation(name='shift_dataset', args=(-1,))]
112     },
113     'special_kpoint_labels': {
114         'h5path': '/kpts/specialPointLabels',
115         'transforms': [Transformation(name='convert_to_str')]
116     },
117     'fermi_energy': {
118         'h5path':
119         '/general',
120         'description':
121         'fermi_energy of the system',
122         'transforms': [
123             Transformation(name='get_attribute', args=('lastFermiEnergy',)),
124             Transformation(name='get_first_element')
125         ]
126     },
127     'spins': {
128         'h5path':
129         '/general',
130         'description':
131         'number of distinct spin directions in the system',
132         'transforms':
133         [Transformation(name='get_attribute', args=('spins',)),
134          Transformation(name='get_first_element')]
135     }
136 }
137 })
138
139 if simple:
140     return recipe
141
142 recipe['datasets']['weights'] = {
143     'h5path':
144     f'/{group}/BS',
145     'transforms': [

```

(continues on next page)

(continued from previous page)

```

146         Transformation(name='get_all_child_datasets', kwargs={'ignore': ['eigenvalues
↪', 'kpts']})),
147         AttribTransformation(name='add_partial_sums',
148                             attrib_name='atoms_groups',
149                             args=(f'{atom_prefix}{{{}}}{weight_atom_terminator}'.
↪format,)),
150                             kwargs={
151                                 'make_set': True,
152                                 'replace_format': f'{atom_prefix}{{{}}}'.format
153                             }),
154         Transformation(name='split_array', kwargs={'suffixes': ['up', 'down']}),
155         Transformation(name='flatten_array')
156     ],
157     'unpack_dict':
158     True
159 }
160
161     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 `masci_tools.io.parsers.hdf5.transforms`. These are added to an entry by adding a list of namedtuples (`reader.Transformation` for general transformations; `reader.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()`: Multiply 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.

4.1.5 Plotting Fleur DOS/bandstructures

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

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*.

Contents

- *Plotting Fleur DOS/bandstructures*
 - *Reading banddos.hdf files*
 - *Bandstructures*
 - * *Standard bandstructure*
 - * *Bandstructure with weights*
 - * *Plotting options for bandstructure plots*
 - *Plotting bandstructure without spinpolarization*
 - *Selecting a specific spin channel*
 - *Density of States*
 - * *Standard density of states plot*
 - * *Plotting options for DOS plots*
 - *Selecting specific DOS components*
 - *Plotting DOS without spinpolarization*
 - *Selecting a specific spin channel*

4.1.5.1 Reading banddos.hdf files

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 *reader.HDF5Reader*. For a detailed explanation of the capabilities of this tool refer to *General HDF5 file reader*. Here we show the basic usage:

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

with HDF5Reader('files/banddos_bands.hdf') as h5reader:
    data, attributes = h5reader.read(recipe=FleurBands)

print(f"The following datasets were read: {list(data.keys())}")
print(f"The following attributes were read: {list(attributes.keys())}")
```

```
The following datasets were read: ['eigenvalues_up', 'kpath', 'INT_up', 'MT:1d_up',
↪ 'MT:1f_up', 'MT:1p_up', 'MT:1s_up', 'Sym_up', 'Total_up', 'MT:1_up']
```

(continues on next page)

(continued from previous page)

The following attributes were read: ['group_name', 'kpoints', 'nkpts', 'nbands', 'atoms_elements', 'n_types', 'atoms_position', 'atoms_groups', 'reciprocal_cell', 'bravais_matrix', 'special_kpoint_indices', 'special_kpoint_labels', 'fermi_energy', 'spins']

4.1.5.2 Bandstructures

Compatible Recipes for the `reader.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
- `recipes.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-psinpolarized 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 masci_tools.io.parsers.hdf5 import HDF5Reader
from masci_tools.io.parsers.hdf5.recipes import FleurBands
from masci_tools.vis.fleur import plot_fleur_bands

#Read in data
with HDF5Reader('files/banddos_bands.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,
                      limits={'y': (-13, 5)},
                      markersize=10)
```

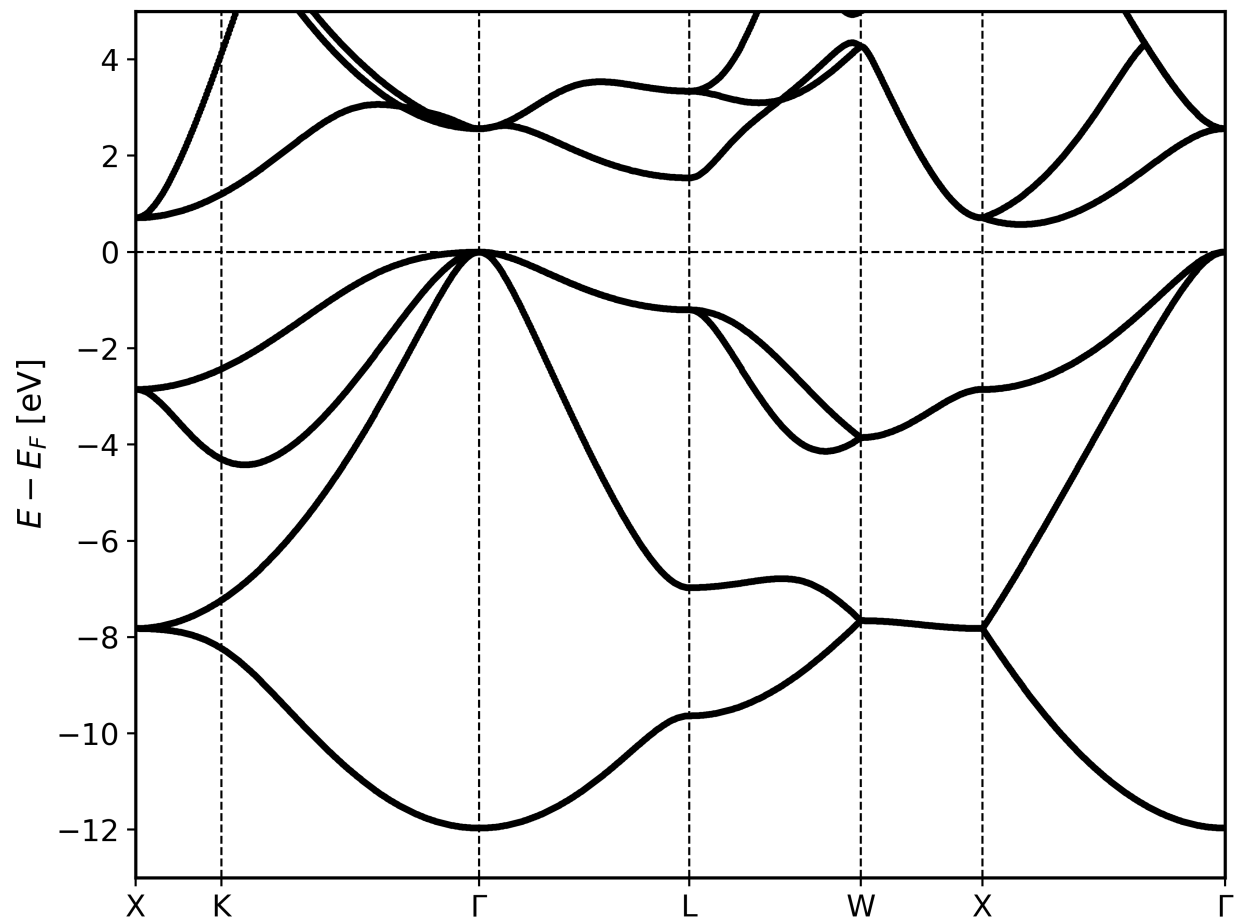


Fig. 1: Non spinpolarized bandstructure for a bulk Si structure

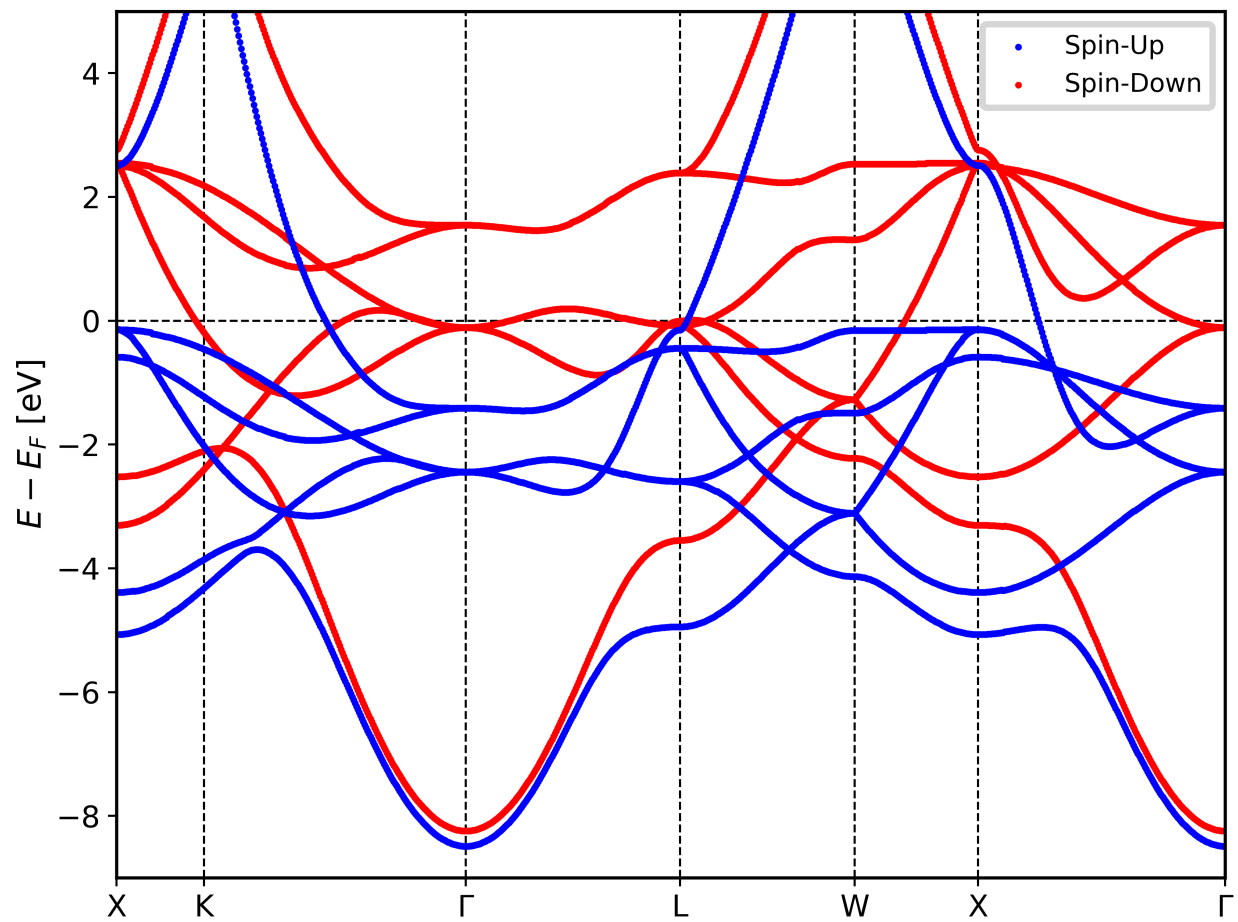


Fig. 2: Spinpolarized bandstructure for a bulk Fe fcc structure

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

#Read in data
with HDF5Reader('files/banddos_bands.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,
                      weight='MT:1s',
                      limits={'y': (-13,5)})
```

Plotting options for bandstructure plots

The `plot_fleur_bands()` 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.

Plotting bandstructure without spinpolarization

Providing `spinpol=False` will display the bandstructure as non spinpolarized, even if there are two spins in the data. Works for both non-weighted and weighted bandstructures.

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

#Read in data
with HDF5Reader('files/banddos_spinpol_bands.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,
```

(continues on next page)

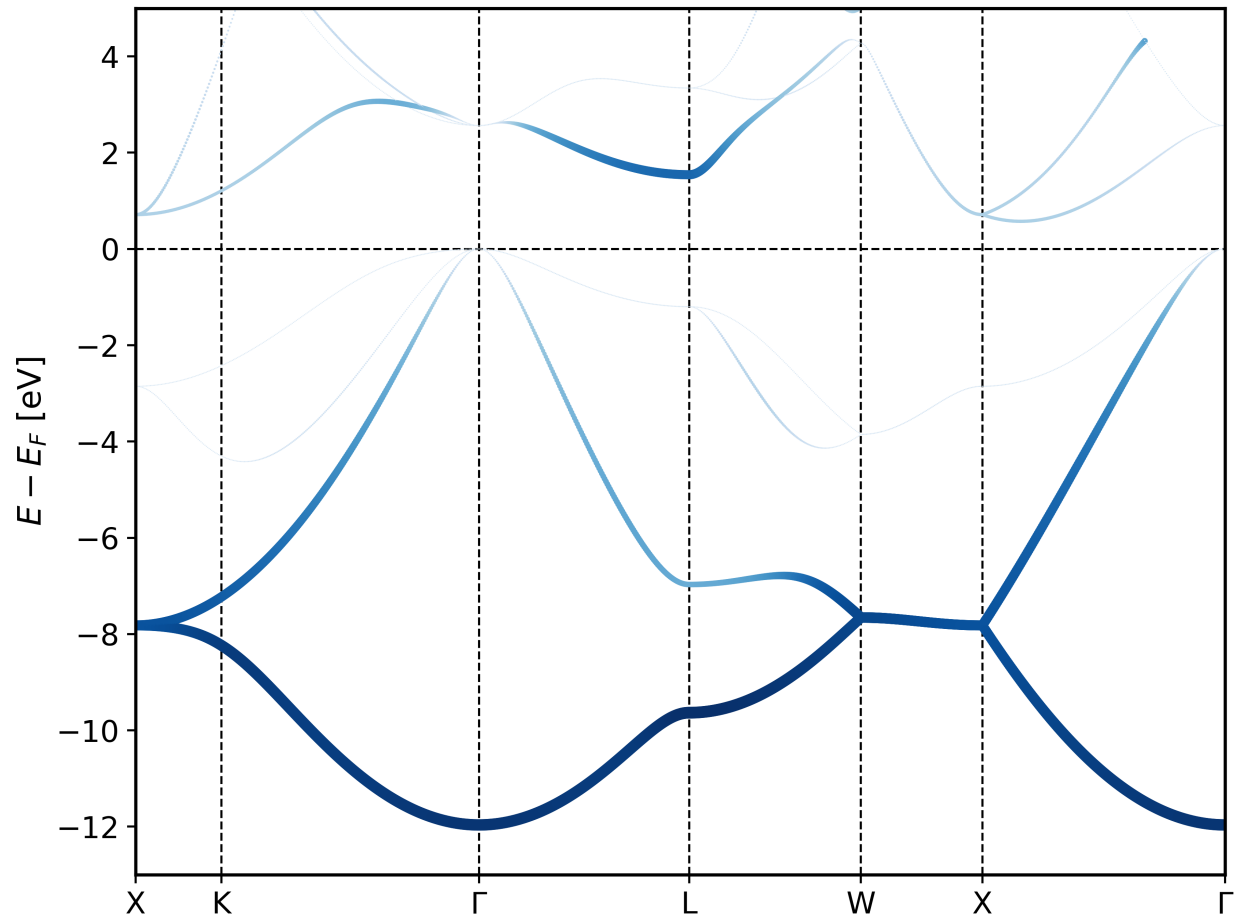


Fig. 3: Non spinpolarized bandstructure for a bulk Si structure. The s-like character inside the Muffin-tin sphere is highlighted

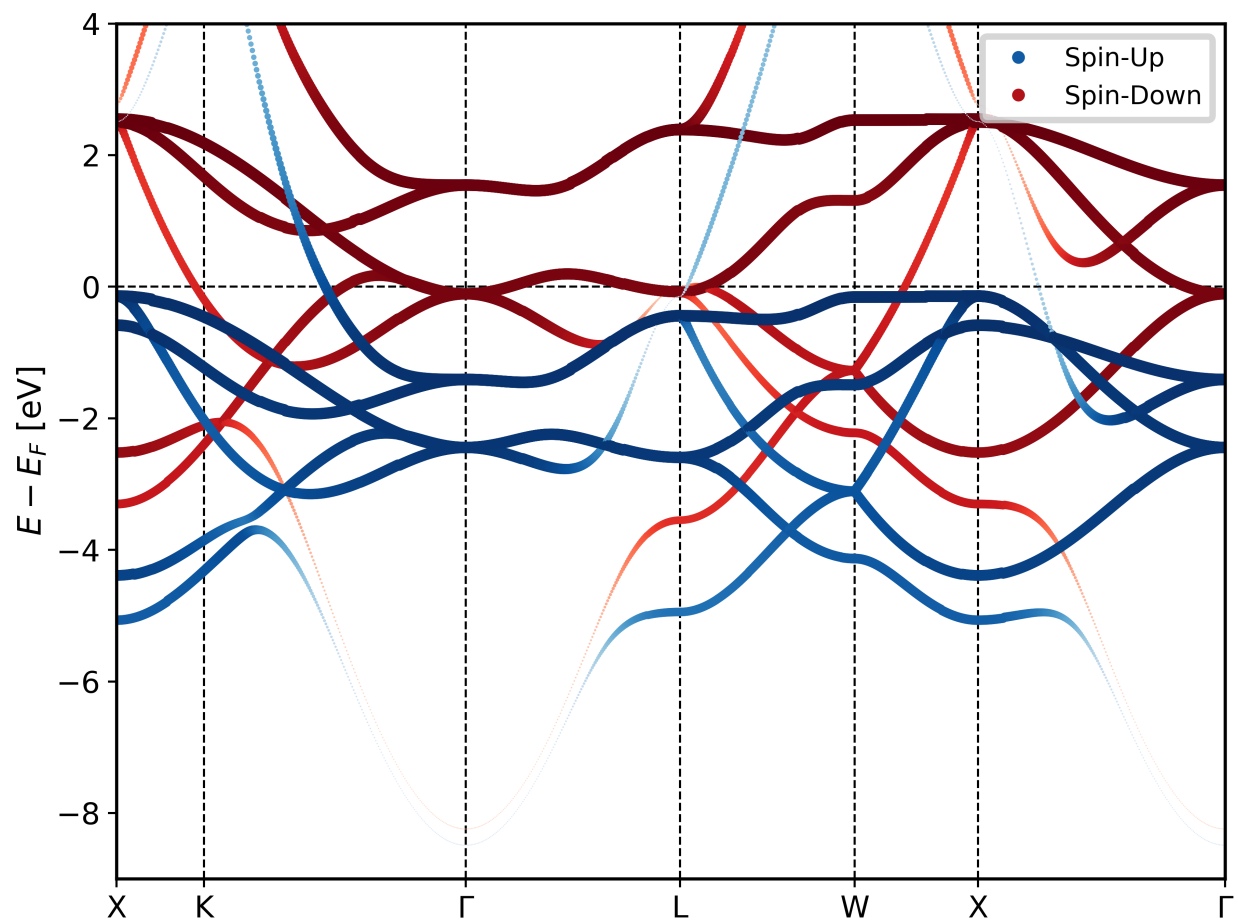


Fig. 4: Spinpolarized bandstructure for a bulk Fe fcc structure. The d-like character inside the Muffin-tin sphere is highlighted

(continued from previous page)

```
limits={'y': (-9,4)},
markersize=10,
spinpol=False)
```

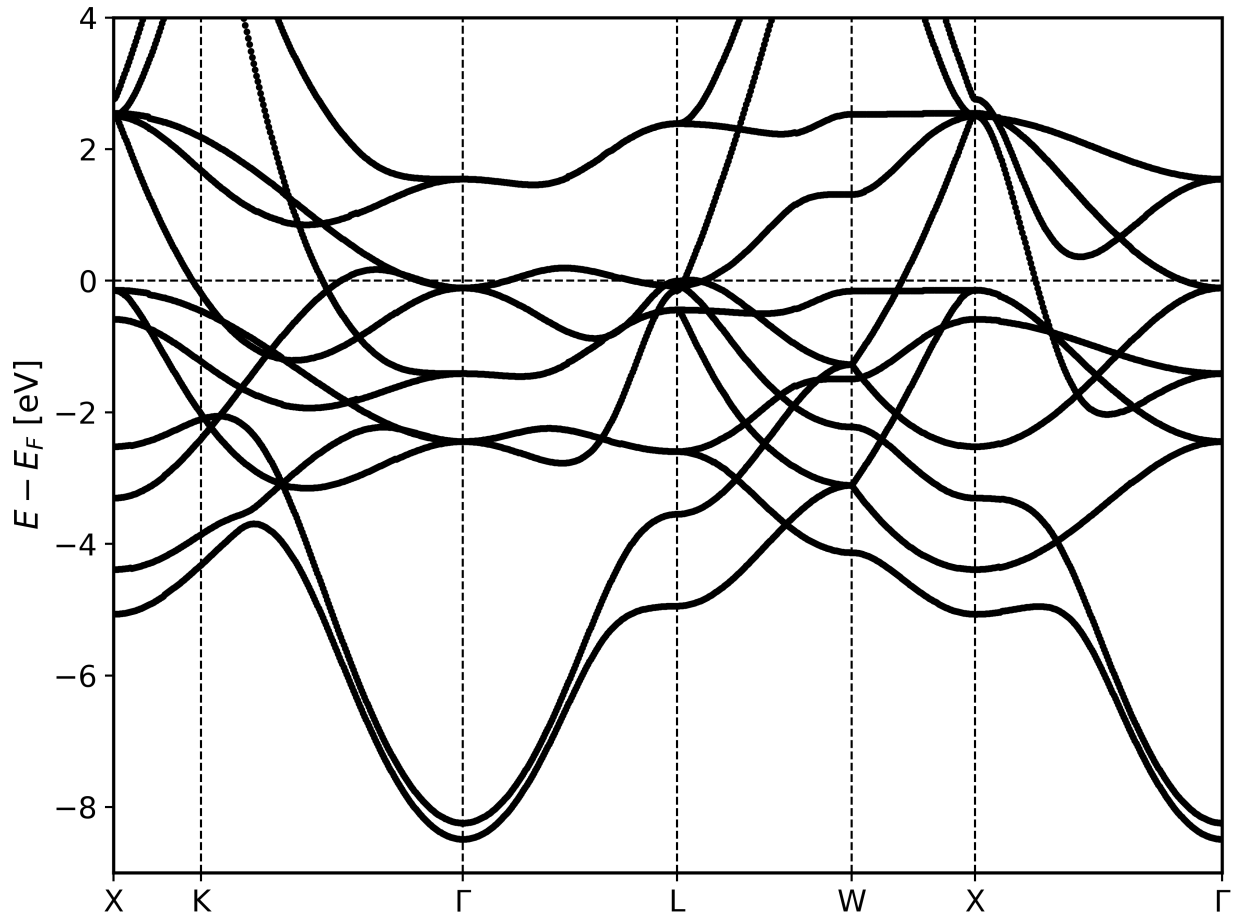


Fig. 5: Non spinpolarized bandstructure for a bulk Fe fcc structure.

Selecting a specific spin channel

Providing only_spin='up' or 'down' will plot only the given spin channel

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

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

#Plot the data
#Notice that you get the axis object of this plot is returned
```

(continues on next page)

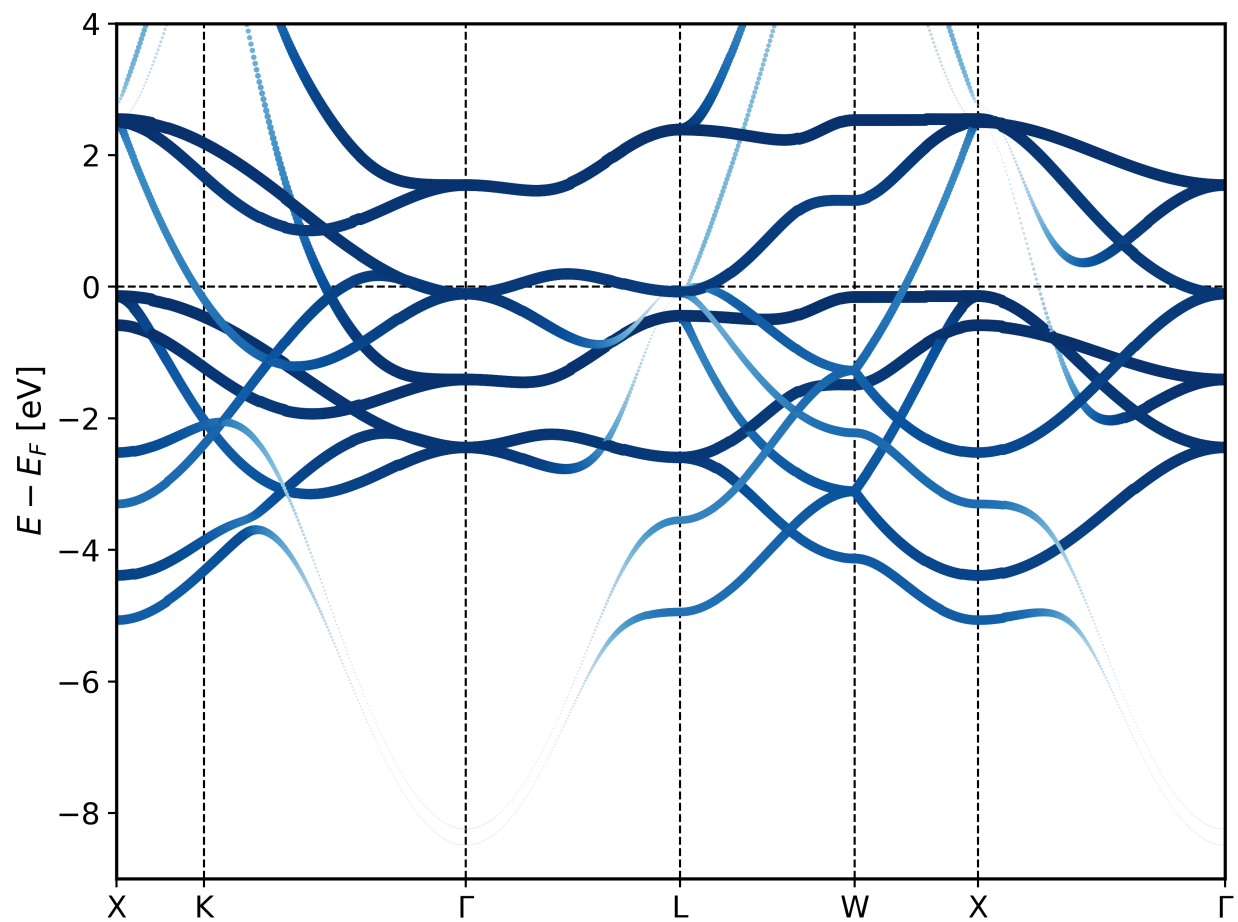


Fig. 6: Non spinpolarized bandstructure for a bulk Fe fcc structure. The d-like character inside the Muffin-tin sphere is highlighted

(continued from previous page)

```
#if you want to make any special additions
ax = plot_fleur_bands(data,
                      attributes,
                      limits={'y': (-9,4)},
                      only_spin='up',
                      markersize=10,
                      color='darkblue')
```

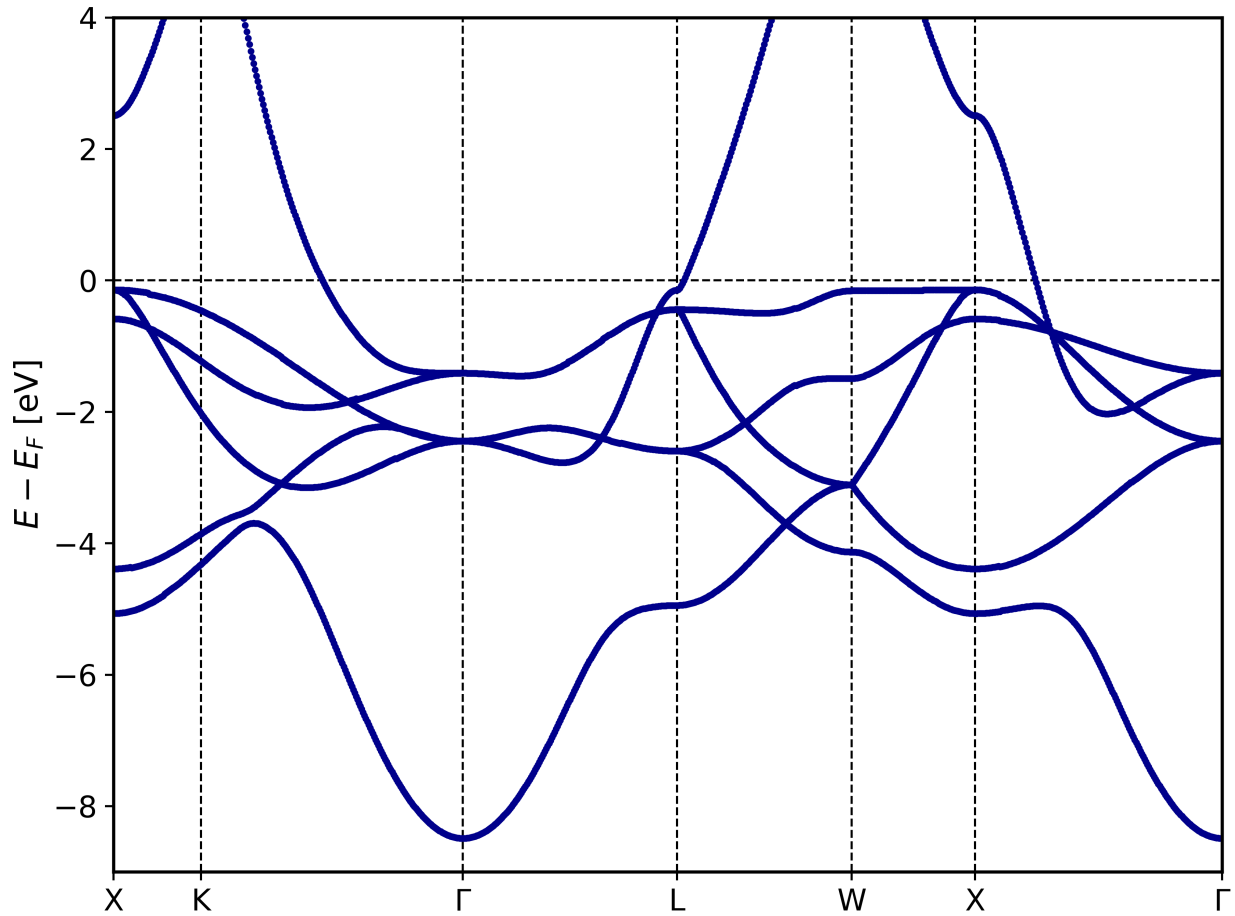


Fig. 7: Bandstructure for a bulk Fe fcc structure (only spin up).

4.1.5.3 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

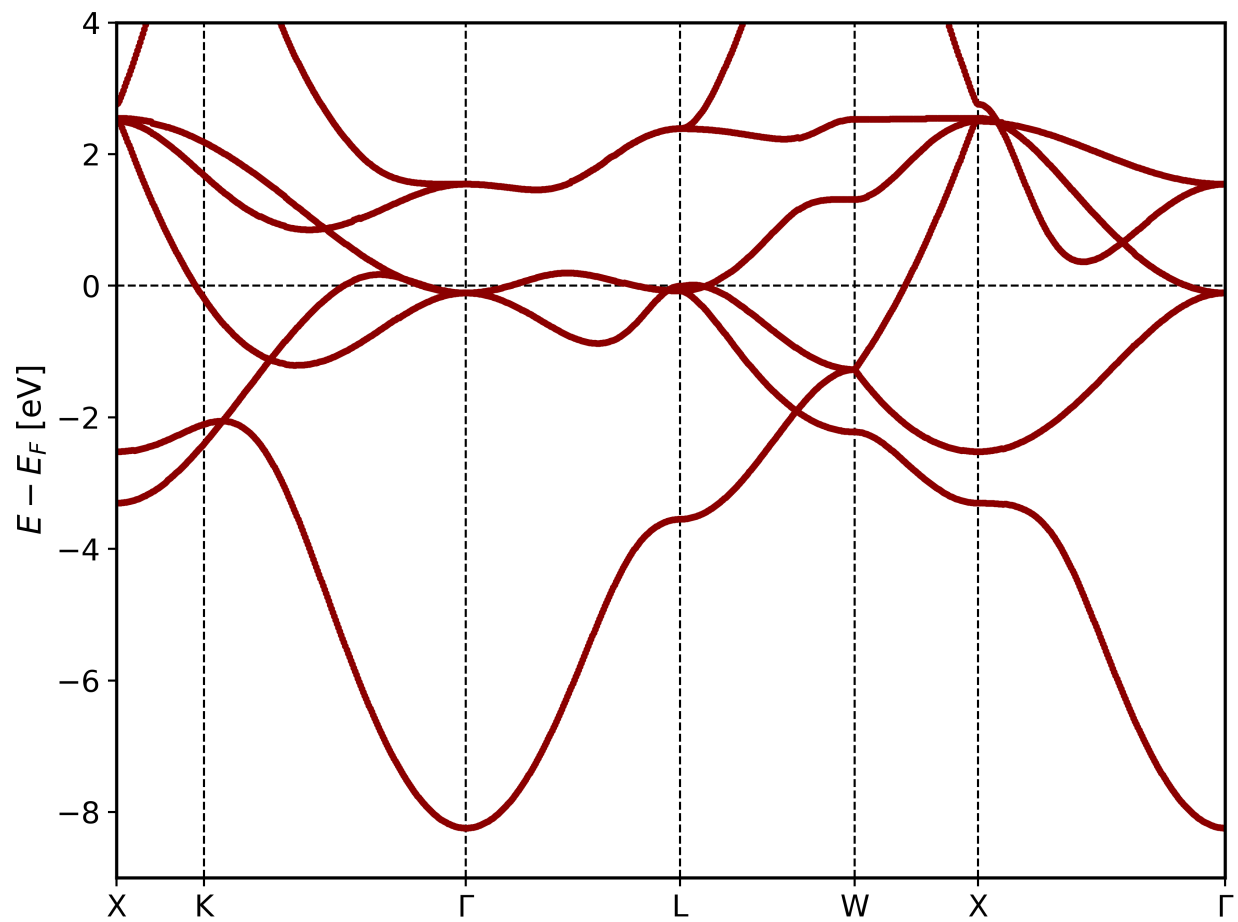


Fig. 8: Bandstructure for a bulk Fe fcc structure (only spin down). The color is changed manually

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

#Read in data
with HDF5Reader('files/banddos_dos.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,
                    limits={'energy': (-13,5)})
```

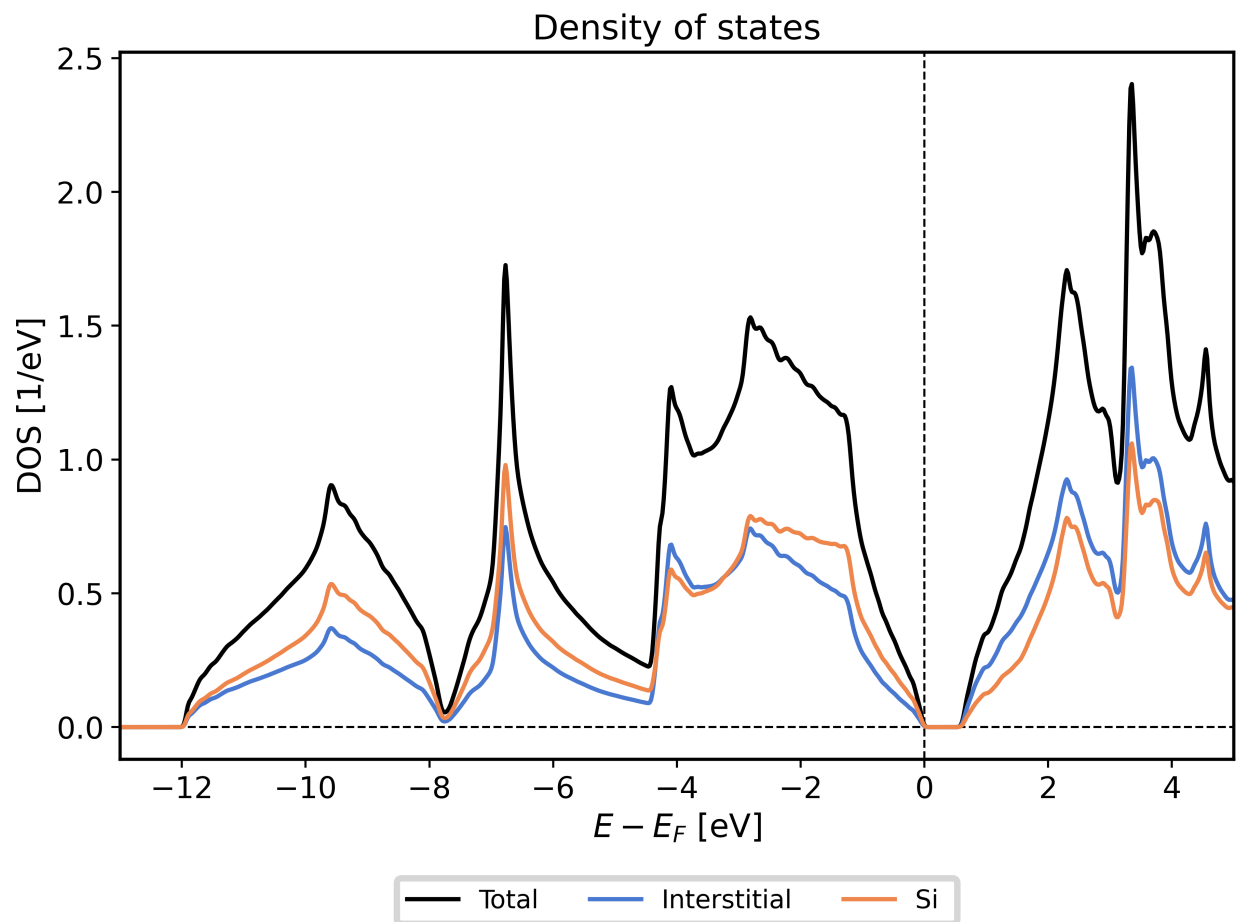


Fig. 9: Non spinpolarized DOS for a bulk Si structure

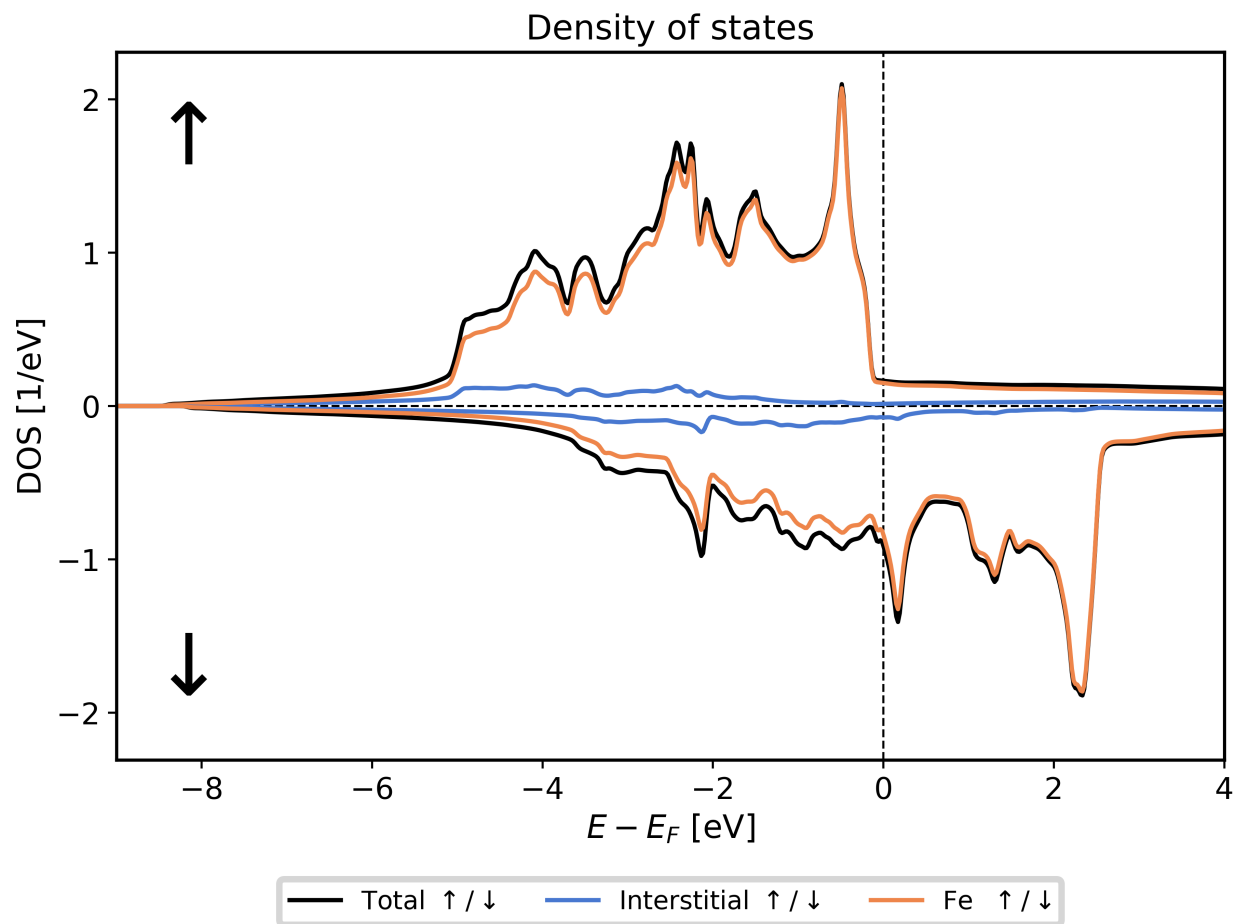


Fig. 10: Spinpolarized DOS for a bulk Fe fcc structure

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.

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 compinents of the density of states

- `plot_keys`: Can be used to provide a 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_lresolved`: 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.

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

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

ax = plot_fleur_dos(data, attributes,
                    show_total=False,
                    show_interstitial=False,
                    show_lresolved='all',
                    limits={'energy': (-13,5)})
```

Plotting DOS without spinpolarization

Providing `spinpol=False` will display the DOS as non spinpolarized, even if there are two spins in the data.

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

#Read in data
with HDF5Reader('files/banddos_spinpol_dos.hdf') as h5reader:
```

(continues on next page)

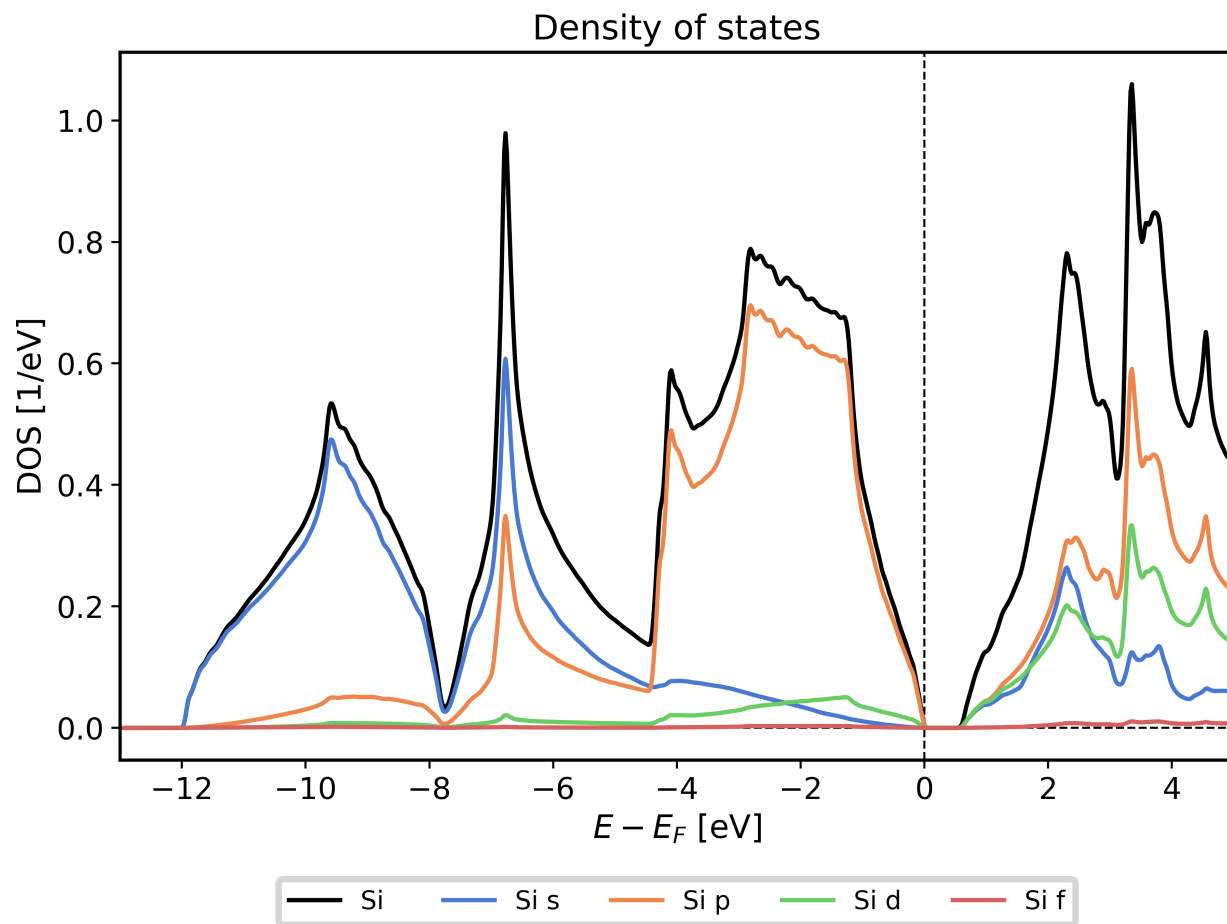


Fig. 11: Non spinpolarized DOS for a bulk Si structure. Only the atom and l-channel projected DOS is shown

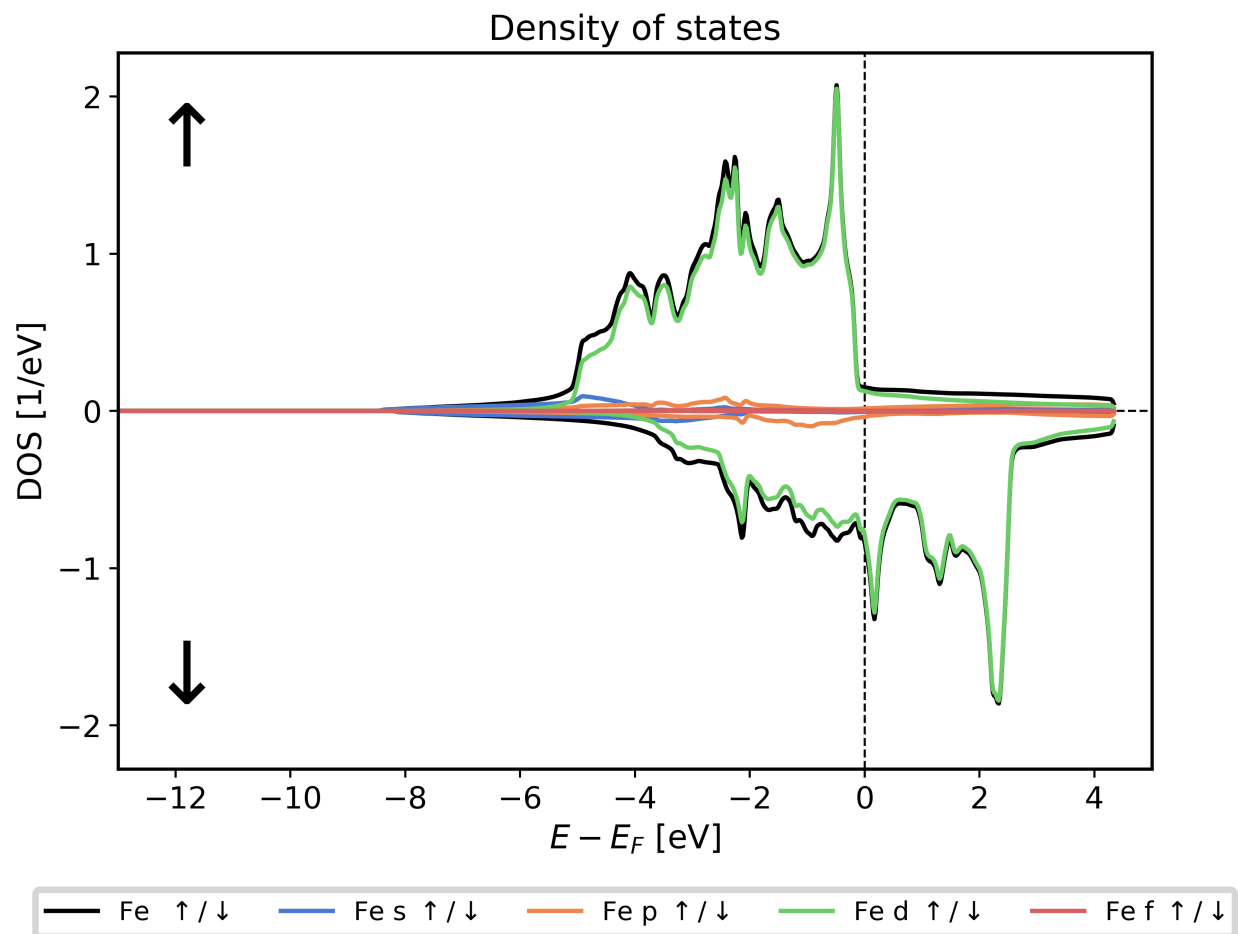


Fig. 12: Non spinpolarized DOS for a bulk Fe fcc structure. Only the atom and l-channel projected DOS is shown

(continued from previous page)

```
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,
                    limits={'energy': (-9,4)},
                    spinpol=False)
```

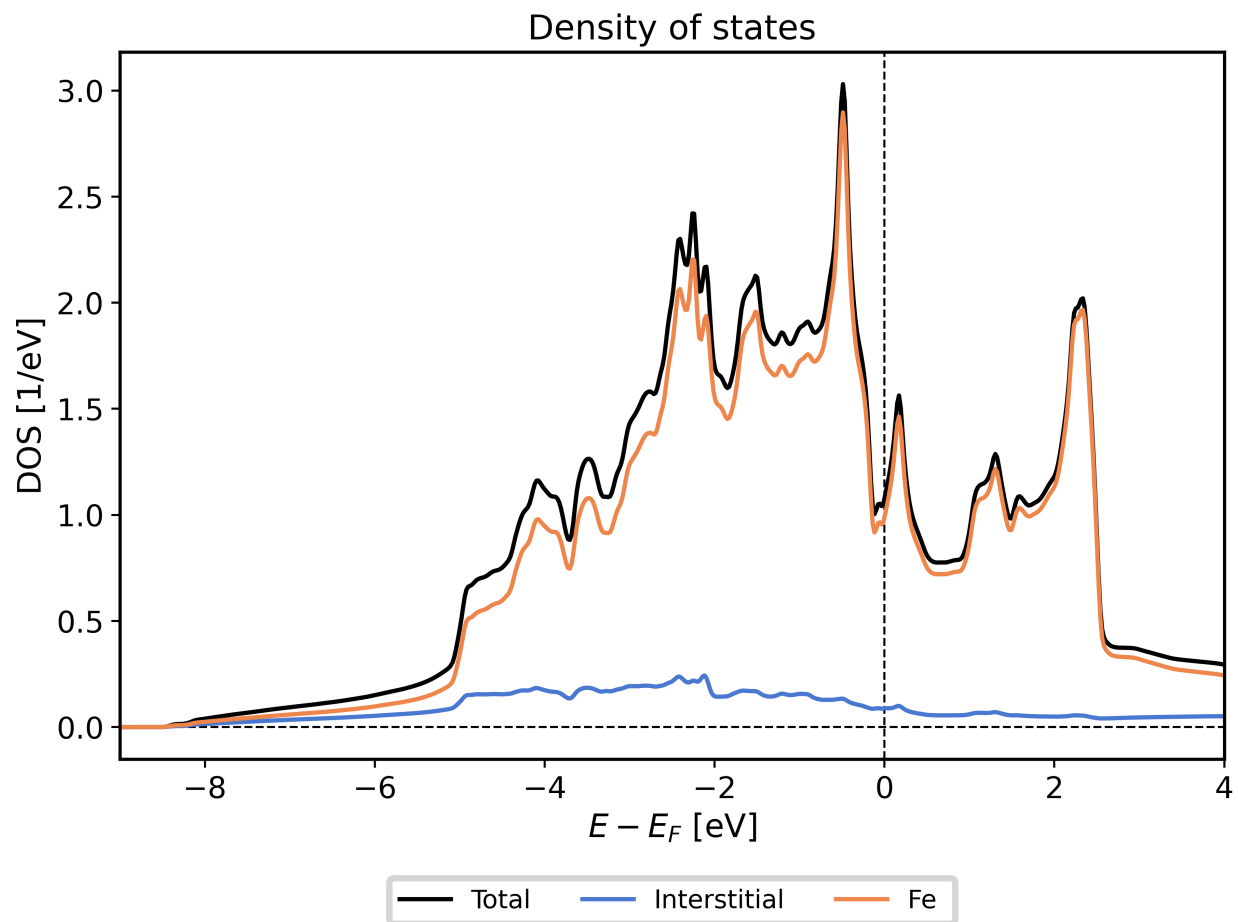


Fig. 13: Non spinpolarized DOS for a bulk Fe fcc structure.

Selecting a specific spin channel

Providing `only_spin='up'` or `'down'` will plot only the given spin channel

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

#Read in data
with HDF5Reader('files/banddos_spinpol_dos.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,
                    limits={'energy': (-9,4)},
                    only_spin='up')
```

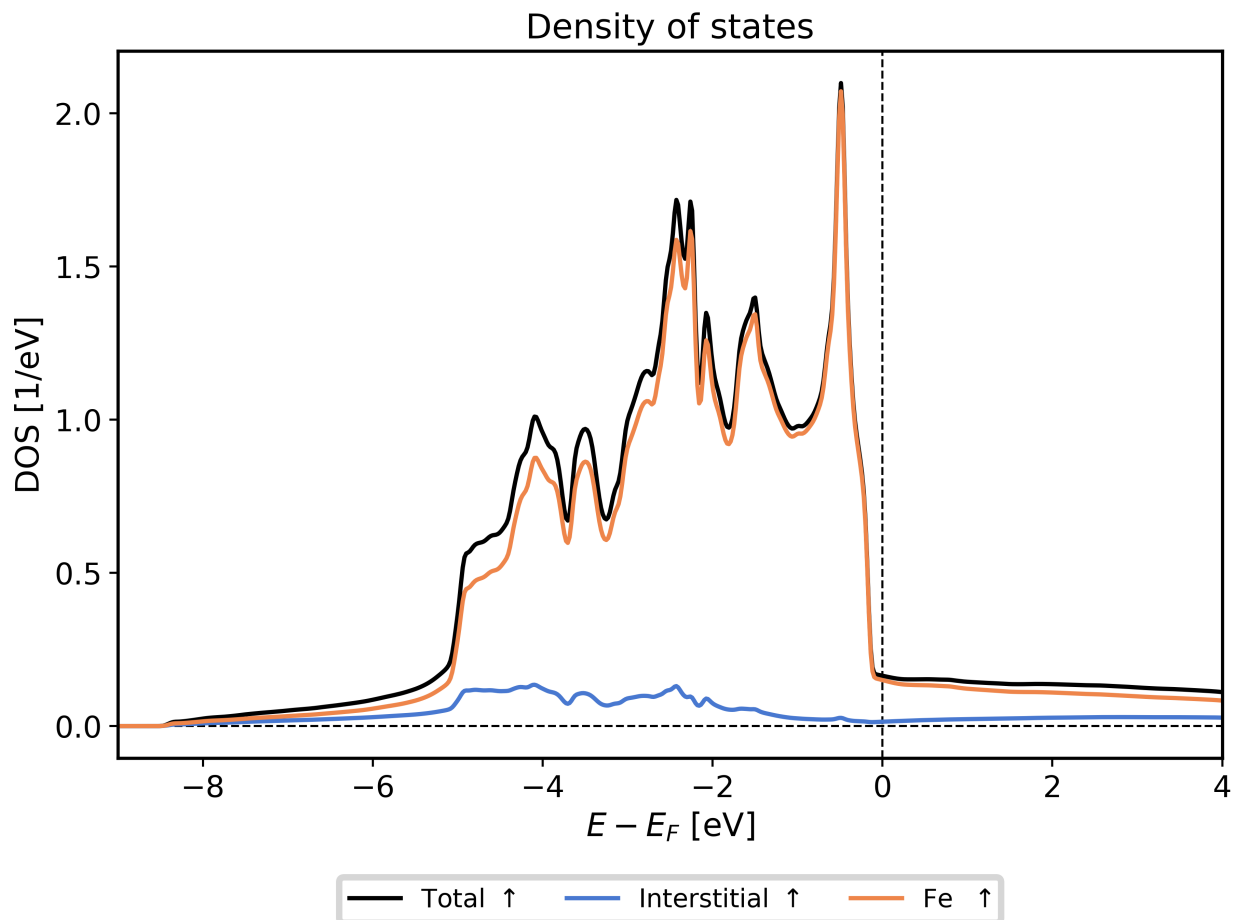


Fig. 14: DOS for a bulk Fe fcc structure (only spin up).

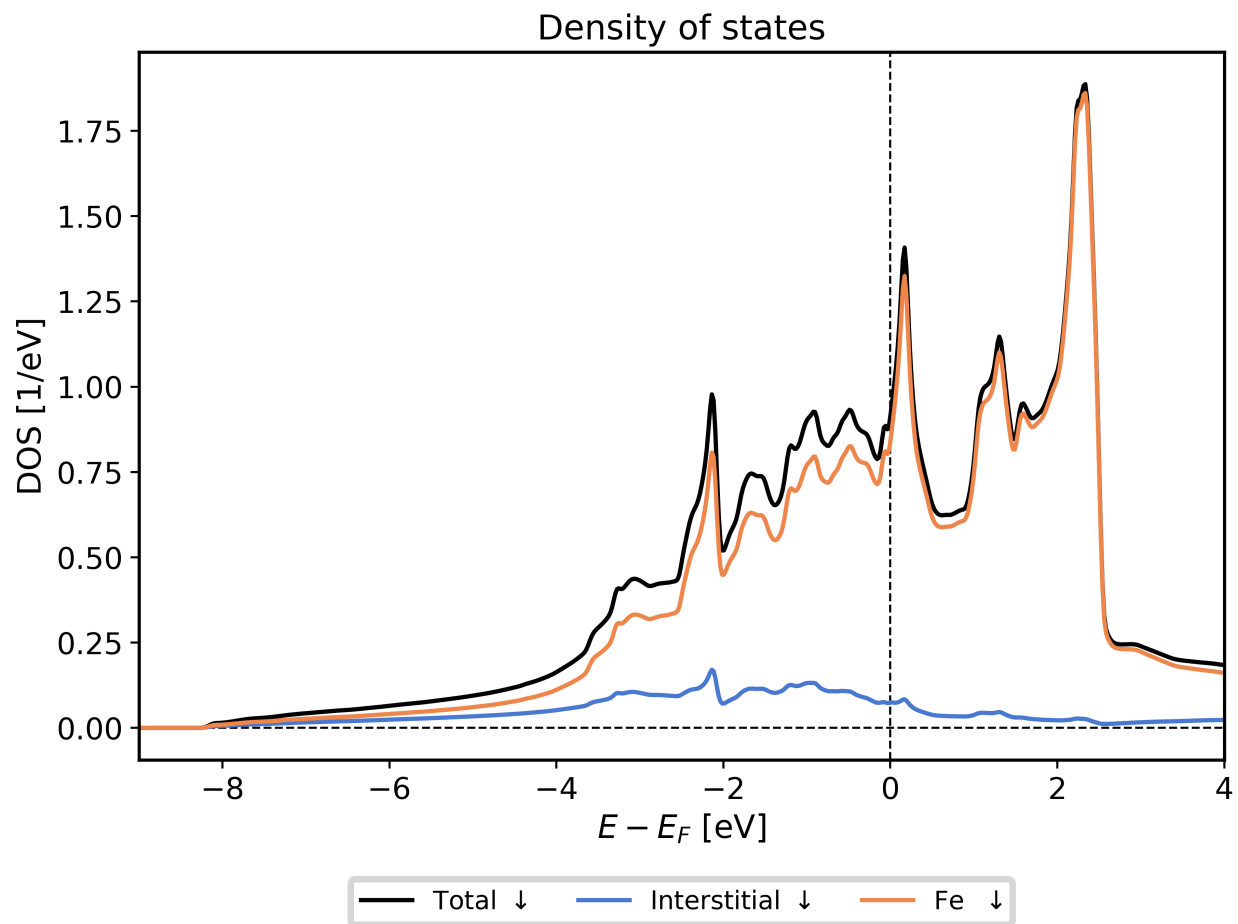


Fig. 15: DOS for a bulk Fe fcc structure (only spin down).

4.1.6 Plotting KKR DOS/bandstructures

This section discusses how the standard output files for density of states and bandstructure calculations of a KKR calculation can be visualized.

Contents

- *Plotting KKR DOS/bandstructures*
 - *Density of states*
 - *Bandstructure*
 - * *Fermi surface*

4.1.6.1 Density of states

```
#Example: KKR DOS

from masci_tools.vis.kkr_plot_dos import dosplot

# the path can be a relative or absolute path to the directory
# where the dos.atom files reside (i.e. the dir where the DOS calculation ran)
dosplot('files/kkr_dos/', color='k', lw=4, marker='v', ls=':', ms=8)
```

```
files/kkr_dos/dos.atom1
```

Where the color, lw, etc inputs are optional settings which customize the plot:

We can also use this to show the l-decomposed DOS (red line are d-orbitals):

```
#Example: KKR DOS, l-resolved

dosplot('files/kkr_dos/', totnonly=False)
```

```
files/kkr_dos/dos.atom1
```

4.1.6.2 Bandstructure

```
#Example: KKR bandstructure

from masci_tools.vis.kkr_plot_bandstruc_qdos import dispersionplot

# the path can be a relative or absolute path to the directory
# where the qdos files reside (i.e. the dir where the qdos calculation ran)
dispersionplot('files/kkr_bandstruc/', ptitle='bulk Cu')
```

Which can also be customized with keyword arguments to the dispersionplot function:

```
#Example: KKR bandstructure with custom color map
```

(continues on next page)

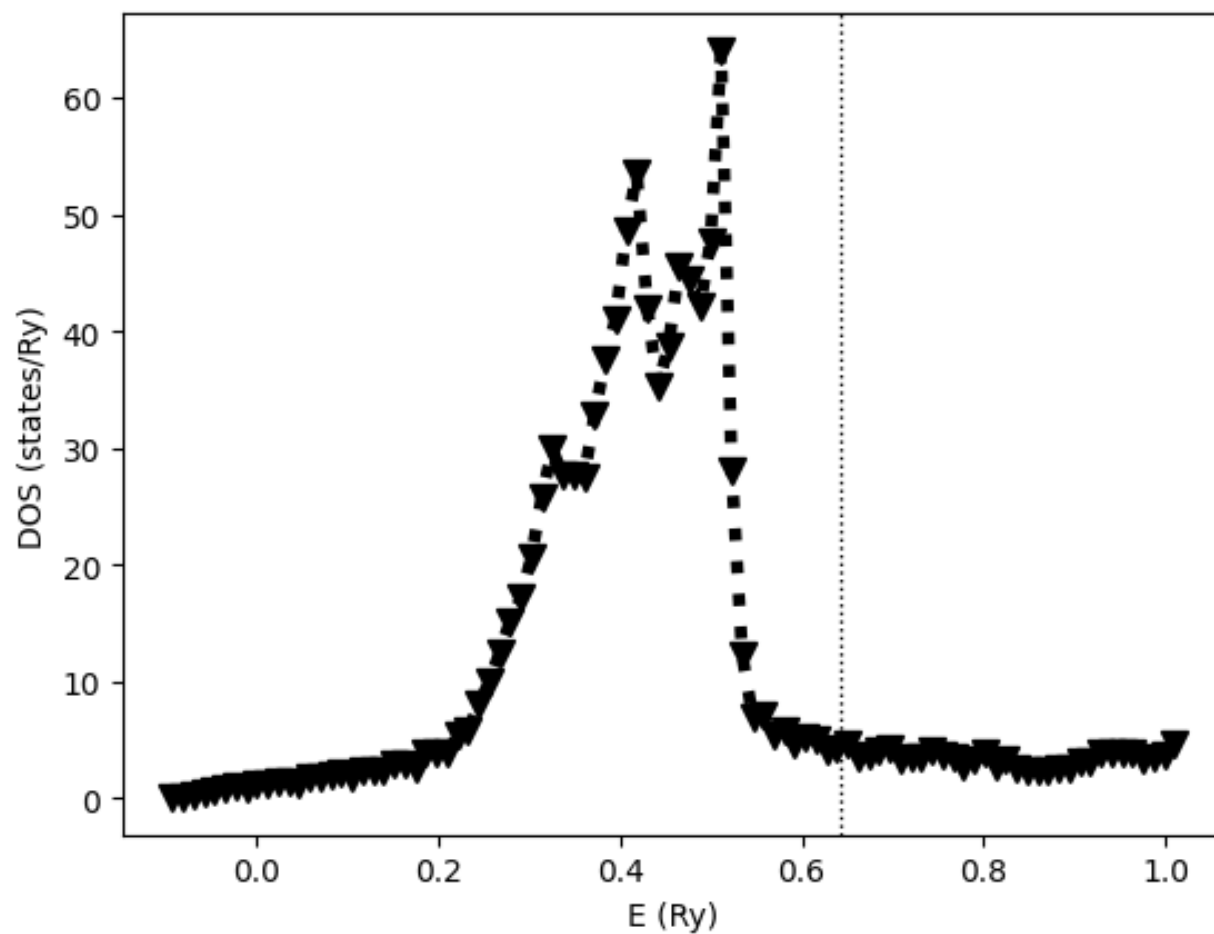


Fig. 16: DOS of bulk fcc Cu.

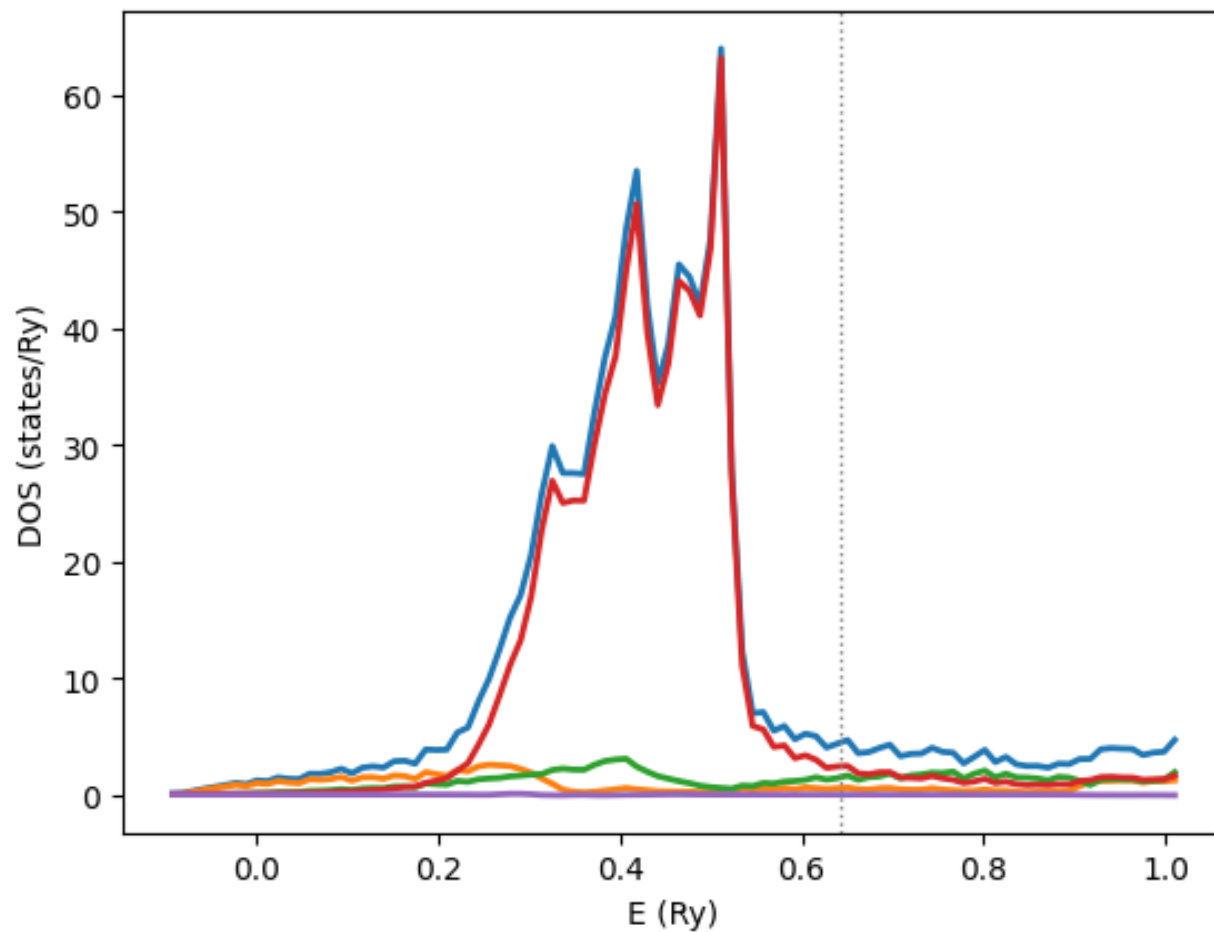


Fig. 17: DOS of bulk fcc Cu, l-resolved.

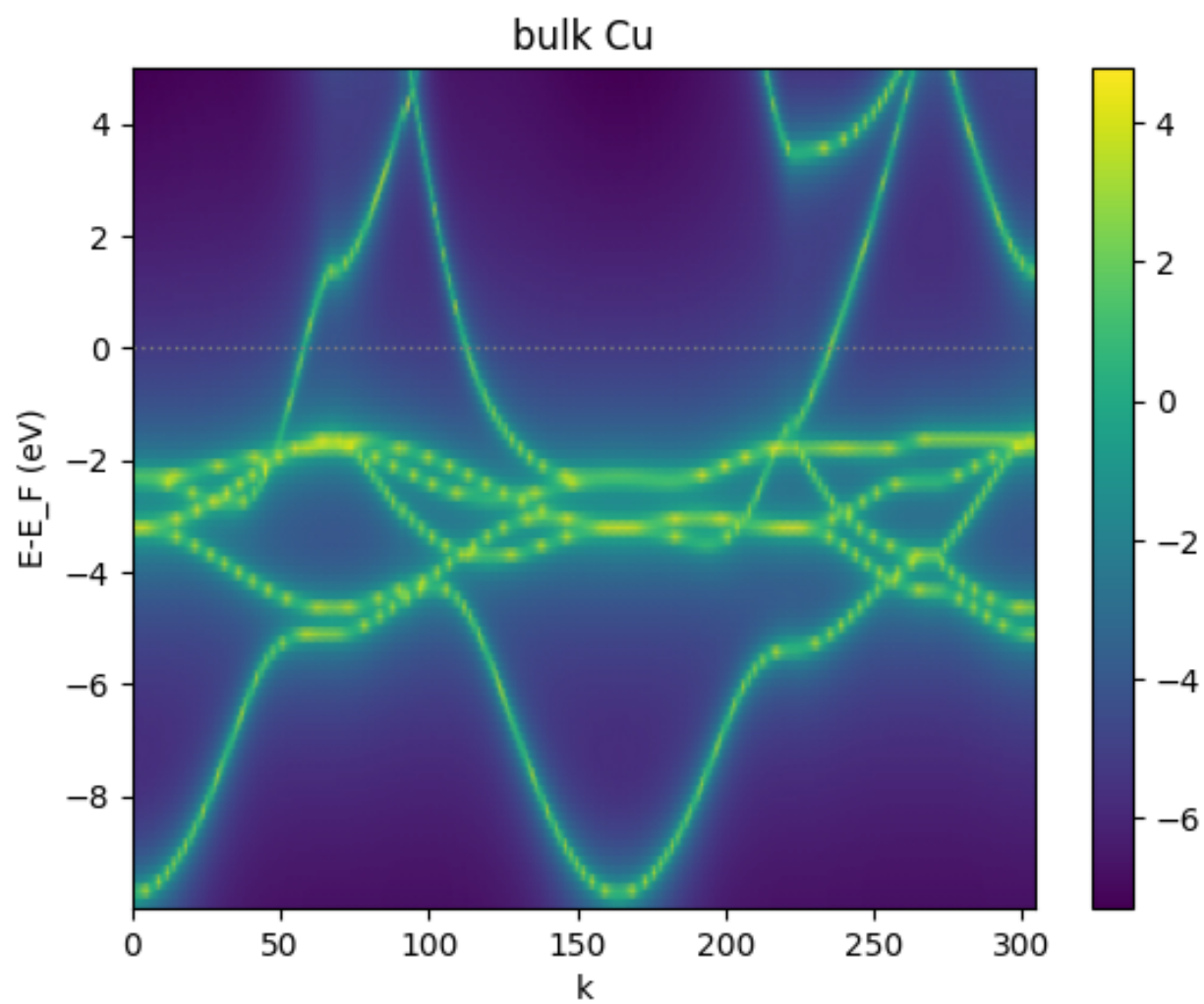


Fig. 18: Bandstructure of bulk fcc Cu.

(continued from previous page)

```
dispersionplot('files/kkr_bandstruc/', ptitle='bulk Cu', cmap='binary', clim=[-2,2],
               clrbar=False)
```

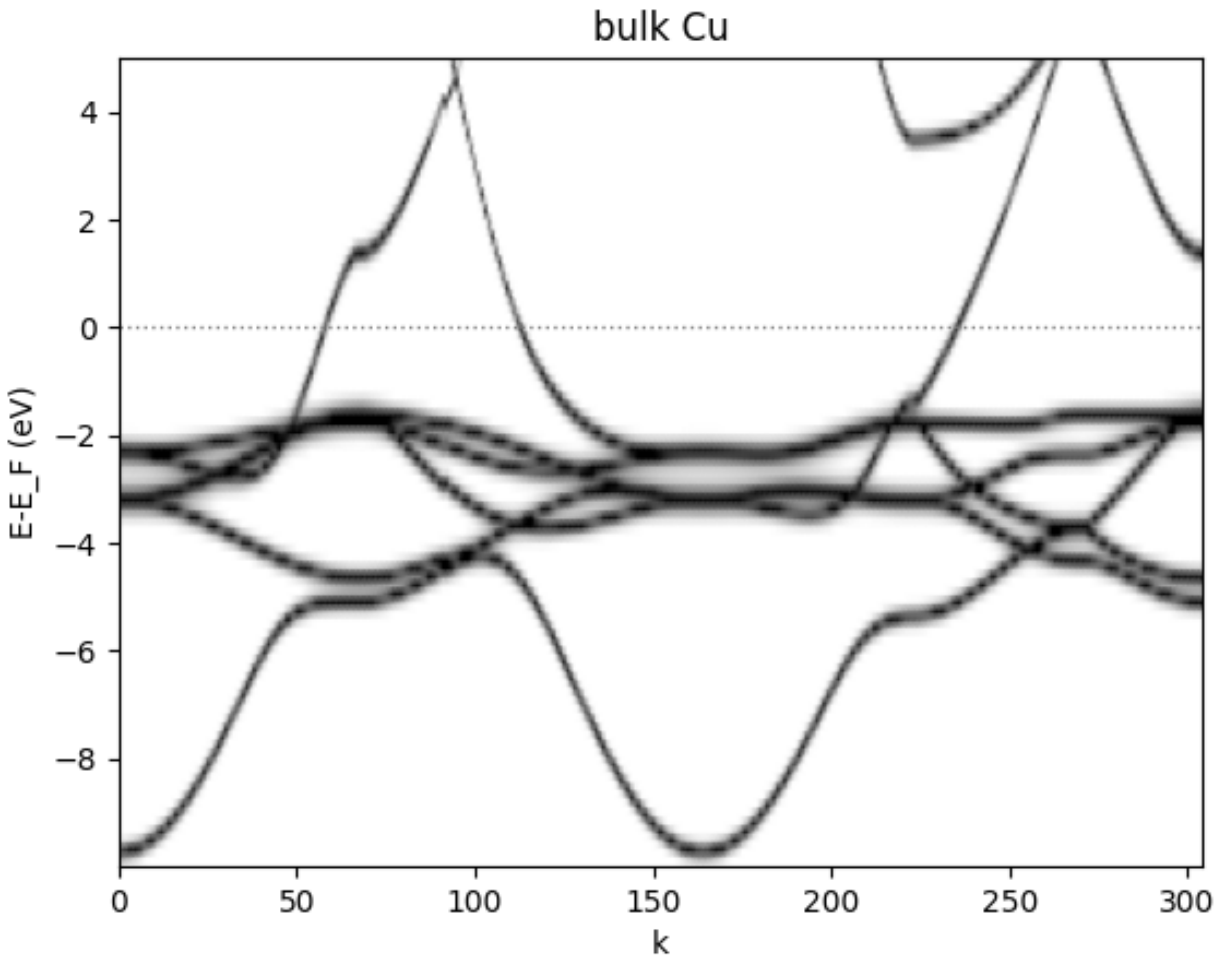


Fig. 19: Bandstructure of bulk fcc Cu.

Fermi surface

Constant energy contours can be calculated by using a single energy point in a qdos calculation with a 2D k-point grid defined in the `qvec.dat` input file to KKRhost. For example, this can then be used to visualize a cut through the Fermi surface of a material.

```
#Example: KKR bandstructure

from masci_tools.vis.kkr_plot_FS_qdos import FSqdos2D

# the path can be a relative or absolute path to the directory
# where the qdos files reside (i.e. the dir where the qdos calculation ran)
FSqdos2D('PATH/TO/OUTPUT-FILES/')
```

4.1.7 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

4.1.7.1 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 of 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()`: Plot the convergence behaviour of charge density distances and energies
- `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)
- `plot_spectral_function()`: Plot a spectral function (colormesh along kpath)

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)
- `bokeh_spectral_function()`: Plot a spectral function (colormesh along kpath)
- `periodic_table_plot()`: Make a interactive plot of data for the periodic table
- `plot_lattice_constant()`: Plot the energy curve with changing unit cell volume
- `plot_convergence()`: Plot the convergence behaviour of charge density distances and energies
- `matrix_plot()`: Plot a grid of rectangles with color scaling and added text

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.

4.1.7.2 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
```

(continues on next page)

(continued from previous page)

```
y2 = 20*np.sin(x)
data = {'x': x, 'y1': y1, 'y2': y2}

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

4.1.7.3 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 [matplotlib_plotter.MatplotlibPlotter](#) and [bokeh_plotter.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 do 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 `plot_methods.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

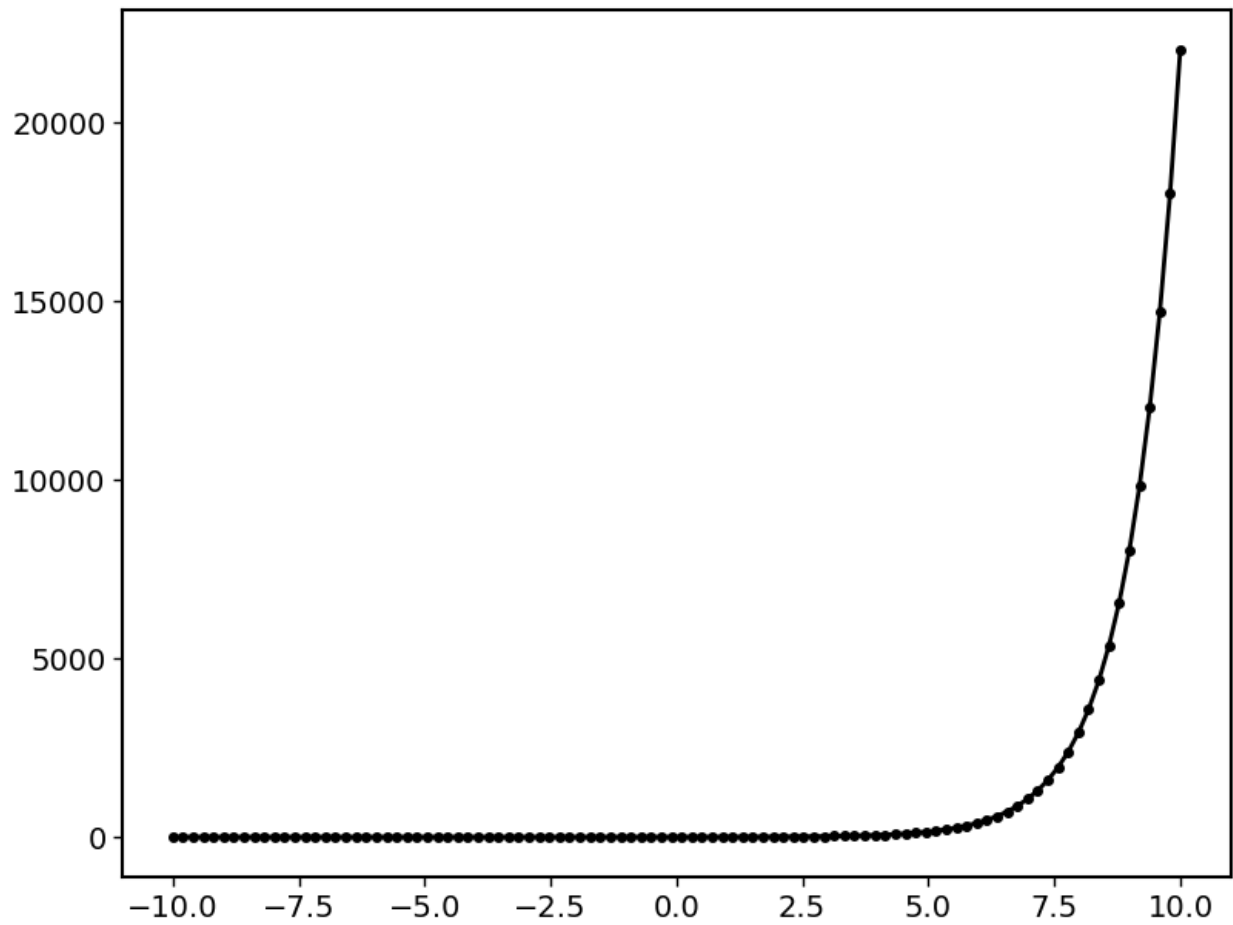
```
from masci_tools.vis.plot_methods import single_scatterplot
import numpy as np

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

(continues on next page)

(continued from previous page)

```
ax = single_scatterplot(x,y)
```

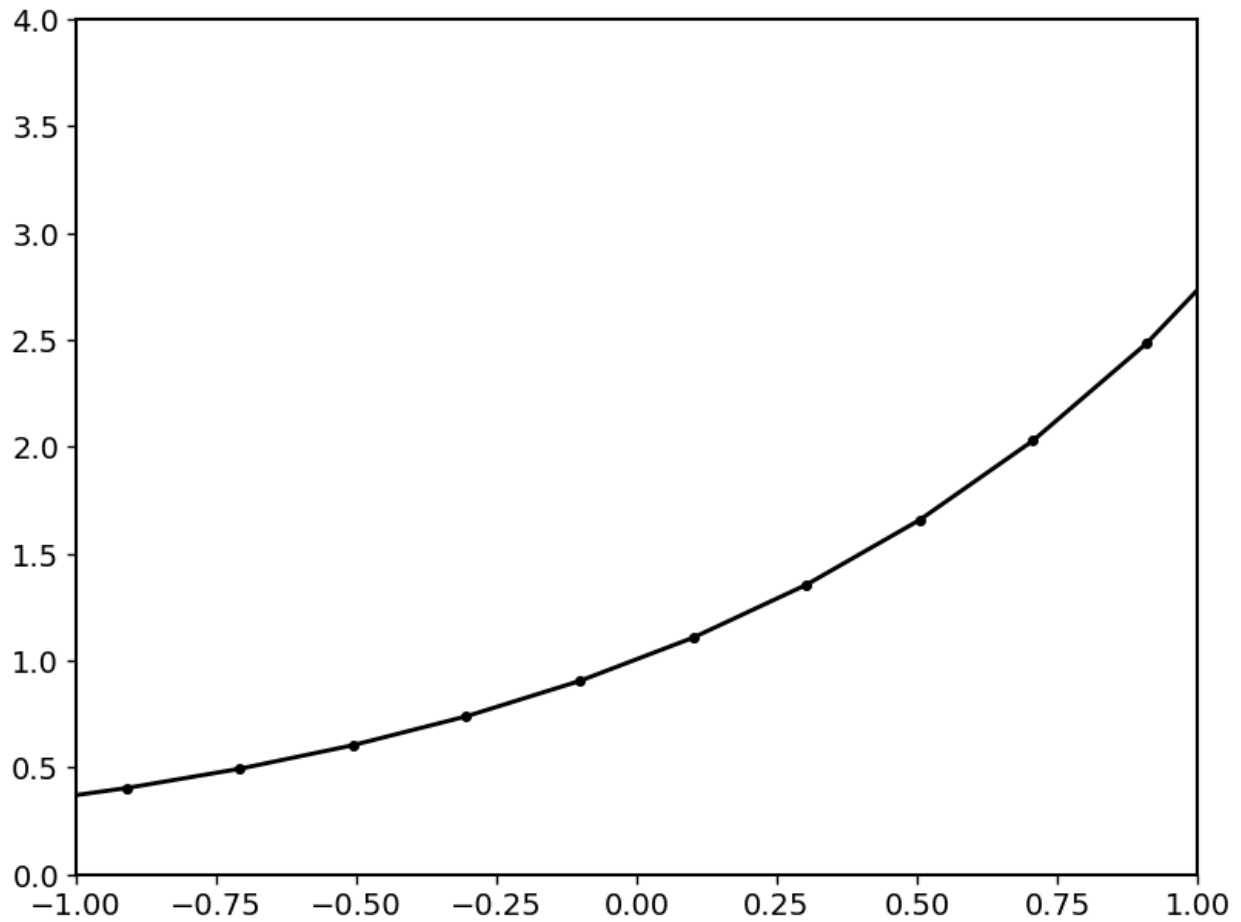


Setting limits

```
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)})
```

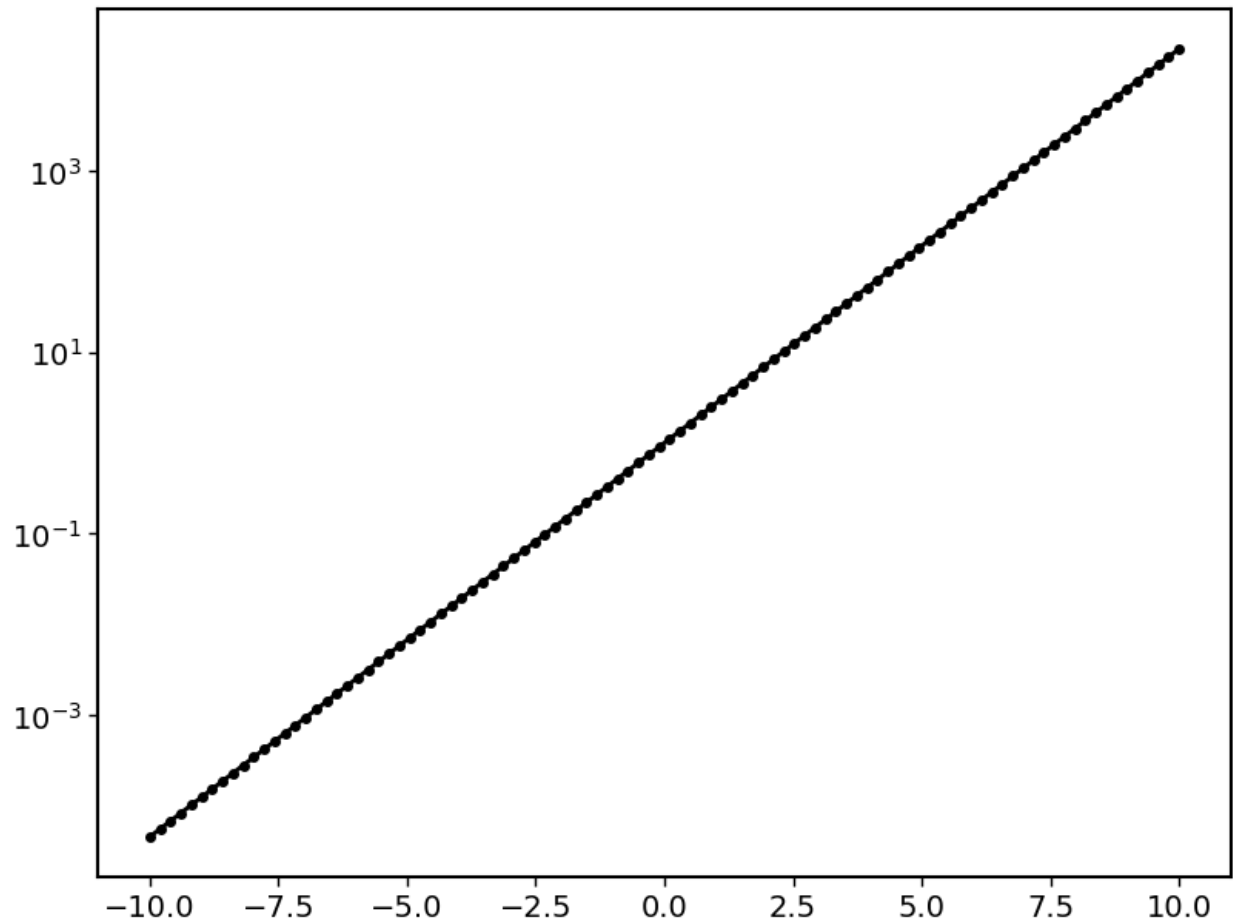


Modifying the scale of the axis

```
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'})
```

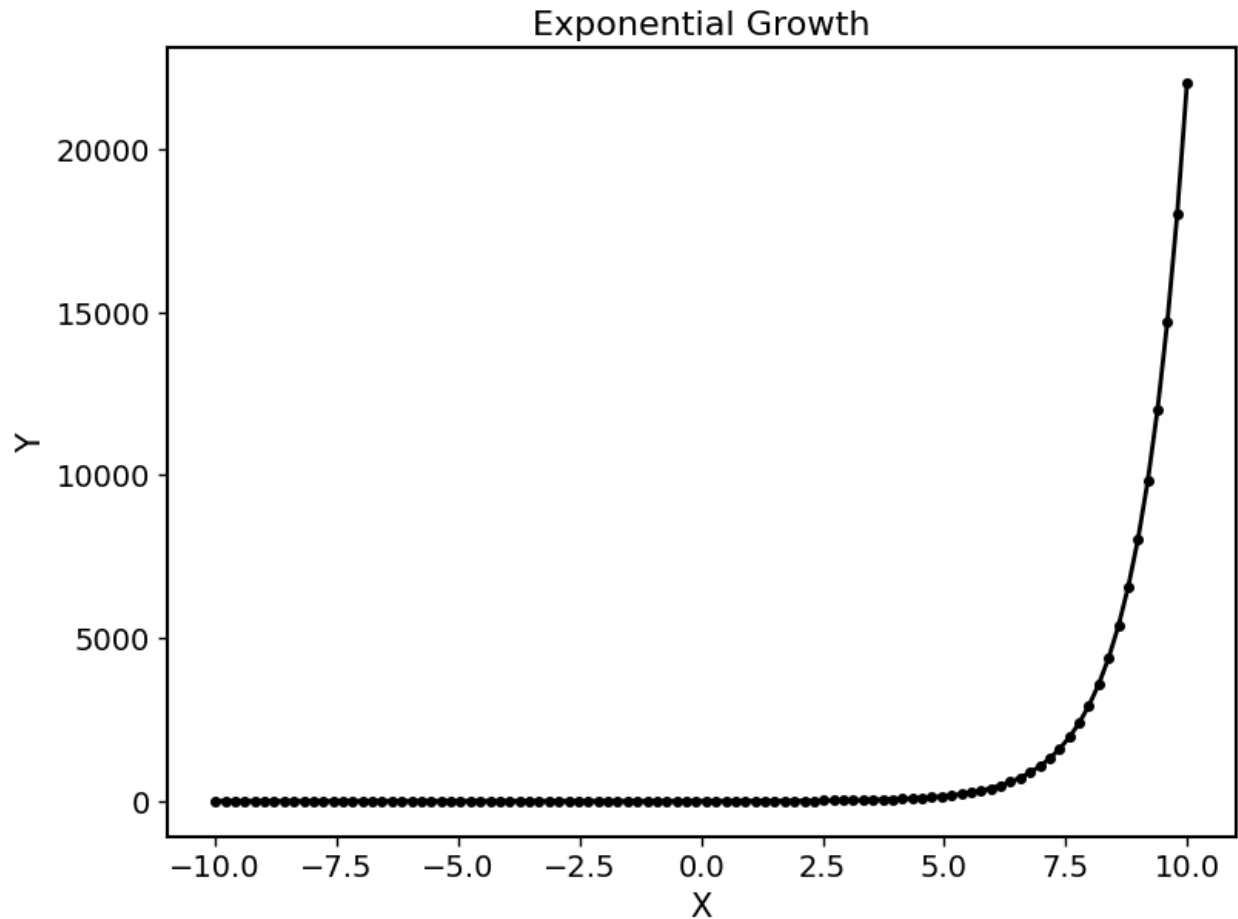


Setting labels on the axis and a title

```
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')
```



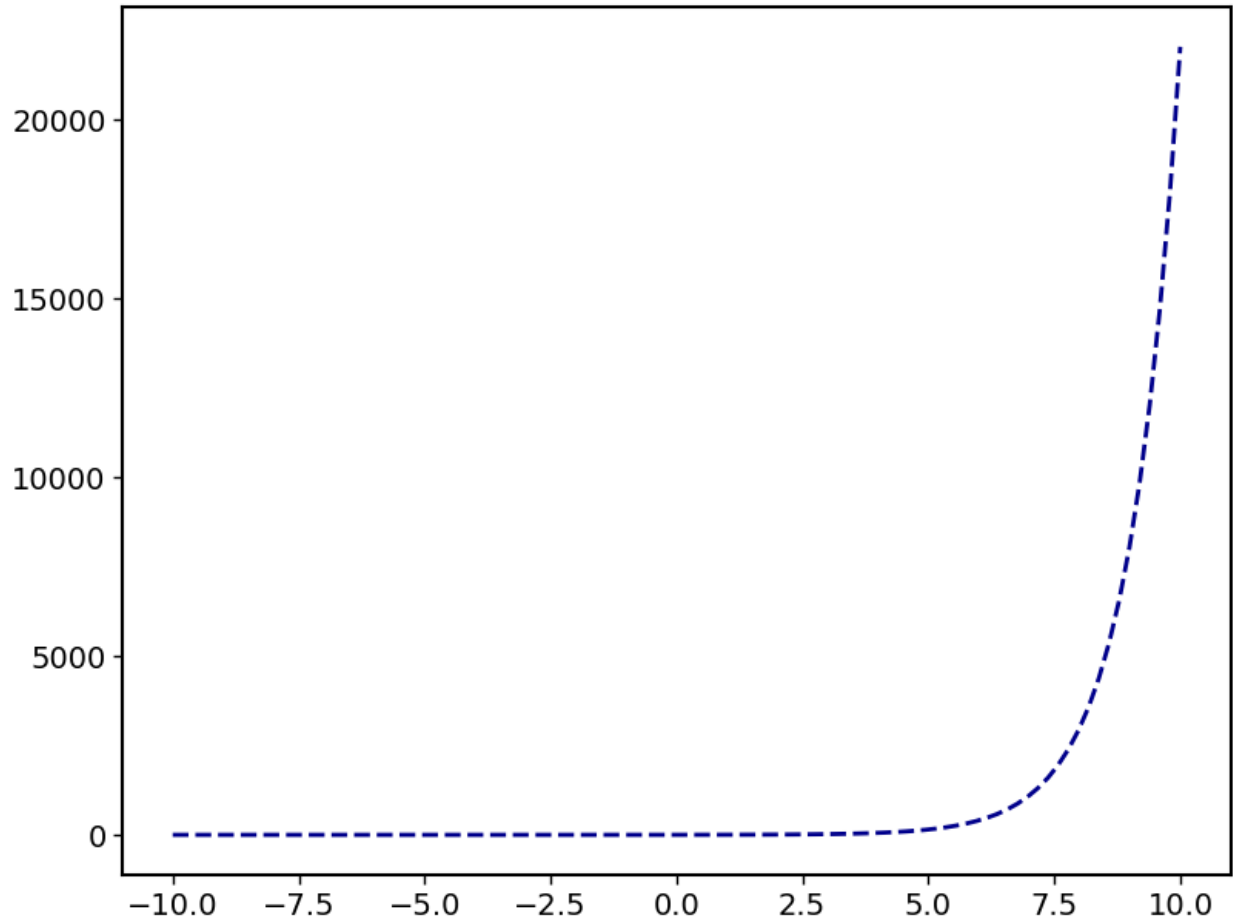
Modifying plot parameters

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

```
from 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, 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 `plot_methods.set_mpl_plot_defaults()` or `bokeh_plots.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 `plot_methods.reset_mpl_plot_defaults()` or `bokeh_plots.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

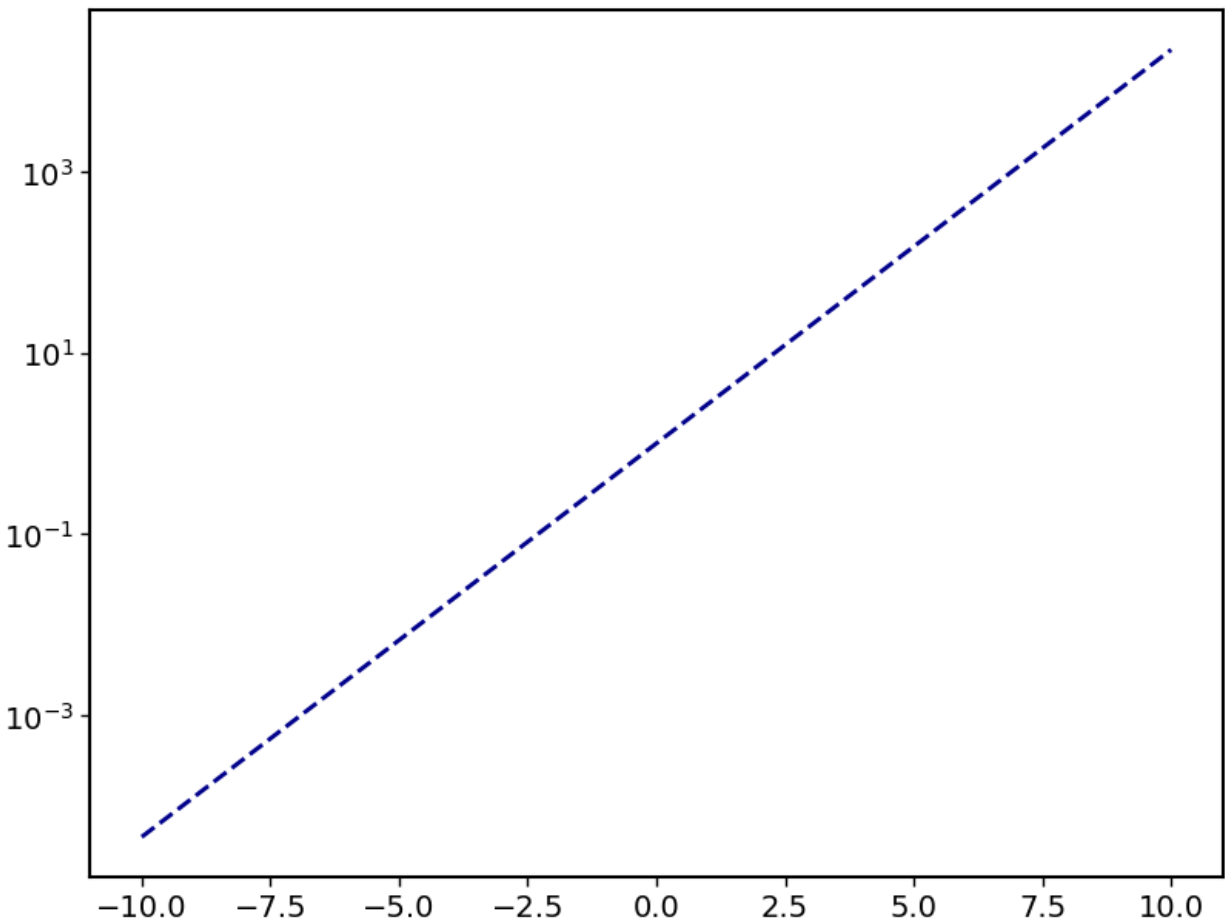
x = np.linspace(-10, 10, 100)
y = np.exp(x)

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

(continues on next page)

(continued from previous page)

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



Resetting defaults:

```
from maschi_tools.vis.plot_methods import reset_mpl_plot_defaults
reset_mpl_plot_defaults()
```

Multiple plots

Many plotting routines accept multiple sets of data to plot. An example of this is the `plot_methods.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:'--'}`

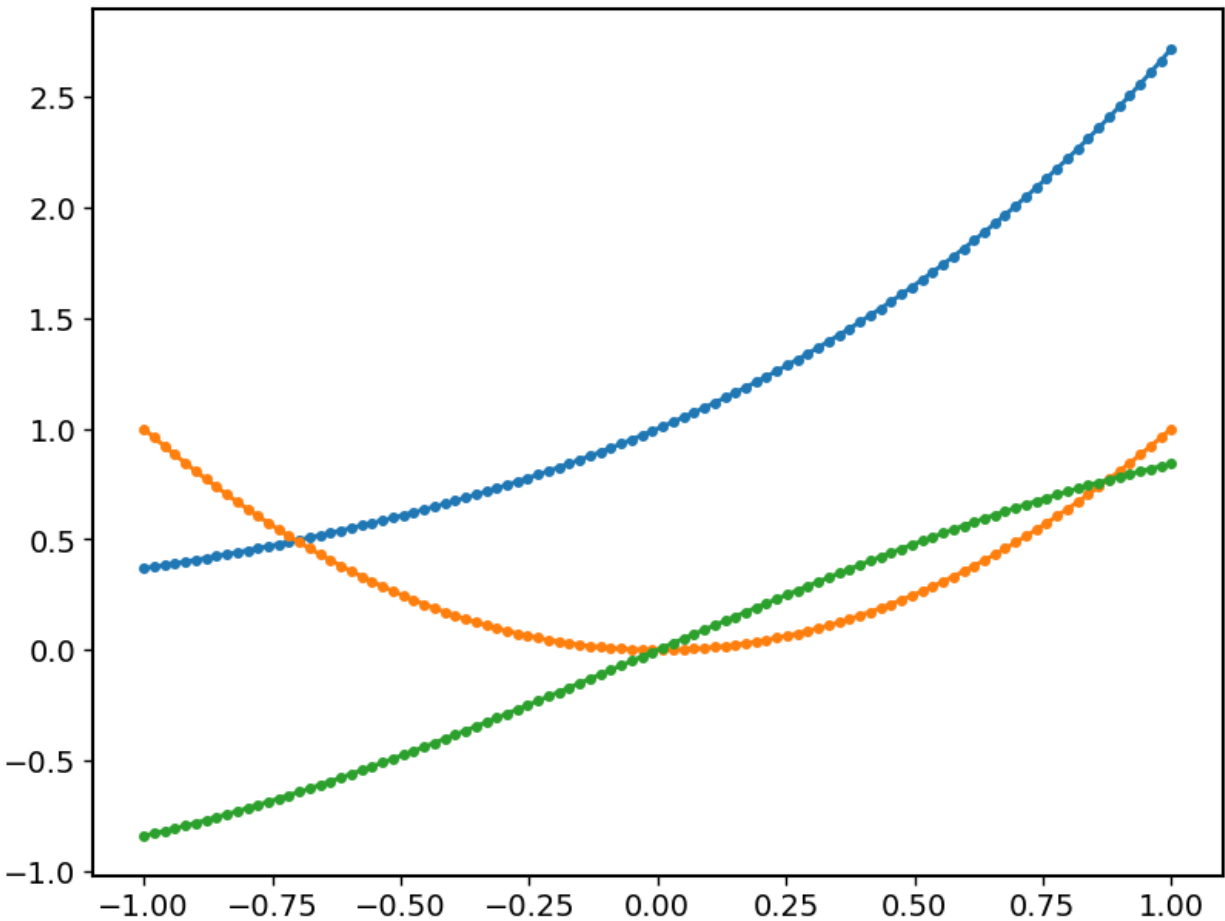
Warning: Specifying parameters for multiple data sets is only valid for the parameters passed into the function. Setting defaults with values for multiple data sets is not supported

Default plot

```
from 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])
```



Changing parameters on all plots

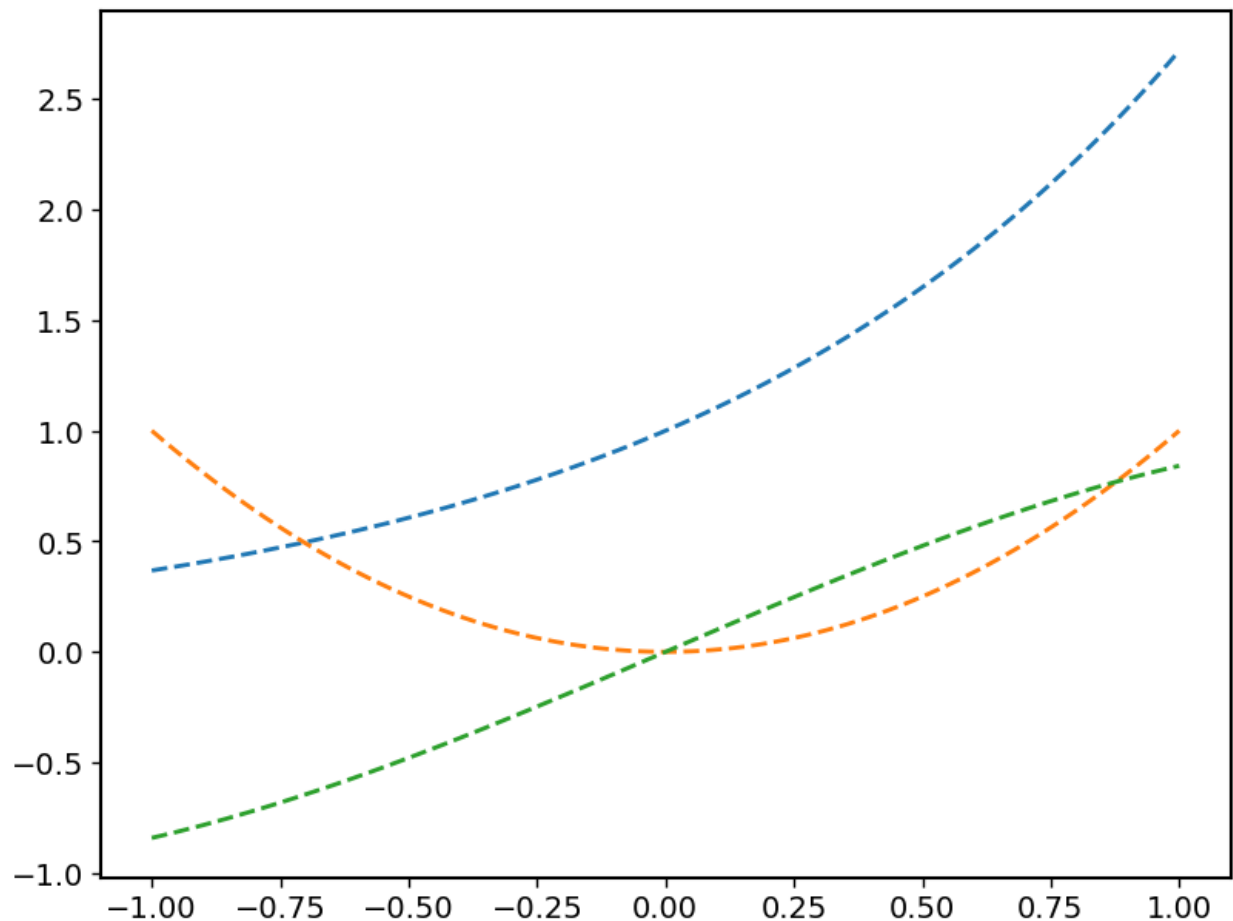
```

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)

```



Changing parameters on specific plots

```

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

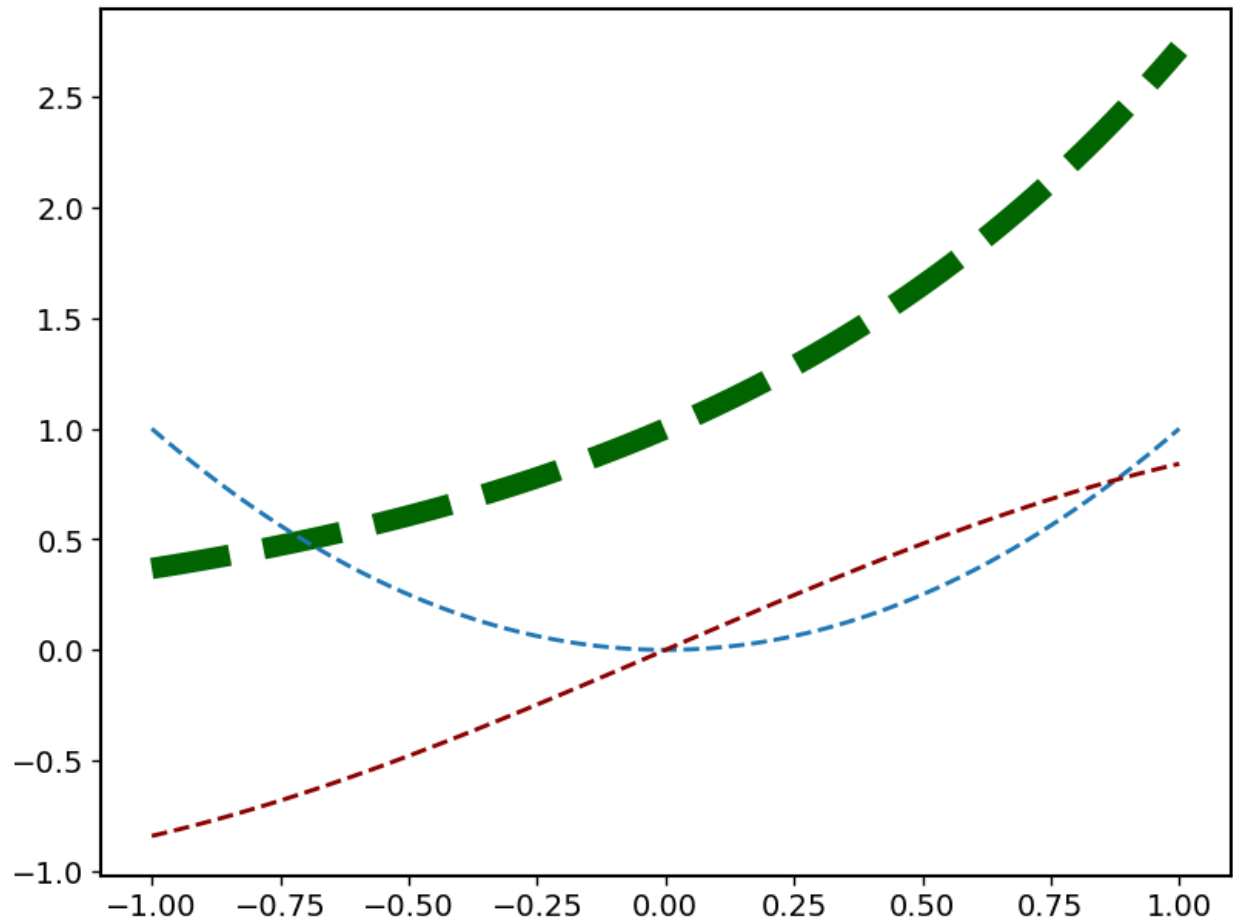
x = np.linspace(-1,1,100)
y = np.exp(x)
y2 = x**2
y3 = np.sin(x)

```

(continues on next page)

(continued from previous page)

```
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 masci-tools

5.1.1 Test suite of masci-tools

5.1.1.1 Installation

Install the package with the `testing` extra:

```
pip install -e .[testing]
```

5.1.1.2 Running the tests

Run pytest in the tests path of masci-tools:

```
cd tests  
pytest .
```

5.1.1.3 Regenerating test files

With some code changes an update of the reference files for the tests is required. This can be done with the `--force-regen` option:

```
pytest --force-regen .
```

5.1.2 Updating or adapting the Fleur Parsers

Note: A more detailed walkthrough of the important design decisions concerning the implementation Fleur XML parsers and IO capabilities can be found in this [document](#)

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 `InputSchemaDict.fromVersion()` or `OutputSchemaDict.fromVersion()` methods by providing the desired version string.

5.1.2.1 Adding/modifying a Fleur Schema:

The command `masci-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.

5.1.2.2 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

xmlGetter

Will execute a function in the module `xml_getters` given in the name entry

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 `[('j spins', 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

5.1.2.3 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 `outxml_parser()` to convert between versions. A usage example is shown below.

```
from masci_tools.io.parsers.fleur import register_migration
import copy

@register_migration(base_version='0.33', target_version='0.34')
def migrate_033_to034(definition_dict):
    """
```

(continues on next page)

(continued from previous page)

```

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

```

5.1.3 Using the Plotter class

5.1.3.1 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 *Plotter.set_defaults()* with `default_type='function'`
2. Global defaults set with *Plotter.set_defaults()*
3. Parameters set with *Plotter.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*)

5.1.3.2 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 a 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)

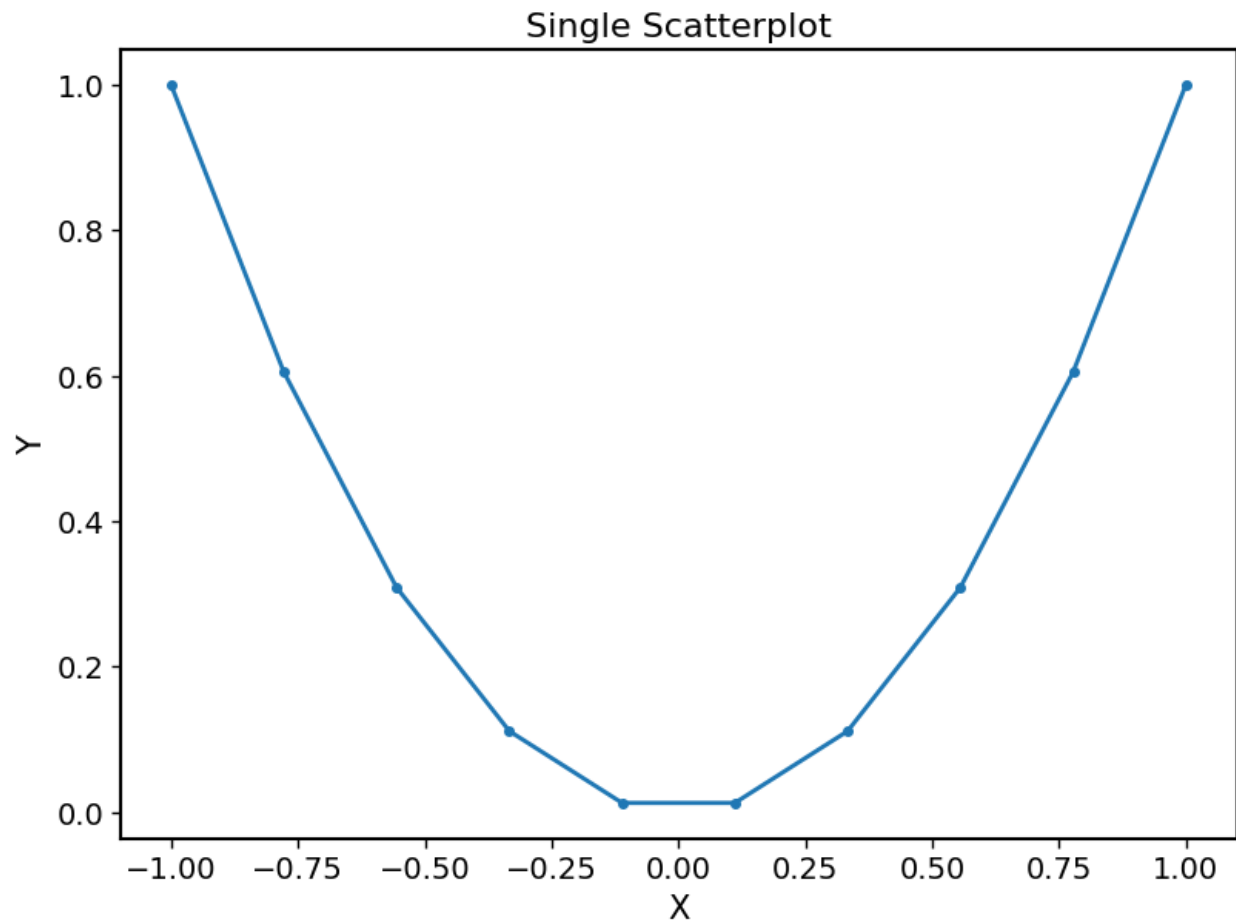
    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)
    plot_params.save_plot('figure')

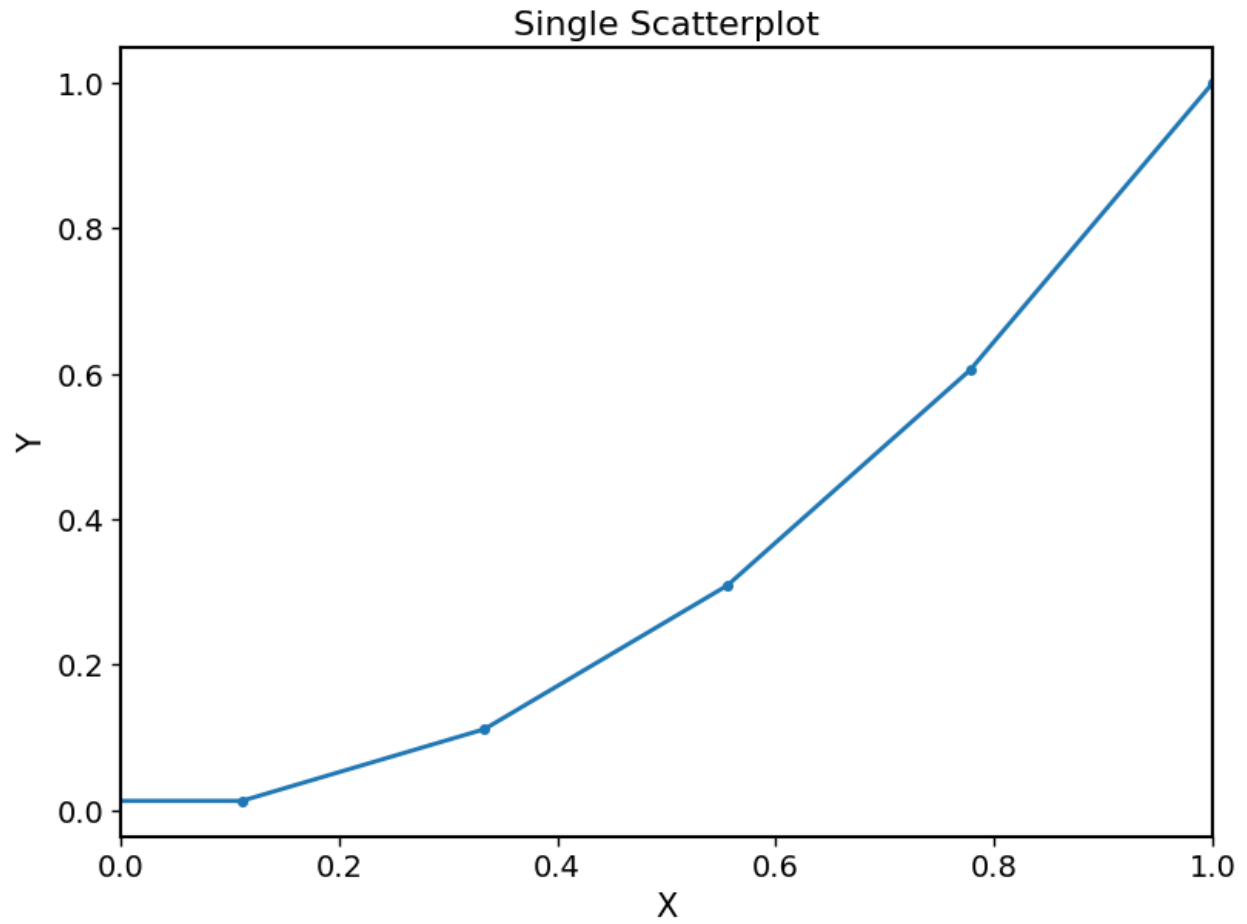
    return ax

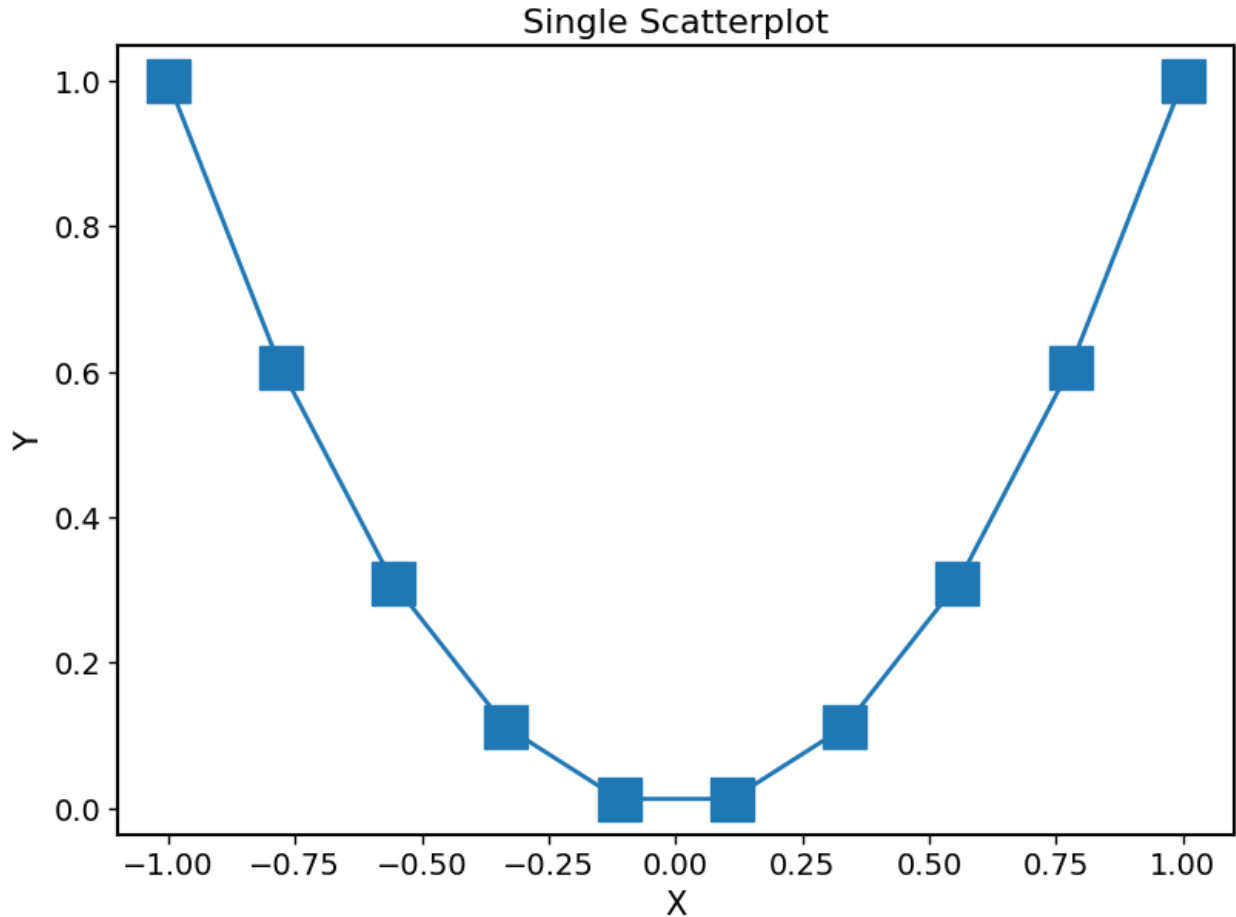
import numpy as np

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)
```







```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

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

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

(continues on next page)

(continued from previous page)

```
#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)

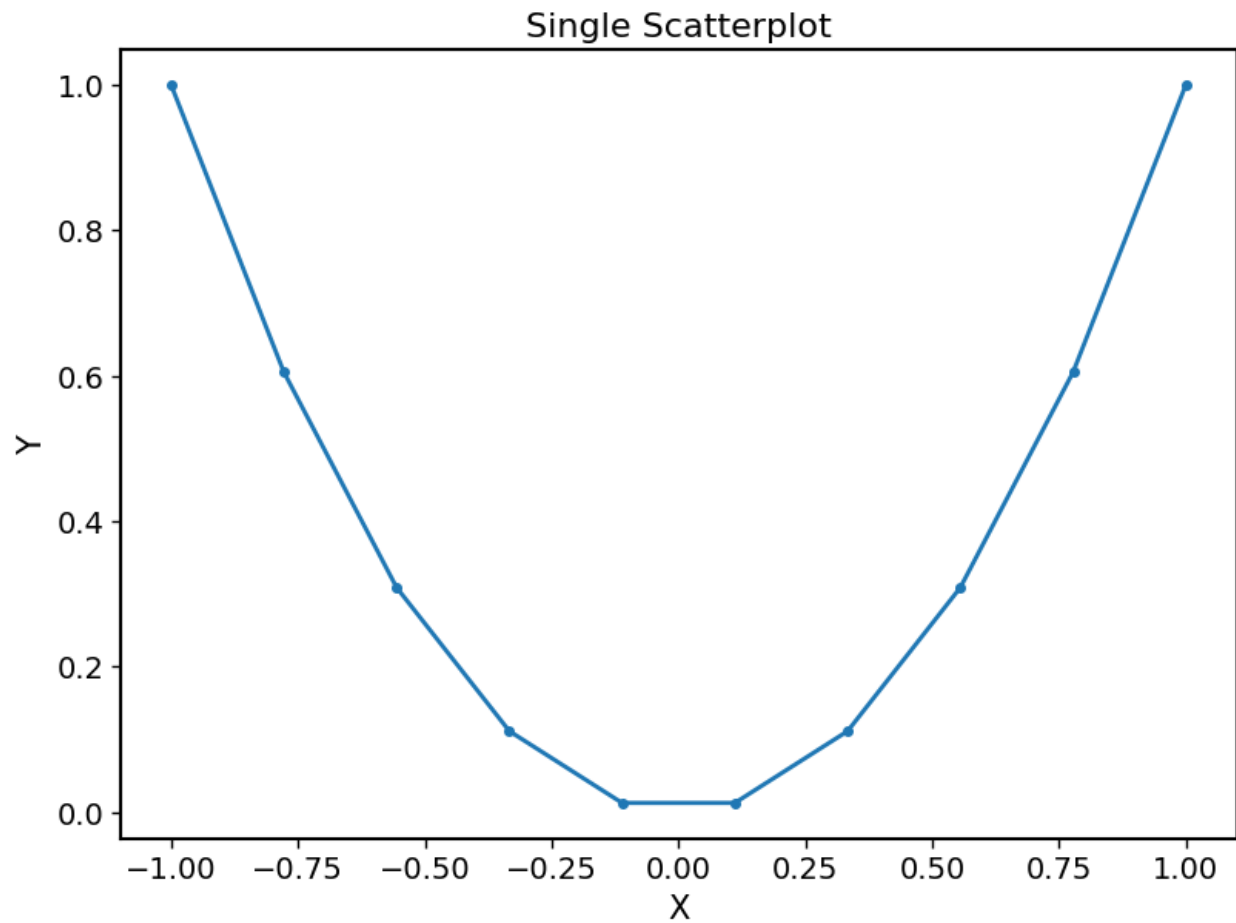
    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)
    plot_params.save_plot('figure')

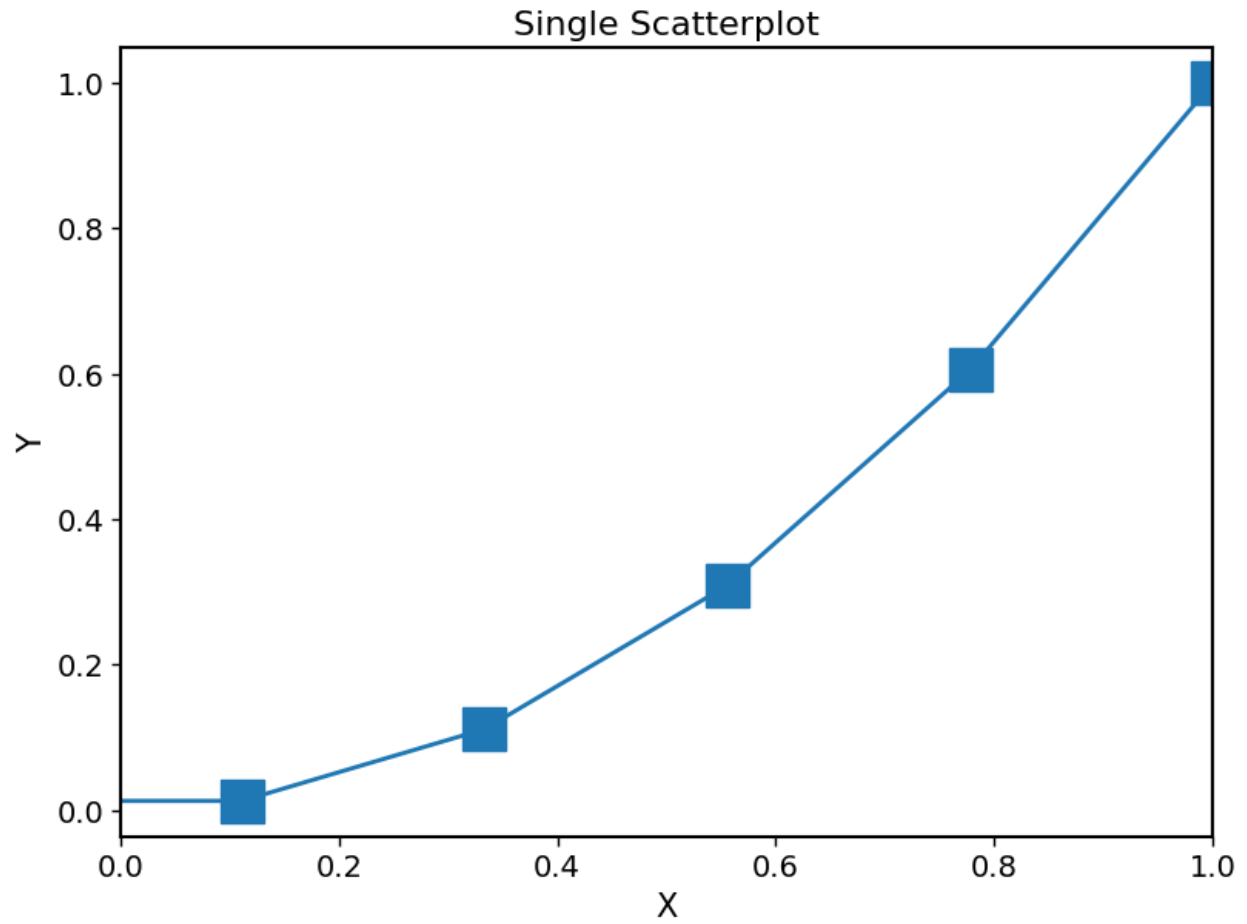
    return ax

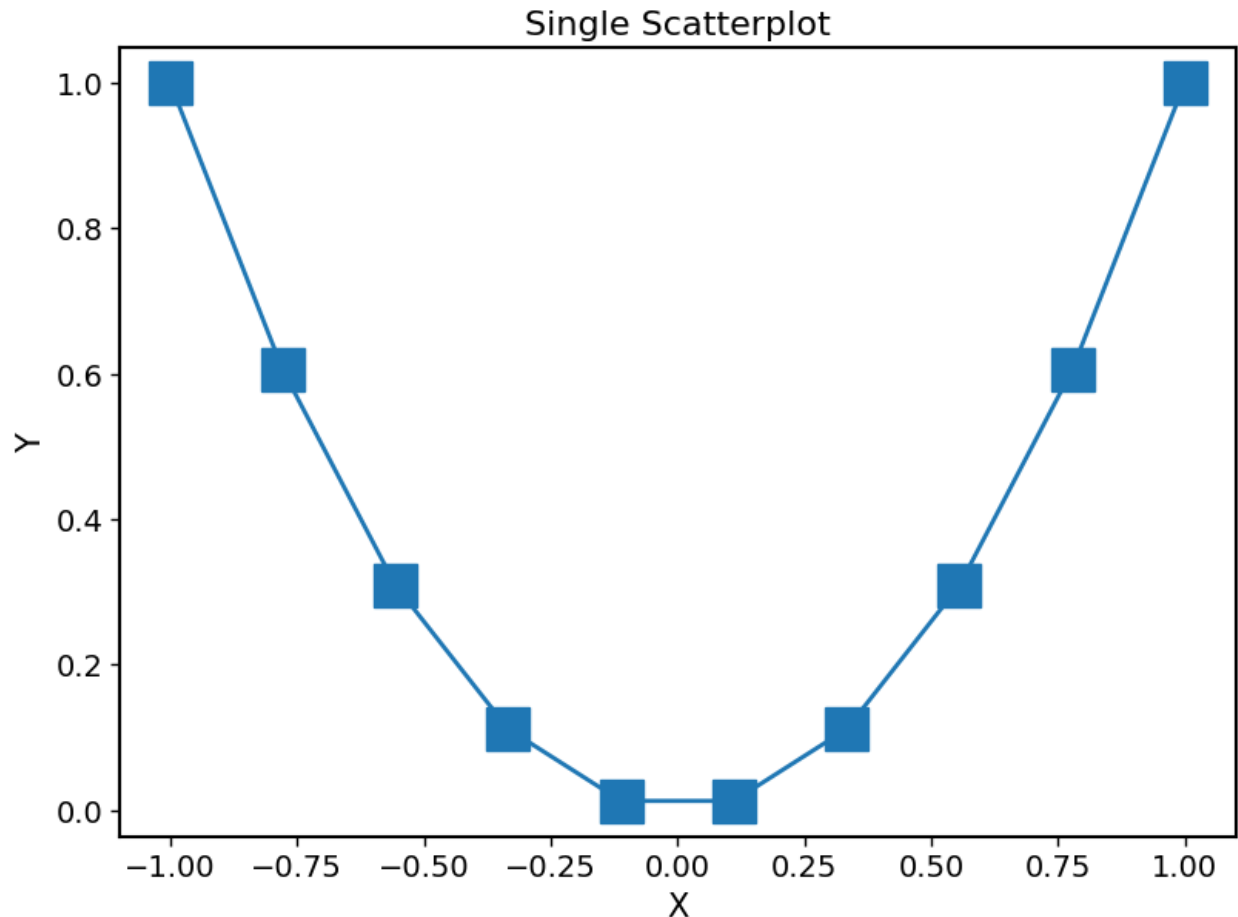
import numpy as np

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

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







```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

The `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.parameters import ensure_plotter_consistency

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

(continues on next page)

(continued from previous page)

```

#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)
    plot_params.save_plot('figure')

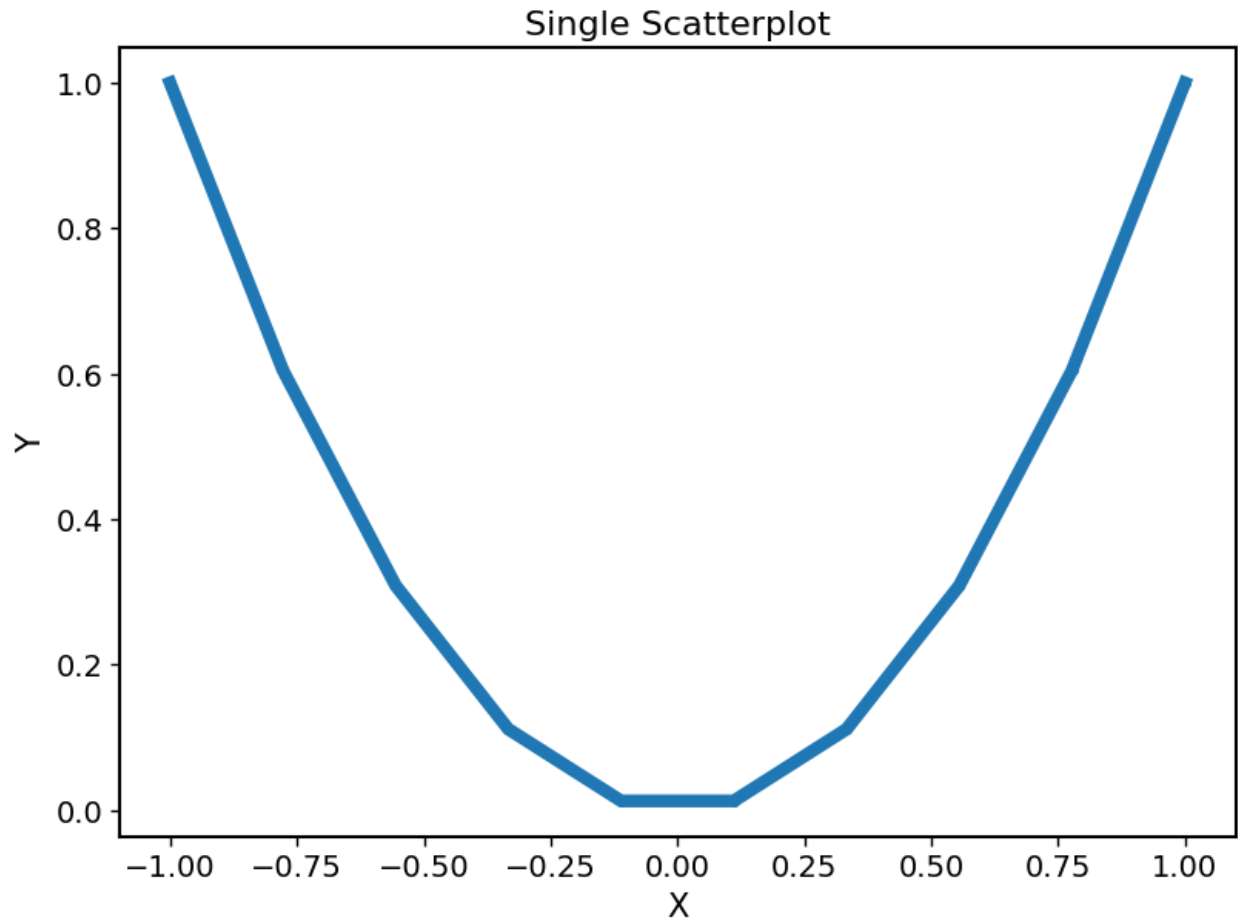
    return ax

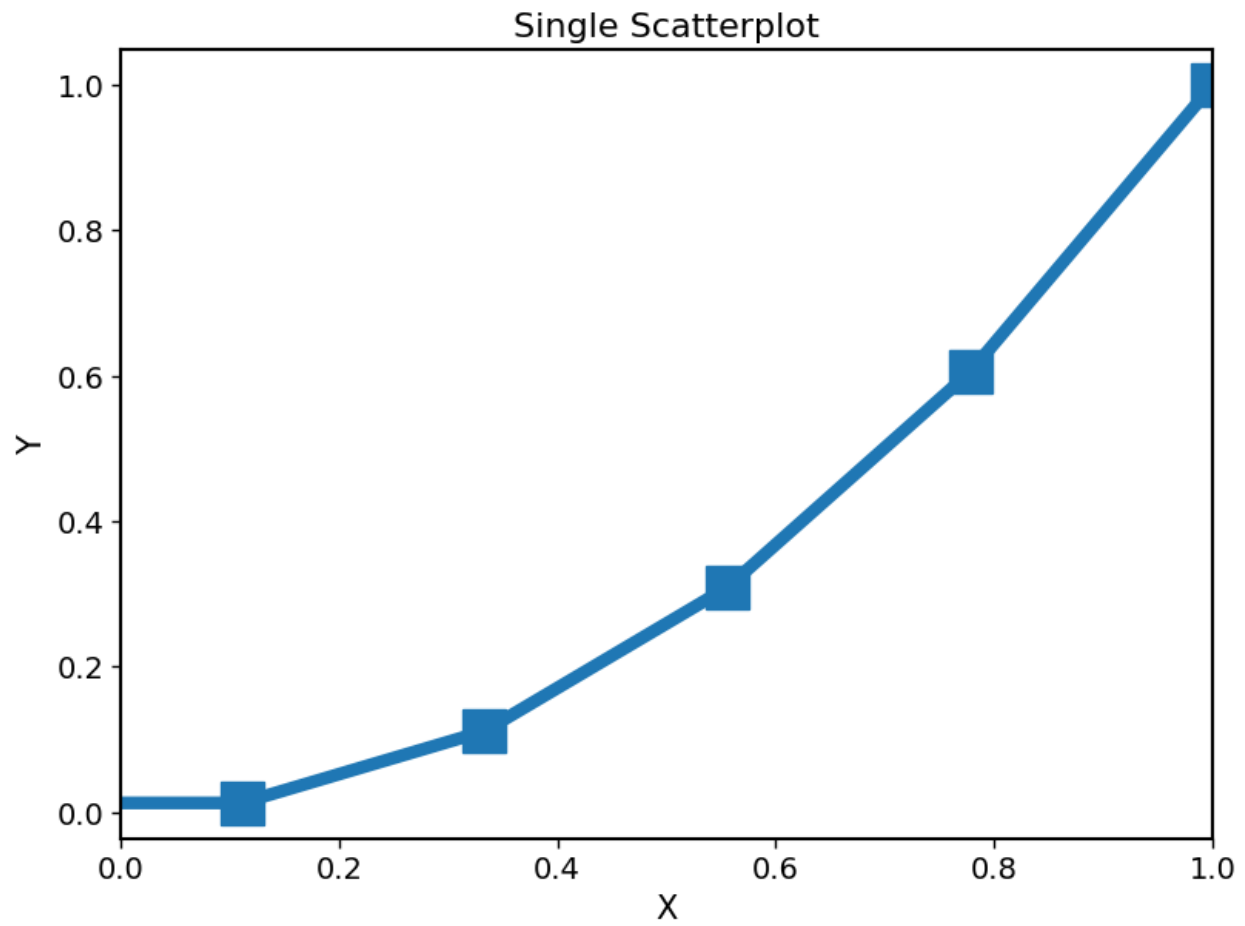
import numpy as np

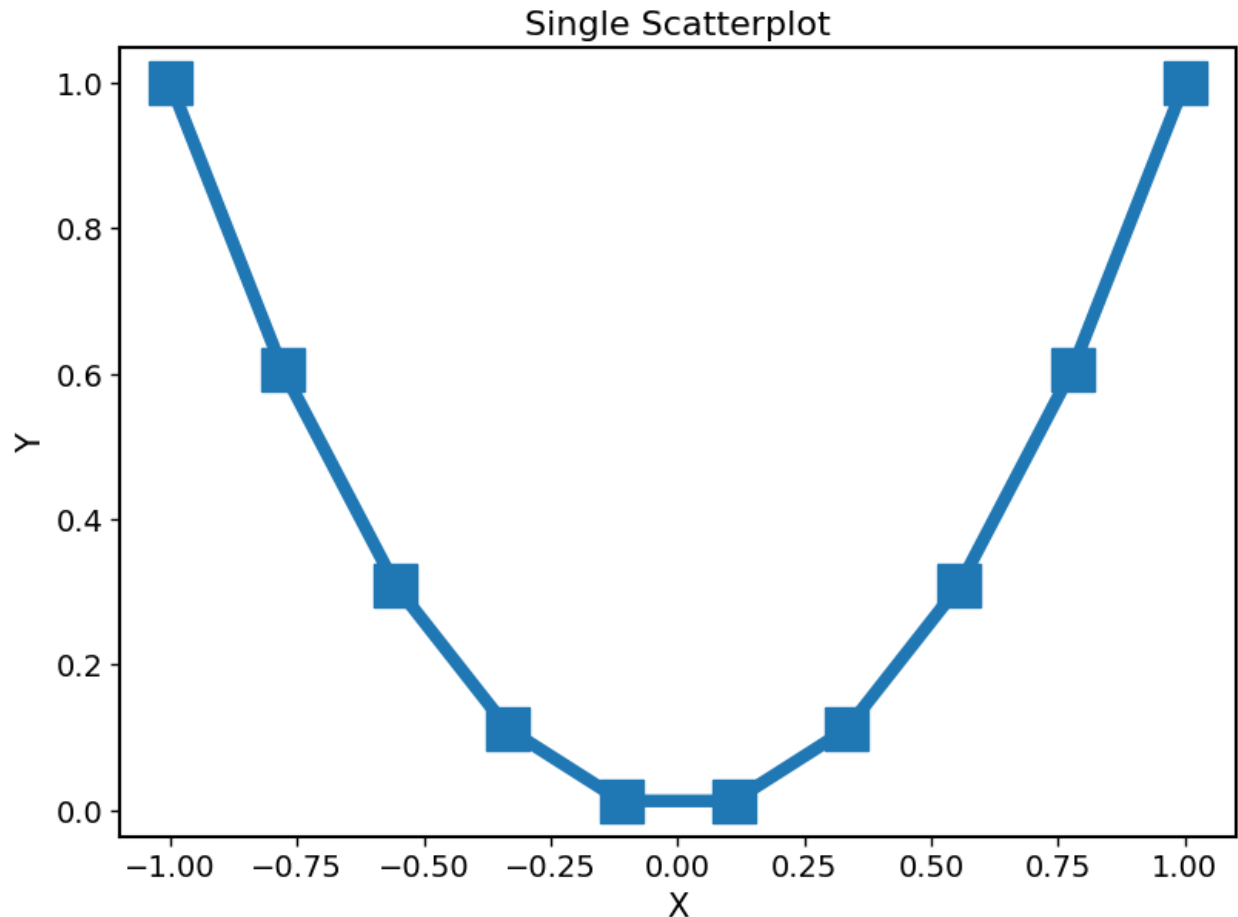
x = np.linspace(-1, 1, 10)
y = x**2

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

```







```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='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 `Plotter.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 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 maschi_tools.vis.matplotlib_plotter import MatplotlibPlotter
from maschi_tools.vis.parameters import ensure_plotter_consistency
```

(continues on next page)

(continued from previous page)

```
#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()

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

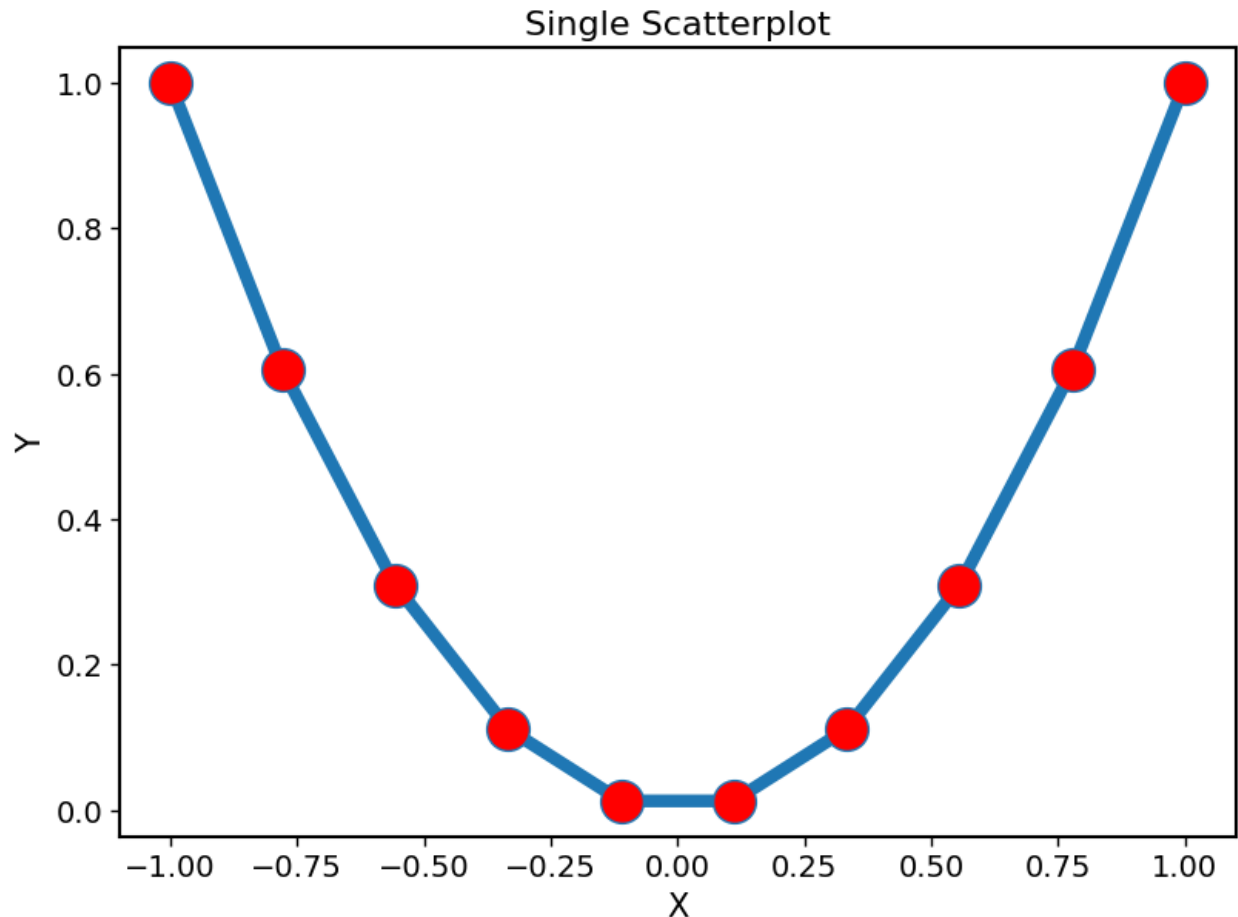
    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)
    plot_params.save_plot('figure')

    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)
```



```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

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

#First we instantiate the MatplotlibPlotter class
```

(continues on next page)

(continued from previous page)

```

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)

    #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)
    plot_params.save_plot('figure')

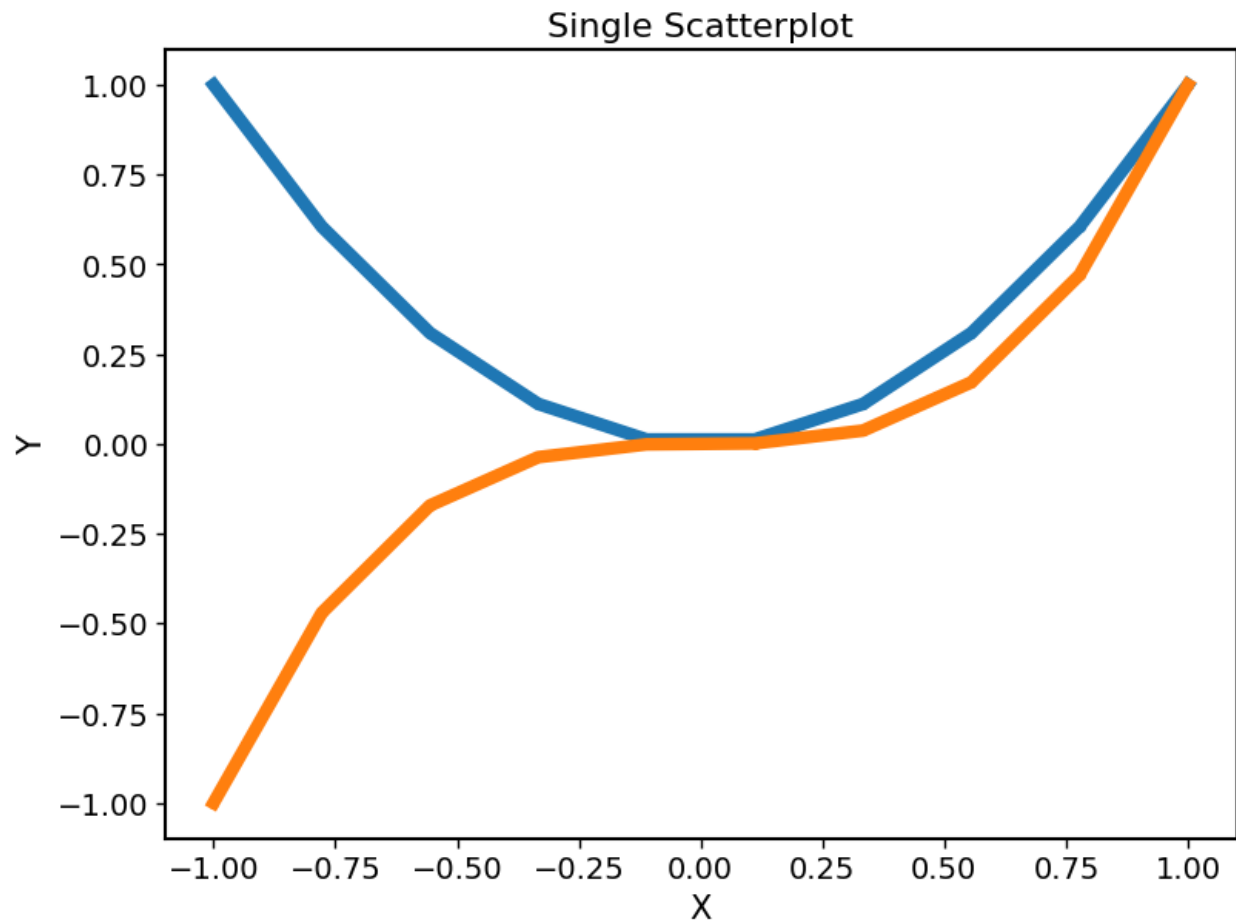
    return ax

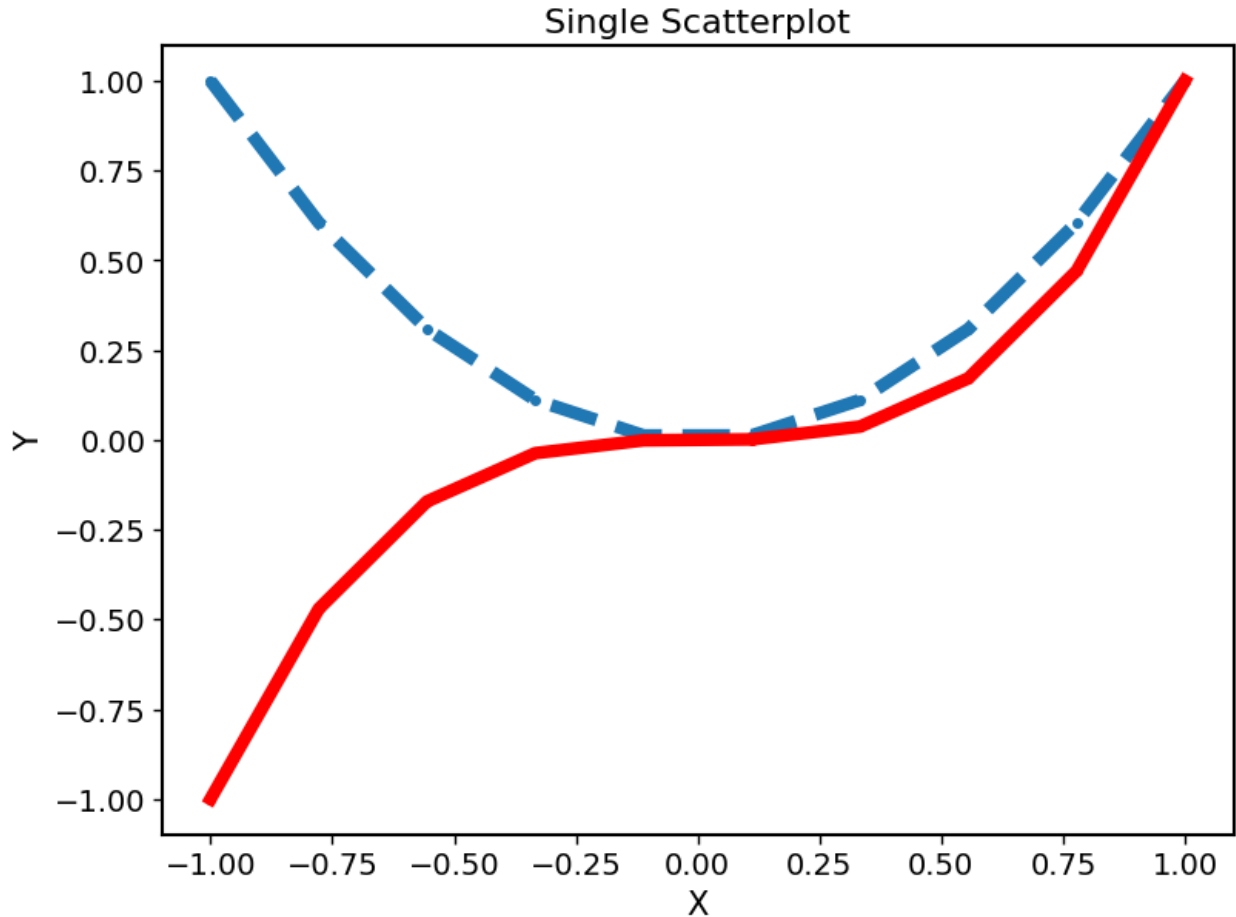
import numpy as np

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])

```





```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

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

(continues on next page)

(continued from previous page)

```
#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)

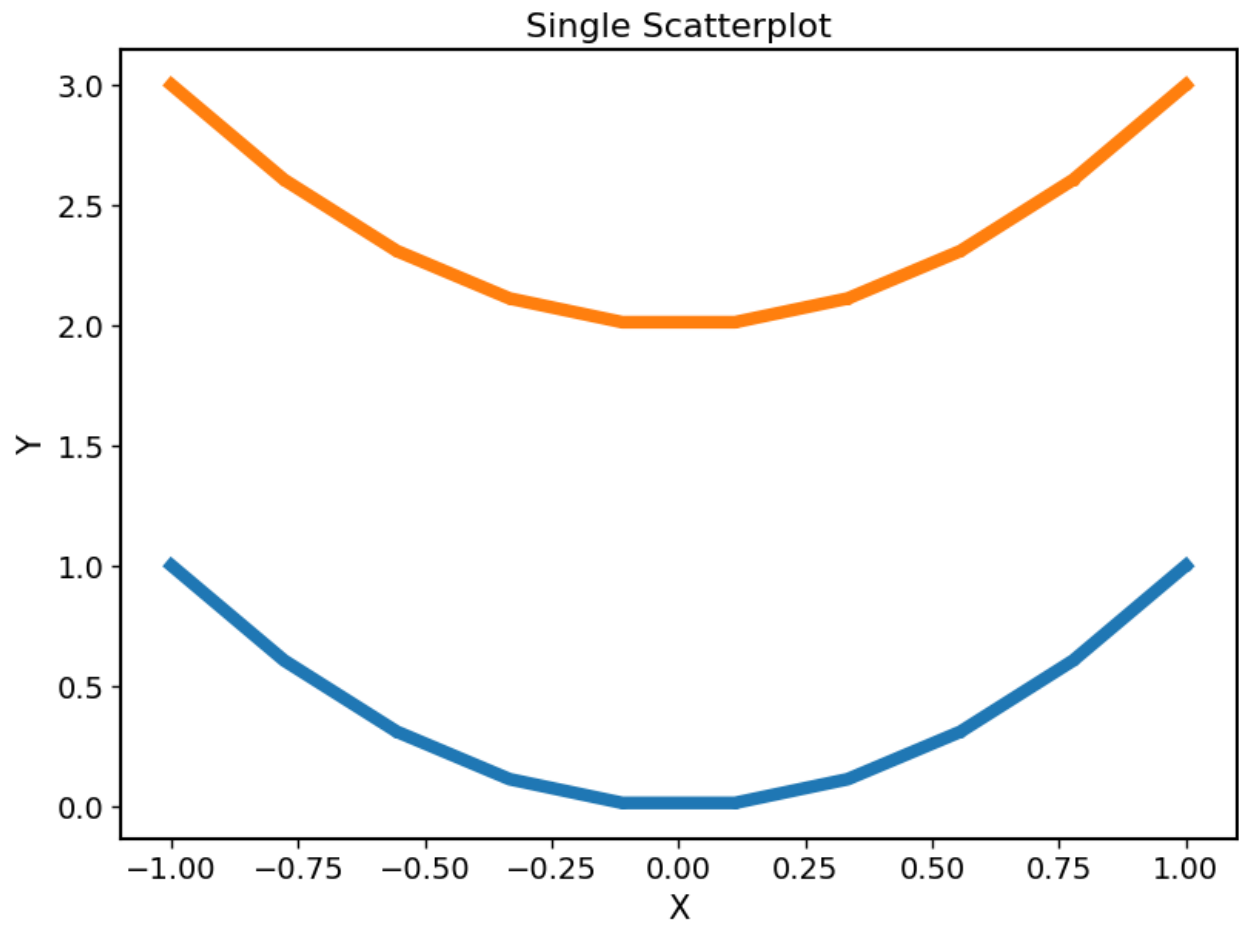
    #The MatplotlibPlotter has a lot of small helper functions
    #In this case we just want to set the limits and scale of the
    #axis if they were given
    plot_params.set_scale(ax)
    plot_params.set_limits(ax)
    plot_params.save_plot('figure')

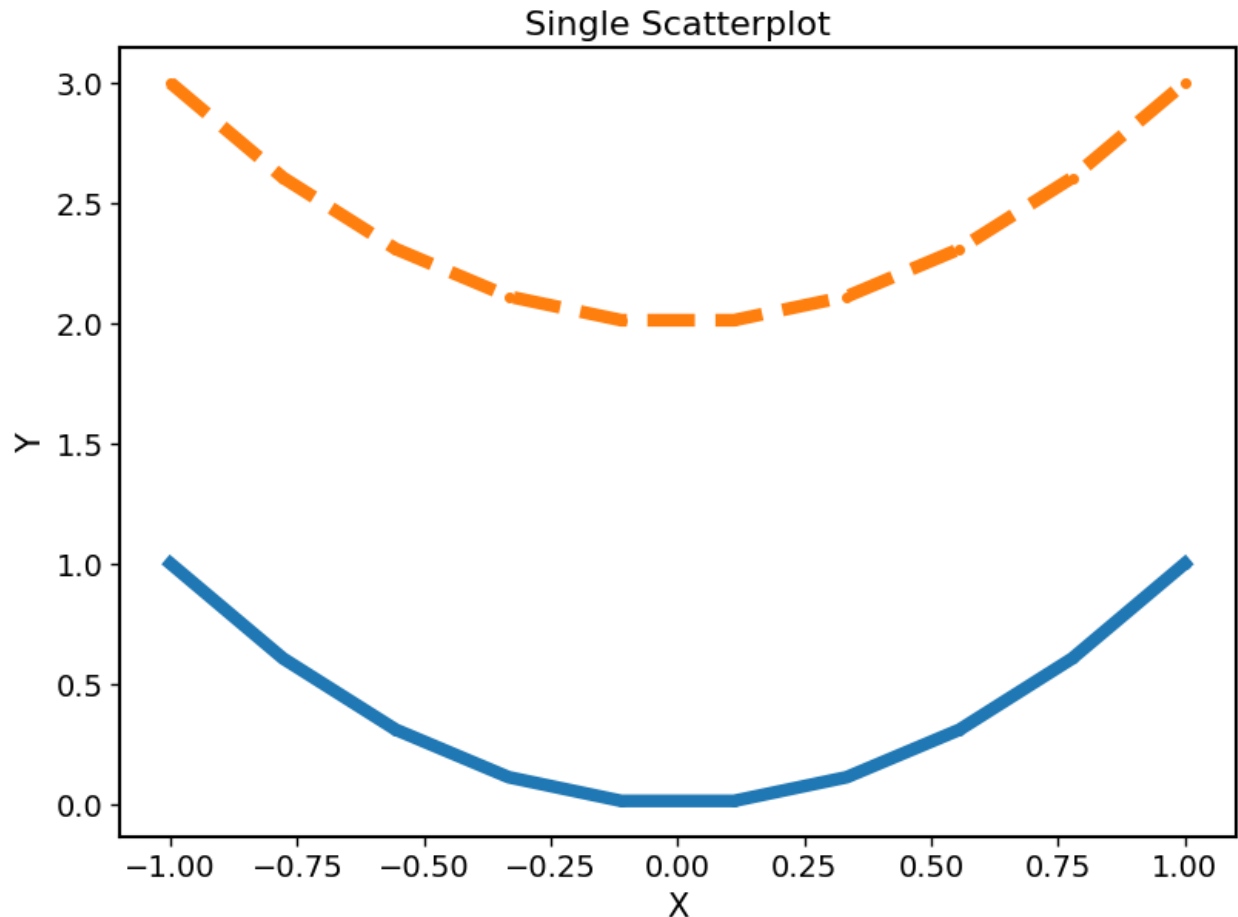
    return ax

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='--')
```



```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

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.parameters import ensure_plotter_consistency
from maschi_tools.vis.parameters import NestedPlotParameters
```

(continues on next page)

(continued from previous page)

```
#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)
    plot_params.save_plot('figure')

    return ax

@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

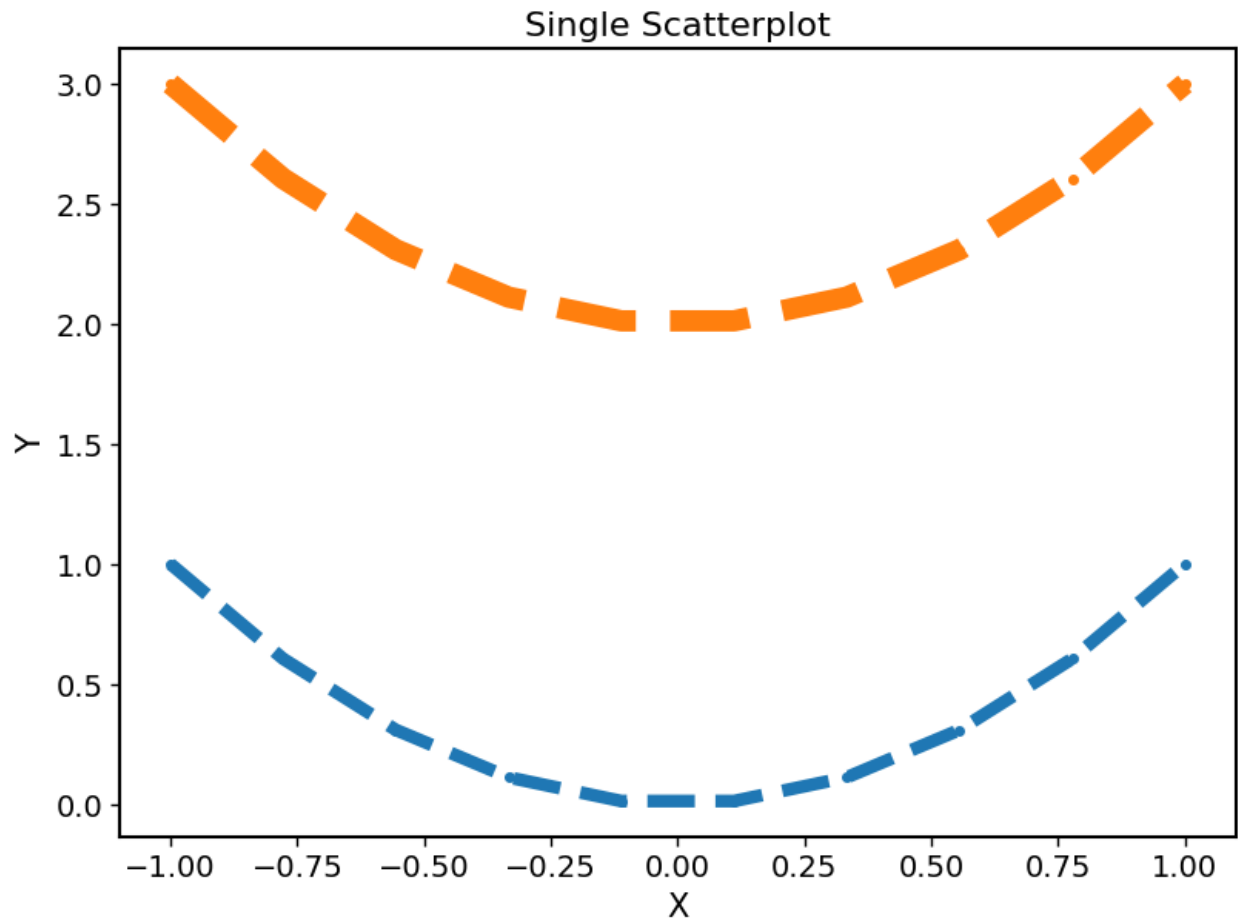
import numpy as np

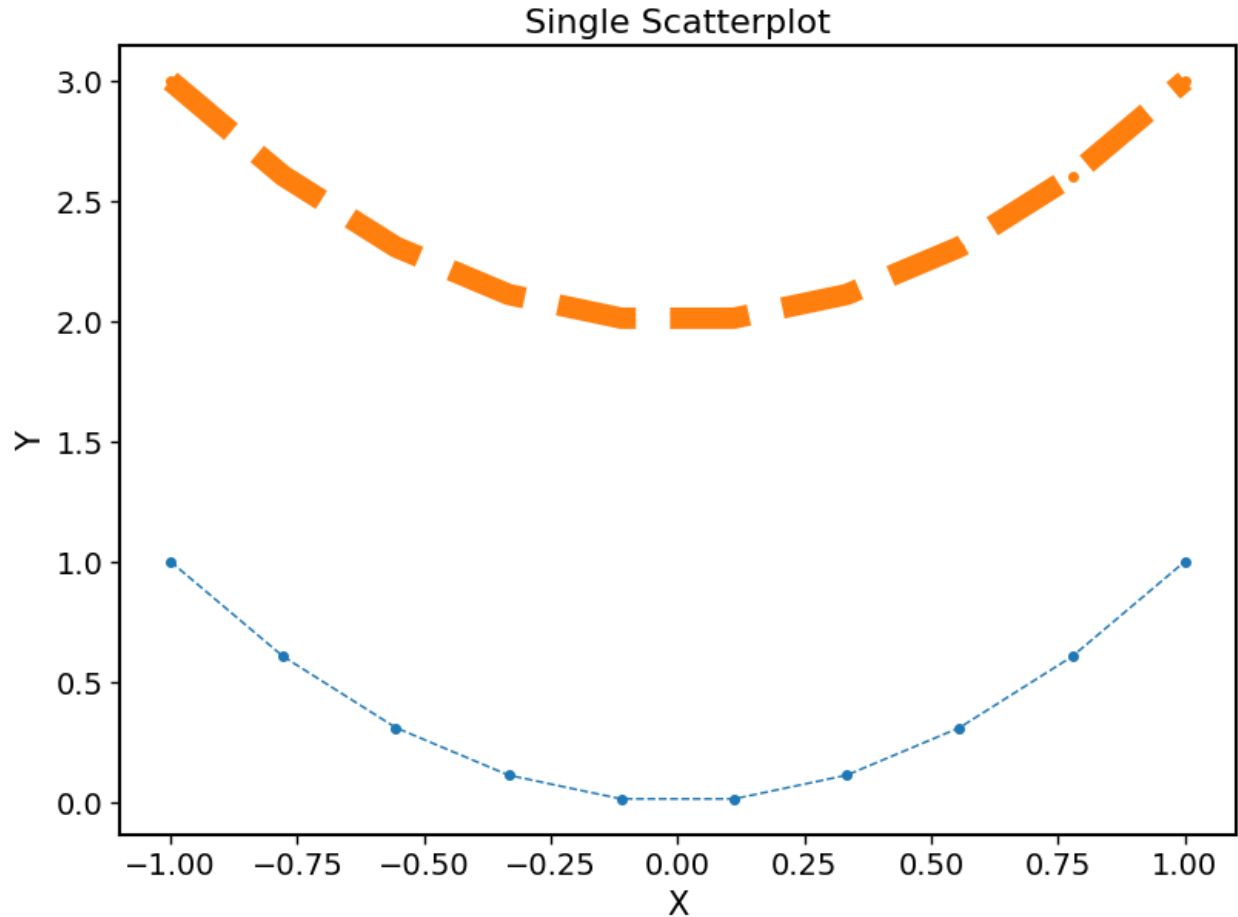
x = np.linspace(-1, 1, 10)
y = x**2
```

(continues on next page)

(continued from previous page)

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





```
<Axes: title={'center': 'Single Scatterplot'}, xlabel='X', ylabel='Y'>
```

5.1.4 Using the PlotData class

5.1.4.1 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 maschi_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
```

(continues on next page)

(continued from previous page)

```
#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

5.1.4.2 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 three rules:

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`

5.1.4.3 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

- `PlotData.get_keys()`: Get all the keys for a given data key in a list
- `PlotData.get_values()`: Get all the values for a given data key in a list
- `PlotData.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
- `PlotData.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
- `PlotData.apply()`: Apply a lambda function to transform the data of a given data key (in-place!!)
- `PlotData.get_function_result()`: Apply a function to a given data key and return the results (Does not change the data)
- `PlotData.sort_data()`: Sort the data by the given data keys
- `PlotData.group_data()`: Group the data by the given data keys
- `PlotData.shift_data()`: Shift the data of a given data key either globally or with different shifts for each plot
- `PlotData.copy_data()`: Copy data to a of one data key to a new data key
- `PlotData.distinct_datasets()`: Return how many different datasets exist for a given data key

Warning: The methods `PlotData.sort_data()` and `PlotData.group_data()` will always convert the data sources to `pd.DataFrame` objects if they are not already.

REFERENCE

6.1 Reference

6.1.1 Changelog

6.1.1.1 latest

[full changelog](#)

Nothing here yet

6.1.1.2 v.0.15.0

[full changelog](#)

Added

- Added support for Python 3.11 [\[#227\]](#)
- Added `FleurInputSchema.xsd` and `FleurOutputSchema.xsd` for the MaX6.2 release of fleur (file version 0.37) [\[#240\]](#)
- KKR parser: Added keywords for Kkrimp BdG [\[#235\]](#)

Bugfixes

- Fixed order of added comments in `readd_inpgen_comments`. Previously they would end up in reverse order
- Fixed extraction of global magnetic moments from `out.xml` [\[#230\]](#)/[\[#231\]](#)

6.1.1.3 v.0.14.0

[full changelog](#)

Added

- Command `masci-tools convert-inpgen` to transform common structure formats, e.g. `cif` into `inpgen` files. Uses `ase` or `pymatgen` with the corresponding fleur plugins `ase-fleur` and `pymatgen-io-fleur` (install additional dependencies `cmdline-extras`)[\[#215\]](#)
- Functions `get_inpgen_comments` and `readd_inpgen_comments` to keep the comments of the `inp.xml` file containing the `inpgen` commandline and file content through functions using `clear_xml`

Improvements

- `write/read_fleur_inpgen` now supports the magnetic moment definitions in the `inpgen` input file introduced in MaX-6.1 (also the `scf` namelist) [\[#213\]](#)
- `get_structuredata` also reads out the magnetic moments [\[#213\]](#)
- `FleurXMLModifier.modify_xmlfile` now keeps the `inpgen` comments by default (controlled by option `keep_inpgen_comments`)

Bugfixes

- Several fixes for Fleur DOS plots [\[#212\]](#):
 - Fixed wrong summation of atom weights for files containing 10 or more atomtypes
 - Fixed crashes using a custom weight not conforming to the naming scheme of weights in the `banddos.hdf`
 - Fixed specification of parameters by DOS label not working if the spin suffix is omitted

6.1.1.4 v.0.13.0

[full changelog](#)

Improvements

- `set_kpointmesh` now also writes out the `nx/ny/nz` attributes for the dimensions of the `kpoint` mesh
- `get_structure_data`, `get_parameter_data` and `get_kpoints_data` are renamed to `get_structuredata`, `get_parameterdata` and `get_kpointsdata` to match the names of the corresponding functions in `aiida-fleur`. Old names are available with deprecations [\[#208\]](#)
- `FleurXMLModifier` now supports changes to input files with not yet available Fleur schemas, if the changes are compatible with the last available file schema [\[#209\]](#)

Bugfixes

- Bugfix in XML setters `set_inpchanges` and `set_attrib_value`, setting the `xcFunctional` key was previously not case-insensitive in contrast with all other keys
- Fixed crash in `get_parameter_data`. This function would previously crash if a kpoint mesh without `nx/ny/nz` attributes was used and the first point in the list was the gamma point

6.1.1.5 v.0.12.0

[full changelog](#)

Added

- Added XPath evaluation functions with runtime type checking of the results of the expressions [\[#181\]](#)
- Command `masci-tools fleur-schema pull <branch>` to update/add Fleur Schema files from the iffgit more easily [\[#184\]](#)
- New XML setters [\[#183\]](#):
 - Setting XC functional explicitly + LibXC support (`set_xcfunctional`),
 - Creating a kpoint path using ase (`set_kpointpath`)
 - Creating a kpoint mesh with symmetry reduction using `spglib`. Should be equivalent to the `gamma@grid=nx,ny,nz` kpoint generator in `inpgen` (`set_kpointmesh`)
- Added `FleurInputSchema.xsd` and `FleurOutputSchema.xsd` for the MaX6.1 release of fleur (file version 0.36) [\[#196\]](#)

Bugfixes

- Add clearer error message if `None` is passed to the `convert_to_xml` functions. This would happen for example using the `set_inpchanges` function with `{'minDistance': None}` [\[#182\]](#)
- Fixed `masci-tools fleur-schema add` with `--from-git` flag. Previously it would still check for the existence of the Schema file locally [\[#184\]](#)
- `get_fleur_modes`: gw mode renamed to `spex` and now stores the actual integer value of the attribute [\[#185\]](#)
- Bugfix in `clear_xml`, when multiple XML comments are present outside the root element [\[#193\]](#)
- Bugfix in `reverse_xinclude`. This would previously break when reexcluding trees already containing a `relaxation` tag and would end up with two `xi:include` tags for the `relax.xml` [\[#194\]](#)
- Bugfix for `FleurXMLModifier`. The `task_list` property would incorrectly enter a `kwargs` key if the modifying function in question has an explicit `**kwargs` argument
- Bugfix in matplotlib plots with placement of multiple colorbars (e.g. weighted spin-polarized bandstructure) [\[#198\]](#)

6.1.1.6 v.0.11.3

[full changelog](#)

Improvements

- Changes to KKR plotting routine `dispersionplot` for compatibility with AiiDA v2.0
- Connecting vectors for intersite `GreensFunction` are now saved in Angstroem. For better interoperability with ase, pymatgen, AiiDA

For Developers

- Relaxed CI requirements for docs build. Nitpicky mode is no longer required to pass but is treated as a hint to look into the warnings

6.1.1.7 v.0.11.2

[full changelog](#)

Improvements

- Several changes in KKR IO functions to make them compatible with `aiida-core` ≥ 2.0 [#175]
- Add function to calculate fourier transform of e.g. J_{ij} constants calculated from Green's functions (`masci_tools.tools.greensf_calculations.heisenberg_reciprocal`)

Bugfixes

- Fixed nondeterministic order in bokeh regression tests if multiple dictionaries with the same values but differing keys in the same list (e.g. Providing the same data for different columns)
- Fixed wrong names for columns entered in `decompose_jij_tensor`, i.e. $J_{ji} \rightarrow J_{ij}$

Deprecations

- Deprecated the unused modules `util/kkr_rms_tracker.py` and `util/modify_potential.py`

6.1.1.8 v.0.11.1

[full changelog](#)

Bugfixes

- Bugfix in `reverse_xinclude`. Version 0.11.0 broke this function for versions, where the `relaxation` tag was not allowed.

6.1.1.9 v.0.11.0

full changelog

This release adds some improvements to the XML and HDF5 handling mainly for better AiiDA-Fleur support. Also major updates to documentation configurations and Green's function calculations.

Added

- Added `FleurElementMaker` class. This can be used to create XML elements compatible with a given version from scratch. Has case-insensitivity and converts values to strings for XML [\[#159\]](#) Example

```
from masci_tools.util.xml.builder import FleurElementMaker
E = FleurElementMaker.fromVersion('0.35')
new_kpointset = E.kpointlist(
    *(
        E.kpoint(kpoint, weight=weight, label=special_labels[indx]) if indx in
↪special_labels else
        E.kpoint(kpoint, weight=weight) for indx, (kpoint, weight) in
↪enumerate(zip(kpoints, weights))
    ),
    name=name,
    count=nkpts,
    type=kpoint_type)
```

- Function `serialize_xml_arguments` to `masci_tools.util.xml.common_functions` to remove XML elements/trees from positional/keyword arguments and replace them with string representations of the XML. Can be used in AiiDA-Fleur
- Module `masci_tools.util.ipython` and `ipython` extension (`%load_ext masci_tools`). Adds syntax highlighted XML tree output and creating HTML syntax highlighted diffs of XML trees [\[#158\]](#)
- Added calculation of Jij Tensor from intersite Green's functions [\[#170\]](#)

Improvements

- Added `name` entry to `SchemaDict.tag_info` which contains the tag name in the original case [\[#159\]](#)
- `convert_to_xml` is made more strict. Conversion `int` to `str` uses the `{:d}` format specifier and string conversion is no longer always attempted [\[#159\]](#)
- Improvements to Colorbar creation in matplotlib plotting methods. Limits are now set consistently with `limits={'color': (low, high)}` in the plot and colorbar. Spinpolarized bandplots now show two colorbars for the two colormaps if requested
- `get_parameter_data` now also extracts the `gamma` switch for kpoint generation for more consistent roundtrips. This is only set if the first kpoint in the mesh is the gamma point and there are multiple
- `load_inpxml` and `load_outxml` now consistently accept the XML file given as a string of the content. The content no longer has to be manually encoded as bytes

- The method `FleurXMLModifier.modify_xmlfile` now always returns two things. The modified XML tree and a dictionary with all additional file contents, e.g. `n_rmp_mat`.
- Support for aligning spin/real-space frames of Green's functions. Several further improvements/bugfixes for Green's function modules [\[#170\]](#)

Bugfixes

- Bugfix for `outxml_parser` returning a nested list for Hubbard 1 distances, where a flat list was expected. Removed `force_list` argument from the parsing task definition
- Fixes in `plot_fleur_bands`, when providing custom weights without spin suffixes, i.e. `_up/_down`
- Fix in `HDF5Reader`. IO like objects without an attached filename would lead to an early error. This is the case for example for some readers in the file repository implementation used in AiiDA v2.0
- Fix in `HDF5Reader`. The file handles for compressed files in the AiiDA v2 repository have to be copied into a temporary file first before they can be used

For Developers

- Docs: Updated `sphinx` and `sphinx-autodoc-typehints` versions and build docs on python 3.10 [\[#156\]](#)
- Docs: Converted to MyST markdown and where appropriate introduced `myst-nb` for executing code cells in the documentation, e.g. generate plotting examples [\[#157\]](#)
- Bokeh regression tests now strip out the bokeh version from the test files
- Added pre-commit hook, which generates the docstrings for the `FleurXMLModifier` registration methods from their XML setter function counterparts [\[#166\]](#)

6.1.1.10 v0.10.1

[full changelog](#)

Bugfixes

- Remove accidentally left in debug print in `outxml_parser`

6.1.1.11 v0.10.0

[full changelog](#)

This release provides several new features in the XML modification/evaluation for Fleur XML files and bugfixes. Multiple problems when working with DFT+U density matrix files are also fixed.

Added

- New XML setter `align_nmmpmat_to_sqa` to rotate the density matrix file according to SQAs specified either for noco or second variation SOC [#140]
- Added `task_list` property to `FleurXMLModifier` to construct a list which can be used to replicate the same `FleurXMLModifier` with the `fromList()` classmethod [#149]
- Added `FleurXMLContext`, which acts as a holder of the XML elements, schema dictionary, constants and logger to reduce the amount of information/clutter in functions evaluating things from the XML file [#152]

Note: The class `ParseTasks` used in the `outxml_parser` was simplified and placed into the `outxml_parser` module and the decorator `register_parsing_function` was removed. This was done without deprecation since they were exclusively used in the `outxml_parser` and were the main cause of cyclic import problems previously

- Added several predefined conversions to/from input version 0.35 to `inpxml_converter` [#153]

Improvements

- Added `inverse` argument to nmmpmat XML setters. These will correctly produce the inverse rotation operation for the given angles. Also allow setting `orbital='all'` in `rotate_nmmpmat` to rotate all blocks by the given angles [#140]
- The XML setters `create_tag`, `replace_tag` and their low-level equivalents now also accept XML strings, i.e. `<example attribute="1"/>`, as arguments for the elements to create/replace [#145]

Bugfixes

- Fix for XML setters operating on the DFT+U density matrix file. Previously these functions would not map the density matrix blocks correctly if multiple atomgroups shared the same species containing `ldaU` tags [#140]
- Added missing prefactor $(-1)^{(m-mp)}$ to `get_wigner_matrix()`
- Added basic tests of `masci_tools.tools.greensfunction` module and fixed several bugs found due to this [#150]
- Fixed bug in XML setters operating on the DFT+U density matrix file not correctly extracting the number of spin blocks when only setting `l_mperp`
- Fixed bug, when using the `FleurXMLModifier` directly (not in `aiida-fleur`), included XML files were not handled
- Fixed bug in `outxml_parser`, when the XML file had to be repaired and more than one iteration was present the wrong iteration was chosen as the last stable iteration [#152]

Deprecated

- The module `masci_tools.io.io_fleurxml` is renamed to `masci_tools.io.fleur_xml` [#152]
- The module `masci_tools.util.parse_task_decorator` is removed. All decorators are now available under `masci_tools.io.parsers.fleur` [#152]

For Developers

- Added `py.typed` marker to `masci-tools`, since a large part of the outside facing code (especially the XML APIs are typed). With this marker other packages can use the typehints in this package

6.1.1.12 v.0.9.1

[full changelog](#)

Added

- Standalone function `masci_tools.tools.fleur_inpxml_converter.convert_inpxml` to allow conversions of `inp.xml` files within a python runtime without needing to go via the commandline

Bugfixes

- Fixed bug in bokeh testing fixtures using the wrong folder for fallback versions
- Fixed bug not correctly converting complex numbers from the Fleur xml files if they have whitespace at beginning/end

6.1.1.13 v.0.9.0

[full changelog](#)

Added

- New bokeh plot routine for matrix plot of rectangle patches with added texts [\[#124\]](#)
- Added TS contribution to free energy to output of `outxml_parser`

Improvements

- Several arguments in XML setter functions were renamed for more consistent signatures. The main changes are
 - `attributedict/change_dict` -> `changes`
 - `attributename/attribv` -> `name/value`
 - `add_number` -> `number_to_add`

The old signatures are still supported with deprecations if called via the `FleurXMLModifier` [\[#118\]](#)

- Remove constraint on bokeh version (previously `<=1.4.0`) [\[#122\]](#)
- Add `only_spin` option and calculate complete non-spinpolarized DOS for `spinpol=False` in `plot_fleur_dos` [\[#125\]](#)
- Refactored `CFCalculation`, i.e. naming of attributes handling of cutoffs. Added classmethod to construct instance directly from numpy arrays [\[#127\]](#)
- Refactored plotting methods for `CFCalculation` to allow the same parameter freedom as for the other matplotlib routines [\[#127\]](#)

- Improvement to labels and legends in DOS/bandstructure plots. Matplotlib plots now put the legend centered below the plot and added latex labels to axis and ticks in bokeh (version 2.4.0 needed) [#133]
- `io_nmmpmat`: Allow negative indices in `read_nmmpmat_block` and raise error for invalid index

Bugfixes

- Fix for signatures of `set_text/set_first_text`. These contained names of attribute setting functions [#118]
- Fix for validating arguments in `FleurXMLModifier` not accepting an argument named `name` when passed by keyword. [#118]
- Fixed problems in `masci_tools.testing.bokeh` when adding files for new bokeh versions [#122]
- Several fixes for `plot_fleur_dos`. Using the `area_plot` or specifying `color` explicitly could mess up the color order [#132]
- Fixed bug in `validate_nmmpmat` and consequently `FleurXMLModifier` not correctly validating density matrix files with certain off-diagonal elements being negative [#135]
- Fix for `HDF5Reader` for compatibility for file handles in `aiida-core` 2.0. The file handles coming from the file repository have no directly attached extension so the check if the file is a hdf file cannot be performed

For developers

- Fixed upload of `pytest-mpl` results artifacts to include the whole directory with images and not just the HTML file [#117]
- Updated typing to newer version of `lxml-stubs` (0.4.0) [#123]

6.1.1.14 v.0.8.0

full changelog

Added

- Added `FleurInputSchema.xsd` and `FleurOutputSchema.xsd` for the MaX6 release of fleur (file version 0.35) [#112]
- New XML getter function: `get_special_kpoints` extracts the labelled kpoints from a kpoint list (for now only implemented for Max5 or later)
- Added extraction of Hubbard `I` input and distances in the `outxml_parser` for fleur (distances only available starting from version 0.35) [#108]
- Added extraction of global vector of magnetic moments in non-collinear calculations in `outxml_parser` under the key `magnetic_vec_moments` starting from version 0.35

Improvements

- Fleur schema parsing functions now recognize a new alias from the fleur schemas `FortranComplex` which is a number of the form `(float, float)`. Converters for complex values are added. (Note: Complex numbers should not yet be used in the `outxml_parser`, since AiiDA (<2.0) does not support complex numbers yet) [#106]
- Added `IncompatibleSchemaVersions` error when a combination of output and input version for `OutputSchemaDict` is given, for which it is known that no XML schema can be compiled
- `xml_getters` functions can now be used in the task definitions of the `outxml_parser` to keep information consistent. This example definition will insert the structure data, i.e. a tuple of atoms, bravais matrix and periodic boundary conditions into the output dictionary. `{'parse_type': 'xmlGetter', 'name': 'get_structure_data'}` [#107]
- The `_conversions` key in the `outxml_parser` now accepts namedtuples `Conversion` to enable passing additional arguments to these functions. [#109]
- Adjusted `get_cell` to understand the `bravaisMatrixFilm` input introduced with the MaX6 release of fleur [#110]
- Improved detection, whether a given xpath contains a tag including stripping predicates. Added function `contains_tag` in `masci_tools.util.xml.common_functions` [#113]
- Refactored bokeh plot routine `periodic_table_plot` to make use of the plot parameters utilities [#114]
- `get_parameter_data` now extracts LOs with higher energy derivatives or HELO type, as they are supported by the newest versions of the inpgen. The old behaviour of dropping all non SCL0 and `eDeriv=""` LOs is available via the option `allow_special_los=False`

Bugfixes

- Fix in `load_inpxml` and `load_outxml` (this also effects the `inpxml/outxml_parser`). Previously file handle like objects not directly subclassing `io.IOBase` would lead to an exception
- Added patch for `OutputSchemaDict` objects with `FleurOutputSchema.xsd` files before version 0.35. The attribute `qPoints` in the DMI output was actually called `qpoints` in these schemas, making it impossible to retrieve this attribute
- Fixed behaviour of relative XPath methods of `SchemaDict` which did not correctly handle root tags, whose names are contained in other tag names, for example `bravaisMatrix` and `bravaisMatrixFilm` from the new file version 0.35

Deprecated

- Passing strings in the `_conversions` key in task definitions for the `outxml_parser`. Use `masci_tools.util.parse_utils.Conversion` instead. [#109]

For developers

- Reorganized visualization tests, making the regeneration of baseline images with `pytest-mpl` easier [#101]
- Switched build system from `setuptools` to `flit`, since this way all the configuration can be specified in the `pyproject.toml` and a lot of duplication of information is avoided (e.g. version numbers) [#102]

6.1.1.15 v.0.7.2

[full changelog](#)

Bugfixes

- Fixed regression in `set_atomgroup_label` and `set_species_label`. These functions could be used in previous versions with atom labels, that do not exist. This is not possible in v.0.7.1. Since some parts of the `aiida-fleur` plugin relied on this the behaviour has to be kept.

6.1.1.16 v.0.7.1

[full changelog](#)

Added

- `XPathBuilder` class for specifying complex conditions on xpaths with a dictionary. Added `filters` argument to `schema_dict_util` and `xml_setters` functions for this dictionary [#96]

Bugfixes

- Fixed issue with `MANIFEST.in`, where non-python files from the `tools` subpackage were not included in the built packages
- Fixed bug not correctly processing the plot limits in `plot_fleur_bands` in excluding points outside the plot area for better performance
- Fix for HDF5 transformation `add_partial_sums` if not all formatted patterns are present in the dataset, e.g. if a bandstructure/DOS is calculated for only selected atoms

For developers

- More strict mypy configuration and moved a lot of the annotations to modern syntax with `from __future__ import annotations`
- Added `pyupgrade` hook to automatically do some easy refactoring, i.e. removing compatibility workarounds move to modern syntax. Set to apply changes compatible with 3.7 and later

6.1.1.17 v.0.7.0

full changelog

Commandline interface, refactoring of SchemaDict/XML functions and major improvements for package configuration/tooling for developers. Added support for python 3.10. Dropped testing for python 3.6.

Added

- Added click command line interface (available as `masci-tools`). Can add Fleur XML Schema files with validation (Also directly pulled from the Fleur git), use the XML parsing functions and interface to the fleur visualization routines [\[#49\]](#)
- Added `Tabulator` for use of creating `pandas.Dataframes` from attributes of python objects. Used in `aiida-jutools` to tabulate attributes of aiida nodes
- Added `optional_tasks` argument to `outxml_parser`. Adds tasks marked with `'_optional': True` to the performed tasks [\[#81\]](#)
- Added visualization routine for spectral functions (colormesh plot with path through Brillouin zone)
- Added tool for converting `inp.xml` files between different file versions (Available through the click CLI `masci-tools inpxml`) [\[#88\]](#)
- Added three new XML setters: `clone_species` (Create and modify a species starting from an existing one), `switch_species/switch_species_label` for switching the species attribute of atom groups with additional checks
- `outxml_parser`: Total Energy is now taken from the output `freeEnergy` in the `out.xml`

Improvements

- Refactored SchemaDict code. Moved routines `get_tag_xpath` and similar to methods on the SchemaDict. If the path cannot be determined custom exceptions `NoPathFound` and `NoUniquePathFound` are now raised [\[#84\]](#)
- Added utility to `OutputSchemaDict` to create absolute paths into iteration elements in `out.xml`. Added support for this option in `schema_dict_util` functions with `iteration_path=True`
- All basic XML modification functions now accept either a `ElementTree` or `Element`. and warn they find no nodes to operate on
- Improved capabilities of green's function tool, can now be used with radially resolved/k-resolved Green's functions
- Improved performance of Fleur XML Schema parsing by switching from the `xpath` method on the `ElementTree` to constructing a `XPathEvaluator` object [\[#89\]](#)
- All `xml_getters` functions can now also be used with `out.xml` files
- `set_atomgroup/set_atomgroup_label` now use `switch_species` if the species attribute is changed
- `set_atomgroup` now supports the `all-<search string>` syntax for species argument, equivalent to `set_species`
- Improved behaviour of spin-polarized DOS plots for duplicating all plot parameters for spin-down components (previously only color was repeated)

Bugfixes

- Fixed several issues in version handling of Schema dictionaries. It is now possible to add a new schema and have it work (with warnings) without needing to change any code (`masci-tools fleur-schema add <path/to/schema>`)
- Bugfix in `evaluate_tag` not handling the combination of options `subtags=True` and `text=True` correctly. Previously some results were overwritten.
- Fixed accidental change in `write_inpgen_file` in comparison to old `aiida-fleur` implementation. Now the species name is always appended to the position in the `inpgen` file if it is not equal to the atom symbol
- Fixed behaviour of `get_parameter_data` for inputs with local orbitals with higher energy derivatives. These cannot be created by the `inpgen` and so are dropped
- Fixes in `xml_setters` to allow consistent creation of multiple tags for setting text or attributes

Deprecated

- The `fleur_schema` subpackage was moved from `masci_tools.io.parsers.fleur` to `masci_tools.io.parsers` to avoid circular import issues [\[#87\]](#)

Removed

- `get_structure_data` now returns `AtomSiteProperties` for the atom information by default. The default value of `site_namedtuple` is now `True`

For developers

- Moved configuration of `yapf`, `pylint` and `pytest` into `pyproject.toml`
- Made test suite executable from the root-folder (Some file paths were not transferrable when changing the execution directory)
- Added `test_file` fixture, which constructs the absolute filepath to files in the `tests/files` folder to reduce the difficulty of moving test files around and reorganizing the `pytest` suite
- Updated `pylint` (2.11), `pytest` (6.0) in `setup.py`
- Added `mypy` pre-commit hook. Checked files are specified explicitly [\[#86\]](#).
- Added typing to majority of XML functions (with stubs package `lxml-stubs`) and large parts of the `io` and `util` subpackages
- Dropped testing for python 3.6 in CI

6.1.1.18 v.0.6.2

[full changelog](#)

Small bugfixes and refactoring for plotting routines

Added

- Common plot routines for equation of states `eos_plot` and plotting scf convergence `convergence_plot`
- Replaced old convergence plot routines with single routines for bokeh/matplotlib named `plot_convergence`

Improvements

- Moved eos and convergence plots to use `PlotData` class
- Fixed some edge cases of bokeh testing fixtures. Previously some plots would crash the code for normalizing the json

Bugfixes

- Fixed bugs in convergence routines for using them in aiida-fleur

6.1.1.19 v.0.6.1

Release fixing a small issue with publishing version 0.6.0 to zenodo

6.1.1.20 v.0.6.0

[full changelog](#)

This release contains major improvements to plotting methods and new tools. Also the fleur parsing functions were improved

Added

- `PlotData` class for handling data passed to plotting methods very flexibly [\[#54\]](#) (For more information see the relevant [users guide](#) or [developers guide](#) sections in the documentation)
- `masci_tools.vis.common` module for plotting methods with common interfaces for bokeh/matplotlib [\[#71\]](#)
- `get_parameter_data` also extracts kpt mesh specifications for the input generator
- Exposed and improved bokeh testing fixtures in `masci_tools.testing.bokeh` for use in higher level packages
- `greensf_calculations` module in `tools` with sample functions for calculating properties with green's functions from fleur
- Added two options `line_plot` and `separate_bands` to bandstructure plots. While `line_plot` is obvious (no weighted bandstructures possible), `separate_bands` allows to set parameters for single selected bands. These options can also be combined

Improvements

- Added option `only_used` to `get_kpoints_data` to get only the `kPointList` referenced in the `kPointListSelection` tag
- Made `constants` argument to `schema_dict_util` functions completely optional. Will raise an exception if a undefined constant is encountered
- Bandstructure plots now exclude points outside the plotting area to speed up these plots significantly for systems with a large number of bands
- Refactored attribute/text type definitions in `SchemaDict` objects. Now unified under one structure. Both attributes and texts can now be recognized to contain multiple values [\[#64\]](#)
- Added `spin_arrows` option to toggle spin arrows in `plot_spinpol_dos` for matplotlib. Previously this was only possible for bokeh
- Added options to create different types of bar plots to `barchart`: Available are 'stacked' (default), 'grouped', 'independent' (positions can be defined for each data set)
- Exceptions occurring in transforms for `HDF5Reader` are now bundled into `HDF5TransformationError` to allow easier error handling
- Added MT keys to `kkparams`

Bugfixes

- Fix for `write_inpgen_file`, which was incorrectly inserting the 'X' (empty sphere) element into inpgen files
- Fix for `read_inpgen_file`, which could not handle inpgen files with comments on certain lines in the inpgen file
- Several fixes for `plot_fleur_dos` not using the standard DOS calculation but orbital decompositions and so on
- Adjusted default `dpi` for matplotlib to 100 to avoid problems with size when using `plt.show()` instead of saving

6.1.1.21 v.0.5.0

full changelog

This release contains fleur inpgen IO capabilities and many major improvements

Added

- IO routines for reading/writing inpgen input files
- `get_structure_data` now returns the relaxed structure if a `relaxation` section is present
- `get_parameter_data` can read in the electron configuration if requested
- `get_symmetry_information` xml getter to get all defined symmetry operations
- `get_structure_data` and `get_parameter_data` now norm the species names to get consistent ids for usage in the inpgen files [\[#70\]](#)

Improvements

- Added another possible value to environment variable `MASCI_TOOLS_USE_OLD_CONSTANTS` to get the values of constants in between commits c171563 and 66953f8 [#66]
- `spex` attribute is now used in addition to old `gw` attribute to determine whether a fleur input file is from a SPEX calculation
- `evaluate_tag` can now also get the text and recursively parse all subtags
- Changed default of `multiply_by_equiv_atoms` in `plot_fleur_dos` to `True`
- `barchart` can now make horizontal bar charts
- `get_cell` now also takes the `scale` attribute into account
- Made conversion to angstroem optional (done by default) in the xml getters. Can be turned off with the options `convert_to/from_angstroem`
- Deprecated: Old atom position output in `get_structure_data`. Use option `site_nametuple=True` to get new output (includes species information)

Bugfixes

- Fixes to make `plot_convex_hull_2d` work
- Fix in `evaluate_text` make `constants` argument optional
- Fix for `get_structure_data`; Added missing unit conversion for z-coordinate of film positions

6.1.1.22 v.0.4.10

[full changelog](#)

This release contains bugfixes for the visualization routines

Bugfixes

- Fixed issue for histogram plot, which prevented usage for e.g. stacked histograms with multiple datasets
- Fixed usage of dictionary arguments in `plot_fleur_dos`, that should not be used to specify parameters for multiple datasets (e.g. limits, lines)
- Updated documentation table of plot parameters to put all dictionaries into literal blocks. Otherwise single quotes appear differently on read the docs

6.1.1.23 v.0.4.9

[full changelog](#)

Improvements

- Added saving/loading of plot defaults in json files to Plotter class (No exposed functions yet in plot_methods or bokeh_plots)
- DOS plot parameters in plot_fleur_dos can now be specified by label, e.g. color={'Total_up': 'blue'}

Bugfixes

- Various Bugfixes and improvements to ChemicalElements class, greensfunction tools and plotting methods
- Fixed wrong atom label generation in DOS plots (#55)

6.1.1.24 v.0.4.8

[full changelog](#)

Added

- Tool to analyze/work with greensfunctions calculated by Fleur `masci_tools.tools.greensfunction`

Improvements

- Several improvements of KKR parsers/parameters [\[#13\]](#)
- Introduced patching functions for the schema dictionaries to manually correct ambiguities [\[#48\]](#)
- Improvements of the HDF5Reader and recipes for Fleur DOS/bandstructure calculation
- For devs:
 - pylint warnings are no longer fatal for the CI jobs
 - tests folder moved outside package directory

Bugfixes

- Bugfixes and improvements in the plotting functions for DOS/bandstructures
- Bugfixes in `xml_setters` and `fleurxmlmodifier` modules

6.1.1.25 v.0.4.7

[full changelog](#)

Added

- Introduced higher level XML modification functions for `xml_delete_tag`, `xml_delete_att`, `xml_replace_tag`

Improvements

- moving conversion factors for energy and bohr to angstroem conversion to NIST values in KKR parts of the code. This can also be disabled for backwards compatibility by setting the environment flag `MASCI_TOOLS_USE_OLD_CONSTANTS`
- Added missing energy unit alias for Fleur input files
- `xml_delete_tag`, `xml_delete_att`, `xml_replace_tag` now also support the `occurrences` argument
- FleurXMLModifier improvements. Modification registration methods will now test the given arguments against the modifying functions immediately to fail early for errors. `fromList` classmethod allows the instantiation with a known list of tasks to perform. These are passed through the same procedure as the normal registration of changes

6.1.1.26 v.0.4.6

[full changelog](#)

Bugfixes

- Fix for the `clear_xml` function, where comments could end up in the set of included tags. This lead to failures in aiida-fleur

6.1.1.27 v.0.4.5

[full changelog](#)

Added

- Introduced function to split a `xmldata` back up into the included subtrees and the main tree with `xi:include` tags
- `clear_xml` now returns a set of the tags, that were included

Bugfixes

- various bugfixes for xml modifying functions
- A special case for `theta`, `phi` and `ef_shift` attributes of `forceTheorem` tags, since they are not correctly typed in the `Inputschema`

6.1.1.28 v.0.4.4

[full changelog](#)

Added

- XML getter methods for number of kpoints and relaxation information
- XML setter methods for manipulating kpoints

Improvements

- top level `create_tag` now also accepts a `etree.Element`
- XML getters now also accept `etree.Element`
- Added `etree.indent` calls to keep modified `inp.xml` clean (raises lxml dependency constraint to 4.5)
- `io_fleurxml` functions now pass keyword arguments to XMLParser
- Re-add `fleur_modes` to `output_dict`

Bugfixes

- Bugfix for relative xpaths

6.1.1.29 v.0.4.3

[full changelog](#)

Added

- Added utility for creating schema dictionary version specific functions
- Added XML getter functions adapted from aiida-fleur fleurinpdata methods [\[#40\]](#)

Improvements

- Improved logging and failure handling of XML parser functions [\[#39\]](#)
- Improved handling of `schema_dict_util` functions with nodes away from the XML root of the file [\[#41\]](#)

6.1.1.30 v.0.4.2

[full changelog](#)

Added

- Added key descriptions to Plotter objects and `get_mpl_help` and `get_bokeh_help` functions in plotting modules for getting descriptions of parameters
- Two new recipes for `HDF5Reader` for bandstructures (for reading in no or specific weights besides eigenvalues)

Improvements

- `MatplotlibPlotter` and `BokehPlotter` now have a autogenerated table of descriptions and default values in the docstring (not as nicely formatted) and in the sphinx build (really nicely formatted)
- `save_format` in matplotlib plots can now be a list of formats
- Various visual improvements to band/DOS plots:
 - Bandstructure size scaling adjusted to not produce massive bands
 - Bandstructure spin up components are now potted on top by default
 - size/color scaling now done with respect to the maximum in the plotting region
 - DOS added spin arrows in spin-polarized case
 - DOS inverted x-axis in vertical plot (spin down now on the right side)
 - DOS added symmetric limits in DOS direction for spin-polarized plots
 - DOS default figsize flipped for vertical plots

6.1.1.31 v.0.4.1

[full changelog](#)

Bugfixes

- Fix for `plot_lattice_constant` to make it work in `aiida-fleur`

6.1.1.32 v.0.4.0

[full changelog](#)

Added

- Parsers for Fleur `inp.xml/out.xml` files. These are robust with respect to changing file versions [#3]
- General parser for `.hdf` files [#30]
- Functionality for modifying `inp.xml` files [#23]
- Fleur visualization routines `plot_fleur_dos` and `plot_fleur_bands`
- IO Module for creating/reading and manipulating `n_mmp_mat` files for LDA+U calculations in fleur [#31]
- Tool for calculating crystal field coefficients [#22]

Improvements

- Refactored parameter handling in plotting methods. Introduced `Plotter` class for consistent behaviour and easy extendability of plotting capabilities [#6]

6.1.2 Source code documentation

6.1.2.1 Visualisation and Plotting

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, only_spin=None,
                                     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
- **only_spin** – optional str, if given only the specified spin components are plotted
- **backend** – specify which plotting library to use ('matplotlib' or 'bokeh')

Arguments for selecting the DOS components to plot: :type plot_keys:

param plot_keys

optional str list of str, defines the labels you want 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

KKR specific Plotting

Tool to plot a Fermi surface calculated with the qdos mode of KKR

`masci_tools.vis.kkr_plot_FS_qdos.FSqdos2D(p0='/', s=20, reload_data=False, clrbar=True, atoms=None, ax=None, nosave=False, noalat=False, cmap=<matplotlib.colors.LinearSegmentedColormap object>, noplots=False, return_data=False, pclmesh=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='/', data_all=None, totnonly=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', verbose=False)`

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

`masci_tools.vis.kkr_plot_bandstruc_qdos.load_data(p0='/', reload_data=False, nosave=False, atoms=None, ratios=False, atoms2=None, atoms3=None, ef=None, verbose=False)`

load the qdos data

Plotting function for KKR dos from files

```
maschi_tools.vis.kkr_plot_dos.dosplot(p0='.', totnonly=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=None, lmdos=False,
lm=None, 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

Tool to visualize the KKR shapefunctions

```
maschi_tools.vis.kkr_plot_shapefun.change_zoom(ax, zoom_range, center=None)
```

Change the zoom of a 3d plot

Author

Philipp Ruessmann

Parameters

- **ax** – axis which is zoomed
- **zoom_range** (`float`) – range to which the image is zoomed, total range from center-zoom_range to center+zoom_range
- **center** (`Optional[list]`) – center of the zoomed region (optional, defaults to origin)

Return type

`None`

```
maschi_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)

```
maschi_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)

General Plotting

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

`masci_tools.vis.parameters.F`

Generic Callable type

alias of `TypeVar('F', bound=Callable[[...], Any])`

`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*) – Plotter instance

Return type

`Generator[None, None, None]`

```
class masci_tools.vis.parameters.Plotter(default_parameters, general_keys=None,
                                         key_descriptions=None, type_kwargs_mapping=None,
                                         kwargs_postprocess_rename=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.

Parameters

- **default_parameters** (`dict[str, Any]`) – dict with hardcoded default parameters
- **general_keys** (`Optional[set[str]]`) – 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
```

(continues on next page)

(continued from previous page)

```

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) – str name of the parameter
- **default_from** (Optional[str]) – str (optional), if given a entry is created in the current defaults with the name and the default value of the key *default_from*

Return type

None

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** (Any) – value passed in, for multiple plots either list or dict with integer keys
- **single_plot** (bool) – bool, if True only a single parameter is allowed
- **num_plots** (int) – int, if single_plot is False this defines the number of plots
- **default** (Optional[Any]) – default value for unspecified entries
- **key** (str) – str of the key to process

Return type

Any

static dict_of_lists_to_list_of_dicts(*dict_of_lists*, *single_plot*, *num_plots*, *repeat_after=None*, *ignore_repeat=None*)

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[str, list[Any]]*) – dict to be converted
- **single_plot** (*bool*) – boolean, if True only a single parameter set is allowed
- **num_plots** (*int*) – int of the number of allowed plots

Return type

list[dict[str, Any]]

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 \leq original_length are expanded. Also expands function defaults

Parameters

- **original_length** – int of the old length
- **kwargs** (*Any*) – arguments to expand

Return type

dict[str, Any]

Returns

expanded kwargs

get_description(*key*)

Get the description of the given key

Parameters

- **key** (*str*) – str of the key, for which the description should be printed

Return type

None

get_dict()

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

Return type

dict[str, Any]

get_multiple_kwargs(*keys*, *ignore=None*)

Get multiple parameters and return them in a dictionary

Parameters

- **keys** (*set[str]*) – set of keys to process
- **ignore** (*Union[str, list[str], None]*) – str or list of str (optional), defines keys to ignore in the creation of the dict

Return type

dict[str, Any]

is_general(*key*)

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

Parameters

key (*str*) – str of the key to check

Return type

bool

Returns

bool, whether the key is general

load_defaults(*filename*='plot_defaults.json')

Load defaults from a json file.

Parameters

filename (*Union[str, bytes, Path, PathLike, IO[Any]]*) – filename, from where the defaults should be taken

Return type

None

property num_plots: *int*

Integer property for number of plots produced

plot_kwargs(*plot_type*='default', *ignore*=None, *extra_keys*=None, *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 of different types

Parameters

- **plot_type** (*str*) – type of plot
- **ignore** (*Union[str, list[str], None]*) – str or list of str (optional), defines keys to ignore in the creation of the dict
- **extra_keys** (*Optional[set[str]]*) – optional set for additional keys to retrieve
- **post_process** (*bool*) – bool, if True the parameters are cleaned up for inserting them directly into bokeh plotting functions

Return type

Any

Kwargs are used to replace values by custom parameters:

Example for using a custom markersize:

```
p = Plotter(type_kwargs_mapping={'default': {'marker'}})
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*

remove_added_parameters()

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

Return type

None

reset_defaults()

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

Return type

`None`

reset_parameters()

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

Return type

`None`

save_defaults(filename='plot_defaults.json', save_complete=False)

Save the current defaults to a json file.

Parameters

- **filename** (`Union[str, bytes, Path, PathLike, IO[Any]]`) – filename, where the defaults should be stored
- **save_complete** (`bool`) – bool if True not only the overwritten user defaults but also the unmodified hardcoded defaults are stored

Return type

`None`

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`) – 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

Return type

`dict[str, Any]`

Kwargs are used to set the defaults.

set_parameters(continue_on_error=False, **kwargs)

Set the current parameters.

Parameters

continue_on_error (`bool`) – bool, if True unknown key are simply skipped and returned

Return type

`dict[str, Any]`

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`) – str of the key to set
- **value** (`Any`) – 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

Return type

None

property single_plot: bool

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

`masci_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*) – Plotter instance to be checked for consistency

Return type

`Callable[[TypeVar(F, bound= Callable[... , Any])], TypeVar(F, bound= Callable[... , Any])]`

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 `masci_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 `masci_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 masci_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
```

(continues on next page)

(continued from previous page)

```
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, replace_value=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 nomralized 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

Collection of routines to be reused in plotting routines

`masci_tools.vis.helpers.exclude_points(plot_data, *data_keys, limits, padding=0.1)`

Exclude points outside the given limits

Parameters

- **plot_data** (*PlotData*) – PlotData instance containing all the data for plots
- **data_keys** (*str*) – str of the keys to consider for excluding points
- **limits** (*Optional[dict[str, tuple[float, float]]*) – dict of the set plot limits
- **padding** (*float*) – float, determines how far beyond the plot limits a point has to lie to be excluded (default 10%)

Return type

None

`masci_tools.vis.helpers.get_special_kpoint_ticks(kpoints, math_mode='$')`

Process the high symmetry kpoints and ggf. replace with appropriate latex symbol

- Gamma/G is replaced with Γ

Parameters

- **kpoints** (*list[tuple[str, float]]*) – list of tuples with label and position of the points
- **math_mode** (*str*) – Determines the symbol to enter math mode in latex

Return type

tuple[list[float], list[str]]

Returns

Ticks and their respective labels

`masci_tools.vis.helpers.mpl_single_line_or_area(axis, entry, source, area=False, area_vertical=False, area_enclosing_line=True, advance_color_cycle=False, area_line_alpha=1.0, **kwargs)`

Create a scatterplot, lineplot or area plot for the given entry on the matplotlib axis

Parameters

- **axis** – Axes to plot on
- **entry** – namedtuple of the entries to plot
- **source** – mapping containing the data to plot
- **area** – bool, if True fill_betweenx/y will be used to create an area plot
- **area_vertical** – bool, if True fill_betweeny will be used
- **area_enclosing_line** – bool if True the area plot will have another line plot around the edge
- **advance_color_cycle** – bool, if True the matplotlib color cycle will be advanced no matter what else is specified
- **area_line_alpha** – if an area plot is done this is the alpha parameter (transparency)

Kwargs are passed to the respective plotting routines

This module 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 `masci_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

`masci_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 instance

`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': True}</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>color_cycle_always_advance</code>	Always advance the color cycle even if the color was specified	False
<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_parameters</code>	How many times as integer the parameters for single plots (except labels) 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

continues on next page

Table 1 – continued from previous page

Name	Description	Default value
shading	Shading to use for pcolormesh plots	gouraud
rasterized	Rasterize the pcolormesh when drawing vector graphics.	True
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	<code>{'linestyle': '--', 'color': 'k', 'linewidth': 1.0}</code>
font_options	Default font options that can be used for text annotations	<code>{'family': 'serif', 'color': 'black', 'weight': 'normal', 'size': 16}</code>
tick_paramsx	Parameters for major ticks on the x-axis (Size, fontsize, ...)	<code>{'size': 4.0, 'width': 1.0, 'labelsize': 14, 'length': 5, 'labelrotation': 0}</code>
tick_paramsy	Parameters for major ticks on the y-axis (Size, fontsize, ...)	<code>{'size': 4.0, 'width': 1.0, 'labelsize': 14, 'length': 5, 'labelrotation': 0}</code>
tick_paramsx_minor	Parameters for minor ticks on the x-axis (Size, fontsize, ...)	<code>{'size': 2.0, 'width': 1.0, 'labelsize': 0, 'length': 2.5}</code>
tick_paramsy_minor	Parameters for minor ticks on the y-axis (Size, fontsize, ...)	<code>{'size': 2.0, 'width': 1.0, 'labelsize': 0, 'length': 2.5}</code>

continues on next page

Table 1 – continued from previous page

Name	Description	Default value
colorbar	If True and the function implements color mapping, a colorbar is shown	True
colorbar_options	Parameters for displaying the colorbar (Fontsize, ...)	<code>{'pad': 0.05}</code>
legend	If True a legend for the plot is shown	False
legend_show_duplicates	If True labels 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 (plot_type='default', ignore=None, extra_keys=None, 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 of 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 # 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)

asymmetric lorentz function

L^α for $x \leq \mu$ L^β 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 `maschi_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)

```
maschi_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 `maschi_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function `pcolormesh`

```
maschi_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, xdatalabel]

`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

`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 description 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 `maschi_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function *hist*

`maschi_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 a, b, z array like inputs TODO: not clear to me how to do this... the identity is only useful if we mangle 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

`maschi_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

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

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

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

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

`maschi_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 `maschi_tools.vis.matplotlib_plotter.MatplotlibPlotter`. If the arguments are not recognized they are passed on to the matplotlib function *scatter*

`maschi_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*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_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

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 diagram

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 positive!

Args:

coreleveldict: dict of corelevels with a list of corelevel energy of atomstypes # (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 references, 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 `fhwm` (float): full width half maximum of peaks (gauss, 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]}
```

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

Plotting function to visualize corelevels and corelevel shifts

```
maschi_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 Kwarg are passed on to the `multiple_scatterplots()` call

```
maschi_tools.vis.plot_methods.plot_lattice_constant(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 Kwarg will be passed on to `multiple_scatterplots()`

```
masci_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]}}
```

```
masci_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. Absolute relaxation in Angstroem of every atom type. Relative realaxation 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

```
masci_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 channels 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 gauss to lorentz, mix* gauss, (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
<code>figure_kwargs</code>	Parameters for creating the bokeh figure. Includes things like axis type (x and y), tools plot width/height	<pre>{'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'}</pre>
<code>additional_tools</code>	tools to add to the tools already specified in <code>figure_kwargs</code> (Has to be in the same format)	No Default
<code>show_tooltips</code>	Switch whether to add hover tooltips	True
<code>global_tooltips</code>	Switch whether to add individual (for each renderer) or global hover tooltips.	False
<code>format_tooltips</code>	Switch whether to enable the processing of formatted strings in tooltips.	True

continues on next page

Table 2 – continued from previous page

Name	Description	Default value
<code>tooltips</code>	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 <code>format_tooltips=False</code> .	<code>[('X', '@{x}'), ('Y', '@{y}')]</code>
<code>additional_tooltips</code>	Additional tooltips to add to the already defined <code>tooltips</code> (See above)	No Default
<code>axis_linewidth</code>	Linewidth for the lines of the axis	2
<code>label_fontsize</code>	Fontsize for the labels of the axis	18pt
<code>tick_label_fontsize</code>	Fontsize for the ticks on the axis	16pt
<code>background_color</code>	Color of the background of the plot	<code>#ffffff</code>
<code>x_axis_formatter</code>	Use this formatter will be used for the ticks on the x-axis	No Default
<code>y_axis_formatter</code>	Use this formatter will be used for the ticks on the y-axis	No Default
<code>x_ticks</code>	Tick specification for the x-axis	No Default
<code>x_ticklabels</code>	Override the labels for the ticks on the x-axis	No Default
<code>y_ticks</code>	Tick specification for the y-axis	No Default
<code>y_ticklabels</code>	Override the labels for the ticks on the y-axis	No Default
<code>x_range_padding</code>	Specifies the amount of padding on the edges of the x-axis	No Default
<code>y_range_padding</code>	Specifies the amount of padding on the edges of the y-axis	No Default
<code>limits</code>	Dict specifying the limits of the axis, e.g. <code>{ 'x': (-5,5) }</code>	No Default
<code>legend_location</code>	Location of the legend inside the plot area	<code>top_right</code>
<code>legend_click_policy</code>	Policy for what happens when labels are clicked in the legend	<code>hide</code>
<code>legend_orientation</code>	Orientation of the legend	<code>vertical</code>
<code>legend_font_size</code>	Font size for the labels inside the legend	14pt
<code>legend_outside_plot</code>	If True the legend will be placed outside of the plot area	False
<code>color_palette</code>	Color palette to use for the plot(s)	No Default
<code>color</code>	Specific colors to use for the plot(s)	No Default
<code>legend_label</code>	Labels to use for the legend of the plot(s)	No Default
<code>alpha</code>	Transparency to use for the plot(s)	1.0
<code>name</code>	Name used for identifying elements in the plot (not shown only internally)	No Default
<code>line_color</code>	Color to use for line plot(s)	No Default
<code>line_alpha</code>	Transparency to use for line plot(s)	1.0
<code>line_dash</code>	Dash styles to use for line plot(s)	No Default
<code>line_width</code>	Line width to use for line plot(s)	2.0
<code>marker</code>	Type of marker to use for scatter plot(s)	<code>circle</code>
<code>marker_size</code>	Marker size to use for scatter plot(s)	6
<code>area_plot</code>	If True h(v)area will be used to produce the plot(s)	False
<code>area_vertical</code>	Determines, whether to use harea (False) or varea (True) for area plots	False
<code>fill_alpha</code>	Transparency to use for the area in area plot(s)	1.0
<code>fill_color</code>	Color to use for the area in area plot(s)	No Default
<code>level</code>	Can be used to specified, which elements are fore- or background	No Default
<code>straight_lines</code>	Dict specifying straight help-lines to draw. For example <code>{ 'vertical': 0, 'horizontal': [-1,1] }</code> will draw a vertical line at 0 and two horizontal at -1 and 1	No Default

continues on next page

Table 2 – continued from previous page

Name	Description	Default value
<code>straight_line_options</code>	Options, and more options for the help-lines	<pre>{'line_color': ↪ 'black', 'line_width': 1.0, 'line_dash': ↪ 'dashed'}</pre>
<code>text_font_size</code>	Fontsize for the text glyphs in the plot	10pt
<code>text_font_style</code>	Fontstyle for the text glyphs in the plot	normal
<code>text_color</code>	text color for the text glyphs in the plot	black
<code>text_align</code>	text alignment for the text glyphs in the plot	left
<code>text_baseline</code>	text baseline for the text glyphs in the plot	middle
<code>save_plots</code>	If True plots will be saved to file (Configuration beforehand is needed)	False
<code>save_format</code>	Formats to save the plots to, can be single or list of formats (html, png or svg)	html
<code>show</code>	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

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 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

```
masci_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\(\)](#)

```
mascki_tools.vis.bokeh_plots.get_bokeh_help(key)
```

Print the description of the given key in the bokeh backend

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

```
mascki_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

```
mascki_tools.vis.bokeh_plots.matrix_plot(text_values, x_axis_data, y_axis_data, positions=None, *,
                                         color_data=None, secondary_color_data=None,
                                         x_offset=-0.47, log_scale=False, color_map=None, data=None,
                                         copy_data=False, title="", xlabel='x', ylabel='y',
                                         saveas='matrix_plot.html', blank_outsiders='both',
                                         blank_color='#c4c4c4', figure=None, categorical_axis=False,
                                         categorical_sort_key=None, block_size=0.95,
                                         block_size_pixel=100, **kwargs)
```

Plot function for an interactive periodic table plot. Heat map and hover tool. source must be a pandas dataframe containing, atom period and group, atomic number and symbol

Parameters

- **values** – data for the text inside each elements box
- **positions** – y positions relative to the middle of the box for each value
- **color_data** – data to display as a heatmap
- **color_map** – color palette to use for the heatmap (default matplotlib plasma)
- **log_scale** – bool, if True the heatmap is done logarithmically
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **saveas** – str, filename for the saved plot
- **blank_outsiders** – either 'both', 'min', 'max' or None, determines, which points outside the color range to color with a default blank color
- **blank_color** – color to replace values outside the color range by
- **include_legend** – if True an additional entry with labels explaining each value entry is added
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure

Additional kwargs are passed on to the label creation for the element box. The kwargs *legend_options* and *color_bar_options* can be used to overwrite default values for these regions of the plot

```
mascki_tools.vis.bokeh_plots.periodic_table_plot(values, positions=None, *, color_data=None,
                                                  log_scale=False, color_map=None, data=None,
                                                  copy_data=False, title="",
                                                  saveas='periodictable.html', blank_outsiders='both',
                                                  blank_color='#c4c4c4', include_legend=True,
                                                  figure=None, **kwargs)
```

Plot function for an interactive periodic table plot. Heat map and hover tool. source must be a pandas dataframe containing, atom period and group, atomic number and symbol

Parameters

- **values** – data for the text inside each elements box
- **positions** – y positions relative to the middle of the box for each value
- **color_data** – data to display as a heatmap
- **color_map** – color palette to use for the heatmap (default matplotlib plasma)
- **log_scale** – bool, if True the heatmap is done logarithmically
- **data** – source for the data of the plot (optional) (pandas Dataframe for example)
- **title** – str, Title of the plot
- **saveas** – str, filename for the saved plot
- **blank_outsiders** – either 'both', 'min', 'max' or None, determines, which points outside the color range to color with a default blank color
- **blank_color** – color to replace values outside the color range by
- **include_legend** – if True an additional entry with labels explaining each value entry is added
- **figure** – bokeh figure (optional), if provided the plot will be added to this figure

Additional kwargs are passed on to the label creation for the element box. The kwargs *legend_options* and *color_bar_options* can be used to overwrite default values for these regions of the plot.

```
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={'width': 600, '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_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](#)

6.1.2.2 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 maschi_tools.tools.cf_calculation.CFCalculation(*, radial_points=4000, reference_radius='pot',
                                                    coefficient_cutoff=0.001, **kwargs)
```

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

- **radial_points** (`int`) – int, number of radial points in the interpolated mesh
- **reference_radius** (`Union[float, Literal['pot', 'cdn']]`) – 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
- **coefficient_cutoff** (`Optional[float]`) – float Defines minimum value that cf coefficients have to have to be considered non-zero

property denNorm

Returns the density normalization

DEPRECATED: Use `density_normalization` instead

```
classmethod from_arrays(charge_density, potentials, radial_mesh, **kwargs)
```

Create a CFCalculation instance from arrays

Warning: Using this classmethod will not check that the charge density and potential have the same bravais matrix

Parameters

- **charge_density** (`ndarray`) – Array for the normed charge density
- **potentials** (`dict[tuple[int, int], ndarray]`) – Data for the potentials (dict mapping (l,m) tuples to the corresponding data)
- **radial_mesh** (`dict[str, ndarray]`) – dict for the data for the used radial meshes (keys ‘cdn’ and ‘pot’)

Return type

`CFCalculation`

Other Kwargs are passed on to the constructor

get_charge_density(*interpolated=True*)

Return the charge density and the corresponding radial mesh

Parameters

- **spin** – which spin to return
- **interpolated** – bool, if True the interpolated mesh and density are returned

Return type

`tuple[ndarray, ndarray]`

get_coefficients(*convention='Stevens', table_fmt=None*)

Performs the integration to obtain the crystal field coefficients. If the data was not already interpolated, the interpolation will be performed beforehand.

Parameters: :type convention: `Literal['Stevens', 'Wybourne']`

param convention

str of the convention to use (Stevens or Wybourne)

Return type

`list[CFCoefficient]`

Returns

list of CFCoefficient objects (namedtuple), with all the necessary information

get_potentials(*spin, interpolated=True, only_nonzero=True, complex_data=True*)

Return the potentials and the corresponding radial mesh

Parameters

- **spin** (`Literal['up', 'down']`) – which spin to return
- **interpolated** (`bool`) – bool, if True the interpolated mesh and potentials are returned
- **only_nonzero** (`bool`) – bool, if True only the potentials corresponding to a non-zero coefficient are returned
- **complex_data** (`bool`) – bool, if False only the real part of the potentials is returned

Return type

`tuple[ndarray, dict[tuple[int, int], ndarray]]`

interpolate()

Interpolate all quantities to a common equidistant radial mesh

Return type

`None`

property nonzero_coefficients_lm: `list[tuple[int, int]]`

Return the lm indices of coefficients that are bigger than the set cutoff

performIntegration(*convert=True*)

DEPRECATED: Use get_coefficients instead

Performs the integration to obtain the crystal field coefficients. If the data was not already interpolated, the interpolation will be performed beforehand.

Parameters: :type convert:

param convert

bool, converts to Stevens's coefficients (if True)

Returns

list of CFCoefficient objects (namedtuple), with all the necessary information

readCDN(*args, **kwargs)

DEPRECATED: Use read_charge_density instead

readPot(*args, **kwargs)

DEPRECATED: Use read_pot

read_charge_density(file, atom_type=None, header=0, atomic_cdn=True)

Reads in the normed charge density for the CF coefficient calculation If hdf files are given also the muffin tin radius is read in

Return type

None

Parameters: :type file: Union[str, bytes, Path, PathLike, IO[Any], File]

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: :type atom_type: Optional[int]

param atom_type

int, Defines the atom_type to read in (only for HDF files)

read_potential(*files, lm_indices=None, atom_type=None, header=0, complex_data=True)

Reads in the potentials for the CF coefficient calculation If hdf files are given also the muffin tin radius is read in

Parameters

- **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)
- **lm_indices** (Optional[list[tuple[int, int]]]) – list of tuples, Defines the l and m indices for the given txt files
- **atom_type** (Optional[int]) – int, Defines the atomType to read in (only for HDF files)
- **header** (int) – int, Define how many lines to skip in the beginning of txt file
- **complex_data** (bool) – bool, Define if the data in the text file is complex

Return type

None

Raises:

ValueError: lm indices list length has to match number of files read in

property spin_polarized: bool

Return whether the potentials have a spin-down component

stevens_prefactor(*l*, *m*)

Gives the lm dependent prefactor for conversion between Blm and Alm coefficients

Args: :type l: `int`

param l

int; orbital quantum number

Return type

`float`

Returns

float prefactor for conversion to Steven's Coefficients

validate()

Validate that the object can be used to execute the calculation Checks that the given bravais matrices are equal if given

Return type

`None`

class `masci_tools.tools.cf_calculation.CFCoefficient`(*l*: `int`, *m*: `int`, *spin_up*: `float` | `complex`,
spin_down: `float` | `complex`, *unit*: `str`,
convention: `str`)

Namedtuple representing an individual crystal field coefficient

convention: `str`

Alias for field number 5

l: `int`

Alias for field number 0

m: `int`

Alias for field number 1

spin_down: `float` | `complex`

Alias for field number 3

spin_up: `float` | `complex`

Alias for field number 2

unit: `str`

Alias for field number 4

`masci_tools.tools.cf_calculation.plot_crystal_field_calculation`(*cfcalc*, *,
saveas='crystal_field_calc',
potential_title='Potential',
density_title='Density',
xlabel='\$R\$ (Bohr)',
potential_ylabel='\$Vpot\$ (Hartree)',
density_ylabel='Density',
density_kwargs=None,
potential=True, *density*=True,
axis_potential=None,
axis_density=None, ***kwargs*)

Plot the given potentials and charge densities used in the given [CFCalculation](#)

Parameters

- **cfcalc** – CFcalculation containing the data to plot
- **saveas** – str, Define the filename to save the figure
- **potential_title** – Title for the potential subplot
- **density_title** – Title for the charge density subplot
- **xlabel** – label for the x axis of both subplots
- **potential_ylabel** – label for the y axis of the potential subplot
- **density_ylabel** – label for the y axis of the charge density subplot
- **density_kwargs** – dict with keyword argument passed to the plotting of the density
- **figure** – Use this preexisting matplotlib figure to produce the plots

All other Kwargs are passed on to the `multiple_scatterplots()` calls for plotting the potentials

```
masci_tools.tools.cf_calculation.plot_crystal_field_potential(cfcoeffs, *,
                                                             saveas='crystal_field_potential_areaplot',
                                                             spin='avg', phi=0.0, figure=None,
                                                             **kwargs)
```

Plots the angular dependence of the calculated CF potential as well as a plane defined by phi.

Parameters

- **cfcoeffs** – list of CFCoefficients to construct the potential
- **saveas** – str, defines the filename to save the figure
- **spin** – str; Either 'up', 'dn' or 'avg'. Which spin direction to plot ('avg' -> ('up'+'dn')/2.0)
- **phi** – float, defines the phi angle of the plane

Raises

- **AssertionError** – When coefficients are provided as wrong types or in the wrong convention
- **ValueError** – When coefficients are provided 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 -l and l
- **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: $\frac{1}{2i} [G(z) - G(z^*)]$ is returned otherwise the real part $\frac{1}{2} [G(z) + G(z^*)]$

Return type

`ndarray`

Returns

numpy array with the selected data

energy_dependence_full_matrix(*imag=True, both_contours=False*)

Get the full matrix $N_{\text{spins}}(2l + 1) \times N_{\text{spins}}(2l' + 1)$

Parameters

- **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 $\frac{1}{2i} [G(z) - G(z^*)]$ is returned otherwise the real part $\frac{1}{2} [G(z) + G(z^*)]$

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

Return type

[GreensFunction](#)

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

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

moment(*n*, *spin=None*)

Calculate the integral

$$M_n = -\frac{1}{4\pi i} \text{Im} \int_{\text{Contour}} dz z^n G(z)$$

Parameters

- **n** (`int`) – power of z in the integral
- **spin** (`Optional[int]`) – optional integer spin between 1 and nspins

Return type

`ndarray`

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

occupation(*spin=None*)

Calculate the 0-th moment of the green's function

$$n = -\frac{1}{4\pi i} \text{Im} \int_{\text{Contour}} dz G(z)$$

Note: Only if the energy contour ends at the fermi energy/is correctly weighted to produce occupations, will this function produce occupations

Parameters

spin (`Optional[int]`) – optional integer spin between 1 and nspins

Return type

`ndarray`

to_global_frame()

Rotate the Green's function into the global real space and spin space frame

Return type

`None`

to_local_frame()

Rotate the Green's function into the local real space and spin space frame

Return type

`None`

static to_m_index(*m*)

Convert between magnetic quantum numbers between -l and l 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=None*, *imag=True*)

Select trace of data with energy dependence

Parameters

- **spin** (*Optional*[*int*]) – integer spin between 1 and nspins
- **imag** (*bool*) – bool if True the imaginary part $\frac{1}{2i} [G(z) - G(z^*)]$ is returned otherwise the real part $\frac{1}{2} [G(z) + G(z^*)]$

Return type

ndarray

Returns

numpy array with the selected and traced over data

class masci_tools.tools.greensfunction.**GreensfElement**(*l: int*, *lp: int*, *atomType: int*, *atomTypep: int*, *sphavg: bool*, *onsite: bool*, *kresolved: bool*, *contour: int*, *nLO: int*, *atomDiff: np.ndarray*)

Namedtuple representing the high-level information about the Green's functions, i.e. what kind, which atoms, which orbitals

atomDiff: *ndarray*

Alias for field number 9

atomType: *int*

Alias for field number 2

atomTypep: *int*

Alias for field number 3

contour: *int*

Alias for field number 7

kresolved: *bool*

Alias for field number 6

l: *int*

Alias for field number 0

lp: *int*

Alias for field number 1

nLO: *int*

Alias for field number 8

onsite: `bool`

Alias for field number 5

sphavg: `bool`

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*, *max_shells=None*)

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 (correspins to the i)
- **show** (`bool`) – if True the elements belonging to a shell are printed in a shell
- **max_shells** (`Optional[int]`) – optional int, if given only the first max_shells shells are constructed

Return type

`list[tuple[floating[Any], 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*, *max_shells=None*)

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 (correspins to the i)
- **show** (`bool`) – if True the elements belonging to a shell are printed in a shell
- **max_shells** (`Optional[int]`) – optional int, if given only the first max_shells shells are constructed

Return type

`Generator[tuple[floating[Any], GreensFunction, GreensFunction], None, None]`

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*, *max_shells=None*)

Construct the green's function pairs to calculate the Jij exchange constants for a given reference atom from a given greensf.hdf file

Parameters

- **hdf**file (Union[str, bytes, Path, PathLike, IO[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
- **max_shells** (Optional[int]) – optional int, if given only the first max_shells shells are constructed

Return type

Generator[tuple[floating[Any], GreensFunction, GreensFunction], None, None]

Returns

flat iterator with distance and the two corresponding *GreensFunction* instances for each Jij calculation

`masci_tools.tools.greensfunction.listElements(hdf, show=False)`

Find the green's function elements contained in the given `greens.hdf` file

Parameters

- **hdf**file (Union[str, bytes, Path, PathLike, IO[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

```
maschi_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

`Generator[GreensFunction, None, None]`

Returns

iterator over the matching *GreensFunction*

```
maschi_tools.tools.greensfunction.select_elements_from_file(hdffile, show=False,
                                                            **selection_params)
```

Construct the green's function matching specified criteria from a given `greensf.hdf` file

Parameters

- **hdffile** (`Union[str, bytes, Path, PathLike, IO[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

`Generator[GreensFunction, None, None]`

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

```
maschi_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

$$J_0 = \frac{1}{4\pi} \text{Im} \text{Tr}_L \int_{-\infty}^{E_F} dz \Delta (G^\uparrow(z) - G^\downarrow(z)) + \Delta^2 G^\uparrow(z) G^\downarrow(z)$$

Parameters

- **greensfunction** (*GreensFunction*) – *GreensFunction* to use for the calculation
- **onsite_delta** (`float`) – onsite exchange splitting to use for the calculation
- **show** (`bool`) – bool if True additional information about the used Greens functions is printed out

Return type

float

Returns

the value of the spin stiffness in meV

`masci_tools.tools.greensf_calculations.calculate_heisenberg_jij(hdffileORgreensfunctions, reference_atom, onsite_delta, max_shells=None)`

Calculate the Heisenberg exchange constants form Green's functions using the formula

$$J_{ij} = \frac{1}{4\pi} \text{Im Tr}_L \int_{-\infty}^{E_F} dz \Delta_i G_{ij}^{\uparrow}(z) \Delta_j G_{ji}^{\downarrow}(z)$$

Parameters

- **hdffileORgreensfunctions** (Union[str, bytes, Path, PathLike, IO[Any], list[GreensFunction]]) – either pat/file-like object for the greensf.hdf file to use or list of *GreensFunction*
- **reference_atom** (int) – integer index of the atom to calculate the Jij's from
- **onsite_delta** (ndarray) – List of floats containing the onsite exchange splitting for each atom type and l-channel
- **max_shells** (Optional[int]) – optional int, if given only the first max_shells shells are constructed

Return type

DataFrame

Returns

pandas DataFrame containing all the Jij constants

`masci_tools.tools.greensf_calculations.calculate_heisenberg_tensor(hdffileORgreensfunctions, reference_atom, onsite_delta, max_shells=None)`

Calculate the Heisenberg exchange tensor **J** from Green's functions using the formula

$$J_{ij}^{\alpha\beta} = \frac{1}{4\pi} \text{Im Tr}_L \int_{-\infty}^{E_F} dz \Delta_i \sigma_{\alpha} G_{ij}(z) \Delta_j \sigma_{\beta} G_{ji}(z)$$

for all α and $\beta = x, y, z$.

Parameters

- **hdffileORgreensfunctions** (Union[str, bytes, Path, PathLike, IO[Any], list[GreensFunction]]) – either pat/file-like object for the greensf.hdf file to use or list of *GreensFunction*
- **reference_atom** (int) – integer index of the atom to calculate the Jij's from
- **onsite_delta** (ndarray) – List of floats containing the onsite exchange splitting for each atom type and l-channel
- **max_shells** (Optional[int]) – optional int, if given only the first max_shells shells are constructed

Return type

DataFrame

Returns

pandas DataFrame containing all the J_xx, J_xy, etc. constants

`masci_tools.tools.greensf_calculations.calculate_hybridization(greensfunction)`

Calculate the hybridization function as

$$\Delta(z) = \frac{1}{2 * l + 1} \text{Tr} G^{-1}(z)$$

Return type

`ndarray`

Returns

numpy array of the hybridization function

`masci_tools.tools.greensf_calculations.decompose_jij_tensor(jij_tensor, moment_direction)`

Decompose the Heisenberg tensor as calculated by `calculate_heisenberg_tensor()` into three parts

- Isotropic $J = \frac{1}{3} \text{Tr} [\mathbf{J}]$
- Symmetric traceless $J_S = \frac{1}{2} (\mathbf{J} + \mathbf{J}^T) - J$
- Antisymmetric $J_A = \frac{1}{2} (\mathbf{J} - \mathbf{J}^T)$

Parameters

`jij_tensor` (`DataFrame`) – Heisenberg tensor

Return type

`DataFrame`

Returns

tuple of the three aforementioned components

`masci_tools.tools.greensf_calculations.heisenberg_reciprocal(qpoints, jij_data, entry='J_ij')`

Calculate the fourier transform of an entry for interaction constants

Example for J_{ij}

$$J(\mathbf{q}) = \sum_{ij} J_{ij} e^{i\mathbf{q} \cdot \mathbf{R}_{ij}}$$

where \mathbf{R}_{ij} is the connecting vector associated with the J_{ij}

Parameters

- `qpoints` (`ndarray`) – numpy array containing the coordinates of the qpoints
- `jij_data` (`DataFrame`) – DataFrame generated by the above calculation functions
- `entry` (`str`) – str of the entry to calculate

Return type

`ndarray`

Returns

numpy array containing the Fourier transform

6.1.2.3 IO helper functions and file parsers

KKR related IO

In this module you find the `kkparams` class that helps defining the KKR input parameters. Also some defaults for the parameters are defined.

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, `kkparams_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'

static split_kkr_options(*valtxt*)

Split keywords after fixed length of 8 :type valtxt: :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()

Reading of shpefun files of KKR

`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 :type out_dict: :param out_dict: dictionary that is filled with parsed output of the KKrimp calculation :type file_dict: :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_enersp_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

Fleur related IO

Input/Output Parser

Load both the outxml_parser and inxml_parser

`masci_tools.io.parsers.fleur.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.

Return type

`TypeVar(F, bound= Callable[..., Any])`

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

`masci_tools.io.parsers.fleur.inxml_parser(inxmlfile, 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 correctly convert to the dictionary

Parameters

- **inxmlfile** (`Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]`) – either path to the inp.xml file, opened file handle (in bytes modes i.e. rb) 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, _Element, str, bytes, Path, PathLike, IO[Any]]`) – either path to the out.xml file, opened file handle (in bytes modes i.e. rb) 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

`masci_tools.io.parsers.fleur.register_migration(base_version, target_version)`

Decorator to add migration for task definition dictionary to the `_TaskParser` 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** (`str | list[str]`) – str or list of str with the versions that work after the migration has been performed

Return type

`Callable[[TypeVar(F, bound= Callable[... , Any])], TypeVar(F, bound= Callable[... , Any])]`

Inputgenerator related IO

This module contains functionality for writing input files for the input generator of fleur

class `masci_tools.io.fleur_inpgen.AtomDictProperties`

TypedDict for the atom properties

class `masci_tools.io.fleur_inpgen.Kinds`

TypedDict for the kinds

`masci_tools.io.fleur_inpgen.read_inpgen_file(file, convert_to_angstroem=True)`

Method which reads in an ingpen input file and parses the structure and name lists information.

Parameters

- **file** (`Union[str, bytes, Path, PathLike, IO[Any]]`) – path to the file to read or opened file handle
- **convert_to_angstroem** (`bool`) – bool if True the bravais matrix (and atom positions) are converted to angstroem

Return type

`tuple[Optional[ndarray], list[AtomSiteProperties], tuple[bool, bool, bool], dict[str, Any]]`

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, significant_figures_magnetic_moments=4, convert_from_angstroem=True)`

Write an input file for the fleur inputgenerator 'inpgen' from given inputs

Parameters

- **cell** (`ndarray | list[list[float]]`) – 3x3 arraylike. The bravais matrix of the structure, in Angstrom by default
- **atom_sites** (`Union[Sequence[AtomSiteProperties], Sequence[tuple[list[float], str, str]], Sequence[AtomDictProperties]]`) – 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** (`Optional[Iterable[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** (`Union[str, bytes, Path, PathLike, IO[Any]]`) – Path or filehandle where the file should be written to. Defaults to ‘inpgen.in’ in the current folder.
- **pb** (`tuple[bool, bool, bool]`) – tuple of boolean length 3, optional, Periodic boundary conditions of the structure. Defaults to (True, True, True).
- **input_params** (`Optional[dict[str, Any]]`) – Optional dict containing further namelist which should be written to the file. Defaults to None.
- **significant_figures_cell** (`int`) – int, how many decimal places should be written for the bravais matrix (default: 9)
- **significant_figures_positions** (`int`) – int, how many decimal places should be written for the atom positions (default: 10)
- **convert_from_angstroem** (`bool`) – 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.

Return type

`Optional[str]`

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.

Note: The docstrings for the setter methods are generated from their actual implementations in the [xml](#) modules via a pre-commit hook. Changes in the docstrings here will be overwritten

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)
```

(continues on next page)

(continued from previous page)

```
#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, additional_files = fmode.modify_xmlfile('/path/to/input/file/inp.xml')
```

add_number_to_attr(**args*, ***kwargs*)

Add/Multiply values for multiple occurrences in the XML tree

Parameters

- **name** – the attribute name to change
- **number_to_add** – number to add/multiply with the old attribute value
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **mode** – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*
- **occurrences** – int or list of int. Which occurrence of the node to set. By default all are set.

Return type

None

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*

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.add_number_to_attrb('itmax', 10)
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.add_number_to_attrb('radius', 1.05, mode='rel')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.add_number_to_attrb('radius', 1.05, mode='rel', contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.add_number_to_attrb('spinUp',
                      -1,
                      contains='species',
                      filters={
                          'species': {
                              'atomicNumber': {'>': 30}
                          }
                      })
```

This registration method does not modify the file immediately but only appends a `add_number_to_attrb()` to the list of tasks that will be done on the xmltree.

add_number_to_first_attrb(*args, **kwargs)

Add/Multiply value for the first occurrences of an attribute in the XML tree

Parameters

- **name** – the attribute name to change
- **number_to_add** – 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/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

None

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`

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.add_number_to_first_attrb('itmax', 10)
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.add_number_to_first_attrb('radius', 1.05, mode='rel')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.add_number_to_first_attrb('radius', 1.05, mode='rel', contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.add_number_to_first_attrb('spinUp',
                             -1,
                             contains='species',
                             filters={
                                 'species': {
                                     'atomicNumber': {'>': 30}
                                 }})
```

This registration method does not modify the file immediately but only appends a *add_number_to_first_attrb()* to the list of tasks that will be done on the xmltree.

add_task_list(task_list)

Add a list of tasks to be added

Parameters

task_list (`list[tuple[str, dict[str, Any]]]`) – list of tuples first index is the name of the method second is defining the arguments by keyword in a dict

Return type

`None`

align_nmmpmat_to_sqa(*args, **kwargs)

Align the density matrix with the given SQA of the associated species

Parameters

- **species_name** – string, name of the species you want to change
- **orbital** – integer or string ('all'), orbital quantum number of the LDA+U procedure to be modified
- **phi_before** – float or list of floats, angle (radian), values for phi for the previous alignment of the density matrix
- **theta_before** – float or list of floats, angle (radian), values for theta for the previous alignment of the density matrix
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type

`None`

This registration method does not modify the file immediately but only appends a [align_nmmpmat_to_sqa\(\)](#) to the list of tasks that will be done on the xmltree.

classmethod apply_modifications(*xmltree*, *nmmp_lines*, *modification_tasks*, *validate_changes=True*, *adjust_version_for_dev_version=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** (`_ElementTree`) – a lxml tree to be modified (IS MODIFIED INPLACE)
- **nmmp_lines** (`Optional[list[str]]`) – a n_mmp_mat file to be modified (IS MODIFIED INPLACE)
- **modification_tasks** (`list[ModifierTask]`) – a list of modification tuples
- **validate_changes** (`bool`) – bool optional (default True), if True after all tasks are performed both the xmltree and nmmp_lines are checked for consistency
- **adjust_version_for_dev_version** (`bool`) – bool optional (default True), if True and the schema_dict and file version differ, e.g. a development version is used the version is temporarily modified to swallow the validation error that would occur

Return type

`tuple[_ElementTree, Optional[list[str]]]`

Returns

a modified lxml tree and a modified n_mmp_mat file

changes()

Prints out all changes currently registered on this instance

Return type

`list[ModifierTask]`

clone_species(*args, **kwargs)

Clone (and possibly change) a species in the XML tree

For reference of the changes dictionary look at [set_species\(\)](#)

Parameters

- **species_name** – string, name of the specie you want to clone Has to correspond to one single species (no ‘all’/‘all-<search_string>’)
- **new_name** – new name of the cloned species
- **changes** – a optional python dict specifying what you want to change.

Return type

`None`

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.clone_species('Fe-1', 'Fe-clone-1')
```

Modifying the cloned species

The *changes* dictionary is passed on to [set_species](#) to modify the cloned species

```
fm.clone_species('Fe-1', 'Fe-clone-1', {'mtSphere' : {'radius' : 2.7}})
```

This registration method does not modify the file immediately but only appends a [clone_species\(\)](#) to the list of tasks that will be done on the xmltree.

create_tag(*args, **kwargs)

Create a tag with a given name in the XML tree 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

- **tag** – str of the tag to create or etree Element or string representing the XML element with the same name to insert
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **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.

Return type

`None`

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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.create_tag('mtnocoparams')
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.create_tag('lo')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.create_tag('lo', contains='species')
```

Nested creation

With create_parents=True evtl. missing parent tags are also created

```
fm.create_tag('diagElements',
              create_parents=True,
              contains='species',
              not_contains='torque')
```

Creation of pre-created XML element

Passing a XML element instead of a name will insert this element at the correct position

```
fm.create_tag(etree.Element('lo', n='5', l='0', type='SCL0'),
              contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.create_tag(etree.Element('lo', n='5', l='0', type='SCL0'),
              contains='species',
              filters={
                  'species': {
                      'name': {'contains': 'Fe'}
                  }
              })
```

This registration method does not modify the file immediately but only appends a `create_tag()` to the list of tasks that will be done on the xmltree.

`delete_att(*args, **kwargs)`

Delete an attribute with a given name from the XML tree

Parameters

- **name** – str of the attribute to delete
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **occurrences** – int or list of int. Which occurrence of the parent nodes to delete a attribute. By default all nodes are used.

Return type

`None`

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`

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.delete_att('mixparam')
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.delete_att('alpha')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.delete_att('alpha', contains='nocoParams', not_contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.delete_att('alpha',
              contains='nocoParams',
              not_contains='species',
              filters={
                  'atomGroup': {
                      'species': {'not-contains': 'Fe'}
                  }
              })
```

This registration method does not modify the file immediately but only appends a `delete_att()` to the list of tasks that will be done on the xmltree.

`delete_tag(*args, **kwargs)`

Delete a tag with a given name from the XML tree

Parameters

- **tag** – str of the tag to delete
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **occurrences** – int or list of int. Which occurrence of the parent nodes to delete a tag. By default all nodes are used.

Return type

`None`

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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.delete_tag('expertModes')
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.delete_tag('lo')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.delete_tag('lo', contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.delete_tag('lo',
              contains='species',
              filters={
                  'species': {
                      'name': {'contains': 'Fe'}
                  }
              })
```

This registration method does not modify the file immediately but only appends a *delete_tag()* to the list of tasks that will be done on the xmltree.

classmethod fromList(*task_list*, *args, **kwargs)

Instantiate the FleurXMLModifier from a list of tasks to be added immediately

Parameters

task_list (list[tuple[str, dict[str, Any]]]) – 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

Return type

FleurXMLModifier

Returns

class with the task list instantiated

get_avail_actions()

Returns the allowed functions from FleurXMLModifier

Return type

`dict[str, Callable]`

modify_xmlfile(*original_inpxmlfile*, *original_nmmp_file*=None, *validate_changes*=True, *adjust_version_for_dev_version*=True, *keep_inpgen_comments*=True)

Applies the registered modifications to a given inputfile

Parameters

- **original_inpxmlfile** (`Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]`) – either path to the inp.xml file, opened file handle or a xml etree to be parsed
- **original_nmmp_file** (`Union[str, bytes, Path, PathLike, IO[Any], list[str], None]`) – path or list of str to a corresponding density matrix file

Raises

ValueError – if the parsing of the input file

Return type

`tuple[_ElementTree, dict[str, str]]`

Returns

a modified xmltree and if existent a modified density matrix file

replace_tag(*args, **kwargs)

Replace a tag with a given name with a different XML element

Parameters

- **tag** – str of the tag to replace
- **element** – etree Element or string representing the XML element to replace the tag
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **occurrences** – int or list of int. Which occurrence of the parent nodes to replace a tag. By default all nodes are used.

Return type

`None`

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

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.replace_tag('expertmodes', etree.Element('greensFunction'))
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.replace_tag('lo', etree.Element('ldaU', l='3'))
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.replace_tag('lo', etree.Element('ldaU', l='3'), contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.replace_tag('lo',
               etree.Element('ldaU', l='3'),
               contains='species',
               filters={
                   'species': {
                       'atomicNumber': {'>': 30}
                   }
               })
```

This registration method does not modify the file immediately but only appends a `replace_tag()` to the list of tasks that will be done on the xmltree.

rotate_nmmpmat(*args, **kwargs)

Rotate the density matrix with the given angles phi and theta

Parameters

- **species_name** – string, name of the species you want to change
- **orbital** – integer or string ('all'), 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
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type

None

This registration method does not modify the file immediately but only appends a `rotate_nmmpmat()` to the list of tasks that will be done on the xmltree.

set_atomgroup(*args, **kwargs)

Set the attributes of a atom group, specified by species name, position, etc.

Parameters

- **changes** – 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
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

None

changes 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:

```
'changes': {'nocoParams': {'beta': val}}
```

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, species='Fe-1')
```

Specifying the number of the atomgroup

Providing the `position` argument will modify the n-th atomgroup in the `inp.xml` file. (Indexing starts at 1)

```
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, position=1)
```

Modifying all species

Providing ‘all’ to either `position` or `species`` will modify all atomgroups

```
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, species='all')
```

This registration method does not modify the file immediately but only appends a [set_atomgroup\(\)](#) to the list of tasks that will be done on the xmltree.

set_atomgroup_label(*args, **kwargs)

Set the attributes of a atom group, given by an atom label occuring in it’s definition

This method calls [set_atomgroup\(\)](#) method for a certain atom species that corresponds to an atom with a given label.

Parameters

- **atom_label** – string, a label of the atom which specie will be changed. ‘all’ to change all the species
- **changes** – a python dict specifying what you want to change.

Return type*None*

changes 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:

```
'changes': {'nocoParams': {'beta': val}}
```

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_atomgroup_label('222', {'nocoParams': {'alpha': 1.57}})
```

Modifying all species

Providing ‘all’ as the first argument applies the changes to all species

```
fm.set_atomgroup_label('all', {'nocoParams': {'alpha': 1.57}})
```

This registration method does not modify the file immediately but only appends a *set_atomgroup_label()* to the list of tasks that will be done on the xmltree.

set_attr_value(*args, **kwargs)

Set the value of an attribute possibly occurring multiple times

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

- **name** – the attribute name to set
- **value** – value or list of values to set
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **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

Return type*None***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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_attr_value('itmax', 180)
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_attr_value('radius', 1.05)
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_attr_value('radius', 1.05, contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.set_attr_value('spinUp',
                  1,
                  contains='species',
                  filters={
                      'species': {
                          'atomicNumber': {'>': 30}
                      }
                  })
```

This registration method does not modify the file immediately but only appends a *set_attr_value()* to the list of tasks that will be done on the xmltree.

set_complex_tag(*args, **kwargs)

Set attributes, text and subtags of a given tag in the XML tree

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

- **tag_name** – name of the tag to set
- **changes** – 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
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **create** – bool optional (default False), if True and the path, where the complex tag is set does not exist it is created

Return type

[None](#)

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

This registration method does not modify the file immediately but only appends a [set_complex_tag\(\)](#) to the list of tasks that will be done on the xmltree.

set_first_attr_value(*args, **kwargs)

Set the value of the first occurrence of an attribute

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

- **name** – the attribute name to set
- **value** – value or list of values to set
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **create** – bool optional (default False), if True the tag is created if is missing

Return type

[None](#)

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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_first_attrib_value('itmax', 180)
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_first_attrib_value('radius', 1.05)
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_first_attrib_value('radius', 1.05, contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.set_first_attrib_value('spinUp',
                           1,
                           contains='species',
                           filters={
                               'species': {
                                   'atomicNumber': {'>': 30}
                               }})
```

This registration method does not modify the file immediately but only appends a *set_first_attrib_value()* to the list of tasks that will be done on the xmltree.

set_first_text(*args, **kwargs)

Set the value of text of the first occurrence of a tag in the XML tree

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

- **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
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create** – bool optional (default False), if True the tag is created if is missing

Return type

None

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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_first_text('kpoint', [1,2,3])
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_first_text('valenceConfig', '(1s1/2)')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_first_text('valenceConfig', '(1s1/2)', contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.set_first_text('valenceConfig',
                  '(1s1/2)',
                  contains='species',
                  filters={
                      'species': {
                          './lo/@n': {'>': 4}
                      }
                  })
```

This registration method does not modify the file immediately but only appends a `set_first_text()` to the list of tasks that will be done on the xmltree.

set_inpchanges(*args, **kwargs)

Set the value of attributes/text occurring once in the XML tree

The first occurrence of the attribute/tag is set

Parameters

- **changes** – dictionary {attrib_name : value} with all the wanted changes.
- **path_spec** – dict, with ggf. necessary further specifications for the path of the attribute

Return type

None

An example of changes:

```
changes = {
    'itmax' : 1,
    'l_noco': True,
    'ctail': False,
    'l_ss': True
}
```

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_inpchanges({'itmax' : 1, 'kmax': 4.3})
```

Attribute selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_inpchanges({'itmax' : 1, 'theta': 1.57})
```

Attribute selection

Selection can be done by adding conditions on what the XPath should(n't) contain in the `path_spec` argument

```
fm.set_inpchanges({'itmax' : 1, 'theta': 1.57},
                  path_spec={'theta': {'contains': 'soc'}})
```

This registration method does not modify the file immediately but only appends a `set_inpchanges()` to the list of tasks that will be done on the xmltree.

set_kpath(*args, **kwargs)

Sets a k-path directly as an alternative kpoint set with purpose ‘bands’

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

Return type

None

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_kpath({'Point': [0, 0, 0], 'Point2': [1, 1, 1]}, 200)
```

This registration method does not modify the file immediately but only appends a `set_kpath()` to the list of tasks that will be done on the xmltree.

set_kpointlist(*args, **kwargs)

Create a k-point list with the given points and weights

Note: If no name is given, a name of the form default-<number> is generated

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

Return type

None

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_kpointlist([[0,0,0],[0.5,0.5,0.5]],
                  [1,1], switch=True)
```

MaX 4 compatibility

For input files before the MaX 5 release the previous kpoint list will always be overwritten and the `switch` argument has no effect

```
fm.set_kpointlist([[0,0,0],[0.5,0.5,0.5]],
                  [1,1])
```

This registration method does not modify the file immediately but only appends a `set_kpointlist()` to the list of tasks that will be done on the xmltree.

`set_kpointmesh(*args, **kwargs)`

Create a kpoint mesh using spglib

for details see `get_stabilized_reciprocal_mesh()`

Parameters

- **mesh** – list-like with three elements, giving the size of the kpoint set in each direction
- **use_symmetry** – bool if True the available symmetry operations in the inp.xml will be used to reduce the kpoint set otherwise only the identity matrix is used
- **name** – Name of the created kpoint list. If not given a name is generated
- **switch** – bool if True the kpoint list is directly set as the used set
- **overwrite** – if True and a kpoint list of the given name already exists it will be overwritten
- **shift** – shift the center of the kpoint set
- **time_reversal** – bool if True time reversal symmetry will be used to reduce the kpoint set
- **map_to_first_bz** – bool if True the kpoints are mapped into the [0,1] interval

Return type

None

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage


```
fm.set_kpointmesh([4, 4, 1], switch=True)
```

This registration method does not modify the file immediately but only appends a `set_kpointmesh()` to the list of tasks that will be done on the xmltree.

set_kpointpath(*args, **kwargs)

Create a kpoint list for a bandstructure calculation (using ASE kpath generation)

The path can be defined explicitly (see `bandpath()`) or derived from the unit cell

Parameters

- **path** – str, list of str or None defines the path to interpolate (for syntax `bandpath()`)
- **nkpts** – int number of kpoints in the path
- **density** – float number of kpoints per Angstrom
- **name** – Name of the created kpoint list. If not given a name is generated
- **switch** – bool if True the kpoint list is directly set as the used set
- **overwrite** – if True and a kpoint list of the given name already exists it will be overwritten
- **special_points** – dict mapping names to coordinates for special points to use

Return type

None

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_kpointpath(nkpts=50, switch=True)
```

Specifying custom points

By default the path is chosen by ASE. By providing path this can be modified

```
fm.set_kpointpath(path='CA',
                  nkpts=25,
                  special_points={
                      'C': [0, 0, 0],
                      'A': [0, 0, 0.5]
                  })
```

This registration method does not modify the file immediately but only appends a `set_kpointpath()` to the list of tasks that will be done on the xmltree.

set_nkpts(*args, **kwargs)

Sets a k-point mesh directly specific for inputs of version Max4

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

Return type

None

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_nkpts(200)
```

Include Gamma point

```
fm.set_nkpts(200, gamma=True)
```

This registration method does not modify the file immediately but only appends a *set_nkpts()* to the list of tasks that will be done on the xmltree.

set_nmmpmat(*args, **kwargs)

Routine sets the block in the n_mmp_mat file specified by species_name, orbital and spin to the desired density matrix

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
- **filters** – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Raises

- **ValueError** – If something in the input is wrong
- **KeyError** – If no LDA+U procedure is found on a species

Return type

None

This registration method does not modify the file immediately but only appends a *set_nmmpmat()* to the list of tasks that will be done on the xmltree.

set_simple_tag(*args, **kwargs)

Sets one or multiple simple tag(s) (no subtags/text) 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

- **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
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **create_parents** – bool optional (default False), if True and the path, where the simple tags are set does not exist it is created

Return type

None

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

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.set_simple_tag('soc', {'theta': 0.1, 'phi': 0.2, 'l_soc': True})
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_simple_tag('lo', [{'l': 0, 'n': 6, 'type': 'SCL0'},
                        {'l': 1, 'n': 6, 'type': 'SCL0'}])
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_simple_tag('lo', [{'l': 0, 'n': 6, 'type': 'SCL0'},
                        {'l': 1, 'n': 6, 'type': 'SCL0'}],
                    contains='species')
```

Nested creation

With `create_parents=True` if the parents of the tag are missing they are created

```
fm.set_simple_tag('realAxis', {'ne': 1300, 'ellow': -1.0, 'elup': 1.0}, create_
↳parents=True)
```

Added filters

The filters argument allows to be more specific

```
fm.set_simple_tag('lo', [{'l': 0, 'n': 6, 'type': 'SCL0'},
                        {'l': 1, 'n': 6, 'type': 'SCL0'}],
                  contains='species',
                  filters={
                      'species': {
                          'atomicNumber': {'>': 30}
                      }})
```

This registration method does not modify the file immediately but only appends a `set_simple_tag()` to the list of tasks that will be done on the xmltree.

set_species(*args, **kwargs)

Set the attributes of a species, specified by name position, etc.

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)
- **changes** – a python dict specifying what you want to change.
- **create** – bool, if species does not exist create it and all subtags?
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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 xmltree

xml etree of the new inp.xml

Return type

None

changes 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

Simple Usage

```
fm.set_species('Fe-1', {'mtSphere' : {'radius' : 2.7}})
```

Adding child elements

The *changes* dictionary is not limited to setting attributes Child elements can also be created. Notice here that if multiple elements of a given name are allowed all previously existing elements are deleted and replaced with the ones specified in *changes*

```
fm.set_species('Fe-1', {'ldaU' : {'l' : 3, 'U' : 4.0, 'J' : 0.5, 'l_amf' : True},
                        'lo' : [{'l' : 0, 'n' : 6, 'type' : 'SCL0'},
                                {'l' : 1, 'n' : 6, 'type' : 'SCL0'}]})
```

Modifying all species

Providing 'all' as the first argument applies the changes to all species

```
fm.set_species('all', {'mtSphere' : {'radius' : 2.7}})
```

Modifying a subset of species

Providing 'all-<search string>' as the first argument applies the changes to all species which contain the search string in it's name

```
fm.set_species('all-Pt', {'mtSphere' : {'radius' : 2.7},
                          'lo' : {'l' : 0, 'n' : 6, 'type' : 'SCL0'},})
```

Example species. All these attributes tags can be modified. Additional keys can for example be found in the FLEUR documentation flapw.de

```
<species name="Fe-1" element="Fe" atomicNumber="26">
  <mtSphere radius="2.200000000" gridPoints="787" logIncrement=".016000000"/>
  <atomicCutoffs lmax="10" lnonsphr="6"/>
  <electronConfig>
    <coreConfig>[Ne]</coreConfig>
    <valenceConfig>(3s1/2) (3p1/2) (3p3/2) (4s1/2) (3d3/2) (3d5/2)</
    ↪ valenceConfig>
    <stateOccupation state="(3d3/2)" spinUp="2.000000000" spinDown="1.000000000
    ↪"/>
    <stateOccupation state="(3d5/2)" spinUp="3.000000000" spinDown=".000000000"/
    ↪>
  </electronConfig>
  <energyParameters s="4" p="4" d="3" f="4"/>
  <lo type="SCL0" l="0" n="3" eDeriv="0"/>
  <lo type="SCL0" l="1" n="3" eDeriv="0"/>
</species>
```

This registration method does not modify the file immediately but only appends a `set_species()` to the

list of tasks that will be done on the xmltree.

set_species_label(*args, **kwargs)

Set the attributes of a species, given by an atom label occurring in it's atom type

This method calls [set_species\(\)](#) method for a certain atom species that corresponds to an atom with a given label

Parameters

- **atom_label** – string, a label of the atom which specie will be changed. 'all' to change all the species
- **changes** – a python dict specifying what you want to change.
- **create** – bool, if species does not exist create it and all subtags?

Return type

None

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.set_species_label('222', {'mtSphere' : {'radius' : 2.7}})
```

Adding child elements

The *changes* dictionary is not limited to setting attributes Child elements can also be created. Notice here that if multiple elements of a given name are allowed all previously existing elements are deleted and replaced with the ones specified in *changes*

```
fm.set_species_label('222', {'ldaU' : {'l' : 3, 'U' : 4.0, 'J' : 0.5, 'l_amf': ↵
↵True},
                               'lo': [{'l': 0, 'n': 6, 'type': 'SCL0'},
                                       {'l': 1, 'n': 6, 'type': 'SCL0']})
```

Modifying all species

Providing 'all' as the first argument applies the changes to all species

```
fm.set_species_label('all', {'mtSphere' : {'radius' : 2.7}})
```

This registration method does not modify the file immediately but only appends a [set_species_label\(\)](#) to the list of tasks that will be done on the xmltree.

set_text(*args, **kwargs)

Set the value of text of tags in the XML tree possibly occurring mutiple times

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

- **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
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **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

Return type

`None`

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

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.set_text('kpoint', [[1,2,3], [4,5,6]])
```

Setting one value

If only one value for the text is given all elements are set of the same value in this case only one list of three coordinates is set

```
fm.set_text('kpoint', [4,4,4])
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_text('q', '1 1 1')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_text('q', '1 1 1', contains='spinSpiral')
```

Added filters

The filters argument allows to be more specific

```
fm.set_text('valenceConfig',
            '(1s1/2)',
            contains='species',
            filters={
                'species': {
                    './lo/@n': {'>': 4}
                }
            })
```

This registration method does not modify the file immediately but only appends a `set_text()` to the list of tasks that will be done on the xmltree.

set_xcfunctional(*args, **kwargs)

Set the Exchange Correlation potential tag

Parameters

- **xc_functional** – str or dict. If str it is the name of a inbuilt XC functional. If it is a dict it specifies either the name or id for LibXC functionals for the keys `'exchange'`, `'correlation'`, `'etot_exchange'` and `'etot_correlation'`
- **xc_functional_options** – dict with further general changes to the `xcFunctional` tag
- **libxc** – bool if True the functional is a LibXC functional

Return type

None

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_xcfunctional('pbe')
```

Setting a LibXC functional

```
fm.set_xcfunctional({'exchange': 'lda_x', 'correlation': 'lda_c_xalpha'},
                    libxc=True)
```

This registration method does not modify the file immediately but only appends a `set_xcfunctional()` to the list of tasks that will be done on the xmltree.

shift_value(*args, **kwargs)

Add/Multiply values of attributes occurring once in the XML tree

The first occurrence of the attribute is shifted

Parameters

- **changes** – a python dictionary with the keys to shift and the shift values.

- **mode** – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with the given value *abs/absolute* adds the old value and the given value
- **path_spec** – dict, with ggf. necessary further specifications for the path of the attribute

Return type

None

An example of changes:

```
changes = {'itmax' : 1, 'dVac': -0.123}
```

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.shift_value({'itmax' : 5, 'mindistance': 1})
```

Relative shift

Passing mode="rel" or mode="relative" will multiply the initial attribute values with the provided number instead of adding them

```
fm.shift_value({'itmax' : 2, 'mindistance': 3}, mode="relative")
```

This registration method does not modify the file immediately but only appends a *shift_value()* to the list of tasks that will be done on the xmltree.

shift_value_species_label(*args, **kwargs)

Add/Multiply numeric attributes in a or multiple atom species Shifts the value of an attribute on a species by label if atom_label contains 'all' then applies to all species

Parameters

- **atom_label** – string, a label of the atom which specie will be changed. 'all' if set up all species
- **attribute_name** – name of the attribute to change
- **number_to_add** – value to add or to multiply by
- **mode** – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*

Return type

None

Kwargs if the attribute_name 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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.shift_value_species_label('222', 'radius', 1)
```

Relative shift

Passing `mode="rel"` or `mode="relative"` will multiply the initial attribute value with the provided number instead of adding them

```
fm.shift_value_species_label('222', 'radius', 1.2, mode="relative")
```

Modifying all species

Providing `'all'` as the first argument applies the changes to all atomgroups

```
fm.shift_value_species_label('all', 'lmax', 2)
```

This registration method does not modify the file immediately but only appends a `shift_value_species_label()` to the list of tasks that will be done on the xmltree.

switch_kpointset(*args, **kwargs)

Switch the used k-point set

Warning: This method is only supported for input versions after the Max5 release

Parameters

list_name – name of the kPoint set to use

Return type

`None`

This registration method does not modify the file immediately but only appends a `switch_kpointset()` to the list of tasks that will be done on the xmltree.

switch_species(*args, **kwargs)

Change the species of an atom group, identified by species name, position, etc.

Parameters

- **new_species_name** – name of the species to switch to
- **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
- **clone** – if True and the new species name does not exist and it corresponds to changing from one species the species will be cloned with `clone_species()`
- **changes** – changes to do if the species is cloned

- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

None

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.switch_species('Pt-1', species='Fe-1')
```

Specifying the number of the atomgroup

Providing the `position` argument will modify the `n`-th atomgroup in the `inp.xml` file. (Indexing starts at 1)

```
fm.switch_species('Pt-1', position=1)
```

Create a clone of the old species

Specifying a non-existent new species name and `clone=True` will create a clone of the old species with that name. Additionally the `changes` argument can then be used to apply changes to the new clone

```
fm.switch_species('Fe-clone-2', species='Fe-1', clone=True, changes={'mtSphere':  
↪: {'radius' : 2.7}})
```

Modifying all species

Providing `'all'` to either `position` or `species`` will modify the species of all atomgroups

```
fm.switch_species('Fe-1', species='all')
```

This registration method does not modify the file immediately but only appends a [switch_species\(\)](#) to the list of tasks that will be done on the xmltree.

switch_species_label(*args, **kwargs)

Change the species of an atom group, identified by the lable of an atom in it's group

Parameters

- **atom_label** – string, a label of the atom which group will be changed. `'all'` to change all the groups
- **new_species_name** – name of the species to switch to
- **clone** – if True and the new species name does not exist and it corresponds to changing from one species the species will be cloned with [clone_species\(\)](#)
- **changes** – changes to do if the species is cloned

Return type

`None`

This registration method does not modify the file immediately but only appends a `switch_species_label()` to the list of tasks that will be done on the xmltree.

property task_list: `list[tuple[str, dict[str, Any]]]`

Return the current changes in a format accepted by `add_task_list()` and `fromList()`

undo(*revert_all=False*)

Cancels the last change or all of them

Parameters

revert_all (`bool`) – set True if need to cancel all the changes, False if the last one.

Return type

`list[ModifierTask]`

xml_create_tag(**args, **kwargs*)

Creates a tag under the given XPath 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

- **xpath** – a path where to place a new tag
- **element** – a tag name, etree Element or string representing the XML 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, ...)

Return type

`None`

This registration method does not modify the file immediately but only appends a `xml_create_tag()` to the list of tasks that will be done on the xmltree.

xml_delete_att(**args, **kwargs*)

Deletes an attribute on the tags of the given XPath in the XML tree

Parameters

- **xpath** – a path to the attribute to be deleted
- **name** – the name of an attribute to delete
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type

None

This registration method does not modify the file immediately but only appends a `xml_delete_att()` to the list of tasks that will be done on the xmltree.

xml_delete_tag(*args, **kwargs)

Deletes a tag under the XPath in the XML tree.

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.

Return type

None

This registration method does not modify the file immediately but only appends a `xml_delete_tag()` to the list of tasks that will be done on the xmltree.

xml_replace_tag(*args, **kwargs)

Replace XML tags under the XPath by a given tag

Parameters

- **xpath** – a path to the tag to be replaced
- **element** – an Element or string representing the Element to replace the found tags with
- **occurrences** – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type

None

This registration method does not modify the file immediately but only appends a `xml_replace_tag()` to the list of tasks that will be done on the xmltree.

xml_set_attrib_value_no_create(*args, **kwargs)

Sets the value of an attribute under the XPath By default the attribute will be set on all nodes returned for the specified xpath.

Parameters

- **xpath** – a path where to set the attributes
- **name** – the attribute name to set
- **value** – 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

Return type

None

This registration method does not modify the file immediately but only appends a `xml_set_attrib_value_no_create()` to the list of tasks that will be done on the xmltree.

xml_set_text_no_create(*args, **kwargs)

Sets the text of a tag under the XPath By default the text will be set on all nodes returned for the specified xpath.

Parameters

- **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

Return type

None

This registration method does not modify the file immediately but only appends a *xml_set_text_no_create()* to the list of tasks that will be done on the xmltree.

class masci_tools.io.fleurxmlmodifier.**ModifierTask**(name, args, kwargs)

args: *tuple*[Any, ...]

Alias for field number 1

kwargs: *dict*[str, Any]

Alias for field number 2

name: *str*

Alias for field number 0

Functions/Classes for loading/validating fleur XML files

This module provides the classes for easy access to information from the fleur input and output xsd schema files

masci_tools.io.parsers.fleur_schema.schema_dict.F

Generic Type variable for callable

alias of *TypeVar('F', bound=Callable[[...], Any])*

exception masci_tools.io.parsers.fleur_schema.schema_dict.**IncompatibleSchemaVersions**

Exception raised when it is known that a given output version and input version cannot be compiled into a complete fleur output xml schema

class masci_tools.io.parsers.fleur_schema.schema_dict.**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 <)

`exception maschi_tools.io.parsers.fleur_schema.schema_dict.NoPathFound`

Exception raised when no path is found for a given tag/attribute

`exception maschi_tools.io.parsers.fleur_schema.schema_dict.NoUniquePathFound`

Exception raised when no unique path is found for a given tag/attribute

```
class masci_tools.io.parsers.fleur_schema.schema_dict.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

iteration_tags

Names of the elements that can contain all iteration tags

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 <)

iteration_attr_xpath(*name*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the attribute and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

Parameters

- **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 `iteration_tag_xpath()`
- **iteration_tag** (`str`) – name of the tag containing the iteration information

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

iteration_tag_xpath(*name*, *contains=None*, *not_contains=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the `schema_dict` based on the given name of the tag and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

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
- **iteration_tag** (`str`) – name of the tag containing the iteration information

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

property out_version: `tuple[int, int]`

Returns the output version as an integer for comparisons (> or <)

relative_iteration_attr_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*, *iteration_tag='iteration'*)

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
- **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
- **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_iteration_tag_xpath()`
- **iteration_tag** (`str`) – name of the tag containing the iteration information

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_iteration_tag_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

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
- **iteration_tag** (`str`) – name of the tag containing the iteration information

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

class `masci_tools.io.parsers.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
- **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_attrib_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
- **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
- **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*)

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

`TagInfo`

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

validate(*xmltree*, *logger=None*, *header=""*)

Validate the given XML tree against the schema

Parameters

- **xmltree** (`_ElementTree`) – XML tree to validate
- **logger** (`Optional[Logger]`) – Logger to relay evlt warnings/errors

Return type

`None`

class `masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDictDispatch(*args, **kwargs)`

Protocol representing function decorated by the `schema_dict_version_dispatch` decorator

register(*min_version=Ellipsis*, *max_version=Ellipsis*)

Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

Return type

`Callable[[TypeVar(F, bound= Callable[... Any]), TypeVar(F, bound= Callable[... Any])]`

`masci_tools.io.parsers.fleur_schema.schema_dict.list_available_versions(output_schema)`

List the available versions for the schema

Parameters

output_schema (`bool`) – bool, if True search for `FleurOutputSchema.xsd` otherwise `FleurInputSchema.xsd`

Return type

`list[str]`

Returns

list version string of the available versions

`masci_tools.io.parsers.fleur_schema.schema_dict.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[[TypeVar(F, bound= Callable[... , Any])], SchemaDictDispatch[TypeVar(F,
bound= Callable[... , Any])]]
```

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.fleur_xml.FleurXMLContext(etree_or_element, schema_dict, constants=None,
                                         logger=None)
```

Contextmanager to hold the relevant datastructures for evaluating values from XML files

This holds:

Return type

```
Generator[_EvalContext, None, None]
```

The following methods are available:

- `_EvalContext.attribute()`: Evaluate attribute values (has an additional argument *default* to provide values for missing attributes)
- `_EvalContext.text()`: Evaluate text of tags
- `_EvalContext.all_attributes()`: Evaluate all attributes of a tag
- `_EvalContext.parent_attributes()`: Evaluate attribute values of parent of given tag
- `_EvalContext.single_value()`: Evaluate *value* and *unit* attribute of tag
- `_EvalContext.tag_exists()`: Evaluate whether a given tag exists
- `_EvalContext.number_nodes()`: Evaluate how many elements of the tag are present
- `_EvalContext.attribute_exists()`: Evaluate whether an attribute exists
- `_EvalContext.simple_xpath()`: Evaluate the simple xpath expression for a given tag
- *Nested Context* `_EvalContext.find()`: Find the first occurrence of the tag and provide a nested context to that element
- *Nested Context* `_EvalContext.iter()`: Find all occurrences of the tag and provide a nested context to these elements when iterated over
- *Nested Context* `_EvalContext.nested()`: Explicitly inherit all the context except the XML tree to a new context with the XML element replaced by the given argument

Example Usage:

```
from masci_tools.io.fleur_xml import load_inpxml, FleurXMLContext
xmltree, schema_dict = load_inpxml('/path/to/inp.xml')

with FleurXMLContext(xmltree, schema_dict) as root:
    spins = root.attribute('jspins')
    noco = root.attribute('l_noco', default=False)
```

(continues on next page)

(continued from previous page)

```
#Not nesting the context we need to specify which elements are meant
mt_radrii = root.attribute('radius', contains='species')

#Nesting using find
with root.find('atomspecies') as all_species:
    mt_radrii = all_species.attribute('radius')

#Nesting using iter
mt_radrii = []
for species in root.iter('species'):
    mt_radrii.append(species.attribute('radius'))
```

```
class masci_tools.io.fleur_xml._EvalContext(etree_or_element, schema_dict, constants=None,
                                             logger=None)
```

Contextmanager to hold the relevant datastructures for evaluating values from XML files

This holds:

```
all_attributes(name, **kwargs)
```

Alias for `evaluate_tag()`

Parameters

- **name** (`str`) – str, name of the tag
- **subtags** – optional bool, if True the subtags of the given tag are evaluated
- **text** – optional bool, if True the text of the tag is also parsed
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

`Any`

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

attribute(*name*, *default=None*, ***kwargs*)

Alias for [evaluate_attribute\(\)](#)

Parameters

- **name** (*str*) – str, name of the attribute
- **default** (*Optional[Any]*) – value to return, if there are no values found
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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`

attribute_exists(*name*, ***kwargs*)

Alias for [attrib_exists\(\)](#)

Parameters

- **name** (*str*) – str, name of the attribute
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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

find(name, **kwargs)

Finds the first element for the given name and constraints and gives a nested context for this element, i.e. inheriting the schema_dict, constants and logger

Example:

```
with FleurXMLContext(xmltree, schema_dict) as root:
    #Operations happen on the root here
    jspins = root.attribute('jspins')
    with root.find('species') as species:
        #Operations in this block happen on the first species node
        radius = species.attribute('radius')
```

Parameters

- **name** (str) – str, name of the tag
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

_GeneratorContextManager[_EvalContext]

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

iter(name, **kwargs)

Finds all elements for the given name and constraints and gives nested contexts for these elements to be iterated over, i.e. inheriting the schema_dict, constants and logger

Example:

```
with FleurXMLContext(xmltree, schema_dict) as root:
    #Operations happen on the root here
    jspins = root.attribute('jspins')
    for species in root.iter('species'):
        #Operations happen on one species node in each iteration of the
        #for loop. The order is the same as they appear in the XML file
        radius = species.attribute('radius')
```

Parameters

- **name** (`str`) – str, name of the tag
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

`Generator[_EvalContext, None, None]`

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

nested(*etree_or_element*)

Create a nested context from the current one inheriting the schema_dict, constants and logger only replacing the XML element

Example of explicit usage:

```
with FleurXMLContext(xmltree, schema_dict) as root:
    #Operations happen on the root here
    jspins = root.attribute('jspins')

    all_species_tag = root.simple_xpath('atomspecies')
    with root.nested(all_species_tag) as all_species:
        #Operations happen on the 'atomSpecies' tag
        mt_radII = all_species.attribute('radius')
```

Parameters

etree_or_element (`Union[_Element, _ElementTree]`) – Element to use for evaluation in the nested context

Return type

`Generator[_EvalContext, None, None]`

number_nodes(*name*, ***kwargs*)

Alias of [get_number_of_nodes\(\)](#)

Parameters

name (`str`) – str, name of the tag

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 iteration_path

bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters

Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

`int`

Returns

number of nodes of the given tag

parent_attributes(*name*, ***kwargs*)

Alias for [evaluate_parent_tag\(\)](#)

Parameters

- **name** (`str`) – str, name of the tag
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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`

simple_xpath(*name*, ***kwargs*)

Alias for [eval_simple_xpath\(\)](#)

Parameters

- **name** (*str*) – str, name of the tag
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **list_return** – bool, if True a list is always returned

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

[list](#)[[_Element](#)] | [_Element](#)

Returns

etree Elements obtained via the simple xpath expression

single_value(*name*, ***kwargs*)

Alias for [evaluate_single_value_tag\(\)](#)

Parameters

- **name** (*str*) – str, name of the tag
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation

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

param iteration_path

bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters

Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

[Any](#)

Returns

value and unit, both converted in `convert_xml_attribute`

tag_exists(*name*, ***kwargs*)

Alias for [tag_exists\(\)](#)

Parameters

- **name** ([str](#)) – str, name of the tag
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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 tag exists

text(*name*, ***kwargs*)

Alias for [evaluate_text\(\)](#)

Parameters

- **name** ([str](#)) – str, name of the tag
- **complex_xpath** – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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.io.fleur_xml.get_constants(xmltree, 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** – root of the etree of the inp.xml file
- **schema_dict** (*InputSchemaDict* | *OutputSchemaDict*) – 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.io.fleur_xml.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, _Element, str, bytes, Path, PathLike, IO[Any]]) – either path to the inp.xml file, opened file handle (in bytes modes i.e. rb) 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.fleur_xml.load_outxml(outxmlfile, logger=None, base_url=None, **kwargs)`

Loads a out.xml file for fleur together with its corresponding schema dictionary. Also returns whether the XML file had to be parsed with *recover=True*

Parameters

outxmlfile (*Union*[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]) – either path to the out.xml file, opened file handle (in bytes modes i.e. rb) 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

`masci_tools.io.fleur_xml.load_outxml_and_check_for_broken_xml(outxmlfile, logger=None, base_url=None, **kwargs)`

Loads a out.xml file for fleur together with its corresponding schema dictionary. Also returns whether the XML file had to be parsed with *recover=True*

Parameters

outxmlfile (`Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]`) – either path to the out.xml file, opened file handle (in bytes modes i.e. `rb`) or a xml etree to be parsed

Return type

`tuple[_ElementTree, OutputSchemaDict, bool]`

Returns

parsed xmltree of the outxmlfile and the schema dictionary for the corresponding output version and bool indicating whether the outxml is broken

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, inverse=False)`

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_nmmmpmat.write_nmmmpmat(orbital, denmat, phi=None, theta=None, inverse=False)`

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_nmmmpmat.write_nmmmpmat_from_orbitals(orbital, orbital_occupations, phi=None, theta=None, inverse=False)`

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
- $\frac{1}{\sqrt{2}} (Y_l^{-1} + Y_l^1)$
- $\frac{i}{\sqrt{2}} (Y_l^{-1} - Y_l^1)$
- $\frac{1}{\sqrt{2}} (Y_l^{-2} + Y_l^2)$
- $\frac{i}{\sqrt{2}} (Y_l^{-2} - Y_l^2)$
- 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, inverse=False)
```

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

General HDF5 parser

This module contains a generic HDF5 reader

```
class masci_tools.io.parsers.hdf5.reader.AttribTransformation(name, attrib_name, args, kwargs)
```

args: `tuple[Any, ...]`

Alias for field number 2

attrib_name: `str`

Alias for field number 1

kwargs: `dict[str, Any]`

Alias for field number 3

name: `str`

Alias for field number 0

```
class masci_tools.io.parsers.hdf5.reader.HDF5LimitedTransformation
```

```
class masci_tools.io.parsers.hdf5.reader.HDF5Reader(file, move_to_memory=True,
                                                    filename='UNKNOWN')
```

Class for reading in data from hdf5 files using a specified recipe

Parameters

- **file** (`Union[str, bytes, Path, PathLike, IO[Any]]`) – filepath to hdf file or opened file handle (mode ‘rb’)
- **move_to_memory** (`bool`) – bool if True after reading and transforming the data all leftover h5py.Datasets are moved into np.arrays
- **filename** (`str`) – Name of the file. Only used for logging. If not given and the file provides the information extract it from there

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 maschi_tools.io.parsers.hdf5 import HDF5Reader
import maschi_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 (Optional[HDF5Recipe]) – dict with the format given in *recipes*

Return type

tuple[dict[str, Any], dict[str, Any]]

Returns

two dicts with the datasets/attributes read in and transformed according to the recipe

class maschi_tools.io.parsers.hdf5.reader.HDF5Recipe

class maschi_tools.io.parsers.hdf5.reader.HDF5Transformation

class maschi_tools.io.parsers.hdf5.reader.Transformation(*name, args, kwargs*)

args: tuple[Any, ...]

Alias for field number 1

kwargs: dict[str, Any]

Alias for field number 2

name: str

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 maschi_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': {
```

(continues on next page)

(continued from previous page)

```

        '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 correpond 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** ([Literal](#)['Local', 'jDOS', 'Orbcomp', 'MCD']) – str of the group the weights should be taken from
- **simple** ([bool](#)) – bool, if True no additional weights are retrieved with the produced recipe

Return type

[HDF5Recipe](#)

Returns

dict of the recipe to retrieve a bandstructure calculation

`masci_tools.io.parsers.hdf5.recipes.dos_recipe_format(group)`

Format for denisty of states calculations retrieving the DOS from the given group

Parameters

group ([Literal](#)['Local', 'jDOS', 'Orbcomp', 'MCD']) – str of the group the DOS should be taken from

Return type

[HDF5Recipe](#)

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** (`str` | `list[str]`) – key or list of keys of the weight(s) to retrieve
- **group** (`Literal['Local', 'jDOS', 'Orbcomp', 'MCD']`) – optional str (default Local) name of the group from where to take the weights

Return type

`HDF5Recipe`

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

`masci_tools.io.parsers.hdf5.transforms.F`

Generic Callable type

alias of `TypeVar('F', bound=Callable[[...], Any])`

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_format=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:{}'format` 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_format** – callable returning a formatted string This will be called with every entry in the `attribute_value` list

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) – bool if True this function takes a previously processed attribute value and is therefore only available for the entries in datasets

Return type

`Callable[[TypeVar(F, bound= Callable[... Any]), TypeVar(F, bound= Callable[... Any])]`

`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: `:type transpose:`

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

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 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

If `parse_type` is not equal to `xmlGetter` the following key is required:

param path_spec

dict with all the arguments that should be passed to `tag_xpath` or `attrib_xpath` to get the correct path

param kwargs

additional arguments to pass to the parsing function

In the case of `xmlGetter` the following keys are allowed:

param name

name of the function in `masci_tools.util.xml.xml_getters` (required)

param result_names

list of str defining the keys under which to enter the outputs of the function

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 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 from masci_tools.util.parse_utils import Conversion
2
3 __working_out_versions__ = {'0.34', '0.35', '0.36', '0.37'}
4 __base_version__ = '0.34'
5
6 TASKS_DEFINITION = {
7     #-----Definitions for general info from outfile (start, endtime, number_
8     ↪ iterations)-----
9     'general_out_info': {
10         '_general': True,
11         '_minimal': True,
12         '_conversions': [Conversion(name='calculate_walltime')],
13         'creator_name': {
14             'parse_type': 'attrib',
15             'path_spec': {
16                 'name': 'version',
17                 'not_contains': 'git'
18             }
19         },
20         'creator_target_architecture': {
21             'parse_type': 'text',
22             'path_spec': {
23                 'name': 'targetComputerArchitectures'
24             }
25         }
26     },
27 }
```

(continues on next page)

(continued from previous page)

```

25     'output_file_version': {
26         'parse_type': 'attrib',
27         'path_spec': {
28             'name': 'fleurOutputVersion'
29         }
30     },
31     'number_of_iterations': {
32         'parse_type': 'numberNodes',
33         'path_spec': {
34             'name': 'iteration'
35         }
36     },
37     'number_of_atoms': {
38         'parse_type': 'attrib',
39         'path_spec': {
40             'name': 'nat'
41         }
42     },
43     'number_of_atom_types': {
44         'parse_type': 'attrib',
45         'path_spec': {
46             'name': 'ntype'
47         }
48     },
49     'number_of_kpoints': {
50         'parse_type': 'attrib',
51         'path_spec': {
52             'name': 'count',
53             'contains': 'numericalParameters'
54         }
55     },
56     'start_date': {
57         'parse_type': 'allAttribs',
58         'path_spec': {
59             'name': 'startDateAndTime'
60         },
61         'flat': False,
62         'kwargs': {
63             'ignore': ['zone'],
64         }
65     },
66     'end_date': {
67         'parse_type': 'allAttribs',
68         'path_spec': {
69             'name': 'endDateAndTime'
70         },
71         'flat': False,
72         'kwargs': {
73             'ignore': ['zone'],
74         }
75     }
76 },

```

(continues on next page)

(continued from previous page)

```

77  #-----Defintions for general info from input section of outfile (kmax, symmetries,
78  ↪ ..)-----
79  'general_inp_info': {
80      '_general': True,
81      '_minimal': True,
82      'title': {
83          'parse_type': 'text',
84          'path_spec': {
85              'name': 'comment'
86          }
87      },
88      'kmax': {
89          'parse_type': 'attrib',
90          'path_spec': {
91              'name': 'Kmax'
92          }
93      },
94      'gmax': {
95          'parse_type': 'attrib',
96          'path_spec': {
97              'name': 'Gmax'
98          }
99      },
100      'number_of_spin_components': {
101          'parse_type': 'attrib',
102          'path_spec': {
103              'name': 'jspins'
104          }
105      },
106      'number_of_symmetries': {
107          'parse_type': 'numberNodes',
108          'path_spec': {
109              'name': 'symOp'
110          }
111      },
112      'number_of_species': {
113          'parse_type': 'numberNodes',
114          'path_spec': {
115              'name': 'species'
116          }
117      },
118      'film': {
119          'parse_type': 'exists',
120          'path_spec': {
121              'name': 'filmPos'
122          }
123      },
124      #-----Defintions for lda+u info from input section (species, ldau tags)-----
125      'ldau_info': {
126          '_general': True,
127          '_modes': [('ldau', True)],

```

(continues on next page)

(continued from previous page)

```

128     '_conversions': [Conversion(name='convert_ldau_definitions')],
129     'parsed_ldau': {
130         'parse_type': 'allAttribs',
131         'path_spec': {
132             'name': 'ldaU',
133             'contains': 'species'
134         },
135         'subdict': 'ldau_info',
136         'flat': False,
137         'kwargs': {
138             'only_required': True
139         }
140     },
141     'ldau_species': {
142         'parse_type': 'parentAttribs',
143         'path_spec': {
144             'name': 'ldaU',
145             'contains': 'species'
146         },
147         'subdict': 'ldau_info',
148         'flat': False,
149         'kwargs': {
150             'only_required': True
151         }
152     }
153 },
154     'ldahia_info': {
155         '_general': True,
156         '_modes': [('ldahia', True)],
157         '_conversions': [Conversion(name='convert_ldahia_definitions')],
158         'parsed_ldahia': {
159             'parse_type': 'allAttribs',
160             'path_spec': {
161                 'name': 'ldaHIA',
162                 'contains': 'species'
163             },
164             'subdict': 'ldahia_info',
165             'flat': False,
166         },
167         'ldahia_species': {
168             'parse_type': 'parentAttribs',
169             'path_spec': {
170                 'name': 'ldaHIA',
171                 'contains': 'species'
172             },
173             'subdict': 'ldahia_info',
174             'flat': False,
175             'kwargs': {
176                 'only_required': True
177             }
178         }
179     },

```

(continues on next page)

(continued from previous page)

```

180 #-----Defintions for relaxation info from input section (bravais matrix, atompos)
181 #-----for Bulk and film
182 'relax_info': {
183     '_general': True,
184     '_modes': [
185         ('relax', True),
186     ],
187     '_conversions': [Conversion(name='convert_relax_info')],
188     'parsed_relax_info': {
189         'parse_type': 'xmlGetter',
190         'name': 'get_structuredata',
191         'kwargs': {
192             'convert_to_angstroem': False,
193             'include_relaxations': False
194         }
195     },
196 },
197 'hubbard1_distances': {
198     '_general': True, #General because not every iteration contains a hubbard1_
199     iteration
200     '_minimum_version': '0.35',
201     '_minimal': True,
202     '_modes': [('ldahia', True)],
203     'occupation_distance': {
204         'parse_type': 'attrib',
205         'path_spec': {
206             'name': 'occupationDistance'
207         },
208         'kwargs': {
209             'iteration_path': True,
210             'list_return': True
211         },
212         'subdict': 'ldahia_info',
213     },
214     'element_distance': {
215         'parse_type': 'attrib',
216         'path_spec': {
217             'name': 'elementDistance'
218         },
219         'kwargs': {
220             'iteration_path': True,
221             'list_return': True
222         },
223         'subdict': 'ldahia_info',
224     },
225 },
226 #----General iteration tasks
227 # iteration number
228 # total energy (only total or also contributions, also lda+u correction)
229 # distances (nonmagnetic and magnetic, lda+u density matrix)
230 # charges (total, interstitial, mt sphere)
231 # fermi energy and bandgap

```

(continues on next page)

(continued from previous page)

```

231 # magnetic moments
232 # orbital magnetic moments
233 # forces
234 'iteration_number': {
235     '_minimal': True,
236     'number_of_iterations_total': {
237         'parse_type': 'attrib',
238         'path_spec': {
239             'name': 'overallNumber'
240         },
241         'overwrite_last': True,
242     }
243 },
244 'total_energy': {
245     '_minimal':
246     True,
247     '_conversions': [
248         Conversion(name='convert_htr_to_ev', kwargs={
249             'name': 'energy_hartree',
250             'converted_name': 'energy',
251         }),
252         Conversion(name='convert_htr_to_ev',
253             kwargs={
254                 'name': 'ts_energy_hartree',
255                 'converted_name': 'ts_energy',
256                 'pop': True
257             })
258     ],
259     'energy_hartree': {
260         'parse_type': 'attrib',
261         'path_spec': {
262             'name': 'value',
263             'tag_name': 'freeEnergy'
264         }
265     },
266     'energy_hartree_units': {
267         'parse_type': 'attrib',
268         'path_spec': {
269             'name': 'units',
270             'tag_name': 'totalEnergy'
271         },
272         'overwrite_last': True,
273     },
274     'ts_energy_hartree': {
275         'parse_type': 'attrib',
276         'path_spec': {
277             'name': 'value',
278             'tag_name': 'tkbtimesentropy'
279         }
280     }
281 },
282 'distances': {

```

(continues on next page)

(continued from previous page)

```

283     '_minimal': True,
284     'density_convergence': {
285         'parse_type': 'attrib',
286         'path_spec': {
287             'name': 'distance',
288             'tag_name': 'chargeDensity'
289         }
290     },
291     'density_convergence_units': {
292         'parse_type': 'attrib',
293         'path_spec': {
294             'name': 'units',
295             'tag_name': 'densityConvergence',
296         },
297     'overwrite_last': True,
298     }
299 },
300 'magnetic_distances': {
301     '_minimal': True,
302     '_modes': [('jspin', 2)],
303     'overall_density_convergence': {
304         'parse_type': 'attrib',
305         'path_spec': {
306             'name': 'distance',
307             'tag_name': 'overallChargeDensity'
308         }
309     },
310     'spin_density_convergence': {
311         'parse_type': 'attrib',
312         'path_spec': {
313             'name': 'distance',
314             'tag_name': 'spinDensity'
315         }
316     }
317 },
318 'total_energy_contributions': {
319     'sum_of_eigenvalues': {
320         'parse_type': 'singleValue',
321         'path_spec': {
322             'name': 'sumOfEigenvalues'
323         },
324     'kwargs': {
325         'only_required': True
326     }
327 },
328 'energy_core_electrons': {
329     'parse_type': 'singleValue',
330     'path_spec': {
331         'name': 'coreElectrons',
332         'contains': 'sumOfEigenvalues'
333     },
334     'kwargs': {

```

(continues on next page)

(continued from previous page)

```

335         'only_required': True
336     },
337 },
338 'energy_valence_electrons': {
339     'parse_type': 'singleValue',
340     'path_spec': {
341         'name': 'valenceElectrons'
342     },
343     'kwargs': {
344         'only_required': True
345     }
346 },
347 'charge_den_xc_den_integral': {
348     'parse_type': 'singleValue',
349     'path_spec': {
350         'name': 'chargeDenXCDenIntegral'
351     },
352     'kwargs': {
353         'only_required': True
354     }
355 },
356 },
357 'ldau_energy_correction': {
358     '_modes': [('ldau', True)],
359     'ldau_energy_correction': {
360         'parse_type': 'singleValue',
361         'path_spec': {
362             'name': 'dftUCorrection'
363         },
364         'subdict': 'ldau_info',
365         'kwargs': {
366             'only_required': True
367         }
368     },
369 },
370 'nmmmp_distances': {
371     '_minimal': True,
372     '_modes': [('ldau', True)],
373     'density_matrix_distance': {
374         'parse_type': 'attrib',
375         'path_spec': {
376             'name': 'distance',
377             'contains': 'ldaUDensityMatrixConvergence'
378         },
379         'subdict': 'ldau_info'
380     },
381 },
382 'fermi_energy': {
383     'fermi_energy': {
384         'parse_type': 'singleValue',
385         'path_spec': {
386             'name': 'FermiEnergy'

```

(continues on next page)

(continued from previous page)

```

387         },
388     },
389 },
390 'bandgap': {
391     '_modes': [('bz_integration', 'hist')],
392     'bandgap': {
393         'parse_type': 'singleValue',
394         'path_spec': {
395             'name': 'bandgap'
396         },
397     },
398 },
399 'magnetic_moments': {
400     '_modes': [('jspin', 2)],
401     'magnetic_moments': {
402         'parse_type': 'allAttribs',
403         'path_spec': {
404             'name': 'magneticMoment'
405         },
406         'base_value': 'moment',
407         'kwargs': {
408             'ignore': ['atomType'],
409         }
410     },
411 },
412 'orbital_magnetic_moments': {
413     '_modes': [('jspin', 2), ('soc', True)],
414     'orbital_magnetic_moments': {
415         'parse_type': 'allAttribs',
416         'path_spec': {
417             'name': 'orbMagMoment'
418         },
419         'base_value': 'moment',
420         'kwargs': {
421             'ignore': ['atomType'],
422         }
423     },
424 },
425 'global_magnetic_moments': {
426     '_minimum_version': '0.35',
427     '_modes': [('noco', True)],
428     'magnetic_vec_moments': {
429         'parse_type': 'attrib',
430         'path_spec': {
431             'name': 'vec',
432             'contains': 'magnetic',
433             'tag_name': 'globalMagMoment'
434         },
435     },
436 },
437 'magnetic_moment': {
438     '_minimum_version': '0.36',

```

(continues on next page)

(continued from previous page)

```

439     '_modes': [('jspin', 2), ('noco', False)],
440     'magnetic_vec_moments': {
441         'parse_type': 'attrib',
442         'path_spec': {
443             'name': 'vec',
444             'contains': 'magneticMomentsIn',
445             'not_contains': 'local'
446         }
447     },
448 },
449 'forces': {
450     '_minimal': True,
451     '_modes': [('relax', True)],
452     '_conversions': [Conversion(name='convert_forces')],
453     'force_units': {
454         'parse_type': 'attrib',
455         'path_spec': {
456             'name': 'units',
457             'tag_name': 'totalForcesOnRepresentativeAtoms'
458         },
459     },
460     'overwrite_last': True
461 },
462 'parsed_forces': {
463     'parse_type': 'allAttribs',
464     'path_spec': {
465         'name': 'forceTotal'
466     },
467     'flat': False,
468     'kwargs': {
469         'only_required': True
470     }
471 },
472 'charges': {
473     '_conversions': [Conversion(name='calculate_total_magnetic_moment')],
474     'spin_dependent_charge': {
475         'parse_type': 'allAttribs',
476         'path_spec': {
477             'name': 'spinDependentCharge',
478             'contains': 'allElectronCharges',
479             'not_contains': 'fixed'
480         },
481     },
482     'kwargs': {
483         'only_required': True
484     }
485 },
486 'total_charge': {
487     'parse_type': 'singleValue',
488     'path_spec': {
489         'name': 'totalCharge',
490         'contains': 'allElectronCharges',
491         'not_contains': 'fixed'

```

(continues on next page)

(continued from previous page)

```

491         },
492         'kwargs': {
493             'only_required': True
494         }
495     },
496 },
497 #-----Tasks for forcetheorem Calculations
498 # DMI, JIJ, MAE, SSDISP
499 'forcetheorem_dmi': {
500     '_special': True,
501     'dmi_force': {
502         'parse_type': 'allAttribs',
503         'path_spec': {
504             'name': 'Entry',
505             'contains': 'DMI'
506         }
507     },
508     'dmi_force_so': {
509         'parse_type': 'allAttribs',
510         'path_spec': {
511             'name': 'allAtoms',
512             'contains': 'DMI'
513         }
514     },
515     'dmi_force_qs': {
516         'parse_type': 'attrib',
517         'path_spec': {
518             'name': 'qpoints',
519             'contains': 'Forcetheorem_DMI'
520         }
521     },
522     'dmi_force_angles': {
523         'parse_type': 'attrib',
524         'path_spec': {
525             'name': 'Angles',
526             'contains': 'Forcetheorem_DMI'
527         }
528     },
529     'dmi_force_units': {
530         'parse_type': 'attrib',
531         'path_spec': {
532             'name': 'units',
533             'contains': 'Forcetheorem_DMI'
534         }
535     }
536 },
537 'forcetheorem_ssdisp': {
538     '_special': True,
539     'spst_force': {
540         'parse_type': 'allAttribs',
541         'path_spec': {
542             'name': 'Entry',

```

(continues on next page)

(continued from previous page)

```

543         'contains': 'SSDISP'
544     },
545     'spst_force_qs': {
546         'parse_type': 'attrib',
547         'path_spec': {
548             'name': 'qvectors',
549             'contains': 'Forcetheorem_SSDISP'
550         }
551     },
552     'spst_force_units': {
553         'parse_type': 'attrib',
554         'path_spec': {
555             'name': 'units',
556             'contains': 'Forcetheorem_SSDISP'
557         }
558     }
559 },
560 'forcetheorem_mae': {
561     '_special': True,
562     'mae_force': {
563         'parse_type': 'allAttribs',
564         'path_spec': {
565             'name': 'Angle',
566             'contains': 'MAE'
567         }
568     },
569     'mae_force_units': {
570         'parse_type': 'attrib',
571         'path_spec': {
572             'name': 'units',
573             'contains': 'Forcetheorem_MAE'
574         }
575     }
576 },
577 'forcetheorem_jij': {
578     '_special': True,
579     'jij_force': {
580         'parse_type': 'allAttribs',
581         'path_spec': {
582             'name': 'Config',
583             'contains': 'JIJ'
584         }
585     },
586     'jij_force_units': {
587         'parse_type': 'attrib',
588         'path_spec': {
589             'name': 'units',
590             'contains': 'Forcetheorem_JIJ'
591         }
592     }
593 },
594 },

```

(continues on next page)

(continued from previous page)

```

595 'torques': {
596     '_minimum_version': '0.35', #Typo torque/torque before
597     '_optional': True,
598     'torque_x': {
599         'parse_type': 'attrib',
600         'path_spec': {
601             'name': 'sigma_x',
602             'contains': 'noncollinearTorque'
603         }
604     },
605     'torque_y': {
606         'parse_type': 'attrib',
607         'path_spec': {
608             'name': 'sigma_y',
609             'contains': 'noncollinearTorque'
610         }
611     }
612 },
613 'noco_angles': {
614     '_general': True,
615     '_optional': True,
616     'noco_alpha': {
617         'parse_type': 'attrib',
618         'path_spec': {
619             'name': 'alpha',
620             'tag_name': 'nocoParams',
621             'contains': 'Group'
622         }
623     },
624     'noco_beta': {
625         'parse_type': 'attrib',
626         'path_spec': {
627             'name': 'beta',
628             'tag_name': 'nocoParams',
629             'contains': 'Group'
630         }
631     }
632 },
633 'corelevels': {
634     '_optional': True,
635     'corestates': {
636         'parse_type': 'allAttribs',
637         'path_spec': {
638             'name': 'coreStates'
639         },
640         'kwargs': {
641             'subtags': True,
642         },
643         'flat': False
644     }
645 }
646 }

```

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

Return type

`dict[str, dict[str, Any]]`

Changes:

- LDA+U density matrix distance output did not exist

`masci_tools.io.parsers.fleur.task_migrations.migrate_034_to_033(definition_dict)`

Migrate definitions for MaX5 bugfix release to MaX5 release

Return type

`dict[str, dict[str, Any]]`

Changes:

- forcetheorem units attribute did not exist (get from ‘sumValenceSingleParticleEnergies’)

6.1.2.4 Commandline interface (CLI)

masci_tools

CLI for the *masci-tools* library.

```
masci_tools [OPTIONS] COMMAND [ARGS]...
```

Options

-v, --version

Show the version and exit.

convert-inpgen

Convert the given file to an fleur inpgen file

```
masci_tools convert-inpgen [OPTIONS] INPUT_FILE OUTPUT_FILE
```

Options

-c, --converter <converter>

Which library is used to read in the given file format

Options

ase | pymatgen

Arguments

INPUT_FILE

Required argument

OUTPUT_FILE

Required argument

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_schema* corresponding to its version number

```
masci_tools fleur-schema add [OPTIONS] SCHEMA_FILE
```

Options

--overwrite

Overwrite any existing schema-file

--branch <branch>

If the file does not exist the branch can be specified in the fleur git

--api-key <api_key>

API key for access to the Iff Gitlab instance

--test-xml-file <test_xml_file>

Example xmlfile for this schema version to test the file parser against

--from-git

Add the schema from the fleur git repository

Arguments

SCHEMA_FILE

Required argument

list

Show the available fleur schemas

```
masci_tools fleur-schema list [OPTIONS]
```

pull

Pull the default XML schema files from the iffgit and store them in the subfolder of *masci_tools/io/parsers/fleur_schema* corresponding to its version number

```
masci_tools fleur-schema pull [OPTIONS] BRANCH
```

Options

--api-key <api_key>

API key for access to the Iff Gitlab instance

--test-xml-file <test_xml_file>

Example xmlfile for this schema version to test the file parser against

Arguments

BRANCH

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

inpxml

Tool for converting inp.xml files to different versions

```
masci_tools inpxml [OPTIONS] COMMAND [ARGS]...
```

convert

Convert the given XML_FILE file to version TO_VERSION

XML_FILE is the file to convert TO_VERSION is the file version of the finale input file

```
masci_tools inpxml convert [OPTIONS] XML_FILE TO_VERSION
```

Options

-o, --output-file <output_file>

Name of the output file

--overwrite

If the flag is given and the file already exists it is overwritten

Arguments

XML_FILE

Required argument

TO_VERSION

Required argument

generate-conversion

Generate the conversions from FROM_VERSION to TO_VERSION

FROM_VERSION is the file version of the initial input file TO_VERSION is the file version of the finale input file

```
masci_tools inpxml generate-conversion [OPTIONS] FROM_VERSION TO_VERSION
```

Options

--show, --no-show

Show a summary of the conversion at the end

Arguments

FROM_VERSION

Required argument

TO_VERSION

Required argument

show-conversion

Show the actions for an already created conversion from FROM_VERSION to TO_VERSION

FROM_VERSION is the file version of the initial input file TO_VERSION is the file version of the finale input file

```
masci_tools inpxml show-conversion [OPTIONS] FROM_VERSION TO_VERSION
```

Arguments

FROM_VERSION

Required argument

TO_VERSION

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

```
masci_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 of the given xml file

```
masci_tools parse cell [OPTIONS] XML_FILE
```

Arguments

XML_FILE

Required argument

constants

Parse the mathematical constants used in the given xml-file

```
masci_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

```
maschi_tools parse kpoints [OPTIONS] XML_FILE
```

Arguments

XML_FILE

Required argument

nkpts

Extract the number of kpoints used in the given xml file

```
maschi_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

```
masci_tools parse out-file [OPTIONS] XML_FILE
```

Options

--ignore-validation

Arguments

XML_FILE

Required argument

parameters

Parse the calculation parameters of the given xml file

```
masci_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

```
masci_tools parse symmetry [OPTIONS] XML_FILE
```

Arguments

XML_FILE

Required argument

tag-exists

Return whether the specified tag exists in the given xml file

```
masci_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

```
masci_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 | FleurMCDBands

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

6.1.2.5 Utility Functions/Classes

Custom Datatypes

This module defines subclasses of UserDict and UserList to be able to prevent unintended modifications

`maschi_tools.util.lockable_containers.LockContainer(lock_object)`

Contextmanager for temporarily locking a lockable object. Object is unfrozen when exiting with block

Parameters

lock_object (*LockableList*[Any] | *LockableDict*[Any, Any]) – lockable container (not yet frozen)

Return type

Iterator[None]

class `maschi_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[*TypeVar*(S), *TypeVar*(T_co, covariant=True)]

property locked: bool

Returns whether the object is locked

```
class masci_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[TypeVar(T)]`

insert(*i, item*)

S.insert(index, value) – insert value before index

Return type

`None`

property locked: `bool`

Returns whether the object is locked

pop(*i=-1*)

return the value at index i (default last) and remove it from list

Return type

`TypeVar(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`

`masci_tools.util.lockable_containers.S`

Type variable for the key type of the dictionary

alias of `TypeVar('S')`

`masci_tools.util.lockable_containers.T`

Type variable for the value type of the list

alias of `TypeVar('T')`

`masci_tools.util.lockable_containers.T_co`

Type variable for the value type of the dictionary

alias of `TypeVar('T_co', covariant=True)`

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, recursive=True, **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[TypeVar(T_co, covariant=True)]]`) – 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[TypeVar(T_co, covariant=True)]`

intersection(*others)

Return the intersection of two sets as a new set.

(i.e. all elements that are in both sets.)

Return type

`CaseInsensitiveFrozenSet[TypeVar(T_co, covariant=True)]`

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[TypeVar(T_co, covariant=True)]`

union(*others)

Return the union of sets as a new set.

(i.e. all elements that are in either set.)

Return type

`CaseInsensitiveFrozenSet[Union[TypeVar(T_co, covariant=True), TypeVar(_S)]]`

`masci_tools.util.case_insensitive_dict.S`

Generic Type

alias of `TypeVar('S')`

`masci_tools.util.case_insensitive_dict.T_co`

Generic Type

alias of `TypeVar('T_co', covariant=True)`

This module defines some aliases used in typing

`masci_tools.util.typing.FileLike`

Type used for functions accepting file-like objects, i.e. handles or file paths

alias of `Union[str, bytes, Path, PathLike, IO[Any]]`

`masci_tools.util.typing.TXPathLike`

Type for xpath expressions

alias of `TypeVar('XPathLike', bound=Union[str, bytes, XPath, XPathBuilder])`

`masci_tools.util.typing.XMLFileLike`

Type used for functions accepting xml-file-like objects, i.e. handles or file paths or already parsed xml objects

alias of `Union[_ElementTree, _Element, str, bytes, Path, PathLike, IO[Any]]`

`masci_tools.util.typing.XMLLike`

Type used for functions accepting xml objects from lxml

alias of `Union[_Element, _ElementTree]`

`masci_tools.util.typing.XPathLike`

Type for xpath expressions

alias of `Union[str, bytes, XPath, XPathBuilder]`

Common XML utility

Common functions for parsing input/output files or XMLschemas from FLEUR

`masci_tools.util.xml.common_functions.T`

Generic Type

alias of `TypeVar('T')`

`masci_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

`masci_tools.util.xml.common_functions.add_tag(xpath, tag)`

Add tag to xpath

Note: etree.XPath objects could lose context in here, i.e. non-default options passed at init

Parameters

- **xpath** (`TypeVar(TXPathLike, bound= Union[str, bytes, XPath, XPathBuilder])`) – xpath to change
- **tag** (`str`) – str of the tag to add

Return type

`TypeVar(TXPathLike, bound= Union[str, bytes, XPath, XPathBuilder])`

Returns

xpath with the form {old_xpath}/tag

`masci_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** (`Union[_Element, _ElementTree, XPathElementEvaluator]`) – root node of an etree or an etree
- **base_xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – str of the xpath without complex syntax
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – str of the xpath to check

Raises

ValueError – If the complex_xpath does not produce a subset of the results of the base_xpath

Return type

`None`

`masci_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

`masci_tools.util.xml.common_functions.contains_tag(xpath, tag)`

Return whether a given xpath contains a given tag This assumes that predicates of xpaths can't be nested since otherwise the regex for removing them could fail

This function will only return True if one of the tags exactly matches the tag argument not if one tag contains the given name in it's name

Parameters

- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – xpath expression
- **tag** (`str`) – tag to check for

Return type

`bool`

Returns

whether a tag is contained in the xpath

`masci_tools.util.xml.common_functions.eval_xpath(node, xpath, logger=None, list_return=False, namespaces=None, **variables)`

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_inpgen_comments(xmltree)`

Get the XML comment element appended after the root of the inp.xml file

These contain at the moment the inpgen command line and the content of the inpgen file

Parameters

xmltree (`_ElementTree`) – representation of the inp.xml

Return type

`list[_Element]`

Returns

list of XML comments, which appear after the fleurInput tag

`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.is_valid_tag(tag)`

Return whether the given string is a valid XML tag name

Parameters

tag (`str`) – tag to check

Return type

`bool`

`masci_tools.util.xml.common_functions.normalize_xmllike(xmllike)`

Returns the root of the xmltree

Return type

`_Element`

`masci_tools.util.xml.common_functions.process_xpath_argument(simple_xpath, complex_xpath, filters)`

Process the simple and complex Xpath expressions and given filters Used for unifying the logic for all xml setters/evaluators using these arguments

Parameters

- **simple_xpath** (`str` | `bytes` | `XPath`) – The simple XPath (no predicates) expression to base the paths on
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – Optional XPath given with no restrictions
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type

`Union[str, bytes, XPath, XPathBuilder]`

Returns

Complex XPath expression

`masci_tools.util.xml.common_functions.readd_inpgen_comments(xmltree, comments)`

Add the given comments after the fleurInput tag of the inp.xml

These contain at the moment the inpgen command line and the content of the inpgen file

Parameters

- **xmltree** (`_ElementTree`) – representation of the inp.xml
- **comments** (`list[_Element]`) – list of XML comments

Return type

`_ElementTree`

`masci_tools.util.xml.common_functions.reverse_xinclude(xmltree, schema_dict, included_tags, **kwargs)`

DEPRECATED ALIAS: Moved to `masci_tools.util.schema_dict_util`

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** – an xml-tree which will be processed
- **schema_dict** – Schema dictionary containing all the necessary information
- **included_tags** – Iterable of str, containing the names of the tags to be excluded

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.serialize_xml_objects(args, kwargs)`

Convert every XML element/tree in the given args/kwargs to string using `lxml.etree.tostring()`

Parameters

- **args** (`tuple[Any, ...]`) – positional arguments
- **kwargs** (`dict[str, Any]`) – keyword arguments

Return type

`tuple[tuple[Any, ...], dict[str, Any]]`

`masci_tools.util.xml.common_functions.split_off_attrib(xpath)`

Splits off attribute of the given xpath (part after @)

Note: `etree.XPath` objects could lose context in here, i.e. non-default options passed at init

Parameters

xpath (`TypeVar(TXPathLike, bound= Union[str, bytes, XPath, XPathBuilder])`) – xpath to split up

Return type

`tuple[TypeVar(TXPathLike, bound= Union[str, bytes, XPath, XPathBuilder]), str]`

`masci_tools.util.xml.common_functions.split_off_tag(xpath)`

Splits off the last part of the given xpath

Note: `etree.XPath` objects could lose context in here, i.e. non-default options passed at init

Parameters

xpath (`TypeVar(TXPathLike, bound= Union[str, bytes, XPath, XPathBuilder])`) – xpath to split up

Return type

`tuple[TypeVar(TXPathLike, bound= Union[str, bytes, XPath, XPathBuilder]), 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, allow_special_los=True)`

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 (`str` | `bool`) – a string ('t', 'f', 'F', 'T')

Return type

`bool`

Returns

boolean (either True or False)

`masci_tools.util.xml.converters.convert_from_fortran_complex(number_str)`

Converts a string of the form (float,float) to a complex number

Parameters

number_str (`str`) – string to convert

Return type

`complex`

Returns

complex number

`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 an 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, complex, list[Union[int, float, bool, str, complex]], list[Union[int, float, bool, str, complex, list[Union[int, float, bool, str, complex]]]]], 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, complex, list[Union[int, float, bool, str, complex]], list[Union[int, float, bool, str, complex, list[Union[int, float, bool, str, complex]]]]], 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[Literal['int', 'switch', 'string', 'float', 'float_expression', 'complex'], ...]`) – 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, complex]], 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 a 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 (`bool` | `str`) – either a boolean or a string ('True', 'False', 'F', 'T')

Return type

`Literal['T', '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[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[str | list[str], bool]`

Returns

The converted value of the first successful conversion

`masci_tools.util.xml.converters.convert_to_xml_single_values(value, possible_types, logger=None, float_format='.10')`

Tries to converts a given attributevalue 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[Literal['int', 'switch', 'string', 'float', 'float_expression', 'complex'], ...]`) – 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 successful conversion

This module contains Classes for building complex XPath expressions based on general attribute conditions from simple XPath expressions

`masci_tools.util.xml.xpathbuilder.FilterType`

Type for filters argument for XPathBuilder

alias of `Dict[str, Any]`

class `masci_tools.util.xml.xpathbuilder.XPathBuilder`(*simple_path, filters=None, compile_path=False, strict=False, **kwargs*)

Class for building a complex xpath (restricted to adding filters) from a simple xpath expression

Note: passing in an etree.XPath object will not respect the options passed into it. Only the kwargs in `__init__` are used to compile the path if `compile_path=True`

Note: Filters/Constraints (or predicates like they are called for XPaths) can either be added by providing the `filters` argument in the constructor or by calling the `add_filter()` method.

The `filters` argument is a dictionary with the tag names, where to apply the condition, as keys and the condition as values while the `add_filter()` method takes these as its two arguments. The tag name has to be a part of the original simple xpath expression. The condition is a dictionary with one key specifying the kind of condition and the value for the condition. The condition can also be the name of an attribute or path, in which case the value can be another condition dictionary. The following conditions operators i.e. keys in the dictionary are supported:

- `=/==`: equal to
- `!=`: not equal to
- `<`: less than
- `>`: greater than
- `<=`: less than or equal to
- `>=`: greater than or equal to
- `contains`: attribute/tag contains the given value (case sensitive)
- `not-contains`: attribute/tag does not contains the given value
- `starts-with`: attribute/tag starts with the given value (case sensitive)
- `ends-with`: attribute/tag ends with the given value (case sensitive)
- `index`: Select tags based on their index in the parent tag (either explicit index or another condition)
- `has`: Select tags based on the presence of the given attribute/tag
- `has-not`: Select tags based on the absence of the given attribute/tag
- `number-nodes`: Compute the number of nodes in the previous path and select based on further criteria
- `and`: Provide multiple conditions in a list joined by `and`
- `or`: Provide multiple conditions in a list joined by `or`
- `in`: Select tags if the value of the path is in a given list of values
- `not-in`: Select tags if the value of the path is not in a given list of values
- `<string>`: All other strings are interpreted as paths to attributes/tags specifying conditions on their value
- `<tuple of paths>`: Multiple strings are interpreted as multiple node sets, which are joined with `|`

Example:

```
from maschi_tools.util.xml.xpathbuilder import XPathBuilder

# XPath selecting all lo tags for SCL0 type LOs and Iron species
xpath = XPathBuilder('/fleurInput/atomSpecies/species/lo',
                    filters = {'species': {
                        'name': {'contains': 'Fe'},
                    },
                        'lo': {
                            'type': 'SCL0'
                        }
                    })
```

Parameters

- **simple_path** (`str` | `bytes` | `XPath`) – basic simple XPath expression to start from
- **filters** (`Optional[dict[str, Dict[str, Any]]]`) – dictionary with filters
- **compile_path** (`bool`) – bool if True the path property will be compiled as `etree.XPath`
- **strict** (`bool`) – bool if True the `__str__` conversion will raise an error

Other Kwargs will be passed on to the `etree.XPath` compilation if `compile_path=True`

add_filter(*tag, conditions*)

Add a filter to the filters dictionary

Parameters

- **tag** (`str`) – str name of the tag name to add a filter to
- **conditions** (`Union[Dict[str, Any], Any]`) – dictionary specifying the filter

Return type

`None`

append_tag(*tag*)

Append another tag to the end of the simple xpath expression

Parameters

- **tag** (`str`) – str name of the tag to append

Return type

`None`

get_predicate(*tag, condition, compound=False, path='.', process_path=False*)

Construct the predicate for the given tag and condition

Parameters

- **tag** (`str`) – str name of the tag
- **condition** (`Any`) – condition specified, either dict or single value
- **compound** (`bool`) – bool if True the enclosing condition is a compound condition, forbidding any other compound condition
- **path** (`str` | `tuple[str, ...]`) – path, to which to apply the condition
- **process_path** (`bool`) – bool if True the path will taken apart into its components and the components will be checked with XPath variables

Return type

`str`

property path: `str` | `lxml.etree.XPath`

Property for constructing the complex Xpath

process_condition(*tag, operator, content, path, process_path=False*)

Process the condition for the given tag and condition

Parameters

- **tag** (`str`) – str name of the tag
- **operator** (`str`) – operator for condition
- **content** (`Any`) – content of condition

- **path** (`str` | `tuple[str, ...]`) – path, to which to apply the condition
- **process_path** (`bool`) – bool if True the path will taken apart into its components and the components will be checked with XPath variables

Return type

`str`

strip_off_tag()

Strip off the last tag of the simple xpath expression

Return type

`str`

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, name,
                                                            number_to_add,
                                                            complex_xpath=None, filters=None,
                                                            mode='absolute', occurrences=None,
                                                            **kwargs)
```

Add/Multiply values for multiple occurrences in the XML tree

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **name** (`str`) – the attribute name to change
- **number_to_add** (`Any`) – number to add/multiply with the old attribute value
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **mode** (`Literal['abs', 'absolute', 'rel', 'relative']`) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*
- **occurrences** (`Union[int, Iterable[int], None]`) – 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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.add_number_to_attr('itmax', 10)
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.add_number_to_attr('radius', 1.05, mode='rel')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.add_number_to_attr('radius', 1.05, mode='rel', contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.add_number_to_attr('spinUp',
                    -1,
                    contains='species',
                    filters={
                        'species': {
                            'atomicNumber': {'>': 30}
                        }
                    })
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmldata with shifted attribute

```
masci_tools.util.xml.xml_setters_names.add_number_to_first_attr(xmldata, schema_dict, name,
                                                                number_to_add,
                                                                complex_xpath=None,
                                                                filters=None,
                                                                mode='absolute', **kwargs)
```

Add/Multiply value for the first occurrences of an attribute in the XML tree

Parameters

- **xmldata** (`Union[_Element, _ElementTree]`) – an xmldata that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **name** (`str`) – the attribute name to change
- **number_to_add** (`Any`) – number to add/multiply with the old attribute value
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **mode** (`Literal['abs', 'absolute', 'rel', 'relative']`) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details

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`

Usage Examples (fm refers to an instance of `FleurXMLModifier`)

Simple Usage

```
fm.add_number_to_first_attr('itmax', 10)
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.add_number_to_first_attr('radius', 1.05, mode='rel')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.add_number_to_first_attr('radius', 1.05, mode='rel', contains='species')
```

Added filters

The `filters` argument allows to be more specific

```
fm.add_number_to_first_attrb('spinUp',
                             -1,
                             contains='species',
                             filters={
                                 'species': {
                                     'atomicNumber': {'>': 30}
                                 }}))
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with shifted attribute

`masci_tools.util.xml.xml_setters_names.clone_species(xmltree, schema_dict, species_name, new_name, changes=None)`

Clone (and possibly change) a species in the XML tree

For reference of the changes dictionary look at `set_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
- **species_name** (`str`) – string, name of the specie you want to clone Has to correspond to one single species (no ‘all’/’all-<search_string>’)
- **new_name** (`str`) – new name of the cloned species
- **changes** (`Optional[dict[str, Any]]`) – a optional python dict specifying what you want to change.

Returns xmltree

xml etree of the new inp.xml

Return type

`Union[_Element, _ElementTree]`

Usage Examples (fm refers to an instance of `FleurXMLModifier`)

Simple Usage

```
fm.clone_species('Fe-1', 'Fe-clone-1')
```

Modifying the cloned species

The `changes` dictionary is passed on to `set_species` to modify the cloned species

```
fm.clone_species('Fe-1', 'Fe-clone-1', {'mtSphere' : {'radius' : 2.7}})
```

```
masci_tools.util.xml.xml_setters_names.create_tag(xmltree, schema_dict, tag, complex_xpath=None,
                                                  filters=None, create_parents=False,
                                                  occurrences=None, **kwargs)
```

Create a tag with a given name in the XML tree 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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **tag** (`QName | str | _Element`) – str of the tag to create or etree Element or string representing the XML element with the same name to insert
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **create_parents** (`bool`) – bool optional (default False), if True and the given xpath has no results the the parent tags are created recursively
- **occurrences** (`Union[int, Iterable[int], None]`) – 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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.create_tag('mtnocoparams')
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.create_tag('lo')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.create_tag('lo', contains='species')
```

Nested creation

With create_parents=True evtl. missing parent tags are also created

```
fm.create_tag('diagElements',
              create_parents=True,
              contains='species',
              not_contains='torque')
```

Creation of pre-created XML element

Passing a XML element instead of a name will insert this element at the correct position

```
fm.create_tag(etree.Element('lo', n='5', l='0', type='SCL0'),
              contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.create_tag(etree.Element('lo', n='5', l='0', type='SCL0'),
              contains='species',
              filters={
                  'species': {
                      'name': {'contains': 'Fe'}
                  }
              })
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with created tags

`masci_tools.util.xml.xml_setters_names.delete_att(xmltree, schema_dict, name, complex_xpath=None, filters=None, occurrences=None, **kwargs)`

Delete an attribute with a given name from the XML tree

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **name** (`str`) – str of the attribute to delete

- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **occurrences** (`Union[int, Iterable[int], None]`) – 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`

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.delete_att('mixparam')
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.delete_att('alpha')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.delete_att('alpha', contains='nocoParams', not_contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.delete_att('alpha',
              contains='nocoParams',
              not_contains='species',
              filters={
```

(continues on next page)

(continued from previous page)

```
'atomGroup': {  
    'species': {'not-contains': 'Fe'}  
}  
})
```

Return type`Union[_Element, _ElementTree]`**Returns**

xmlltree with deleted attributes

```
masci_tools.util.xml.xml_setters_names.delete_tag(xmlltree, schema_dict, tag_name,  
                                                  complex_xpath=None, filters=None,  
                                                  occurrences=None, **kwargs)
```

Delete a tag with a given name from the XML tree

Parameters

- **xmlltree** (`Union[_Element, _ElementTree]`) – an xmlltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **tag** – str of the tag to delete
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **occurrences** (`Union[int, Iterable[int], None]`) – 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

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.delete_tag('expertModes')
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.delete_tag('lo')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.delete_tag('lo', contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.delete_tag('lo',
              contains='species',
              filters={
                  'species': {
                      'name': {'contains': 'Fe'}
                  }
              })
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with deleted tags

`masci_tools.util.xml.xml_setters_names.replace_tag(xmltree, schema_dict, tag_name, element, complex_xpath=None, filters=None, occurrences=None, **kwargs)`

Replace a tag with a given name with a different XML element

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **tag** – str of the tag to replace
- **element** (`str | _Element`) – etree Element or string representing the XML element to replace the tag
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **occurrences** (`Union[int, Iterable[int], None]`) – 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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.replace_tag('expertmodes', etree.Element('greensFunction'))
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.replace_tag('lo', etree.Element('ldaU', l='3'))
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.replace_tag('lo', etree.Element('ldaU', l='3'), contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.replace_tag('lo',
               etree.Element('ldaU', l='3'),
               contains='species',
               filters={
                   'species': {
                       'atomicNumber': {'>': 30}
                   }
               })
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with replaced tags

`masci_tools.util.xml.xml_setters_names.set_atomgroup(xmltree, schema_dict, changes, position=None, species=None, filters=None)`

Set the attributes of a atom group, specified by species name, position, etc.

Parameters

- **xmldata** (`Union[_Element, _ElementTree]`) – xml etree of the inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **changes** (`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
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

Return type`Union[_Element, _ElementTree]`**Returns**

xml etree of the new inp.xml

changes 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:

```
'changes': {'nocoParams': {'beta': val}}
```

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, species='Fe-1')
```

Specifying the number of the atomgroup

Providing the **position** argument will modify the n-th atomgroup in the inp.xml file. (Indexing starts at 1)

```
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, position=1)
```

Modifying all species

Providing **'all'** to either **position** or **species** will modify all atomgroups

```
fm.set_atomgroup({'nocoParams': {'alpha': 1.57}}, species='all')
```

```
masci_tools.util.xml.xml_setters_names.set_atomgroup_label(xmldata, schema_dict, atom_label,
                                                            changes)
```

Set the attributes of a atom group, given by an atom label occuring in it's definition

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
- **changes** (`dict[str, Any]`) – a python dict specifying what you want to change.

Return type`Union[_Element, _ElementTree]`**Returns**

xml etree of the new inp.xml

changes 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:

```
'changes': {'nocoParams': {'beta': val}}
```

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_atomgroup_label('222', {'nocoParams': {'alpha': 1.57}})
```

Modifying all species

Providing ‘all’ as the first argument applies the changes to all species

```
fm.set_atomgroup_label('all', {'nocoParams': {'alpha': 1.57}})
```

```
masci_tools.util.xml.xml_setters_names.set_attr_value(xmltree, schema_dict, name, value,
                                                         complex_xpath=None, filters=None,
                                                         occurrences=None, create=False,
                                                         **kwargs)
```

Set the value of an attribute possibly occurring multiple times

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** (`Union[_Element, _ElementTree]`) – an xmldata that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **name** (`str`) – the attribute name to set
- **value** (`Any`) – value or list of values to set
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation

- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** (`bool`) – 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`

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.set_attr_value('itmax', 180)
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_attr_value('radius', 1.05)
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_attr_value('radius', 1.05, contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.set_attr_value('spinUp',
                  1,
                  contains='species',
                  filters={
                      'species': {
```

(continues on next page)

(continued from previous page)

```
        'atomicNumber': {'>': 30}
    })
```

Return type`Union[_Element, _ElementTree]`**Returns**

xmlltree with set attribute

```
maschi_tools.util.xml.xml_setters_names.set_complex_tag(xmlltree, schema_dict, tag_name, changes,
                                                         complex_xpath=None, filters=None,
                                                         create=False, **kwargs)
```

Setattributes, text and subtags of a given tag in the XML tree

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

- **xmlltree** (`Union[_Element, _ElementTree]`) – an xmlltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **tag_name** (`str`) – name of the tag to set
- **changes** (`dict[str, Any]`) – 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** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **create** (`bool`) – 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

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with changes to the complex tag

```
masci_tools.util.xml.xml_setters_names.set_first_attrib_value(xmltree, schema_dict, name, value,
                                                                complex_xpath=None,
                                                                filters=None, create=False,
                                                                **kwargs)
```

Set the value of the first occurrence of an attribute

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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **name** (`str`) – the attribute name to set
- **value** (`Any`) – value or list of values to set
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details
- **create** (`bool`) – 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`

Usage Examples (fm refers to an instance of `FleurXMLModifier`)

Simple Usage

```
fm.set_first_attrib_value('itmax', 180)
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_first_attr_value('radius', 1.05)
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_first_attr_value('radius', 1.05, contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.set_first_attr_value('spinUp',
                        1,
                        contains='species',
                        filters={
                            'species': {
                                'atomicNumber': {'>': 30}
                            }
                        })
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with set attribute

```
masci_tools.util.xml.xml_setters_names.set_first_text(xmltree, schema_dict, tag_name, text,
                                                       complex_xpath=None, filters=None,
                                                       create=False, **kwargs)
```

Set the value of text of the first occurrence of a tag in the XML tree

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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **tag_name** (`str`) – str name of the tag, where the text should be set
- **text** (`Any`) – value or list of values to set
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation

- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **create** (`bool`) – 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

Usage Examples (fm refers to an instance of [FleurXMLModifier](#))

Simple Usage

```
fm.set_first_text('kpoint', [1,2,3])
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_first_text('valenceConfig', '(1s1/2)')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_first_text('valenceConfig', '(1s1/2)', contains='species')
```

Added filters

The filters argument allows to be more specific

```
fm.set_first_text('valenceConfig',
                  '(1s1/2)',
                  contains='species',
                  filters={
                      'species': {
                          './lo/@n': {'>': 4}
                      }
                  })
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with set text

```
masci_tools.util.xml.xml_setters_names.set_inpchanges(xmltree, schema_dict, changes,
                                                         path_spec=None)
```

Set the value of attributes/text occurring once in the XML tree

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
- **changes** (`dict[str, Any]`) – dictionary {attrib_name : value} with all the wanted changes.
- **path_spec** (`Optional[dict[str, Any]]`) – dict, with ggf. necessary further specifications for the path of the attribute

An example of changes:

```
changes = {
    '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.

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_inpchanges({'itmax' : 1, 'kmax': 4.3})
```

Attribute selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_inpchanges({'itmax' : 1, 'theta': 1.57})
```

Attribute selection

Selection can be done by adding conditions on what the XPath should(n't) contain in the path_spec argument

```
fm.set_inpchanges({'itmax' : 1, 'theta': 1.57},
                   path_spec={'theta': {'contains': 'soc'}})
```

`masci_tools.util.xml.xml_setters_names.set_kpath(xmltree, schema_dict, kpath, count, gamma=False)`
 Sets a k-path directly as an 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.

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_kpath({'Point': [0, 0, 0], 'Point2': [1, 1, 1]}, 200)
```

`masci_tools.util.xml.xml_setters_names.set_kpath_max4(xmltree, schema_dict, kpath, count, gamma=False)`

Sets a k-path directly as an 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,
                                                       additional_attributes=None)
```

Create a k-point list with the given points and weights

Note: If no name is given, a name of the form `default-<number>` is generated

Warning: For input versions Max4 and older **all** keyword arguments are not valid (*name*, *kpoint_type*, *special_labels*, *switch* and *overwrite*)

Parameters

- **xmltree** (`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 xmltree of the inp.xml file with changes.

Usage Examples (fm refers to an instance of `FleurXMLModifier`)

Simple Usage

```
fm.set_kpointlist([[0,0,0],[0.5,0.5,0.5]],
                  [1,1], switch=True)
```

MaX 4 compatibility

For input files before the MaX 5 release the previous kpoint list will always be overwritten and the switch argument has no effect

```
fm.set_kpointlist([[0,0,0],[0.5,0.5,0.5]],
                  [1,1])
```

```
masci_tools.util.xml.xml_setters_names.set_kpointlist_max4(xmltree, schema_dict, kpoints, weights,
                                                            name=None, kpoint_type='path',
                                                            special_labels=None, switch=False,
                                                            overwrite=False,
                                                            additional_attributes=None)
```

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

- **xmltree** (`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 xmltree of the inp.xml file with changes.

```
masci_tools.util.xml.xml_setters_names.set_kpointmesh(xmltree, schema_dict, mesh, name=None,
                                                       use_symmetries=True, switch=False,
                                                       overwrite=False, shift=None,
                                                       time_reversal=True, map_to_first_bz=True)
```

Create a kpoint mesh using spglib

for details see `get_stabilized_reciprocal_mesh()`

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – xml tree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **mesh** (`Collection[int]`) – list-like with three elements, giving the size of the kpoint set in each direction
- **use_symmetry** – bool if True the available symmetry operations in the inp.xml will be used to reduce the kpoint set otherwise only the identity matrix is used
- **name** (`Optional[str]`) – Name of the created kpoint list. If not given a name is generated
- **switch** (`bool`) – bool if True the kpoint list is directly set as the used set
- **overwrite** (`bool`) – if True and a kpoint list of the given name already exists it will be overwritten

- **shift** (`Optional[Iterable[float]]`) – shift the center of the kpoint set
- **time_reversal** (`bool`) – bool if True time reversal symmetry will be used to reduce the kpoint set
- **map_to_first_bz** (`bool`) – bool if True the kpoints are mapped into the [0,1] interval

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with a created kpoint path

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_kpointmesh([4, 4, 1], switch=True)
```

```
masci_tools.util.xml.xml_setters_names.set_kpointpath(xmltree, schema_dict, path=None,
                                                         nkpts=None, density=None, name=None,
                                                         switch=False, overwrite=False,
                                                         special_points=None)
```

Create a kpoint list for a bandstructure calculation (using ASE kpath generation)

The path can be defined explicitly (see `bandpath()`) or derived from the unit cell

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** (`Union[str, list[str], None]`) – str, list of str or None defines the path to interpolate (for syntax `bandpath()`)
- **nkpts** (`Optional[int]`) – int number of kpoints in the path
- **density** (`Optional[float]`) – float number of kpoints per Angstroem
- **name** (`Optional[str]`) – Name of the created kpoint list. If not given a name is generated
- **switch** (`bool`) – bool if True the kpoint list is directly set as the used set
- **overwrite** (`bool`) – if True and a kpoint list of the given name already exists it will be overwritten
- **special_points** (`Optional[dict[str, Iterable[float]]]`) – dict mapping names to coordinates for special points to use

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with a created kpoint path

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_kpointpath(nkpts=50, switch=True)
```

Specifying custom points

By default the path is chosen by ASE. By providing path this can be modified

```
fm.set_kpointpath(path='CA',
                  nkpts=25,
                  special_points={
                      'C': [0, 0, 0],
                      'A': [0, 0, 0.5]
                  })
```

`masci_tools.util.xml.xml_setters_names.set_nkpts(xmltree, schema_dict, count, gamma=False)`

Sets a k-point mesh directly specific for inputs of version Max4

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
- **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.

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_nkpts(200)
```

Include Gamma point

```
fm.set_nkpts(200, gamma=True)
```

`masci_tools.util.xml.xml_setters_names.set_nkpts_max4(xmltree, schema_dict, count, gamma=False)`

Sets a k-point mesh directly specific for inputs of version Max4

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 xmldata of the inp.xml file with changes.

```
masci_tools.util.xml.xml_setters_names.set_simple_tag(xmldata, schema_dict, tag_name, changes,
                                                       complex_xpath=None, filters=None,
                                                       create_parents=False, **kwargs)
```

Sets one or multiple simple tag(s) (no subtags/text) in an xmldata.

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

- **xmldata** (`Union[_Element, _ElementTree]`) – an xmldata that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **tag_name** (`str`) – str name of the tag to modify/set
- **changes** (`list[dict[str, Any]] | dict[str, Any]`) – list of dicts or dict with the changes. Elements in list describe multiple tags. Keys in the dictionary correspond to {'attribute-name': attributevalue}
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details
- **create_parents** (`bool`) – 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

Usage Examples (fm refers to an instance of `FleurXMLModifier`)

Simple Usage

```
fm.set_simple_tag('soc', {'theta': 0.1, 'phi': 0.2, 'l_soc': True})
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_simple_tag('lo', [{ 'l': 0, 'n': 6, 'type': 'SCL0'},
                          { 'l': 1, 'n': 6, 'type': 'SCL0'}])
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_simple_tag('lo', [{ 'l': 0, 'n': 6, 'type': 'SCL0'},
                          { 'l': 1, 'n': 6, 'type': 'SCL0'}],
                  contains='species')
```

Nested creation

With `create_parents=True` if the parents of the tag are missing they are created

```
fm.set_simple_tag('realAxis', { 'ne': 1300, 'elbow': -1.0, 'elup': 1.0}, create_
↳parents=True)
```

Added filters

The filters argument allows to be more specific

```
fm.set_simple_tag('lo', [{ 'l': 0, 'n': 6, 'type': 'SCL0'},
                          { 'l': 1, 'n': 6, 'type': 'SCL0'}],
                  contains='species',
                  filters={
                      'species': {
                          'atomicNumber': { '>': 30}
                      }
                  })
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with set simple tags

`masci_tools.util.xml.xml_setters_names.set_species(xmltree, schema_dict, species_name, changes, filters=None, create=False)`

Set the attributes of a species, specified by name position, etc.

Parameters

- **xmldata** (`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)
- **changes** (`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?
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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 xmldata

xml etree of the new inp.xml

Return type

`Union[_Element, _ElementTree]`

changes 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

Simple Usage

```
fm.set_species('Fe-1', {'mtSphere' : {'radius' : 2.7}})
```

Adding child elements

The *changes* dictionary is not limited to setting attributes Child elements can also be created. Notice here that if multiple elements of a given name are allowed all previously existing elements are deleted and replaced with the ones specified in *changes*

```
fm.set_species('Fe-1', {'ldaU' : {'l' : 3, 'U' : 4.0, 'J' : 0.5, 'l_amf' : True},
                        'lo' : [{'l' : 0, 'n' : 6, 'type' : 'SCL0'},
                               {'l' : 1, 'n' : 6, 'type' : 'SCL0'}]})
```

Modifying all species

Providing 'all' as the first argument applies the changes to all species

```
fm.set_species('all', {'mtSphere' : {'radius' : 2.7}})
```

Modifying a subset of species

Providing 'all-<search string>' as the first argument applies the changes to all species which contain the search string in it's name

```
fm.set_species('all-Pt', {'mtSphere' : {'radius' : 2.7},
                           'lo' : {'l': 0, 'n': 6, 'type': 'SCL0'},})
```

Example species. All these attributes tags can be modified. Additional keys can for example be found in the FLEUR documentation flapw.de

```
<species name="Fe-1" element="Fe" atomicNumber="26">
  <mtSphere radius="2.200000000" gridPoints="787" logIncrement=".016000000"/>
  <atomicCutoffs lmax="10" lnonsphr="6"/>
  <electronConfig>
    <coreConfig>[Ne]</coreConfig>
    <valenceConfig>(3s1/2) (3p1/2) (3p3/2) (4s1/2) (3d3/2) (3d5/2)</valenceConfig>
    <stateOccupation state="(3d3/2)" spinUp="2.000000000" spinDown="1.000000000"/>
    <stateOccupation state="(3d5/2)" spinUp="3.000000000" spinDown=".000000000"/>
  </electronConfig>
  <energyParameters s="4" p="4" d="3" f="4"/>
  <lo type="SCL0" l="0" n="3" eDeriv="0"/>
  <lo type="SCL0" l="1" n="3" eDeriv="0"/>
</species>
```

```
masci_tools.util.xml.xml_setters_names.set_species_label(xmltree, schema_dict, atom_label,
                                                         changes, create=False)
```

Set the attributes of a species, given by an atom label occurring in it's atom type

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
- **changes** (`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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_species_label('222', {'mtSphere' : {'radius' : 2.7}})
```

Adding child elements

The *changes* dictionary is not limited to setting attributes Child elements can also be created. Notice here that if multiple elements of a given name are allowed all previously existing elements are deleted and replaced with the ones specified in *changes*

```
fm.set_species_label('222', {'ldaU' : {'l' : 3, 'U': 4.0, 'J': 0.5, 'l_amf': True},
                             'lo': [{'l': 0, 'n': 6, 'type': 'SCL0'},
                                    {'l': 1, 'n': 6, 'type': 'SCL0'}]})
```

Modifying all species

Providing *'all'* as the first argument applies the changes to all species

```
fm.set_species_label('all', {'mtSphere' : {'radius' : 2.7}})
```

```
masci_tools.util.xml.xml_setters_names.set_text(xmltree, schema_dict, tag_name, text,
                                                complex_xpath=None, filters=None,
                                                occurrences=None, create=False, **kwargs)
```

Set the value of text of tags in the XML tree possibly occurring mutliple times

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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **tag_name** (`str`) – str name of the tag, where the text should be set
- **text** (`Any`) – value or list of values to set
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** (`bool`) – 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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage


```
fm.set_text('kpoint', [[1,2,3], [4,5,6]])
```

Setting one value

If only one value for the text is given all elements are set to the same value in this case only one list of three coordinates is set

```
fm.set_text('kpoint', [4,4,4])
```

Tag selection not unique

If no or multiple locations could be possible an error is raised

```
fm.set_text('q', '1 1 1')
```

Tag selection

Selection can be done by adding conditions on what the XPath should(n't) contain

```
fm.set_text('q', '1 1 1', contains='spinSpiral')
```

Added filters

The filters argument allows to be more specific

```
fm.set_text('valenceConfig',
            '(1s1/2)',
            contains='species',
            filters={
                'species': {
                    './lo/@n': {'>': 4}
                }
            })
```

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with set text

`masci_tools.util.xml.xml_setters_names.set_xcfunctional(xmltree, schema_dict, xc_functional, xc_functional_options=None, libxc=False)`

Set the Exchange Correlation potential tag

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – XML tree that represents inp.xml

- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xc_functional** (*str* | *dict[str, int | str]*) – str or dict. If str it is the name of a inbuilt XC functional. If it is a dict it specifies either the name or id for LibXC functionals for the keys *'exchange'*, *'correlation'*, *'etot_exchange'* and *'etot_correlation'*
- **xc_functional_options** (*Optional[dict[str, Any]]*) – dict with further general changes to the *xcFunctional* tag
- **libxc** (*bool*) – bool if True the functional is a LibXC functional

Return type

Union[_Element, _ElementTree]

Returns

an xmltree with modified xcFunctional tag

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.set_xcfunctional('pbe')
```

Setting a LibXC functional

```
fm.set_xcfunctional({'exchange': 'lda_x', 'correlation': "lda_c_xalpha"}, libxc=True)
```

`masci_tools.util.xml.xml_setters_names.shift_value(xmltree, schema_dict, changes, mode='absolute', path_spec=None)`

Add/Multiply values of attributes occurring once in the XML tree

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
- **changes** (*dict[str, Any]*) – a python dictionary with the keys to shift and the shift values.
- **mode** (*Literal['abs', 'absolute', 'rel', 'relative']*) – str (either *rellrelative* or *abs/absolute*). *rellrelative* multiplies the old value with the given value *abs/absolute* adds the old value and the given value
- **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 changes:

```
changes = {'itmax' : 1, 'dVac': -0.123}
```

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.shift_value({'itmax' : 5, 'mindistance': 1})
```

Relative shift

Passing mode="rel" or mode="relative" will multiply the initial attribute values with the provided number instead of adding them

```
fm.shift_value({'itmax' : 2, 'mindistance': 3}, mode="relative")
```

```
masci_tools.util.xml.xml_setters_names.shift_value_species_label(xmltree, schema_dict,
                                                                    atom_label, attribute_name,
                                                                    number_to_add,
                                                                    mode='absolute', **kwargs)
```

Add/Multiply numeric attributes in a or multiple atom species 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
- **attribute_name** (*str*) – name of the attribute to change
- **number_to_add** (*Any*) – value to add or to multiply by
- **mode** (*Literal*['abs', 'absolute', 'rel', 'relative']) – str (either *rellrelative* or *abs/absolute*). *rellrelative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*

Kwargs if the attribute_name 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

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.shift_value_species_label('222', 'radius', 1)
```

Relative shift

Passing mode="rel" or mode="relative" will multiply the initial attribute value with the provided number instead of adding them

```
fm.shift_value_species_label('222', 'radius', 1.2, mode="relative")
```

Modifying all species

Providing 'all' as the first argument applies the changes to all atomgroups

```
fm.shift_value_species_label('all', 'lmax', 2)
```

`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)`

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_species(xmltree, schema_dict, new_species_name, position=None, species=None, filters=None, clone=False, changes=None)`

Change the species of an atom group, identified by species name, position, etc.

Parameters

- **xmltree** (Union[_Element, _ElementTree]) – xml etree of the inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **new_species_name** (*str*) – name of the species to switch to
- **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
- **clone** (bool) – if True and the new species name does not exist and it corresponds to changing from one species the species will be cloned with *clone_species()*
- **changes** (Optional[dict[str, Any]]) – changes to do if the species is cloned
- **filters** (Optional[Dict[str, Any]]) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type

Union[_Element, _ElementTree]

Returns

xml etree of the new inp.xml

Usage Examples (fm refers to an instance of *FleurXMLModifier*)

Simple Usage

```
fm.switch_species('Pt-1', species='Fe-1')
```

Specifying the number of the atomgroup

Providing the position argument will modify the n-th atomgroup in the `inp.xml` file. (Indexing starts at 1)

```
fm.switch_species('Pt-1', position=1)
```

Create a clone of the old species

Specifying a non-existent new species name and `clone=True` will create a clone of the old species with that name. Additionally the `changes` argument can then be used to apply changes to the new clone

```
fm.switch_species('Fe-clone-2', species='Fe-1', clone=True, changes={'mtSphere' : {
↪ 'radius' : 2.7}})
```

Modifying all species

Providing `'all'` to either `position` or `species`` will modify the species of all atomgroups

```
fm.switch_species('Fe-1', species='all')
```

```
masci_tools.util.xml.xml_setters_names.switch_species_label(xmltree, schema_dict, atom_label,
                                                             new_species_name, clone=False,
                                                             changes=None)
```

Change the species of an atom group, identified by the lable of an atom in it's group

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 group will be changed. `'all'` to change all the groups
- **new_species_name** (`str`) – name of the species to switch to
- **clone** (`bool`) – if True and the new species name does not exist and it corresponds to changing from one species the species will be cloned with `clone_species()`
- **changes** (`Optional[dict[str, Any]]`) – changes to do if the species is cloned

Return type

`Union[_Element, _ElementTree]`

Returns

xml etree of the new inp.xml

This module contains useful methods for initializing or modifying a `n_mmp_mat` file for LDA+U

```
class masci_tools.util.xml.xml_setters_nmmpmat.LDAUElement(species: str, orbital: int, group_index:
                                                             int)
```

Contains the important information needed to locate the associated density matrix blocks

group_index: `int`

Alias for field number 2

orbital: `int`

Alias for field number 1

species: `str`

Alias for field number 0

```
masci_tools.util.xml.xml_setters_nmmpmat.align_nmmpmat_to_sqa(xmltree, nmmpmlines, schema_dict,
                                                             species_name='all', orbital='all',
                                                             phi_before=0.0, theta_before=0.0,
                                                             filters=None)
```

Align the density matrix with the given SQA of the associated species

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **nmmpmlines** (`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 | str`) – integer or string ('all'), orbital quantum number of the LDA+U procedure to be modified
- **phi_before** (`float | list[float]`) – float or list of floats, angle (radian), values for phi for the previous alignment of the density matrix
- **theta_before** (`float | list[float]`) – float or list of floats, angle (radian), values for theta for the previous alignment of the density matrix
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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 nmmpmlines

```
masci_tools.util.xml.xml_setters_nmmpmat.rotate_nmmpmat(xmltree, nmmpmlines, schema_dict,
                                                         species_name, orbital, phi, theta,
                                                         inverse=False, filters=None)
```

Rotate the density matrix with the given angles phi and theta

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **nmmpmlines** (`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 | str`) – integer or string ('all'), 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
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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 nmmlines

```
masci_tools.util.xml.xml_setters_nmmpmat.set_nmmpmat(xmltree, nmmlines, schema_dict,
                                                    species_name, orbital, spin,
                                                    state_occupations=None,
                                                    orbital_occupations=None, denmat=None,
                                                    phi=None, theta=None, inverse=False,
                                                    align_to_sqa=False, filters=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
- **nmmlines** (`Optional[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
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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 nmmlines


```
masci_tools.util.xml.xml_setters_nmmpmat.validate_nmmpmat(xmltree, nmmp_lines, schema_dict)
```

Checks that the given nmmp_lines 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
- **nmmp_lines** (`Optional[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, number_nodes=1)
```

Evaluates and xpath and creates tag if the result is empty

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to place a new tag
- **base_xpath** (`str`) – path where to place a new tag without complex syntax ([] conditions and so on)
- **create_parents** (`bool`) – bool optional (default False), if True also the parents of the tag are created if they are missing
- **occurrences** (`Union[int, Iterable[int], None]`) – 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
- **number_nodes** (`int`) – how many identical nodes to create

Return type

`list[_Element]`

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, name,
                                                                number_to_add,
                                                                mode='absolute',
                                                                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

- **xmldata** (`Union[_Element, _ElementTree]`) – an xmldata that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the attributes
- **base_xpath** (`str`) – path where to place a new tag without complex syntax ([] conditions and so on)
- **name** (`str`) – the attribute name to change
- **number_to_add** (`Any`) – number to add/multiply with the old attribute value
- **mode** (`Literal['abs', 'absolute', 'rel', 'relative']`) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*
- **occurrences** (`Union[int, Iterable[int], None]`) – 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

Return type

`Union[_Element, _ElementTree]`

Returns

xmldata with shifted attribute

```
masci_tools.util.xml.xml_setters_xpaths.xml_add_number_to_first_attrb(xmldata, schema_dict,
                                                                        xpath, base_xpath,
                                                                        name, number_to_add,
                                                                        mode='absolute')
```

Adds a given number to the first occurrence of a attribute value in a xmldata. If there are no nodes under the specified xpath an error is raised

Parameters

- **xmldata** (`Union[_Element, _ElementTree]`) – an xmldata that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the attributes
- **base_xpath** (`str`) – path where to place a new tag without complex syntax ([] conditions and so on)
- **name** (`str`) – the attribute name to change
- **number_to_add** (`Any`) – number to add/multiply with the old attribute value
- **mode** (`Literal['abs', 'absolute', 'rel', 'relative']`) – str (either *rel/relative* or *abs/absolute*). *rel/relative* multiplies the old value with *number_to_add* *abs/absolute* adds the old value and *number_to_add*

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

Return type

`Union[_Element, _ElementTree]`

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,
                                                                    number_nodes=1,
                                                                    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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to place a new tag
- **base_xpath** (`str`) – path where to place a new tag without complex syntax ([] conditions and so on)
- **element** (`QName | str | _Element`) – a tag name or etree Element or string representing the XML element to be created
- **create_parents** (`bool`) – bool optional (default False), if True and the given xpath has no results the the parent tags are created recursively
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.
- **number_nodes** (`int`) – how many identical nodes to create

Raises

ValueError – If the nodes are missing and *create_parents=False*

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with created tags

```
masci_tools.util.xml.xml_setters_xpaths.xml_set_attrib_value(xmltree, schema_dict, xpath,
                                                             base_xpath, name, value,
                                                             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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the attributes
- **base_xpath** (`str`) – path where to place a new tag without complex syntax ([] conditions and so on)
- **name** (`str`) – the attribute name to set
- **value** (`Any`) – value or list of values to set
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** (`bool`) – 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 name is not allowed on the base_xpath

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with set attribute

`masci_tools.util.xml.xml_setters_xpaths.xml_set_complex_tag(xmltree, schema_dict, xpath, base_xpath, changes, create=False)`

Recursive function to correctly set tags/attributes for a given tag and it's subtags. Goes through the changes dictionary 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

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the attributes
- **base_xpath** (`str`) – path where to place a new tag without complex syntax ([] conditions and so on)
- **tag_name** – name of the tag to set
- **changes** (`dict[str, Any]`) – 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*) – bool optional (default False), if True and the path, where the complex tag is set does not exist it is created

Return type

`Union[_Element, _ElementTree]`

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, name, value,
                                                                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** (`Union[_Element, _ElementTree]`) – an xmldata that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the attribute
- **base_xpath** (*str*) – path where to place a new tag without complex syntax ([] conditions and so on)
- **name** (*str*) – the attribute name to set
- **value** (*Any*) – value or list of values to set
- **create** (*bool*) – 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

Return type

`Union[_Element, _ElementTree]`

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** (`Union[_Element, _ElementTree]`) – an xmldata that represents inp.xml
- **schema_dict** (`SchemaDict`) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the text

- **base_xpath** (*str*) – path where to place a new tag without complex syntax ([] conditions and so on)
- **text** (*Any*) – value or list of values to set
- **create** (*bool*) – 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*

Return type

`Union[_Element, _ElementTree]`

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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the attributes
- **base_xpath** (*str*) – path where to place a new tag without complex syntax ([] conditions and so on)
- **tag_name** (*str*) – name of the tag to set
- **changes** (`list[dict[str, Any]] | dict[str, Any]`) – list of dicts or dict with the changes. Elements in list describe multiple tags. Keys in the dictionary correspond to { 'name': value }
- **create_parents** (*bool*) – bool optional (default False), if True and the path, where the simple tags are set does not exist it is created

Return type

`Union[_Element, _ElementTree]`

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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **schema_dict** (*SchemaDict*) – InputSchemaDict containing all information about the structure of the input

- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the text
- **base_xpath** (`str`) – path where to place a new tag without complex syntax ([] conditions and so on)
- **text** (`Any`) – value or list of values to set
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the node to set. By default all are set.
- **create** (`bool`) – 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*

Return type

`Union[_Element, _ElementTree]`

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

```
masci_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)
```

Creates a tag under the given XPath 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** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to place a new tag
- **element** (`QName | str | _Element`) – a tag name, etree Element or string representing the XML element to be created
- **place_index** (`Optional[int]`) – defines the place where to put a created tag
- **tag_order** (`Optional[list[str]]`) – defines a tag order
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.
- **correct_order** (`bool`) – 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`) – 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, ...)

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with created tags

`masci_tools.util.xml.xml_setters_basic.xml_delete_att(xmltree, xpath, name, occurrences=None)`

Deletes an attribute on the tags of the given XPath in the XML tree

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path to the attribute to be deleted
- **name** (`str`) – the name of an attribute to delete
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with deleted attribute

`masci_tools.util.xml.xml_setters_basic.xml_delete_tag(xmltree, xpath, occurrences=None)`

Deletes a tag under the XPath in the XML tree.

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path to the tag to be deleted
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with deleted tag

`masci_tools.util.xml.xml_setters_basic.xml_replace_tag(xmltree, xpath, element, occurrences=None)`

Replace XML tags under the XPath by a given tag

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path to the tag to be replaced
- **element** (`str | _Element`) – an Element or string representing the Element to replace the found tags with
- **occurrences** (`Union[int, Iterable[int], None]`) – int or list of int. Which occurrence of the parent nodes to create a tag. By default all nodes are used.

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with replaced tag

`masci_tools.util.xml.xml_setters_basic.xml_set_attrib_value_no_create(xmltree, xpath, name, value, occurrences=None)`

Sets the value of an attribute under the XPath By default the attribute will be set on all nodes returned for the specified xpath.

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the attributes
- **name** (`str`) – the attribute name to set
- **value** (`Any`) – value or list of values to set (if not str they will be converted with `str(value)`)
- **occurrences** (`Union[int, Iterable[int], None]`) – 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

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with set attribute

```
masci_tools.util.xml.xml_setters_basic.xml_set_text_no_create(xmltree, xpath, text,
                                                             occurrences=None)
```

Sets the text of a tag under the XPath By default the text will be set on all nodes returned for the specified xpath.

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – an xmltree that represents inp.xml
- **xpath** (`Union[str, bytes, XPath, XPathBuilder]`) – a path where to set the text
- **text** (`Any`) – value or list of values to set (if not str they will be converted with `str(value)`)
- **occurrences** (`Union[int, Iterable[int], None]`) – 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

Return type

`Union[_Element, _ElementTree]`

Returns

xmltree with set text

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** (`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** (`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?
- *spex*: 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(*args, **kwargs)
```

RENAMED TO `get_kpointsdata`

Return type

```
tuple[list[list[float]] | dict[str, list[list[float]]], list[float] | dict[str, list[float]], ndarray, tuple[bool, bool, bool]]
```

```
masci_tools.util.xml.xml_getters.get_kpointsdata(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 arguments *name*, *index* and *only_used* have no effect

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema_dict** (`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[list[list[float]] | dict[str, list[list[float]]], 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_kpointsdata_max4(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.

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** (`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
- **name** (`Optional[str]`) – (Has no effect for Max4)
- **index** (`Optional[int]`) – (Has no effect for Max4)

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. **pbc**
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** (`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** (`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(*args, **kwargs)`

RENAMED TO `get_parameterdata`

Return type

`dict[str, Any]`

`masci_tools.util.xml.xml_getters.get_parameterdata(xmltree, schema_dict, inpgen_ready=True, write_ids=True, extract_econfig=False, allow_special_los=True, logger=None)`

This routine returns an python dictionary produced from the inp.xml file, which contains all the parameters needed to setup a new inp.xml from a 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** (`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** (`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** (`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_special_kpoints(xmltree, schema_dict, name=None, index=None, only_used=False, logger=None)`

Extract the labeled special kpoints from the given kpointlist

Warning: Only implemented for versions starting with Max5

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema_dict** (`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

Return type

`list[tuple[int, str]] | dict[str, list[tuple[int, str]]]`

Returns

list of tuples (index, label) for multiple kpoint sets a dict with the names containing the list of tuples is returned

```
masci_tools.util.xml.xml_getters.get_special_kpoints_max4(xmltree, schema_dict, name=None,
                                                         index=None, only_used=False,
                                                         logger=None)
```

Extract the labeled special kpoints from the given kpointlist

Warning: Only implemented for versions starting with Max5

Parameters

- **xmltree** (`Union[_Element, _ElementTree]`) – etree representing the fleur xml file
- **schema_dict** (`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

Return type

`list[tuple[int, str]] | dict[str, list[tuple[int, str]]]`

Returns

list of tuples (index, label) for multiple kpoint sets a dict with the names containing the list of tuples is returned

```
masci_tools.util.xml.xml_getters.get_structure_data(*args, **kwargs)
```

RENAMED TO get_structuredata

Return type

`tuple[list[AtomSiteProperties], ndarray, tuple[bool, bool, bool]]`

```
masci_tools.util.xml.xml_getters.get_structuredata(xmltree, schema_dict, include_relaxations=True,
                                                    convert_to_angstroem=True,
                                                    normalize_kind_name=True,
                                                    extract_magnetic_moments=True, logger=None,
                                                    **kwargs)
```

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** (`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
- **extract_magnetic_moments** (`bool`) – bool, if True (default) the magnetic moments are also extracted and put onto the `atom_data` output

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

Changed in version 0.7.0: The default for `site_namedtuple` is set to `True`

Changed in version 0.10.0: The argument `site_namedtuple` was deprecated. The old output is no longer supported. If the argument `site_namedtuple` is passed a deprecation warning is shown

`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** (`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

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: list[float], symbol: str, kind: str,
                                                    magnetic_moment: Literal['up', 'down'] |
                                                    float | list[float] | None = None)
```

namedtuple used for input output of atom sites

kind: `str`

Alias for field number 2

magnetic_moment: `Union[Literal['up', 'down'], float, list[float], None]`

Alias for field number 3

position: `list[float]`

Alias for field number 0

symbol: `str`

Alias for field number 1

`masci_tools.io.common_functions._TVectorType`

Generic type variable for atom position types

alias of `TypeVar('_TVectorType', bound=Union[Tuple[float, float, float], List[float], ndarray])`

`masci_tools.io.common_functions.abs_to_rel(vector, cell)`

Converts a position vector in absolute coordinates to relative coordinates.

Parameters

- **vector** (`TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])`) – list or np.array of length 3, vector to be converted
- **cell** (`list[list[float]] | ndarray`) – Bravais matrix of a crystal 3x3 Array, List of list or np.array

Return type

`TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])`

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** (`TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])`) – list or np.array of length 3, vector to be converted
- **cell** (`list[list[float]] | ndarray`) – Bravais matrix of a crystal 3x3 Array, List of list or np.array
- **pbc** (`tuple[bool, bool, bool]`) – Boundary conditions, List or Tuple of 3 Boolean

Return type

`TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])`

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

Return type

`ndarray`

`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

Return type

`str`

`masci_tools.io.common_functions.convert_to_fortran(val, quote_strings=True)`

Parameters

val (`Any`) – the value to be read and converted to a Fortran-friendly string.

Return type

`str`

`masci_tools.io.common_functions.convert_to_fortran_string(string)`

converts some parameter strings to the format for the inpgen :type string: `str` :param string: some string :rtype: `str` :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.

Return type

`Any`

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

Return type

`int`

```
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`) – dict to filter

Return type

`dict`

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** (`Union[Tuple[float, float, float], List[float], ndarray]`) – vector to rotate
- **to_pos** (`Union[Tuple[float, float, float], List[float], ndarray]`) – vector to rotate to
- **rotations** (`list[ndarray]`) – list of np.arrays with the given symmetry rotations
- **shifts** (`list[ndarray]`) – list of np.arrays with the given shifts for the symmetry operations
- **cell** (`list[list[float]] | ndarray`) – Bravais matrix of a crystal 3x3 Array, List of list or np.array
- **relative_pos** (`bool`) – bool if True the given vectors are assumed to be in internal coordinates
- **film** (`bool`) – bool if True the vectors are assumed to be film coordinates

Return type

`tuple[ndarray, ndarray]`

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

Return type

`tuple[list, list, list]`

`masci_tools.io.common_functions.get_ef_from_potfile(potfile)`

extract fermi energy from potfile

Return type

`float`

`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

Return type

`tuple[int, float, str]`

`masci_tools.io.common_functions.get_outfile_txt(outfile)`

Get the content of a file In case the outfile is a file handle, we just roll it back and read everything in again. For an ordinary file path we open the file in a context manager and then read it.

`masci_tools.io.common_functions.get_pauli_matrix(direction, alpha=0.0, beta=0.0)`

Get the pauli matrix with additional rotation applied

Parameters

- **direction** (`Literal['x', 'y', 'z']`) – str (x,y or z) for which pauli matrix to return
- **alpha** (`float`) – angle in radians
- **beta** (`float`) – angle in radians

Return type

`ndarray`

`masci_tools.io.common_functions.get_spin_rotation(alpha, beta)`

Get matrix to rotate the spin frame by the given angles alpha/beta

Parameters

- **alpha** (`float`) – angle in radians
- **beta** (`float`) – angle in radians

Return type

`ndarray`

`masci_tools.io.common_functions.get_wigner_matrix(l, alpha, beta, gamma=0.0, inverse=False)`

Produces the wigner rotation matrix for the density matrix

Parameters

- **l** (`int`) – int, orbital quantum number
- **alpha** (`float`) – float, angle (radian) corresponds to euler angle alpha
- **beta** (`float`) – float, angle (radian) corresponds to euler angle beta
- **gamma** (`float`) – float, angle (radian) corresponds to euler angle gamma

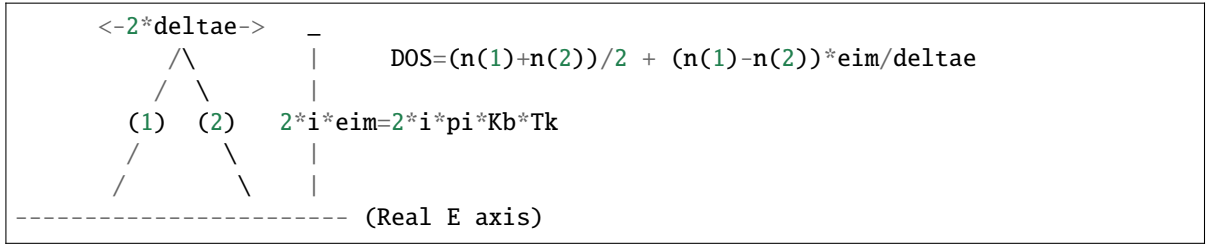
Return type

`ndarray`

`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:

```
with open_general(outfile) as f:
    txt = f.readlines()
```

Return type

`IO[Any]`

`masci_tools.io.common_functions.rel_to_abs(vector, cell)`

Converts a position vector in internal coordinates to absolute coordinates in Angstrom.

Parameters

- **vector** (`TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])`) – list or np.array of length 3, vector to be converted
- **cell** (`list[list[float]] | ndarray`) – Bravais matrix of a crystal 3x3 Array, List of list or np.array

Return type

`TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])`

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).

Return type

`TypeVar(_TVectorType, bound= Union[Tuple[float, float, float], List[float], ndarray])`

`masci_tools.io.common_functions.skipHeader(seq, n)`

Iterate over a sequence skipping the first n elements

Return type

`Generator[Any, None, None]`

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)

Return type

`tuple[ndarray, ndarray, ndarray] | tuple[float, float, float]`

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

Return type

`None`

`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

Return type

`None`

`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** (Group) – current hdf group to process
- **flatten** (`bool`) – bool, if True the dictionary will be flattened (does not check for lost information)

Return type

`tuple[dict[str, Any], dict[str, Any]]`

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`) – bool, if True the dictionary will be flattened (does not check for lost information)

Return type

`tuple[dict[str, Any], dict[str, Any]]`

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

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.

Return type

`None`

class `masci_tools.util.logging_util.OutParserLogAdapter(logger, extra=None)`

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.

Return type

`tuple[str, dict[str, Any]]`

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, iteration_path=False, filters=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, XPathElementEvaluator]`) – 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

- **logger** (`Optional[Logger]`) – logger object for logging warnings, errors, if not provided all errors will be raised
- **iteration_path** (`bool`) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

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.ensure_relaxation_xinclude(xmltree, schema_dict)`

Ensure that the xinclude tag for the relax.xml is added if no relaxation is present in the inp.xml

Parameters

- **xmltree** (`_ElementTree`) – an xml-tree which will be processed
- **schema_dict** (`InputSchemaDict`) – Schema dictionary containing all the necessary information

Return type

`None`

Returns

xmltree, which either contains the relaxation section or a xinclude tag

`masci_tools.util.schema_dict_util.eval_simple_xpath(node, schema_dict, name, logger=None, iteration_path=False, filters=None, list_return=False, **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** (`Union[_Element, _ElementTree, XPathElementEvaluator]`) – 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
- **iteration_path** (`bool`) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details
- **list_return** (`bool`) – bool, if True a list is always returned

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

`_Element` | `list[_Element]`

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, complex_xpath=None,
                                                    filters=None, iteration_path=False, **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, XPathElementEvaluator]`) – 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
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** (`bool`) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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, complex_xpath=None,
                                                    iteration_path=False, filters=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, XPathElementEvaluator]`) – 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
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** (`bool`) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details

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,
                                                           complex_xpath=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, XPathElementEvaluator]`) – 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
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation

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

param iteration_path

bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters

Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details

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,
                                              complex_xpath=None, iteration_path=False,
                                              filters=None, **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, XPathElementEvaluator]`) – 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
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** (`bool`) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details

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, complex_xpath=None,
                                                iteration_path=False, filters=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, XPathElementEvaluator]`) – 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
- **complex_xpath** (`Union[str, bytes, XPath, XPathBuilder, None]`) – an optional xpath to use instead of the simple xpath for the evaluation
- **iteration_path** (`bool`) – bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed
- **filters** (`Optional[Dict[str, Any]]`) – Dict specifying constraints to apply on the xpath. See `XPathBuilder` for details

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_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, XPathElementEvaluator]`) – 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

param iteration_path

bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters

Dict specifying constraints to apply on the xpath. See *XPathBuilder* for details

Return type

int

Returns

number of nodes for the given tag

`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, XPathElementEvaluator]*) – 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.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

- **xmldata** (`_ElementTree`) – an xml-tree which will be processed
- **schema_dict** (`InputSchemaDict`) – 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[PathLike | str, _ElementTree]]`

Returns

xmldata 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 xmldata

`masci_tools.util.schema_dict_util.tag_exists(node, schema_dict, name, logger=None, **kwargs)`

Evaluates whether the tag exists in the xmldata based on the given name and additional further specifications with the available type information

Parameters

- **node** (`Union[_Element, _ElementTree, XPathElementEvaluator]`) – 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

param iteration_path

bool if True and the SchemaDict is of an output schema an absolute path into the iteration element is constructed

param filters

Dict specifying constraints to apply on the xpath. See [XPathBuilder](#) for details

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`

Exception raised when a constant appearing in a expression is not defined

`masci_tools.util.fleur_calculate_expression.calculate_expression(expression, constants=None)`

Recursively evaluates the given expression string with the given defined constants

Parameters

- **expression** (`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 | int`

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** (`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 | int, 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[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 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_htr_to_ev(out_dict, name, converted_name=None, pop=False, add_unit=True, logger=None)`

Convert value from htr to eV

Return type

`dict[str, Any]`

`masci_tools.io.parsers.fleur.outxml_conversions.convert_ldahia_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]`

`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]`

`masci_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]`

Functions for expanding/splitting or converting electron configuration strings

`masci_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

`masci_tools.util.econfig.get_coreconfig(element, full=False)`

returns the econfiguration as a string of an element.

Parameters

- **element** (`str | int`) – element string
- **full** (`bool`) – a bool if True the econfig without [He]... is returned

Return type

`Optional[str]`

Returns

coreconfig string

`masci_tools.util.econfig.get_econfig(element, full=False)`

returns the econfiguration as a string of an element.

Parameters

- **element** (`str` | `int`) – element string
- **full** (`bool`) – a bool if True the econfig without [He]... is returned

Return type

`Optional[str]`

Returns

a econfig string

`masci_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

6.1.2.6 Basic Fleur Schema parser functions

Load all fleur schema related functions

class `masci_tools.io.parsers.fleur_schema.AttributeType`(*base_type: str, length: int | Literal['unbounded'] | None*)

Type for describing the types of attributes/text

exception `masci_tools.io.parsers.fleur_schema.IncompatibleSchemaVersions`

Exception raised when it is known that a given output version and input version cannot be compiled into a complete fleur output xml schema

class `masci_tools.io.parsers.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 <)

exception masci_tools.io.parsers.fleur_schema.NoPathFound

Exception raised when no path is found for a given tag/attribute

exception masci_tools.io.parsers.fleur_schema.NoUniquePathFound

Exception raised when no unique path is found for a given tag/attribute

class masci_tools.io.parsers.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

iteration_tags

Names of the elements that can contain all iteration tags

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 <)

iteration_attr_xpath(*name*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the attribute and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

Parameters

- **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 *iteration_tag_xpath()*
- **iteration_tag** (*str*) – name of the tag containing the iteration information

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

iteration_tag_xpath(*name*, *contains=None*, *not_contains=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the schema_dict based on the given name of the tag and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

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
- **iteration_tag** (*str*) – name of the tag containing the iteration information

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

property out_version: *tuple*[*int*, *int*]

Returns the output version as an integer for comparisons (> or <)

relative_iteration_attr_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *exclude=None*, *tag_name=None*, *iteration_tag='iteration'*)

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
- **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

- **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_iteration_tag_xpath()*
- **iteration_tag** (*str*) – name of the tag containing the iteration information

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_iteration_tag_xpath(*name*, *root_tag*, *contains=None*, *not_contains=None*, *iteration_tag='iteration'*)

Tries to find a unique path from the *schema_dict* based on the given name of the tag and additional further specifications in the iteration section of the out.xml and returns the absolute path to it

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
- **iteration_tag** (*str*) – name of the tag containing the iteration information

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

class *masci_tools.io.parsers.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
- **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
- **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
- **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*)

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

`TagInfo`

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

validate(`xmltree`, `logger=None`, `header=""`)

Validate the given XML tree against the schema

Parameters

- **xmltree** (`_ElementTree`) – XML tree to validate
- **logger** (`Optional[Logger]`) – Logger to relay evlt warnings/errors

Return type

`None`

`masci_tools.io.parsers.fleur_schema.list_available_versions(output_schema)`

List the available versions for the schema

Parameters

output_schema (`bool`) – bool, if True search for FleurOutputSchema.xsd otherwise FleurInputSchema.xsd

Return type

`list[str]`

Returns

list version string of the available versions

`masci_tools.io.parsers.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[[TypeVar(F, bound= Callable[... Any]), SchemaDictDispatch[TypeVar(F, bound= Callable[... Any])]]`

functions to extract information about the fleur schema input or output

```
class masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.AttributeType(base_type:
                                                    str,
                                                    length:
                                                    int |
                                                    Literal['unbounded']
                                                    |
                                                    None)
```

Type for describing the types of attributes/text

```
class masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.TagInfo
```

Dict representing the entries for the tag information.

```
masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.convert_str_version_number(version_str)
```

Convert the version number as a 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.io.parsers.fleur_schema.fleur_schema_parser_functions.eval_single_string_attribute(xmlschema
                                                                                               xpath,
                                                                                               **variables)
```

Wrapper around the xpath calls in this module. Makes sure the return value is a single string (not cached)

Parameters

- **xmlschema_evaluator** – etree.XPathEvaluator for the schema
- **xpath** – str, xpath expression to evaluate

```
masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.extract_attribute_types(xmlschema_evaluator,
                                                                                          **kwargs)
```

Determine the required type of all attributes

Parameters

xmlschema_evaluator (*XPathDocumentEvaluator*) – etree.XPathEvaluator for the schema

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_schema.fleur_schema_parser_functions.extract_text_types(xmlschema_evaluator,
                                                                                       **kwargs)
```

Determine the required type of all elements with text

Parameters

xmlschema_evaluator (*XPathDocumentEvaluator*) – etree.XPathEvaluator for the schema

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_schema.fleur_schema_parser_functions.get_basic_types(xmlschema_evaluator,
                                                                                     in-
                                                                                     put_basic_types=None,
                                                                                     **kwargs)
```

find all types, which can be traced back directly to a base_type

Parameters

xmlschema_evaluator (`XPathDocumentEvaluator`) – etree.XPathEvaluator for the schema

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_schema.fleur_schema_parser_functions.get_input_tag(xmlschema_evaluator,
                                                                                   **kwargs)
```

Returns the tag for the input type element of the outxmlschema

Parameters

xmlschema_evaluator (`XPathDocumentEvaluator`) – etree.XPathEvaluator for the schema

Return type

`str`

Returns

name of the element with the type 'FleurInputType'

```
masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.get_iteration_tags(xmlschema_evaluator,
                                                                                       **kwargs)
```

Returns the tags that can contain the information from a SCF iteration

Parameters

xmlschema_evaluator (`XPathDocumentEvaluator`) – etree.XPathEvaluator for the schema

Return type

`CaseInsensitiveFrozenSet[str]`

Returns

set of tag names that contain elements from the group 'GeneralIterationType'

```
masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.get_omittable_tags(xmlschema_evaluator,
                                                                                       **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_evaluator (`XPathDocumentEvaluator`) – etree.XPathEvaluator for the schema

Return type

`list[str]`

Returns

list of tags, containing only a sequence of one allowed tag

`masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.get_other_attribs(xmlschema_evaluator, **kwargs)`

Determine all other attributes not contained in `settable` or `settable_contains`

Parameters

`xmlschema_evaluator` (`XPathDocumentEvaluator`) – `etree.XPathEvaluator` for the schema

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_schema.fleur_schema_parser_functions.get_root_tag(xmlschema_evaluator, **kwargs)`

Returns the tag for the root element of the `xmlschema`

Parameters

`xmlschema_evaluator` (`XPathDocumentEvaluator`) – `etree.XPathEvaluator` for the schema

Return type

`str`

Returns

name of the single element defined in the first level of the schema

`masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.get_tag_info(xmlschema_evaluator, **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_evaluator` (`XPathDocumentEvaluator`) – `etree.XPathEvaluator` for the schema

Return type

`dict[str, TagInfo]`

Returns

dictionary with the tag information

`masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.get_tag_paths(xmlschema_evaluator, **kwargs)`

Determine simple `xpaths` to all possible tags

Parameters

`xmlschema_evaluator` (`XPathDocumentEvaluator`) – `etree.XPathEvaluator` for the schema

Return type

`CaseInsensitiveDict[str, 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_schema.fleur_schema_parser_functions.get_text_tags(xmlschema_evaluator, **kwargs)`

find all elements, who can contain text

Parameters

xmlschema_evaluator (`XPathDocumentEvaluator`) – xmltree representing the schema

Return type

`CaseInsensitiveFrozenSet[str]`

Returns

dictionary with tags and their corresponding type_definition meaning a dictionary with possible base types and evtl. length restriction

`masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.get_unique_attribs(xmlschema_evaluator, **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_evaluator (`XPathDocumentEvaluator`) – etree.XPathEvaluator for the schema

Return type

`CaseInsensitiveDict[str, str]`

Returns

dictionary with all settable attributes and the corresponding path to the tag

`masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions.get_unique_path_attribs(xmlschema_evaluator, **kwargs)`

Determine all attributes, with multiple possible path that do have at least one path with all contained tags max-Occurs!=1

Parameters

xmlschema_evaluator (`XPathDocumentEvaluator`) – etree.XPathEvaluator for the schema

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_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]`

6.1.2.7 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  # NIST https://physics.nist.gov/cgi-bin/cuu/Value?hrev
2  HTR_TO_EV = 27.211386245988 #(53)
3  RY_TO_EV = 13.605693122994 #(26)
4  BOHR_A = 0.5291772108
5  HTR_TO_KELVIN = 315_775.02480407
6  #Scipy bohr 5.29177210903e-11 m
7  #Scipy htr 27.211386245988 eV
8  # NIST BOHR 0.529177210903 #(80)
9  #https://physics.nist.gov/cgi-bin/cuu/Value?bohrrada0
10
11 #KKR constants versions. See module docstring for details.
12 _MASCI_TOOLS_USE_OLD_CONSTANTS = os.environ.get('MASCI_TOOLS_USE_OLD_CONSTANTS', None)
13 if _MASCI_TOOLS_USE_OLD_CONSTANTS and _MASCI_TOOLS_USE_OLD_CONSTANTS.lower() in ['old',
14 ↪ 'true']:
15     ANG_BOHR_KKR = 1.8897261254578281
16     RY_TO_EV_KKR = 13.605693009
17 elif _MASCI_TOOLS_USE_OLD_CONSTANTS and _MASCI_TOOLS_USE_OLD_CONSTANTS.lower() in [
18 ↪ 'interim']:
19     ANG_BOHR_KKR = 1.8897261249935897
20     RY_TO_EV_KKR = RY_TO_EV
21 else:
22     #Set the constants to the NIST values
23     ANG_BOHR_KKR = 1.8897261246257702
24     RY_TO_EV_KKR = RY_TO_EV
25
26 #Fleur
27 #htr_eV = 27.21138602
28 #bohr=0.5291772108
29 #bohrtocm=0.529177e-8
30 #pymatgen uses scipy.constants
31 #ase: Bohr 0.5291772105638411
32 #Hartree 27.211386024367243

```

(continues on next page)

(continued from previous page)

```

31 #Rydberg 13.605693012183622
32 #1/Bohr
33 #1.8897261258369282
34 #aiida-core units:
35 #bohr_to_ang = 0.52917720859
36
37 #Predefined constants in the Fleur Code (These are accepted in the inp.xml)
38 FLEUR_DEFINED_CONSTANTS = {
39     'Pi': np.pi,
40     'Deg': 2 * np.pi / 360.0,
41     'Ang': 1.8897261247728981,
42     'nm': 18.897261247728981,
43     'pm': 0.018897261247728981,
44     'Bohr': 1.0,
45     'Htr': 1.0,
46     'eV': 1.0 / HTR_TO_EV,
47     'Ry': 0.5
48 }
49
50 PERIODIC_TABLE_ELEMENTS = {
51     0: { # This is for empty spheres etc.
52         'mass': 1.000000,
53         'name': 'Unknown',
54         'symbol': 'X'
55     },
56     1: {
57         'mass': 1.00794,
58         'name': 'Hydrogen',
59         'symbol': 'H',
60         'econfig': '1s1',
61         'fleur_default_econfig': '| 1s1',
62         'lo': '',
63         'rmt': 0.65,
64         'lmax': '',
65         'jri': 981,
66         'soc': False,
67         'mag': False
68     },
69     2: {
70         'mass': 4.002602,
71         'name': 'Helium',
72         'symbol': 'He',
73         'econfig': '1s2',
74         'fleur_default_econfig': '| 1s2',
75         'lo': '',
76         'rmt': 1.2,
77         'lmax': '',
78         'jri': 981
79     },
80     3: {
81         'mass': 6.941,
82         'name': 'Lithium',

```

(continues on next page)

(continued from previous page)

```

83     'symbol': 'Li',
84     'econfig': '1s2 | 2s1',
85     'fleur_default_econfig': '1s2 | 2s1',
86     'lo': '',
87     'rmt': 2.13,
88     'lmax': '',
89     'jri': 981
90 },
91 4: {
92     'mass': 9.012182,
93     'name': 'Beryllium',
94     'symbol': 'Be',
95     'econfig': '1s2 | 2s2',
96     'fleur_default_econfig': '1s2 | 2s2',
97     'lo': '',
98     'rmt': 1.87,
99     'lmax': '',
100    'jri': 981
101 },
102 5: {
103     'mass': 10.811,
104     'name': 'Boron',
105     'symbol': 'B',
106     'econfig': '1s2 | 2s2 2p1',
107     'fleur_default_econfig': '1s2 | 2s2 2p1',
108     'lo': '',
109     'rmt': 1.4,
110     'lmax': '',
111     'jri': 981
112 },
113 6: {
114     'mass': 12.0107,
115     'name': 'Carbon',
116     'symbol': 'C',
117     'econfig': '[He] 2s2 | 2p2',
118     'fleur_default_econfig': '[He] 2s2 | 2p2',
119     'lo': '',
120     'rmt': 1.2,
121     'lmax': '',
122     'jri': 981
123 },
124 7: {
125     'mass': 14.0067,
126     'name': 'Nitrogen',
127     'symbol': 'N',
128     'econfig': '[He] 2s2 | 2p3',
129     'fleur_default_econfig': '[He] 2s2 | 2p3',
130     'lo': '',
131     'rmt': 1.0,
132     'lmax': '',
133     'jri': 981
134 },

```

(continues on next page)

(continued from previous page)

```
135 8: {
136     'mass': 15.9994,
137     'name': 'Oxygen',
138     'symbol': 'O',
139     'econfig': '[He] 2s2 | 2p4',
140     'fleur_default_econfig': '[He] 2s2 | 2p4',
141     'lo': '',
142     'rmt': 1.1,
143     'lmax': '',
144     'jri': 981
145 },
146 9: {
147     'mass': 18.9984032,
148     'name': 'Fluorine',
149     'symbol': 'F',
150     'econfig': '[He] 2s2 | 2p5',
151     'fleur_default_econfig': '[He] 2s2 | 2p5',
152     'lo': '',
153     'rmt': 1.2,
154     'lmax': '',
155     'jri': 981
156 },
157 10: {
158     'mass': 20.1797,
159     'name': 'Neon',
160     'symbol': 'Ne',
161     'econfig': '[He] 2s2 | 2p6',
162     'fleur_default_econfig': '[He] 2s2 | 2p6',
163     'lo': '',
164     'rmt': 2.1,
165     'lmax': '',
166     'jri': 981
167 },
168 11: {
169     'mass': 22.98977,
170     'name': 'Sodium',
171     'symbol': 'Na',
172     'econfig': '[He] 2s2 | 2p6 3s1',
173     'fleur_default_econfig': '[He] | 2s2 2p6 3s1',
174     'lo': '2s 2p',
175     'rmt': 2.1,
176     'lmax': '',
177     'jri': 981
178 },
179 12: {
180     'mass': 24.305,
181     'name': 'Magnesium',
182     'symbol': 'Mg',
183     'econfig': '[He] 2s2 | 2p6 3s2',
184     'fleur_default_econfig': '[He] 2s2 | 2p6 3s2',
185     'lo': '2p',
186     'rmt': 2.3,
```

(continues on next page)

(continued from previous page)

```

187     'lmax': '',
188     'jri': 981
189 },
190 13: {
191     'mass': 26.981538,
192     'name': 'Aluminium',
193     'symbol': 'Al',
194     'econfig': '[He] 2s2 2p6 | 3s2 3p1',
195     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p1',
196     'lo': '',
197     'rmt': 2.1,
198     'lmax': '',
199     'jri': 981
200 },
201 14: {
202     'mass': 28.0855,
203     'name': 'Silicon',
204     'symbol': 'Si',
205     'econfig': '[He] 2s2 2p6 | 3s2 3p2',
206     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p2',
207     'lo': '',
208     'rmt': 2.0,
209     'lmax': '',
210     'jri': 981
211 },
212 15: {
213     'mass': 30.973761,
214     'name': 'Phosphorus',
215     'symbol': 'P',
216     'econfig': '[He] 2s2 2p6 | 3s2 3p3',
217     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p3',
218     'lo': '',
219     'rmt': 1.9,
220     'lmax': '',
221     'jri': 981
222 },
223 16: {
224     'mass': 32.065,
225     'name': 'Sulfur',
226     'symbol': 'S',
227     'econfig': '[He] 2s2 2p6 | 3s2 3p4',
228     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p4',
229     'lo': '',
230     'rmt': 1.7,
231     'lmax': '',
232     'jri': 981
233 },
234 17: {
235     'mass': 35.453,
236     'name': 'Chlorine',
237     'symbol': 'Cl',
238     'econfig': '[He] 2s2 2p6 | 3s2 3p5',

```

(continues on next page)

(continued from previous page)

```

239     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p5',
240     'lo': '',
241     'rmt': 1.7,
242     'lmax': '',
243     'jri': 981
244 },
245 18: {
246     'mass': 39.948,
247     'name': 'Argon',
248     'symbol': 'Ar',
249     'econfig': '[He] 2s2 2p6 | 3s2 3p6',
250     'fleur_default_econfig': '[He] 2s2 2p6 | 3s2 3p6',
251     'lo': '',
252     'rmt': 1.8,
253     'lmax': '',
254     'jri': 981
255 },
256 19: {
257     'mass': 39.0983,
258     'name': 'Potassium',
259     'symbol': 'K',
260     'econfig': '[Ne] 3s2 | 3p6 4s1',
261     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s1',
262     'lo': '3s 3p',
263     'rmt': 2.0,
264     'lmax': '',
265     'jri': 981
266 },
267 20: {
268     'mass': 40.078,
269     'name': 'Calcium',
270     'symbol': 'Ca',
271     'econfig': '[Ne] 3s2 | 3p6 4s2',
272     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2',
273     'lo': '3s 3p',
274     'rmt': 2.3,
275     'lmax': '',
276     'jri': 981
277 },
278 21: {
279     'mass': 44.955912,
280     'name': 'Scandium',
281     'symbol': 'Sc',
282     'econfig': '[Ne] 3s2 3p6 | 4s2 3d1',
283     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d1',
284     'lo': '3s 3p',
285     'rmt': 2.2,
286     'lmax': '',
287     'jri': 981
288 },
289 22: {
290     'mass': 47.867,

```

(continues on next page)

(continued from previous page)

```
291     'name': 'Titanium',
292     'symbol': 'Ti',
293     'econfig': '[Ne] | 3s2 3p6 4s2 3d2',
294     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d2',
295     'lo': '3s 3p',
296     'rmt': 2.1,
297     'lmax': '',
298     'jri': 981
299 },
300 23: {
301     'mass': 50.9415,
302     'name': 'Vanadium',
303     'symbol': 'V',
304     'econfig': '[Ne] 3s2 3p6 | 4s2 3d3',
305     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d3',
306     'lo': '3s 3p',
307     'rmt': 1.9,
308     'lmax': '',
309     'jri': 981
310 },
311 24: {
312     'mass': 51.9961,
313     'name': 'Chromium',
314     'symbol': 'Cr',
315     'econfig': '[Ne] 3s2 3p6 | 4s1 3d5',
316     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s1 3d5',
317     'lo': '3s 3p',
318     'rmt': 1.8,
319     'lmax': '',
320     'jri': 981
321 },
322 25: {
323     'mass': 54.938045,
324     'name': 'Manganese',
325     'symbol': 'Mn',
326     'econfig': '[Ne] 3s2 3p6 | 4s2 3d5',
327     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d5',
328     'lo': '3s 3p',
329     'rmt': 2.0,
330     'lmax': '',
331     'jri': 981
332 },
333 26: {
334     'mass': 55.845,
335     'name': 'Iron',
336     'symbol': 'Fe',
337     'econfig': '[Ne] 3s2 3p6 | 4s2 3d6',
338     'fleur_default_econfig': '[Ne] | 3s2 3p6 4s2 3d6',
339     'lo': '3s 3p',
340     'rmt': 2.00,
341     'lmax': '',
342     'jri': 981
```

(continues on next page)

(continued from previous page)

```

343 },
344 27: {
345     'mass': 58.933195,
346     'name': 'Cobalt',
347     'symbol': 'Co',
348     'econfig': '[Ne] 3s2 3p6 | 4s2 3d7',
349     'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s2 3d7',
350     'lo': '3p',
351     'rmt': 1.9,
352     'lmax': '',
353     'jri': 981
354 },
355 28: {
356     'mass': 58.6934,
357     'name': 'Nickel',
358     'symbol': 'Ni',
359     'econfig': '[Ne] 3s2 3p6 | 4s2 3d8',
360     'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s2 3d8',
361     'lo': '3p',
362     'rmt': 1.9,
363     'lmax': '',
364     'jri': 981
365 },
366 29: {
367     'mass': 63.546,
368     'name': 'Copper',
369     'symbol': 'Cu',
370     'econfig': '[Ne] 3s2 3p6 | 4s1 3d10',
371     'fleur_default_econfig': '[Ne] 3s2 | 3p6 4s1 3d10',
372     'lo': '3p',
373     'rmt': 2.1,
374     'lmax': '',
375     'jri': 981
376 },
377 30: {
378     'mass': 65.38,
379     'name': 'Zinc',
380     'symbol': 'Zn',
381     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2',
382     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2',
383     'lo': '3d',
384     'rmt': 2.2,
385     'lmax': '',
386     'jri': 981
387 },
388 31: {
389     'mass': 69.723,
390     'name': 'Gallium',
391     'symbol': 'Ga',
392     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p1',
393     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p1',
394     'lo': '3d',

```

(continues on next page)

(continued from previous page)

```
395     'rmt': 2.1,
396     'lmax': '',
397     'jri': 981
398 },
399 32: {
400     'mass': 72.64,
401     'name': 'Germanium',
402     'symbol': 'Ge',
403     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p2',
404     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p2',
405     'lo': '3d',
406     'rmt': 2.1,
407     'lmax': '',
408     'jri': 981
409 },
410 33: {
411     'mass': 74.9216,
412     'name': 'Arsenic',
413     'symbol': 'As',
414     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p3',
415     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p3',
416     'lo': '3d',
417     'rmt': 2.0,
418     'lmax': '',
419     'jri': 981
420 },
421 34: {
422     'mass': 78.96,
423     'name': 'Selenium',
424     'symbol': 'Se',
425     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p4',
426     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p4',
427     'lo': '3d',
428     'rmt': 2.0,
429     'lmax': '',
430     'jri': 981
431 },
432 35: {
433     'mass': 79.904,
434     'name': 'Bromine',
435     'symbol': 'Br',
436     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p5',
437     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p5',
438     'lo': '3d',
439     'rmt': 2.1,
440     'lmax': '',
441     'jri': 981
442 },
443 36: {
444     'mass': 83.798,
445     'name': 'Krypton',
446     'symbol': 'Kr',
```

(continues on next page)

(continued from previous page)

```

447     'econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p6',
448     'fleur_default_econfig': '[Ne] 3s2 3p6 | 3d10 4s2 4p6',
449     'lo': '3d',
450     'rmt': 2.2,
451     'lmax': '',
452     'jri': 981
453 },
454 37: {
455     'mass': 85.4678,
456     'name': 'Rubidium',
457     'symbol': 'Rb',
458     'econfig': '[Ar] 3d10 4s2 | 4p6 5s1',
459     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1',
460     'lo': '4s 4p',
461     'rmt': 2.4,
462     'lmax': '',
463     'jri': 981
464 },
465 38: {
466     'mass': 87.62,
467     'name': 'Strontium',
468     'symbol': 'Sr',
469     'econfig': '[Ar] 3d10 4s2 | 4p6 5s2',
470     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2',
471     'lo': '4s 4p',
472     'rmt': 2.4,
473     'lmax': '',
474     'jri': 981
475 },
476 39: {
477     'mass': 88.90585,
478     'name': 'Yttrium',
479     'symbol': 'Y',
480     'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d1',
481     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d1',
482     'lo': '4s 4p',
483     'rmt': 2.4,
484     'lmax': '',
485     'jri': 981
486 },
487 40: {
488     'mass': 91.224,
489     'name': 'Zirconium',
490     'symbol': 'Zr',
491     'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d2',
492     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d2',
493     'lo': '4s 4p',
494     'rmt': 2.3,
495     'lmax': '',
496     'jri': 981
497 },
498 41: {

```

(continues on next page)

(continued from previous page)

```

499     'mass': 92.90638,
500     'name': 'Niobium',
501     'symbol': 'Nb',
502     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d4',
503     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d4',
504     'lo': '4s 4p',
505     'rmt': 2.1,
506     'lmax': '',
507     'jri': 981
508 },
509 42: {
510     'mass': 95.96,
511     'name': 'Molybdenum',
512     'symbol': 'Mo',
513     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d5',
514     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d5',
515     'lo': '4s 4p',
516     'rmt': 2.0,
517     'lmax': '',
518     'jri': 981
519 },
520 43: {
521     'mass': 98.0,
522     'name': 'Technetium',
523     'symbol': 'Tc',
524     'econfig': '[Ar] 4s2 3d10 4p6 | 5s2 4d5',
525     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s2 4d5',
526     'lo': '4s 4p',
527     'rmt': 2.1,
528     'lmax': '',
529     'jri': 981
530 },
531 44: {
532     'mass': 101.07,
533     'name': 'Ruthenium',
534     'symbol': 'Ru',
535     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d7',
536     'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 5s1 4d7',
537     'lo': '4p',
538     'rmt': 2.1,
539     'lmax': '',
540     'jri': 981
541 },
542 45: {
543     'mass': 102.9055,
544     'name': 'Rhodium',
545     'symbol': 'Rh',
546     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d8',
547     'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 5s1 4d8',
548     'lo': '4p',
549     'rmt': 2.1,
550     'lmax': '',

```

(continues on next page)

(continued from previous page)

```

551     'jri': 981
552 },
553 46: {
554     'mass': 106.42,
555     'name': 'Palladium',
556     'symbol': 'Pd',
557     'econfig': '[Ar] 4s2 3d10 4p6 | 4d10',
558     'fleur_default_econfig': '[Ar] 4s2 3d10 | 4p6 4d10',
559     'lo': '4p',
560     'rmt': 2.1,
561     'lmax': '',
562     'jri': 981
563 },
564 47: {
565     'mass': 107.8682,
566     'name': 'Silver',
567     'symbol': 'Ag',
568     'econfig': '[Ar] 4s2 3d10 4p6 | 5s1 4d10',
569     'fleur_default_econfig': '[Ar] 3d10 | 4s2 4p6 5s1 4d10',
570     'lo': '4s 4p',
571     'rmt': 2.3,
572     'lmax': '',
573     'jri': 981
574 },
575 48: {
576     'mass': 112.411,
577     'name': 'Cadmium',
578     'symbol': 'Cd',
579     'econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2',
580     'fleur_default_econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2',
581     'lo': '4d',
582     'rmt': 2.4,
583     'lmax': '',
584     'jri': 981
585 },
586 49: {
587     'mass': 114.818,
588     'name': 'Indium',
589     'symbol': 'In',
590     'econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2 5p1',
591     'fleur_default_econfig': '[Ar] 4s2 3d10 4p6 | 4d10 5s2 5p1',
592     'lo': '4d',
593     'rmt': 2.2,
594     'lmax': '',
595     'jri': 981
596 },
597 50: {
598     'mass': 118.71,
599     'name': 'Tin',
600     'symbol': 'Sn',
601     'econfig': '[Kr] 4d10 | 5s2 5p2',
602     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p2',

```

(continues on next page)

(continued from previous page)

```

603     'lo': '4d',
604     'rmt': 2.3,
605     'lmax': '',
606     'jri': 981
607 },
608 51: {
609     'mass': 121.76,
610     'name': 'Antimony',
611     'symbol': 'Sb',
612     'econfig': '[Kr] 4d10 | 5s2 5p3',
613     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p3',
614     'lo': '4d',
615     'rmt': 2.3,
616     'lmax': '',
617     'jri': 981
618 },
619 52: {
620     'mass': 127.6,
621     'name': 'Tellurium',
622     'symbol': 'Te',
623     'econfig': '[Kr] 4d10 | 5s2 5p4',
624     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p4',
625     'lo': '4d',
626     'rmt': 2.3,
627     'lmax': '',
628     'jri': 981
629 },
630 53: {
631     'mass': 126.90447,
632     'name': 'Iodine',
633     'symbol': 'I',
634     'econfig': '[Kr] 4d10 | 5s2 5p5',
635     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p5',
636     'lo': '4d',
637     'rmt': 2.2,
638     'lmax': '',
639     'jri': 981
640 },
641 54: {
642     'mass': 131.293,
643     'name': 'Xenon',
644     'symbol': 'Xe',
645     'econfig': '[Kr] 4d10 | 5s2 5p6',
646     'fleur_default_econfig': '[Kr] | 4d10 5s2 5p6',
647     'lo': '4d',
648     'rmt': 2.2,
649     'lmax': '',
650     'jri': 981
651 },
652 55: {
653     'mass': 132.9054519,
654     'name': 'Caesium',

```

(continues on next page)

(continued from previous page)

```

655     'symbol': 'Cs',
656     'econfig': '[Kr] 4d10 5s2 | 5p6 6s1',
657     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s1',
658     'lo': '5s 5p',
659     'rmt': 2.4,
660     'lmax': '',
661     'jri': 981
662 },
663 56: {
664     'mass': 137.327,
665     'name': 'Barium',
666     'symbol': 'Ba',
667     'econfig': '[Kr] 4d10 5s2 | 5p6 6s2',
668     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2',
669     'lo': '5s 5p',
670     'rmt': 2.4,
671     'lmax': '',
672     'jri': 981
673 },
674 57: {
675     'mass': 138.90547,
676     'name': 'Lanthanum',
677     'symbol': 'La',
678     'econfig': '[Kr] 4d10 5s2 | 5p6 6s2 5d1',
679     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 5d1',
680     'lo': '5s 5p',
681     'rmt': 2.2,
682     'lmax': '',
683     'jri': 981
684 },
685 58: {
686     'mass': 140.116,
687     'name': 'Cerium',
688     'symbol': 'Ce',
689     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f1 5d1',
690     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f1 5d1',
691     'lo': '5s 5p',
692     'rmt': 2.2,
693     'lmax': '',
694     'jri': 981
695 },
696 59: {
697     'mass': 140.90765,
698     'name': 'Praseodymium',
699     'symbol': 'Pr',
700     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f3',
701     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f3',
702     'lo': '5s 5p',
703     'rmt': 2.4,
704     'lmax': '',
705     'jri': 981
706 },

```

(continues on next page)

(continued from previous page)

```

707 60: {
708     'mass': 144.242,
709     'name': 'Neodymium',
710     'symbol': 'Nd',
711     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f4',
712     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f4',
713     'lo': '5s 5p',
714     'rmt': 2.1,
715     'lmax': '',
716     'jri': 981
717 },
718 61: {
719     'mass': 145.0,
720     'name': 'Promethium',
721     'symbol': 'Pm',
722     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f5',
723     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f5',
724     'lo': '5s 5p',
725     'rmt': 2.4,
726     'lmax': '',
727     'jri': 981
728 },
729 62: {
730     'mass': 150.36,
731     'name': 'Samarium',
732     'symbol': 'Sm',
733     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f6',
734     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f6',
735     'lo': '5s 5p',
736     'rmt': 2.1,
737     'lmax': '',
738     'jri': 981
739 },
740 63: {
741     'mass': 151.964,
742     'name': 'Europium',
743     'symbol': 'Eu',
744     'econfig': '[Kr] 4d10 | 4f7 5s2 5p6 6s2',
745     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 4f7 6s2',
746     'lo': '5s 5p',
747     'rmt': 2.4,
748     'lmax': '',
749     'jri': 981
750 },
751 64: {
752     'mass': 157.25,
753     'name': 'Gadolinium',
754     'symbol': 'Gd',
755     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f7 5d1',
756     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f7 5d1',
757     'lo': '5s 5p',
758     'rmt': 2.2,

```

(continues on next page)

(continued from previous page)

```

759     'lmax': '',
760     'jri': 981
761 },
762 65: {
763     'mass': 158.92535,
764     'name': 'Terbium',
765     'symbol': 'Tb',
766     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f9',
767     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f8 5d1',
768     'lo': '5s 5p',
769     'rmt': 2.1,
770     'lmax': '',
771     'jri': 981
772 },
773 66: {
774     'mass': 162.5,
775     'name': 'Dysprosium',
776     'symbol': 'Dy',
777     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f10',
778     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f9 5d1',
779     'lo': '5s 5p',
780     'rmt': 2.4,
781     'lmax': '',
782     'jri': 981
783 },
784 67: {
785     'mass': 164.93032,
786     'name': 'Holmium',
787     'symbol': 'Ho',
788     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f11',
789     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f10 5d1',
790     'lo': '5s 5p',
791     'rmt': 2.4,
792     'lmax': '',
793     'jri': 981
794 },
795 68: {
796     'mass': 167.259,
797     'name': 'Erbium',
798     'symbol': 'Er',
799     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f12',
800     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f11 5d1',
801     'lo': '5s 5p',
802     'rmt': 2.5,
803     'lmax': '',
804     'jri': 981
805 },
806 69: {
807     'mass': 168.93421,
808     'name': 'Thulium',
809     'symbol': 'Tm',
810     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f13',

```

(continues on next page)

(continued from previous page)

```

811     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f12 5d1',
812     'lo': '5s 5p',
813     'rmt': 2.4,
814     'lmax': '',
815     'jri': 981
816 },
817 70: {
818     'mass': 173.054,
819     'name': 'Ytterbium',
820     'symbol': 'Yb',
821     'econfig': '[Kr] 4d10 5s2 5p6 | 6s2 4f14',
822     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 6s2 4f13 5d1',
823     'lo': '5s 5p',
824     'rmt': 2.6,
825     'lmax': '',
826     'jri': 981
827 },
828 71: {
829     'mass': 174.9668,
830     'name': 'Lutetium',
831     'symbol': 'Lu',
832     'econfig': '[Kr] 4d10 | 4f14 5s2 5p6 5d1 6s2',
833     'fleur_default_econfig': '[Kr] 4d10 | 5s2 5p6 4f14 6s2 5d1',
834     'lo': '5s 5p',
835     'rmt': 2.5,
836     'lmax': '',
837     'jri': 981
838 },
839 72: {
840     'mass': 178.49,
841     'name': 'Hafnium',
842     'symbol': 'Hf',
843     'econfig': '[Kr] 4d10 | 4f14 5s2 5p6 5d2 6s2',
844     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d2',
845     'lo': '5s 5p',
846     'rmt': 2.3,
847     'lmax': '',
848     'jri': 981
849 },
850 73: {
851     'mass': 180.94788,
852     'name': 'Tantalum',
853     'symbol': 'Ta',
854     'econfig': '[Kr] 4d10 4f14 | 5s2 5p6 5d3 6s2',
855     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d3',
856     'lo': '5s 5p',
857     'rmt': 2.2,
858     'lmax': '',
859     'jri': 981
860 },
861 74: {
862     'mass': 183.84,

```

(continues on next page)

(continued from previous page)

```

863     'name': 'Tungsten',
864     'symbol': 'W',
865     'econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d4',
866     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d4',
867     'lo': '5s 5p',
868     'rmt': 2.1,
869     'lmax': '',
870     'jri': 981
871 },
872 75: {
873     'mass': 186.207,
874     'name': 'Rhenium',
875     'symbol': 'Re',
876     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d5',
877     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d5',
878     'lo': '5s 5p',
879     'rmt': 2.1,
880     'lmax': '',
881     'jri': 981
882 },
883 76: {
884     'mass': 190.23,
885     'name': 'Osmium',
886     'symbol': 'Os',
887     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d6',
888     'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d6',
889     'lo': '5p',
890     'rmt': 2.1,
891     'lmax': '',
892     'jri': 981
893 },
894 77: {
895     'mass': 192.217,
896     'name': 'Iridium',
897     'symbol': 'Ir',
898     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d7',
899     'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d7',
900     'lo': '5p',
901     'rmt': 2.1,
902     'lmax': '',
903     'jri': 981
904 },
905 78: {
906     'mass': 195.084,
907     'name': 'Platinum',
908     'symbol': 'Pt',
909     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d8',
910     'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 | 5p6 6s2 5d8',
911     'lo': '5p',
912     'rmt': 2.1,
913     'lmax': '',
914     'jri': 981

```

(continues on next page)

(continued from previous page)

```

915 },
916 79: {
917     'mass': 196.966569,
918     'name': 'Gold',
919     'symbol': 'Au',
920     'econfig': '[Kr] 4d10 4f14 5p6 | 5s2 6s2 5d9',
921     'fleur_default_econfig': '[Kr] 4d10 4f14 | 5s2 5p6 6s2 5d9',
922     'lo': '5s 5p',
923     'rmt': 2.2,
924     'lmax': '',
925     'jri': 981
926 },
927 80: {
928     'mass': 200.59,
929     'name': 'Mercury',
930     'symbol': 'Hg',
931     'econfig': '[Kr] 5s2 4d10 4f14 | 5p6 5d10 6s2',
932     'fleur_default_econfig': '[Kr] 5s2 4d10 4f14 5p6 | 5d10 6s2',
933     'lo': '5d',
934     'rmt': 2.4,
935     'lmax': '',
936     'jri': 981
937 },
938 81: {
939     'mass': 204.3833,
940     'name': 'Thallium',
941     'symbol': 'Tl',
942     'econfig': '[Xe] 4f14 | 5d10 6s2 6p1',
943     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p1',
944     'lo': '5d',
945     'rmt': 2.4,
946     'lmax': '',
947     'jri': 981
948 },
949 82: {
950     'mass': 207.2,
951     'name': 'Lead',
952     'symbol': 'Pb',
953     'econfig': '[Xe] 4f14 | 5d10 6s2 6p2',
954     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p2',
955     'lo': '5d',
956     'rmt': 2.4,
957     'lmax': '',
958     'jri': 981
959 },
960 83: {
961     'mass': 208.9804,
962     'name': 'Bismuth',
963     'symbol': 'Bi',
964     'econfig': '[Xe] 4f14 | 5d10 6s2 6p3',
965     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p3',
966     'lo': '5d',

```

(continues on next page)

(continued from previous page)

```

967     'rmt': 2.4,
968     'lmax': '',
969     'jri': 981
970 },
971 84: {
972     'mass': 209.0,
973     'name': 'Polonium',
974     'symbol': 'Po',
975     'econfig': '[Xe] 4f14 | 5d10 6s2 6p4',
976     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p4',
977     'lo': '5d',
978     'rmt': 2.2,
979     'lmax': '',
980     'jri': 981
981 },
982 85: {
983     'mass': 210.0,
984     'name': 'Astatine',
985     'symbol': 'At',
986     'econfig': '[Xe] 4f14 | 5d10 6s2 6p5',
987     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p5',
988     'lo': '5d',
989     'rmt': 2.2,
990     'lmax': '',
991     'jri': 981
992 },
993 86: {
994     'mass': 222.0,
995     'name': 'Radon',
996     'symbol': 'Rn',
997     'econfig': '[Xe] 4f14 | 5d10 6s2 6p6',
998     'fleur_default_econfig': '[Xe] 4f14 | 5d10 6s2 6p6',
999     'lo': '5d',
1000     'rmt': 2.2,
1001     'lmax': '',
1002     'jri': 981
1003 }, # TODO: after wards not right
1004 87: {
1005     'mass': 223.0,
1006     'name': 'Francium',
1007     'symbol': 'Fr',
1008     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s1',
1009     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s1',
1010     'lo': '6s 6p',
1011     'rmt': 2.2,
1012     'lmax': '',
1013     'jri': 981
1014 },
1015 88: {
1016     'mass': 226.0,
1017     'name': 'Radium',
1018     'symbol': 'Ra',

```

(continues on next page)

(continued from previous page)

```

1019     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2',
1020     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2',
1021     'lo': '6s 6p',
1022     'rmt': 2.2,
1023     'lmax': '',
1024     'jri': 981
1025 },
1026 89: {
1027     'mass': 227.0,
1028     'name': 'Actinium',
1029     'symbol': 'Ac',
1030     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 6d1',
1031     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1',
1032     'lo': '6s 6p',
1033     'rmt': 2.2,
1034     'lmax': '',
1035     'jri': 981
1036 },
1037 90: {
1038     'mass': 232.03806,
1039     'name': 'Thorium',
1040     'symbol': 'Th',
1041     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 6d1 5f1',
1042     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1 5f1',
1043     'lo': '6s 6p',
1044     'rmt': 2.2,
1045     'lmax': '',
1046     'jri': 981
1047 },
1048 91: {
1049     'mass': 231.03588,
1050     'name': 'Protactinium',
1051     'symbol': 'Pa',
1052     'econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 6d1 5f2',
1053     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 6d1 5f2',
1054     'lo': '6s 6p',
1055     'rmt': 2.2,
1056     'lmax': '',
1057     'jri': 981
1058 },
1059 92: {
1060     'mass': 238.02891,
1061     'name': 'Uranium',
1062     'symbol': 'U',
1063     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f4',
1064     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f4',
1065     'lo': '6s 6p',
1066     'rmt': 2.3,
1067     'lmax': '',
1068     'jri': 981
1069 },
1070 93: {

```

(continues on next page)

(continued from previous page)

```

1071     'mass': 237.0,
1072     'name': 'Neptunium',
1073     'symbol': 'Np',
1074     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f5',
1075     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f5',
1076     'lo': '6s 6p',
1077     'rmt': 2.1,
1078     'lmax': '',
1079     'jri': 981
1080 },
1081 94: {
1082     'mass': 244.0,
1083     'name': 'Plutonium',
1084     'symbol': 'Pu',
1085     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f6',
1086     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f6',
1087     'lo': '6s 6p',
1088     'rmt': 2.2,
1089     'lmax': '',
1090     'jri': 981
1091 },
1092 95: {
1093     'mass': 243.0,
1094     'name': 'Americium',
1095     'symbol': 'Am',
1096     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f7',
1097     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f7',
1098     'lo': '6s 6p',
1099     'rmt': 2.4,
1100     'lmax': '',
1101     'jri': 981
1102 },
1103 96: {
1104     'mass': 247.0,
1105     'name': 'Curium',
1106     'symbol': 'Cm',
1107     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f8',
1108     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f8',
1109     'lo': '6s 6p',
1110     'rmt': 2.4,
1111     'lmax': '',
1112     'jri': 981
1113 },
1114 97: {
1115     'mass': 247.0,
1116     'name': 'Berkelium',
1117     'symbol': 'Bk',
1118     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f9',
1119     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f9',
1120     'lo': '6s 6p',
1121     'rmt': 2.4,
1122     'lmax': '',

```

(continues on next page)

(continued from previous page)

```

1123     'jri': 981
1124 },
1125 98: {
1126     'mass': 251.0,
1127     'name': 'Californium',
1128     'symbol': 'Cf',
1129     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f10',
1130     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f10',
1131     'lo': '6s 6p',
1132     'rmt': 2.4,
1133     'lmax': '',
1134     'jri': 981
1135 },
1136 99: {
1137     'mass': 252.0,
1138     'name': 'Einsteinium',
1139     'symbol': 'Es',
1140     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f11',
1141     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f11',
1142     'lo': '6s 6p',
1143     'rmt': 2.4,
1144     'lmax': '',
1145     'jri': 981
1146 },
1147 100: {
1148     'mass': 257.0,
1149     'name': 'Fermium',
1150     'symbol': 'Fm',
1151     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f12',
1152     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f12',
1153     'lo': '6s 6p',
1154     'rmt': 2.4,
1155     'lmax': '',
1156     'jri': 981
1157 },
1158 101: {
1159     'mass': 258.0,
1160     'name': 'Mendelevium',
1161     'symbol': 'Md',
1162     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f13',
1163     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f13',
1164     'lo': '6s 6p',
1165     'rmt': 2.4,
1166     'lmax': '',
1167     'jri': 981
1168 },
1169 102: {
1170     'mass': 259.0,
1171     'name': 'Nobelium',
1172     'symbol': 'No',
1173     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14',
1174     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f14',

```

(continues on next page)

(continued from previous page)

```

1175     'lo': '6s 6p',
1176     'rmt': 2.4,
1177     'lmax': '',
1178     'jri': 981
1179 },
1180 103: {
1181     'mass': 262.0,
1182     'name': 'Lawrencium',
1183     'symbol': 'Lr',
1184     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d1',
1185     'fleur_default_econfig': '[Xe] 4f14 5d10 | 6s2 6p6 7s2 5f14 6d1',
1186     'lo': '6s 6p 5f',
1187     'rmt': 2.4,
1188     'lmax': '',
1189     'jri': 981
1190 },
1191 104: {
1192     'mass': 267.0,
1193     'name': 'Rutherfordium',
1194     'symbol': 'Rf',
1195     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d2',
1196     'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d2',
1197     'lo': '6p 5f',
1198     'rmt': 2.4,
1199     'lmax': '',
1200     'jri': 981
1201 },
1202 105: {
1203     'mass': 268.0,
1204     'name': 'Dubnium',
1205     'symbol': 'Db',
1206     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d3',
1207     'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 5f14 6d3',
1208     'lo': '6p 5f',
1209     'rmt': 2.4,
1210     'lmax': '',
1211     'jri': 981
1212 },
1213 106: {
1214     'mass': 271.0,
1215     'name': 'Seaborgium',
1216     'symbol': 'Sg',
1217     'econfig': '[Xe] 4f14 5d10 6s2 6p6 | 7s2 5f14 6d4',
1218     'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 | 6p6 7s2 5f14 6d4',
1219     'lo': '6p 5f',
1220     'rmt': 2.4,
1221     'lmax': '',
1222     'jri': 981
1223 },
1224 107: {
1225     'mass': 272.0,
1226     'name': 'Bohrium',

```

(continues on next page)

(continued from previous page)

```

1227     'symbol': 'Bh',
1228     'econfig': '[Rn] 7s2 5f14 | 6d5',
1229     'fleur_default_econfig': '[Xe] 4f14 5d10 6s2 6p6 5f14 | 7s2 6d5',
1230     'lo': '',
1231     'rmt': 2.4,
1232     'lmax': '',
1233     'jri': 981
1234 },
1235 108: {
1236     'mass': 270.0,
1237     'name': 'Hassium',
1238     'symbol': 'Hs',
1239     'econfig': '[Rn] 7s2 5f14 | 6d6',
1240     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d6',
1241     'lo': '',
1242     'rmt': 2.4,
1243     'lmax': '',
1244     'jri': 981
1245 },
1246 109: {
1247     'mass': 276.0,
1248     'name': 'Meitnerium',
1249     'symbol': 'Mt',
1250     'econfig': '[Rn] 7s2 5f14 | 6d7',
1251     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d7',
1252     'lo': '',
1253     'rmt': 2.4,
1254     'lmax': '',
1255     'jri': 981
1256 },
1257 110: {
1258     'mass': 281.0,
1259     'name': 'Darmstadtium',
1260     'symbol': 'Ds',
1261     'econfig': '[Rn] 7s2 5f14 | 6d8',
1262     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d8',
1263     'lo': '',
1264     'rmt': 2.4,
1265     'lmax': '',
1266     'jri': 981
1267 },
1268 111: {
1269     'mass': 280.0,
1270     'name': 'Roentgenium',
1271     'symbol': 'Rg',
1272     'econfig': '[Rn] 7s2 5f14 | 6d9',
1273     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d9',
1274     'lo': '',
1275     'rmt': 2.4,
1276     'lmax': '',
1277     'jri': 981
1278 },

```

(continues on next page)

(continued from previous page)

```

1279 112: {
1280     'mass': 285.0,
1281     'name': 'Copernicium',
1282     'symbol': 'Cn',
1283     'econfig': '[Rn] 7s2 5f14 | 6d10',
1284     'fleur_default_econfig': '[Rn] 5f14 | 7s2 6d10',
1285     'lo': '6d',
1286     'rmt': 2.4,
1287     'lmax': '',
1288     'jri': 981
1289 },
1290 113: {
1291     'mass': 285.0,
1292     'name': 'Nihomium',
1293     'symbol': 'Nh',
1294     'econfig': '[Rn] 7s2 5f14 | 6d10 7p1',
1295     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p1',
1296     'lo': '6d',
1297     'rmt': 2.4,
1298     'lmax': '',
1299     'jri': 981
1300 },
1301 114: {
1302     'mass': 289.0,
1303     'name': 'Flerovium',
1304     'symbol': 'Fl',
1305     'econfig': '[Rn] 7s2 5f14 | 6d10 7p2',
1306     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p2',
1307     'lo': '6d',
1308     'rmt': 2.4,
1309     'lmax': '',
1310     'jri': 981
1311 },
1312 115: {
1313     'mass': 0.0,
1314     'name': 'Mascovium',
1315     'symbol': 'Mc',
1316     'econfig': '[Rn] 7s2 5f14 | 6d10 7p3',
1317     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p3',
1318     'lo': '6d',
1319     'rmt': 2.4,
1320     'lmax': '',
1321     'jri': 981
1322 },
1323 116: {
1324     'mass': 293.0,
1325     'name': 'Livermorium',
1326     'symbol': 'Lv',
1327     'econfig': '[Rn] 7s2 5f14 | 6d10 7p4',
1328     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p4',
1329     'lo': '6d',
1330     'rmt': 2.4,

```

(continues on next page)

(continued from previous page)

```

1331     'lmax': '',
1332     'jri': 981
1333 },
1334 117: {
1335     'mass': 0.0,
1336     'name': 'Tennessine',
1337     'symbol': 'Ts',
1338     'econfig': '[Rn] 7s2 5f14 | 6d10 7p5',
1339     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p5',
1340     'lo': '6d',
1341     'rmt': 2.4,
1342     'lmax': '',
1343     'jri': 981
1344 },
1345 118: {
1346     'mass': 0.0,
1347     'name': 'Oganesson',
1348     'symbol': 'Og',
1349     'econfig': '[Rn] 7s2 5f14 | 6d10 7p6',
1350     'fleur_default_econfig': '[Rn] 7s2 5f14 | 6d10 7p6',
1351     'lo': '6d',
1352     'rmt': 2.4,
1353     'lmax': '',
1354     'jri': 981
1355 }
1356 }
1357
1358 ATOMIC_NUMBERS = {data['symbol']: num for num, data in PERIODIC_TABLE_ELEMENTS.items()}

```

6.1.2.8 Extensions to Sphinx

Sphinx extension for maschi-tools

This adds the ability for giving usage examples atm specific to the FleurXMLModifier. A template can be given, which is filled in with the options and content from *usage-example* directives in the docstrings of the specified methods

Example of the usage-example directive without a connected template.

This is how it's done

Simply do the stuff

```
print('Hello World')
```

`maschi_tools.util.sphinxext.setup(app)`

This function sets up the Sphinx extension. It is called, when `maschi_tools.util.sphinxext` is added to the extensions in a `conf.py` file

Return type

None

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

m

`masci_tools.io.common_functions`, 387
`masci_tools.io.fleur_inpgen`, 229
`masci_tools.io.fleur_xml`, 273
`masci_tools.io.fleurxmlmodifier`, 230
`masci_tools.io.hdf5_util`, 392
`masci_tools.io.io_nmmpmat`, 282
`masci_tools.io.kkr_params`, 223
`masci_tools.io.kkr_read_shapefun_info`, 224
`masci_tools.io.parsers.fleur`, 227
`masci_tools.io.parsers.fleur.default_parse_tasks`, 294
`masci_tools.io.parsers.fleur.outxml_conversions`, 403
`masci_tools.io.parsers.fleur.task_migrations`, 307
`masci_tools.io.parsers.fleur_schema`, 405
`masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions`, 413
`masci_tools.io.parsers.fleur_schema.schema_dict`, 264
`masci_tools.io.parsers.hdf5.reader`, 284
`masci_tools.io.parsers.hdf5.recipes`, 285
`masci_tools.io.parsers.hdf5.transforms`, 287
`masci_tools.io.parsers.kkrimp_parser_functions`, 226
`masci_tools.io.parsers.kkrparser_functions`, 225
`masci_tools.io.parsers.voroparser_functions`, 226
`masci_tools.tools.cf_calculation`, 210
`masci_tools.tools.greensf_calculations`, 220
`masci_tools.tools.greensfunction`, 214
`masci_tools.util.case_insensitive_dict`, 322
`masci_tools.util.constants`, 418
`masci_tools.util.econfig`, 404
`masci_tools.util.fleur_calculate_expression`, 401
`masci_tools.util.lockable_containers`, 320
`masci_tools.util.logging_util`, 393
`masci_tools.util.schema_dict_util`, 393
`masci_tools.util.sphinxext`, 444
`masci_tools.util.typing`, 323
`masci_tools.util.xml`, 324
`masci_tools.util.xml.common_functions`, 324
`masci_tools.util.xml.converters`, 328
`masci_tools.util.xml.xml_getters`, 379
`masci_tools.util.xml.xml_setters_basic`, 377
`masci_tools.util.xml.xml_setters_names`, 335
`masci_tools.util.xml.xml_setters_nmmpmat`, 368
`masci_tools.util.xml.xml_setters_xpaths`, 371
`masci_tools.util.xml.xpathbuilder`, 332
`masci_tools.vis.bokeh_plots`, 201
`masci_tools.vis.bokeh_plotter`, 198
`masci_tools.vis.common`, 175
`masci_tools.vis.data`, 168
`masci_tools.vis.fleur`, 159
`masci_tools.vis.helpers`, 174
`masci_tools.vis.kkr_plot_bandstruc_qdos`, 161
`masci_tools.vis.kkr_plot_dos`, 161
`masci_tools.vis.kkr_plot_FS_qdos`, 161
`masci_tools.vis.kkr_plot_shapefun`, 162
`masci_tools.vis.matplotlib_plotter`, 178
`masci_tools.vis.parameters`, 163
`masci_tools.vis.plot_methods`, 183

Symbols

- `_EvalContext` (class in `masci_tools.io.fleur_xml`), 274
- `_TVectorType` (in module `masci_tools.io.common_functions`), 387
- `--api-key`
 - `masci_tools-fleur-schema-add` command line option, 309
 - `masci_tools-fleur-schema-pull` command line option, 310
- `--atoms`
 - `masci_tools-plot-fleur-dos` command line option, 319
- `--backend`
 - `masci_tools-plot-fleur-bands` command line option, 319
 - `masci_tools-plot-fleur-dos` command line option, 319
- `--branch`
 - `masci_tools-fleur-schema-add` command line option, 309
- `--contains`
 - `masci_tools-parse-all-attrs` command line option, 313
 - `masci_tools-parse-attr` command line option, 313
 - `masci_tools-parse-number-nodes` command line option, 315
 - `masci_tools-parse-parent-attrs` command line option, 316
 - `masci_tools-parse-tag-exists` command line option, 318
 - `masci_tools-parse-text` command line option, 318
- `--converter`
 - `masci_tools-convert-inpgen` command line option, 308
- `--from-git`
 - `masci_tools-fleur-schema-add` command line option, 309
- `--ignore-validation`
 - `masci_tools-parse-out-file` command line option, 316
- `--interstitial`
 - `masci_tools-plot-fleur-dos` command line option, 319
- `--l_resolved`
 - `masci_tools-plot-fleur-dos` command line option, 319
- `--name`
 - `masci_tools-parse-all-attrs` command line option, 313
 - `masci_tools-parse-attr` command line option, 313
 - `masci_tools-parse-number-nodes` command line option, 315
 - `masci_tools-parse-parent-attrs` command line option, 316
 - `masci_tools-parse-tag-exists` command line option, 318
 - `masci_tools-parse-text` command line option, 318
- `--no-show`
 - `masci_tools-inpxml-generate-conversion` command line option, 312
- `--not-contains`
 - `masci_tools-parse-all-attrs` command line option, 313
 - `masci_tools-parse-attr` command line option, 313
 - `masci_tools-parse-number-nodes` command line option, 315
 - `masci_tools-parse-parent-attrs` command line option, 316
 - `masci_tools-parse-tag-exists` command line option, 318
 - `masci_tools-parse-text` command line option, 318
- `--output-file`
 - `masci_tools-inpxml-convert` command line option, 311
- `--overwrite`
 - `masci_tools-fleur-schema-add` command line option, 309
 - `masci_tools-inpxml-convert` command line

option, 311

--recipe
 masci_tools-plot-fleur-bands command
 line option, 319
 masci_tools-plot-fleur-dos command line
 option, 319

--save
 masci_tools-plot-fleur-bands command
 line option, 319
 masci_tools-plot-fleur-dos command line
 option, 319

--show
 masci_tools-inpxml-generate-conversion
 command line option, 312
 masci_tools-plot-fleur-bands command
 line option, 319
 masci_tools-plot-fleur-dos command line
 option, 319

--subtags
 masci_tools-parse-all-attrs command
 line option, 313

--tag
 masci_tools-parse-attr command line
 option, 313

--test-xml-file
 masci_tools-fleur-schema-add command
 line option, 309
 masci_tools-fleur-schema-pull command
 line option, 310

--text
 masci_tools-parse-all-attrs command
 line option, 313

--total
 masci_tools-plot-fleur-dos command line
 option, 319

--version
 masci_tools command line option, 308

--weight
 masci_tools-plot-fleur-bands command
 line option, 319

-c
 masci_tools-convert-inpgen command line
 option, 308
 masci_tools-parse-all-attrs command
 line option, 313
 masci_tools-parse-attr command line
 option, 313
 masci_tools-parse-number-nodes command
 line option, 315
 masci_tools-parse-parent-attrs
 command line option, 316
 masci_tools-parse-tag-exists command
 line option, 318
 masci_tools-parse-text command line

option, 318

-n
 masci_tools-parse-all-attrs command
 line option, 313
 masci_tools-parse-attr command line
 option, 313
 masci_tools-parse-number-nodes command
 line option, 315
 masci_tools-parse-parent-attrs
 command line option, 316
 masci_tools-parse-tag-exists command
 line option, 318
 masci_tools-parse-text command line
 option, 318

-nc
 masci_tools-parse-all-attrs command
 line option, 313
 masci_tools-parse-attr command line
 option, 313
 masci_tools-parse-number-nodes command
 line option, 315
 masci_tools-parse-parent-attrs
 command line option, 316
 masci_tools-parse-tag-exists command
 line option, 318
 masci_tools-parse-text command line
 option, 318

-o
 masci_tools-inpxml-convert command line
 option, 311

-r
 masci_tools-plot-fleur-bands command
 line option, 319
 masci_tools-plot-fleur-dos command line
 option, 319

-t
 masci_tools-parse-attr command line
 option, 313

-v
 masci_tools command line option, 308

-w
 masci_tools-plot-fleur-bands command
 line option, 319

A

abs_to_rel() (in module
 masci_tools.io.common_functions), 387

abs_to_rel_f() (in module
 masci_tools.io.common_functions), 388

abs_to_rel_xpath() (in module
 masci_tools.util.xml.common_functions),
 324

add_data_key() (*masci_tools.vis.data.PlotData*
 method), 169

[add_filter\(\)](#) (*masci_tools.util.xml.xpathbuilder.XPathBuilder* method), 334
[add_number_to_attrib\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 335
[add_number_to_attrib\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 231
[add_number_to_first_attrib\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 336
[add_number_to_first_attrib\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 232
[add_parameter\(\)](#) (*masci_tools.vis.parameters.Plotter* method), 164
[add_partial_sums\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 287
[add_partial_sums_fixed\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 287
[add_tag\(\)](#) (in module *masci_tools.util.xml.common_functions*), 324
[add_task_list\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 233
[add_tooltips\(\)](#) (*masci_tools.vis.bokeh_plotter.BokehPlotter* method), 200
[align_nmmpmat_to_sqa\(\)](#) (in module *masci_tools.util.xml.xml_setters_nmmpmat*), 369
[align_nmmpmat_to_sqa\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 234
[all_attributes\(\)](#) (*masci_tools.io.fleur_xml._EvalContext* method), 274
[angles_to_vec\(\)](#) (in module *masci_tools.io.common_functions*), 388
[append\(\)](#) (*masci_tools.util.lockable_containers.LockableList* method), 321
[append_tag\(\)](#) (*masci_tools.util.xml.xpathbuilder.XPathBuilder* method), 334
[apply\(\)](#) (*masci_tools.vis.data.PlotData* method), 169
[apply_lambda\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 288
[apply_modifications\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* class method), 234
[args](#) (*masci_tools.io.fleurxmlmodifier.ModifierTask* attribute), 264
[args](#) (*masci_tools.io.parsers.hdf5.reader.AttribTransformation* attribute), 284
[args](#) (*masci_tools.io.parsers.hdf5.reader.Transformation* attribute), 285
[asymmetric_lorentz\(\)](#) (in module *masci_tools.vis.plot_methods*), 183
[asymmetric_lorentz_gauss_conv\(\)](#) (in module *masci_tools.vis.plot_methods*), 183
[asymmetric_lorentz_gauss_sum\(\)](#) (in module *masci_tools.vis.plot_methods*), 183
[AtomDictProperties](#) (class in *masci_tools.io.fleur_inpgen*), 229
[atomDiff](#) (*masci_tools.tools.greensfunction.GreensfElement* attribute), 217
[AtomSiteProperties](#) (class in *masci_tools.io.common_functions*), 387
[atomType](#) (*masci_tools.tools.greensfunction.GreensfElement* attribute), 217
[atomTypep](#) (*masci_tools.tools.greensfunction.GreensfElement* attribute), 217
[attrib_exists\(\)](#) (in module *masci_tools.util.schema_dict_util*), 393
[attrib_name](#) (*masci_tools.io.parsers.hdf5.reader.AttribTransformation* attribute), 284
[attrib_xpath\(\)](#) (*masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDict* method), 270
[attrib_xpath\(\)](#) (*masci_tools.io.parsers.fleur_schema.SchemaDict* method), 410
[AttribTransformation](#) (class in *masci_tools.io.parsers.hdf5.reader*), 284
[attribute\(\)](#) (*masci_tools.io.fleur_xml._EvalContext* method), 275
[attribute_exists\(\)](#) (*masci_tools.io.fleur_xml._EvalContext* method), 275
[attributes\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 288
[AttributeType](#) (class in *masci_tools.io.parsers.fleur_schema*), 405
[AttributeType](#) (class in *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_function*), 413

B

[BANDDOS_FILE](#)
[masci_tools-plot-fleur-bands](#) command line option, 319
[masci_tools-plot-fleur-dos](#) command line option, 320
[bands\(\)](#) (in module *masci_tools.vis.common*), 175
[bands_recipe_format\(\)](#) (in module *masci_tools.io.parsers.hdf5.recipes*), 286
[barchart\(\)](#) (in module *masci_tools.vis.plot_methods*), 183
[bokeh_bands\(\)](#) (in module *masci_tools.vis.bokeh_plots*), 201

- bokeh_dos() (in module *masci_tools.vis.bokeh_plots*), 202
- bokeh_line() (in module *masci_tools.vis.bokeh_plots*), 202
- bokeh_multi_scatter() (in module *masci_tools.vis.bokeh_plots*), 203
- bokeh_scatter() (in module *masci_tools.vis.bokeh_plots*), 203
- bokeh_spectral_function() (in module *masci_tools.vis.bokeh_plots*), 203
- bokeh_spinpol_bands() (in module *masci_tools.vis.bokeh_plots*), 204
- bokeh_spinpol_dos() (in module *masci_tools.vis.bokeh_plots*), 205
- BokehPlotter (class in *masci_tools.vis.bokeh_plotter*), 198
- BRANCH
- masci_tools-fleur-schema-pull* command line option, 310
- ## C
- calculate_expression() (in module *masci_tools.util.fleur_calculate_expression*), 402
- calculate_expression_partial() (in module *masci_tools.util.fleur_calculate_expression*), 402
- calculate_heisenberg_j0() (in module *masci_tools.tools.greensf_calculations*), 220
- calculate_heisenberg_jij() (in module *masci_tools.tools.greensf_calculations*), 221
- calculate_heisenberg_tensor() (in module *masci_tools.tools.greensf_calculations*), 221
- calculate_hybridization() (in module *masci_tools.tools.greensf_calculations*), 222
- calculate_norm() (in module *masci_tools.io.parsers.hdf5.transforms*), 288
- calculate_total_magnetic_moment() (in module *masci_tools.io.parsers.fleur.outxml_conversions*), 403
- calculate_walltime() (in module *masci_tools.io.parsers.fleur.outxml_conversions*), 403
- camel_to_snake() (in module *masci_tools.io.common_functions*), 388
- CaseInsensitiveDict (class in *masci_tools.util.case_insensitive_dict*), 322
- CaseInsensitiveFrozenSet (class in *masci_tools.util.case_insensitive_dict*), 322
- CDF_voigt_profile() (in module *masci_tools.vis.plot_methods*), 183
- CFCalculation (class in *masci_tools.tools.cf_calculation*), 210
- CFCoefficient (class in *masci_tools.tools.cf_calculation*), 213
- change_XC_val_kkrimp() (in module *masci_tools.io.kkr_params.kkrparams* method), 223
- change_zoom() (in module *masci_tools.vis.kkr_plot_shapefun*), 162
- changes() (in module *masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 234
- check_complex_xpath() (in module *masci_tools.util.xml.common_functions*), 325
- check_error_category() (in module *masci_tools.io.parsers.kkrparser_functions*), 225
- check_voronoi_output() (in module *masci_tools.io.parsers.voroparser_functions*), 226
- clear() (in module *masci_tools.util.lockable_containers.LockableList* method), 321
- clear_cache() (in module *masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDict* class method), 270
- clear_cache() (in module *masci_tools.io.parsers.fleur_schema.SchemaDict* class method), 411
- clear_xml() (in module *masci_tools.util.xml.common_functions*), 325
- clone_species() (in module *masci_tools.util.xml.xml_setters_names*), 338
- clone_species() (in module *masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 235
- colormesh_plot() (in module *masci_tools.vis.plot_methods*), 184
- colors (class in *masci_tools.tools.greensfunction*), 218
- ColumnDataSourceWrapper (class in *masci_tools.vis.data*), 168
- construct_corelevel_spectrum() (in module *masci_tools.vis.plot_methods*), 184
- contains_tag() (in module *masci_tools.util.xml.common_functions*), 325
- contour (in module *masci_tools.tools.greensfunction.GreensfElement* attribute), 217
- convention (in module *masci_tools.tools.cf_calculation.CFCoefficient* attribute), 213
- conversion_function() (in module *masci_tools.io.parsers.fleur*), 227
- convert_fleur_config_to_econfig() (in module *masci_tools.util.econfig*), 404
- convert_fleur_electronconfig() (in module *masci_tools.util.xml.converters*), 328
- convert_fleur_lo() (in module *masci_tools.util.xml.converters*), 329

`convert_forces()` (in module `masci_tools.io.parsers.fleur.outxml_conversions`), 403
`convert_from_fortran_bool()` (in module `masci_tools.util.xml.converters`), 329
`convert_from_fortran_complex()` (in module `masci_tools.util.xml.converters`), 329
`convert_from_xml()` (in module `masci_tools.util.xml.converters`), 329
`convert_from_xml_explicit()` (in module `masci_tools.util.xml.converters`), 330
`convert_from_xml_single_values()` (in module `masci_tools.util.xml.converters`), 330
`convert_htr_to_ev()` (in module `masci_tools.io.parsers.fleur.outxml_conversions`), 403
`convert_ldahia_definitions()` (in module `masci_tools.io.parsers.fleur.outxml_conversions`), 404
`convert_ldau_definitions()` (in module `masci_tools.io.parsers.fleur.outxml_conversions`), 404
`convert_relax_info()` (in module `masci_tools.io.parsers.fleur.outxml_conversions`), 404
`convert_str_version_number()` (in module `masci_tools.io.parsers.fleur.schema.fleur_schema_parser_fleur_definitions`), 414
`convert_str_version_number()` (in module `masci_tools.util.xml.converters`), 331
`convert_to_complete_list()` (`masci_tools.vis.parameters.Plotter` static method), 164
`convert_to_complex_array()` (in module `masci_tools.io.parsers.hdf5.transforms`), 288
`convert_to_fortran()` (in module `masci_tools.io.common_functions`), 388
`convert_to_fortran_bool()` (in module `masci_tools.util.xml.converters`), 331
`convert_to_fortran_string()` (in module `masci_tools.io.common_functions`), 388
`convert_to_pystd()` (in module `masci_tools.io.common_functions`), 388
`convert_to_str()` (in module `masci_tools.io.parsers.hdf5.transforms`), 288
`convert_to_xml()` (in module `masci_tools.util.xml.converters`), 331
`convert_to_xml_explicit()` (in module `masci_tools.util.xml.converters`), 331
`convert_to_xml_single_values()` (in module `masci_tools.util.xml.converters`), 332
`copy_data()` (`masci_tools.vis.data.PlotData` method), 169
`create_tag()` (in module `masci_tools.util.xml.xml_setters_names`), 339
`create_tag()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 235
`cumulative_sum()` (in module `masci_tools.io.parsers.hdf5.transforms`), 289

D

`data_keys` (`masci_tools.vis.data.PlotData` property), 169
`decompose_jij_tensor()` (in module `masci_tools.tools.greensf_calculations`), 222
`default()` (`masci_tools.vis.common.PlotBackend` static method), 175
`default_histogram()` (in module `masci_tools.vis.plot_methods`), 185
`delete_att()` (in module `masci_tools.util.xml.xml_setters_names`), 340
`delete_att()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 237
`delete_tag()` (in module `masci_tools.util.xml.xml_setters_names`), 340
`delete_tag()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 238
`denNorm` (`masci_tools.tools.cf_calculation.CFCalculation` property), 210
`dict_of_lists_to_list_of_dicts()` (`masci_tools.vis.parameters.Plotter` static method), 164
`DictHandler` (class in `masci_tools.util.logging_util`), 393
`difference()` (`masci_tools.util.case_insensitive_dict.CaseInsensitiveFroz` method), 322
`dispersionplot()` (in module `masci_tools.vis.kkr_plot_bandstruc_qdos`), 161
`distinct_datasets()` (`masci_tools.vis.data.PlotData` method), 169
`doniach_sunjic()` (in module `masci_tools.vis.plot_methods`), 185
`dos()` (in module `masci_tools.vis.common`), 175
`dos_recipe_format()` (in module `masci_tools.io.parsers.hdf5.recipes`), 286
`dosplot()` (in module `masci_tools.vis.kkr_plot_dos`), 161
`draw_lines()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotter` method), 181
`draw_straight_lines()` (`masci_tools.vis.bokeh_plotter.BokehPlotter`

method), 200

E

`emit()` (*masci_tools.util.logging_util.DictHandler method*), 393

`energy_dependence()` (*masci_tools.tools.greensfunction.GreensFunction method*), 214

`energy_dependence_full_matrix()` (*masci_tools.tools.greensfunction.GreensFunction method*), 215

`ensure_plotter_consistency()` (*in module masci_tools.vis.parameters*), 168

`ensure_relaxation_xinclude()` (*in module masci_tools.util.schema_dict_util*), 394

`eval_simple_xpath()` (*in module masci_tools.util.schema_dict_util*), 394

`eval_single_string_attribute()` (*in module masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 414

`eval_xpath()` (*in module masci_tools.util.xml.common_functions*), 325

`eval_xpath_create()` (*in module masci_tools.util.xml.xml_setters_xpaths*), 371

`evaluate_attribute()` (*in module masci_tools.util.schema_dict_util*), 395

`evaluate_bracket()` (*in module masci_tools.util.fleur_calculate_expression*), 402

`evaluate_parent_tag()` (*in module masci_tools.util.schema_dict_util*), 396

`evaluate_single_value_tag()` (*in module masci_tools.util.schema_dict_util*), 397

`evaluate_tag()` (*in module masci_tools.util.schema_dict_util*), 398

`evaluate_text()` (*in module masci_tools.util.schema_dict_util*), 399

`exclude_points()` (*in module masci_tools.vis.helpers*), 174

`expand_parameters()` (*masci_tools.vis.parameters.Plotter method*), 165

`extend()` (*masci_tools.util.lockable_containers.LockableList method*), 321

`extract_attribute_types()` (*in module masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 414

`extract_text_types()` (*in module masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 414

F

`F` (*in module masci_tools.io.parsers.fleur_schema.schema_dict*), 264

`F` (*in module masci_tools.io.parsers.hdf5.transforms*), 287

`F` (*in module masci_tools.vis.parameters*), 163

`fac()` (*in module masci_tools.io.common_functions*), 389

`FileLike` (*in module masci_tools.util.typing*), 323

`fill_keywords_to_inputfile()` (*masci_tools.io.kkr_params.kkrparams method*), 223

`filter_out_empty_dict_entries()` (*in module masci_tools.io.common_functions*), 389

`FilterType` (*in module masci_tools.util.xml.xpathbuilder*), 332

`find()` (*masci_tools.io.fleur_xml._EvalContext method*), 276

`find_symmetry_relation()` (*in module masci_tools.io.common_functions*), 389

`flatten_array()` (*in module masci_tools.io.parsers.hdf5.transforms*), 289

`FleurXMLContext()` (*in module masci_tools.io.fleur_xml*), 273

`FleurXMLModifier` (*class in masci_tools.io.fleurxmlmodifier*), 230

`format_nmmpmat()` (*in module masci_tools.io.io_nmmpmat*), 282

`freeze()` (*masci_tools.util.lockable_containers.LockableDict method*), 320

`freeze()` (*masci_tools.util.lockable_containers.LockableList method*), 321

`from_arrays()` (*masci_tools.tools.cf_calculation.CFCalculation class method*), 210

`from_str()` (*masci_tools.vis.common.PlotBackend static method*), 175

FROM_VERSION

`masci_tools-inpxml-generate-conversion`
command line option, 312

`masci_tools-inpxml-show-conversion`
command line option, 312

`fromFile()` (*masci_tools.tools.greensfunction.GreensFunction class method*), 215

`fromList()` (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier class method*), 239

`fromPath()` (*masci_tools.io.parsers.fleur_schema.InputSchemaDict class method*), 406

`fromPath()` (*in module masci_tools.io.parsers.fleur_schema.OutputSchemaDict class method*), 408

`fromPath()` (*masci_tools.io.parsers.fleur_schema.schema_dict.InputSchemaDict class method*), 265

`fromPath()` (*masci_tools.io.parsers.fleur_schema.schema_dict.OutputSchemaDict class method*), 267

`fromVersion()` (*masci_tools.io.parsers.fleur_schema.InputSchemaDict*

class method), 406
fromVersion() (masci_tools.io.parsers.fleur_schema.OutputSchemaDict
class method), 408
fromVersion() (masci_tools.io.parsers.fleur_schema.schema_dict.InputSchemaDict
class method), 265
fromVersion() (masci_tools.io.parsers.fleur_schema.schema_dict.OutputSchemaDict
class method), 267
FSqdos2D() (in module
masci_tools.vis.kkr_plot_FS_qdos), 161

G

gauss_one() (in module masci_tools.vis.plot_methods),
185
gaussian() (in module masci_tools.vis.plot_methods),
185
get_all_child_datasets() (in module
masci_tools.io.parsers.hdf5.transforms),
289
get_all_mandatory()
(masci_tools.io.kkr_params.kkrparams
method), 223
get_attribute() (in module
masci_tools.io.parsers.hdf5.transforms),
289
get_avail_actions()
(masci_tools.io.fleurxmlmodifier.FleurXMLModifier
method), 239
get_basic_types() (in module
masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions),
415
get_bokeh_help() (in module
masci_tools.vis.bokeh_plots), 205
get_cell() (in module
masci_tools.util.xml.xml_getters), 379
get_charge_density()
(masci_tools.tools.cf_calculation.CFCalculation
method), 210
get_coefficient() (masci_tools.tools.greensfunction.GreensFunction
method), 215
get_coefficients() (masci_tools.tools.cf_calculation.CFCalculation
method), 211
get_constants() (in module masci_tools.io.fleur_xml),
281
get_coreconfig() (in module
masci_tools.util.econfig), 404
get_corestates_from_potential() (in module
masci_tools.io.common_functions), 389
get_description() (masci_tools.io.kkr_params.kkrparams
method), 223
get_description() (masci_tools.vis.parameters.Plotter
method), 165
get_dict() (masci_tools.io.kkr_params.kkrparams
method), 223
get_dict() (masci_tools.vis.parameters.Plotter
method), 165
get_econfig() (in module masci_tools.util.econfig),
404
get_ef_from_potfile() (in module
masci_tools.io.parsers.hdf5.transforms),
289
get_first_element() (in module
masci_tools.io.parsers.hdf5.transforms),
289
get_first_number() (in module
masci_tools.util.fleur_calculate_expression),
402
get_first_string() (in module
masci_tools.util.fleur_calculate_expression),
403
get_fleur_bands_specific_weights() (in module
masci_tools.io.parsers.hdf5.recipes), 286
get_fleur_modes() (in module
masci_tools.util.xml.xml_getters), 380
get_function_result()
(masci_tools.vis.data.PlotData method),
170
get_help() (in module masci_tools.vis.common), 176
get_highest_core_state() (in module
masci_tools.io.common_functions), 390
get_inpgen_comments() (in module
masci_tools.util.xml.common_functions),
326
get_inpwrt_tag() (in module
masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions),
415
get_iteration_tags() (in module
masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions),
415
get_keys() (masci_tools.vis.data.PlotData method),
170
get_KKRcalc_parameter_defaults()
(masci_tools.io.kkr_params.kkrparams class
method), 223
get_kmeshinfo() (in module
masci_tools.io.parsers.kkrparser_functions),
225
get_kpoints_data() (in module
masci_tools.util.xml.xml_getters), 380
get_kpointsdata() (in module
masci_tools.util.xml.xml_getters), 381
get_kpointsdata_max4() (in module
masci_tools.util.xml.xml_getters), 381
get_lattice_vectors() (in module
masci_tools.io.parsers.kkrparser_functions),
225
get_mask() (masci_tools.vis.data.PlotData method),
170
get_missing_keys() (masci_tools.io.kkr_params.kkrparams

- method*), 223
- `get_mpl_help()` (in module *masci_tools.vis.plot_methods*), 185
- `get_multiple_kwargs()` (*masci_tools.vis.parameters.Plotter* *method*), 165
- `get_name()` (in module *masci_tools.io.parsers.hdf5.transforms*), 289
- `get_natom()` (in module *masci_tools.io.parsers.kkrparser_functions*), 225
- `get_nkpts()` (in module *masci_tools.util.xml.xml_getters*), 382
- `get_nkpts_max4()` (in module *masci_tools.util.xml.xml_getters*), 383
- `get_noco_rms()` (in module *masci_tools.io.parsers.kkrparser_functions*), 225
- `get_nspin()` (in module *masci_tools.io.parsers.kkrparser_functions*), 225
- `get_number_of_nodes()` (in module *masci_tools.util.schema_dict_util*), 399
- `get_omittable_tags()` (in module *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 415
- `get_orbmom()` (in module *masci_tools.io.parsers.kkrparser_functions*), 225
- `get_other_attribs()` (in module *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 415
- `get_outfile_txt()` (in module *masci_tools.io.common_functions*), 390
- `get_parameter_data()` (in module *masci_tools.util.xml.xml_getters*), 383
- `get_parameterdata()` (in module *masci_tools.util.xml.xml_getters*), 383
- `get_pauli_matrix()` (in module *masci_tools.io.common_functions*), 390
- `get_plotter()` (in module *masci_tools.vis.common*), 176
- `get_potentials()` (*masci_tools.tools.cf_calculation.CFCalculation* *method*), 211
- `get_predicate()` (*masci_tools.util.xml.xpathbuilder.XPathBuilder* *method*), 334
- `get_relaxation_information()` (in module *masci_tools.util.xml.xml_getters*), 383
- `get_relaxation_information_pre029()` (in module *masci_tools.util.xml.xml_getters*), 384
- `get_rms()` (in module *masci_tools.io.parsers.kkrparser_functions*), 225
- `get_root_tag()` (in module *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 416
- `get_set_values()` (*masci_tools.io.kkr_params.kkrparams* *method*), 224
- `get_shape()` (in module *masci_tools.io.parsers.hdf5.transforms*), 290
- `get_single_particle_energies()` (in module *masci_tools.io.parsers.kkrparser_functions*), 225
- `get_special_kpoint_ticks()` (in module *masci_tools.vis.helpers*), 174
- `get_special_kpoints()` (in module *masci_tools.util.xml.xml_getters*), 384
- `get_special_kpoints_max4()` (in module *masci_tools.util.xml.xml_getters*), 385
- `get_spin_rotation()` (in module *masci_tools.io.common_functions*), 390
- `get_spinmom_per_atom()` (in module *masci_tools.io.parsers.kkrparser_functions*), 225
- `get_structure_data()` (in module *masci_tools.util.xml.xml_getters*), 385
- `get_structuredata()` (in module *masci_tools.util.xml.xml_getters*), 385
- `get_symmetry_information()` (in module *masci_tools.util.xml.xml_getters*), 386
- `get_tag_info()` (in module *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 416
- `get_tag_paths()` (in module *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 416
- `get_text_tags()` (in module *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 416
- `get_type()` (*masci_tools.io.kkr_params.kkrparams* *method*), 224
- `get_unique_attribs()` (in module *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 417
- `get_unique_path_attribs()` (in module *masci_tools.io.parsers.fleur_schema.fleur_schema_parser_functions*), 417
- `get_unlocked()` (*masci_tools.util.lockable_containers.LockableDict* *method*), 320
- `get_unlocked()` (*masci_tools.util.lockable_containers.LockableList* *method*), 321
- `get_valence_min()` (in module *masci_tools.io.parsers.voroparser_functions*), 226
- `get_value()` (*masci_tools.io.kkr_params.kkrparams* *method*), 224

[get_values\(\)](#) (*masci_tools.vis.data.PlotData* method), 170
[get_wigner_matrix\(\)](#) (in module *masci_tools.io.common_functions*), 390
[get_xml_attribute\(\)](#) (in module *masci_tools.util.xml.common_functions*), 326
[GreensfElement](#) (class in *masci_tools.tools.greensfunction*), 217
[GreensFunction](#) (class in *masci_tools.tools.greensfunction*), 214
[group_data\(\)](#) (*masci_tools.vis.data.PlotData* method), 170
[group_index](#) (*masci_tools.util.xml.xml_setters_nmmpmat* attribute), 368

H

[h5dump\(\)](#) (in module *masci_tools.io.hdf5_util*), 392
[hdf5_transformation\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 290
[HDF5LimitedTransformation](#) (class in *masci_tools.io.parsers.hdf5.reader*), 284
[HDF5Reader](#) (class in *masci_tools.io.parsers.hdf5.reader*), 284
[HDF5Recipe](#) (class in *masci_tools.io.parsers.hdf5.reader*), 285
[HDF5Transformation](#) (class in *masci_tools.io.parsers.hdf5.reader*), 285
[HDF5TransformationError](#), 287
[hdfList\(\)](#) (in module *masci_tools.io.hdf5_util*), 392
[heisenberg_reciprocal\(\)](#) (in module *masci_tools.tools.greensf_calculations*), 222
[histogram\(\)](#) (in module *masci_tools.vis.plot_methods*), 185
[hyp2f2\(\)](#) (in module *masci_tools.vis.plot_methods*), 186

I

[IncompatibleSchemaVersions](#), 264, 405
[index_dataset\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 290
[inp_version](#) (*masci_tools.io.parsers.fleur_schema.InputSchemaDict* property), 406
[inp_version](#) (*masci_tools.io.parsers.fleur_schema.OutputSchemaDict* property), 408
[inp_version](#) (*masci_tools.io.parsers.fleur_schema.schema_dict.InputSchemaDict* property), 265
[inp_version](#) (*masci_tools.io.parsers.fleur_schema.schema_dict.OutputSchemaDict* property), 267
[INPUT_FILE](#)
[masci_tools-convert-inp](#) gen command line option, 309
[InputSchemaDict](#) (class in *masci_tools.io.parsers.fleur_schema*), 405
[InputSchemaDict](#) (class in *masci_tools.io.parsers.fleur_schema.schema_dict*), 264
[inpxml_parser\(\)](#) (in module *masci_tools.io.parsers.fleur*), 227
[insert\(\)](#) (*masci_tools.util.lockable_containers.LockableList* method), 321
[interpolate\(\)](#) (*masci_tools.tools.cf_calculation.CFCalculation* method), 211
[interpolate_dos\(\)](#) (in module *masci_tools.io.common_functions*), 390
[Interpolation\(\)](#) (*masci_tools.util.case_insensitive_dict.CaseInsensitiveDict* method), 323
[intersite_shell_indices\(\)](#) (in module *masci_tools.tools.greensfunction*), 218
[intersite_shells\(\)](#) (in module *masci_tools.tools.greensfunction*), 218
[intersite_shells_from_file\(\)](#) (in module *masci_tools.tools.greensfunction*), 218
[is_general\(\)](#) (*masci_tools.vis.parameters.Plotter* method), 165
[is_mandatory\(\)](#) (*masci_tools.io.kkr_params.kkrparams* method), 224
[is_sequence\(\)](#) (in module *masci_tools.io.common_functions*), 391
[is_valid_tag\(\)](#) (in module *masci_tools.util.xml.common_functions*), 326
[isdisjoint\(\)](#) (*masci_tools.util.case_insensitive_dict.CaseInsensitiveFrozenDict* method), 323
[issubset\(\)](#) (*masci_tools.util.case_insensitive_dict.CaseInsensitiveFrozenDict* method), 323
[issuperset\(\)](#) (*masci_tools.util.case_insensitive_dict.CaseInsensitiveFrozenDict* method), 323
[items\(\)](#) (*masci_tools.io.kkr_params.kkrparams* method), 224
[items\(\)](#) (*masci_tools.vis.data.PlotData* method), 170
[iter\(\)](#) (*masci_tools.io.fleur_xml._EvalContext* method), 276
[iteration_attrib_xpath\(\)](#) (*masci_tools.io.parsers.fleur_schema.OutputSchemaDict* method), 408
[iteration_attrib_xpath\(\)](#) (*masci_tools.io.parsers.fleur_schema.schema_dict.OutputSchemaDict* method), 267
[iteration_tag_xpath\(\)](#) (*masci_tools.io.parsers.fleur_schema.OutputSchemaDict* method), 409
[iteration_tag_xpath\(\)](#) (*masci_tools.io.parsers.fleur_schema.schema_dict.OutputSchemaDict* method), 268

K

`keys()` (*masci_tools.vis.data.PlotData* method), 171
`kind` (*masci_tools.io.common_functions.AtomSiteProperties* attribute), 387
`Kinds` (class in *masci_tools.io.fleur_inpgen*), 229
`KkrimpParserFunctions` (class in *masci_tools.io.parsers.kkrimp_parser_functions*), 226
`kkrrparams` (class in *masci_tools.io.kkr_params*), 223
`kresolved` (*masci_tools.tools.greensfunction.GreensfElement* attribute), 217
`kwargs` (*masci_tools.io.fleurxmlmodifier.ModifierTask* attribute), 264
`kwargs` (*masci_tools.io.parsers.hdf5.reader.AttribTransformation* attribute), 284
`kwargs` (*masci_tools.io.parsers.hdf5.reader.Transformation* attribute), 285

L

`l` (*masci_tools.tools.cf_calculation.CFCoefficient* attribute), 213
`l` (*masci_tools.tools.greensfunction.GreensfElement* attribute), 217
`LDAUElement` (class in *masci_tools.util.xml.xml_setters_nmmpmat*), 368
`line()` (in module *masci_tools.vis.common*), 176
`list_available_versions()` (in module *masci_tools.io.parsers.fleur_schema*), 413
`list_available_versions()` (in module *masci_tools.io.parsers.fleur_schema.schema_dict*), 272
`listElements()` (in module *masci_tools.tools.greensfunction*), 219
`load_bokeh_defaults()` (in module *masci_tools.vis.bokeh_plots*), 206
`load_data()` (in module *masci_tools.vis.kkr_plot_bandstruc_qdos*), 161
`load_defaults()` (in module *masci_tools.vis.common*), 176
`load_defaults()` (*masci_tools.vis.parameters.Plotter* method), 166
`load_inpxml()` (in module *masci_tools.io.fleur_xml*), 281
`load_mpl_defaults()` (in module *masci_tools.vis.plot_methods*), 186
`load_outxml()` (in module *masci_tools.io.fleur_xml*), 281
`load_outxml_and_check_for_broken_xml()` (in module *masci_tools.io.fleur_xml*), 281
`LockableDict` (class in *masci_tools.util.lockable_containers*), 320

`LockableList` (class in *masci_tools.util.lockable_containers*), 320
`LockContainer()` (in module *masci_tools.util.lockable_containers*), 320
`locked` (*masci_tools.util.lockable_containers.LockableDict* property), 320
`locked` (*masci_tools.util.lockable_containers.LockableList* property), 321
`lorentzian()` (in module *masci_tools.vis.plot_methods*), 186
`lorentzian_one()` (in module *masci_tools.vis.plot_methods*), 186
`lp` (*masci_tools.tools.greensfunction.GreensfElement* attribute), 217

M

`m` (*masci_tools.tools.cf_calculation.CFCoefficient* attribute), 213
`magnetic_moment` (*masci_tools.io.common_functions.AtomSiteProperties* attribute), 387
`masci_tools` command line option
 `--version`, 308
 `-v`, 308
`masci_tools.io.common_functions` module, 387
`masci_tools.io.fleur_inpgen` module, 229
`masci_tools.io.fleur_xml` module, 273
`masci_tools.io.fleurxmlmodifier` module, 230
`masci_tools.io.hdf5_util` module, 392
`masci_tools.io.io_nmmpmat` module, 282
`masci_tools.io.kkr_params` module, 223
`masci_tools.io.kkr_read_shapefun_info` module, 224
`masci_tools.io.parsers.fleur` module, 227
`masci_tools.io.parsers.fleur.default_parse_tasks` module, 294
`masci_tools.io.parsers.fleur.outxml_conversions` module, 403
`masci_tools.io.parsers.fleur.task_migrations` module, 307
`masci_tools.io.parsers.fleur_schema` module, 405
`masci_tools.io.parsers.fleur_schema.fleur_schema_parser_fu` module, 413
`masci_tools.io.parsers.fleur_schema.schema_dict` module, 264
`masci_tools.io.parsers.hdf5.reader`

- module, 284
- masci_tools.io.parsers.hdf5.recipes
 - module, 285
- masci_tools.io.parsers.hdf5.transforms
 - module, 287
- masci_tools.io.parsers.kkrimp_parser_functions
 - module, 226
- masci_tools.io.parsers.kkrparser_functions
 - module, 225
- masci_tools.io.parsers.voroparser_functions
 - module, 226
- masci_tools.tools.cf_calculation
 - module, 210
- masci_tools.tools.greensf_calculations
 - module, 220
- masci_tools.tools.greensfunction
 - module, 214
- masci_tools.util.case_insensitive_dict
 - module, 322
- masci_tools.util.constants
 - module, 418
- masci_tools.util.econfig
 - module, 404
- masci_tools.util.fleur_calculate_expression
 - module, 401
- masci_tools.util.lockable_containers
 - module, 320
- masci_tools.util.logging_util
 - module, 393
- masci_tools.util.schema_dict_util
 - module, 393
- masci_tools.util.sphinxext
 - module, 444
- masci_tools.util.typing
 - module, 323
- masci_tools.util.xml
 - module, 324
- masci_tools.util.xml.common_functions
 - module, 324
- masci_tools.util.xml.converters
 - module, 328
- masci_tools.util.xml.xml_getters
 - module, 379
- masci_tools.util.xml.xml_setters_basic
 - module, 377
- masci_tools.util.xml.xml_setters_names
 - module, 335
- masci_tools.util.xml.xml_setters_nmmpmat
 - module, 368
- masci_tools.util.xml.xml_setters_xpaths
 - module, 371
- masci_tools.util.xml.xpathbuilder
 - module, 332
- masci_tools.vis.bokeh_plots
 - module, 201
- masci_tools.vis.bokeh_plotter
 - module, 198
- masci_tools.vis.common
 - module, 175
- masci_tools.vis.data
 - module, 168
- masci_tools.vis.fleur
 - module, 159
- masci_tools.vis.helpers
 - module, 174
- masci_tools.vis.kkr_plot_bandstruc_qdos
 - module, 161
- masci_tools.vis.kkr_plot_dos
 - module, 161
- masci_tools.vis.kkr_plot_FS_qdos
 - module, 161
- masci_tools.vis.kkr_plot_shapefun
 - module, 162
- masci_tools.vis.matplotlib_plotter
 - module, 178
- masci_tools.vis.parameters
 - module, 163
- masci_tools.vis.plot_methods
 - module, 183
- masci_tools-convert-inpgen command line
 - option
 - converter, 308
 - c, 308
 - INPUT_FILE, 309
 - OUTPUT_FILE, 309
- masci_tools-fleur-schema-add command line
 - option
 - api-key, 309
 - branch, 309
 - from-git, 309
 - overwrite, 309
 - test-xml-file, 309
 - SCHEMA_FILE, 309
- masci_tools-fleur-schema-pull command line
 - option
 - api-key, 310
 - test-xml-file, 310
 - BRANCH, 310
- masci_tools-fleur-schema-validate-input
 - command line option
 - XML_FILE, 310
- masci_tools-fleur-schema-validate-output
 - command line option
 - XML_FILE, 311
- masci_tools-inpxml-convert command line
 - option
 - output-file, 311
 - overwrite, 311

```

-o, 311
TO_VERSION, 311
XML_FILE, 311
masci_tools-inpxml-generate-conversion
    command line option
    --no-show, 312
    --show, 312
    FROM_VERSION, 312
    TO_VERSION, 312
masci_tools-inpxml-show-conversion command
    line option
    FROM_VERSION, 312
    TO_VERSION, 312
masci_tools-parse-all-attrs command line
    option
    --contains, 313
    --name, 313
    --not-contains, 313
    --subtags, 313
    --text, 313
    -c, 313
    -n, 313
    -nc, 313
    XML_FILE, 313
masci_tools-parse-attr command line
    option
    --contains, 313
    --name, 313
    --not-contains, 313
    --tag, 313
    -c, 313
    -n, 313
    -nc, 313
    -t, 313
    XML_FILE, 313
masci_tools-parse-cell command line option
    XML_FILE, 314
masci_tools-parse-constants command line
    option
    XML_FILE, 314
masci_tools-parse-fleur-modes command line
    option
    XML_FILE, 314
masci_tools-parse-inp-file command line
    option
    XML_FILE, 314
masci_tools-parse-kpoints command line
    option
    XML_FILE, 315
masci_tools-parse-nkpts command line option
    XML_FILE, 315
masci_tools-parse-number-nodes command line
    option
    --contains, 315
    --name, 315
    --not-contains, 315
    -c, 315
    -n, 315
    -nc, 315
    XML_FILE, 315
masci_tools-parse-out-file command line
    option
    --ignore-validation, 316
    XML_FILE, 316
masci_tools-parse-parameters command line
    option
    XML_FILE, 316
masci_tools-parse-parent-attrs command
    line option
    --contains, 316
    --name, 316
    --not-contains, 316
    -c, 316
    -n, 316
    -nc, 316
    XML_FILE, 317
masci_tools-parse-relaxation command line
    option
    XML_FILE, 317
masci_tools-parse-structure command line
    option
    XML_FILE, 317
masci_tools-parse-symmetry command line
    option
    XML_FILE, 317
masci_tools-parse-tag-exists command line
    option
    --contains, 318
    --name, 318
    --not-contains, 318
    -c, 318
    -n, 318
    -nc, 318
    XML_FILE, 318
masci_tools-parse-text command line option
    --contains, 318
    --name, 318
    --not-contains, 318
    -c, 318
    -n, 318
    -nc, 318
    XML_FILE, 318
masci_tools-plot-fleur-bands command line
    option
    --backend, 319
    --recipe, 319
    --save, 319
    --show, 319

```

```

--weight, 319
-r, 319
-w, 319
BANDDOS_FILE, 319
masci_tools-plot-fleur-dos command line
    option
--atoms, 319
--backend, 319
--interstitial, 319
--l_resolved, 319
--recipe, 319
--save, 319
--show, 319
--total, 319
-r, 319
BANDDOS_FILE, 320
mask_data() (masci_tools.vis.data.PlotData method),
    171
MatplotlibPlotter (class in
    masci_tools.vis.matplotlib_plotter), 178
matrix_plot() (in module
    masci_tools.vis.bokeh_plots), 206
max() (masci_tools.vis.data.PlotData method), 171
merge_subgroup_datasets() (in module
    masci_tools.io.parsers.hdf5.transforms),
    290
migrate_033_to_031() (in module
    masci_tools.io.parsers.fleur.task_migrations),
    308
migrate_034_to_033() (in module
    masci_tools.io.parsers.fleur.task_migrations),
    308
min() (masci_tools.vis.data.PlotData method), 171
MissingConstant, 401
ModifierTask (class in
    masci_tools.io.fleurxmlmodifier), 264
modify_xmlfile() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier
    method), 240
module
    masci_tools.io.common_functions, 387
    masci_tools.io.fleur_inpgen, 229
    masci_tools.io.fleur_xml, 273
    masci_tools.io.fleurxmlmodifier, 230
    masci_tools.io.hdf5_util, 392
    masci_tools.io.io_nmmpmat, 282
    masci_tools.io.kkr_params, 223
    masci_tools.io.kkr_read_shapefun_info,
        224
    masci_tools.io.parsers.fleur, 227
    masci_tools.io.parsers.fleur.default_parse_tasks,
        294
    masci_tools.io.parsers.fleur.outxml_conversions,
        403
    masci_tools.io.parsers.fleur.task_migrations, 307
    masci_tools.io.parsers.fleur_schema, 405
    masci_tools.io.parsers.fleur_schema.fleur_schema_parser,
        413
    masci_tools.io.parsers.fleur_schema.schema_dict,
        264
    masci_tools.io.parsers.hdf5.reader, 284
    masci_tools.io.parsers.hdf5.recipes, 285
    masci_tools.io.parsers.hdf5.transforms,
        287
    masci_tools.io.parsers.kkrimp_parser_functions,
        226
    masci_tools.io.parsers.kkrparser_functions,
        225
    masci_tools.io.parsers.voroparser_functions,
        226
    masci_tools.tools.cf_calculation, 210
    masci_tools.tools.greensf_calculations,
        220
    masci_tools.tools.greensfunction, 214
    masci_tools.util.case_insensitive_dict,
        322
    masci_tools.util.constants, 418
    masci_tools.util.econfig, 404
    masci_tools.util.fleur_calculate_expression,
        401
    masci_tools.util.lockable_containers, 320
    masci_tools.util.logging_util, 393
    masci_tools.util.schema_dict_util, 393
    masci_tools.util.sphinxext, 444
    masci_tools.util.typing, 323
    masci_tools.util.xml, 324
    masci_tools.util.xml.common_functions,
        324
    masci_tools.util.xml.converters, 328
    masci_tools.util.xml.xml_getters, 379
    masci_tools.util.xml.xml_setters_basic,
        377
    masci_tools.util.xml.xml_setters_names,
        335
    masci_tools.util.xml.xml_setters_nmmpmat,
        368
    masci_tools.util.xml.xml_setters_xpaths,
        371
    masci_tools.util.xml.xpathbuilder, 332
    masci_tools.vis.bokeh_plots, 201
    masci_tools.vis.bokeh_plotter, 198
    masci_tools.vis.common, 175
    masci_tools.vis.data, 168
    masci_tools.vis.fleur, 159
    masci_tools.vis.helpers, 174
    masci_tools.vis.kkr_plot_bandstruc_qdos,
        161
    masci_tools.vis.kkr_plot_dos, 161

```

[maschi_tools.vis.kkr_plot_FS_qdos](#), 161
[maschi_tools.vis.kkr_plot_shapefun](#), 162
[maschi_tools.vis.matplotlib_plotter](#), 178
[maschi_tools.vis.parameters](#), 163
[maschi_tools.vis.plot_methods](#), 183

[moment\(\)](#) (*maschi_tools.tools.greensfunction.GreensFunction* method), 216

[move_to_memory\(\)](#) (in module *maschi_tools.io.parsers.hdf5.transforms*), 291

[mpl_single_line_or_area\(\)](#) (in module *maschi_tools.vis.helpers*), 174

[multi_scatter_plot\(\)](#) (in module *maschi_tools.vis.plot_methods*), 186

[multiaxis_scatterplot\(\)](#) (in module *maschi_tools.vis.plot_methods*), 186

[multiple_scatterplots\(\)](#) (in module *maschi_tools.vis.plot_methods*), 187

[multiplot_moved\(\)](#) (in module *maschi_tools.vis.plot_methods*), 188

[multiply_array\(\)](#) (in module *maschi_tools.io.parsers.hdf5.transforms*), 291

[multiply_by_attribute\(\)](#) (in module *maschi_tools.io.parsers.hdf5.transforms*), 291

[multiply_scalar\(\)](#) (in module *maschi_tools.io.parsers.hdf5.transforms*), 291

N

[name](#) (*maschi_tools.io.fleurxmlmodifier.ModifierTask* attribute), 264

[name](#) (*maschi_tools.io.parsers.hdf5.reader.AttribTransformation* attribute), 284

[name](#) (*maschi_tools.io.parsers.hdf5.reader.Transformation* attribute), 285

[nested\(\)](#) (*maschi_tools.io.fleur_xml._EvalContext* method), 277

[NestedPlotParameters\(\)](#) (in module *maschi_tools.vis.parameters*), 163

[nLO](#) (*maschi_tools.tools.greensfunction.GreensfElement* attribute), 217

[nonzero_coefficients_lm](#) (*maschi_tools.tools.cf_calculation.CFCalculation* property), 211

[NoPathFound](#), 265, 406

[normalize_list_or_array\(\)](#) (in module *maschi_tools.vis.data*), 172

[normalize_xmllike\(\)](#) (in module *maschi_tools.util.xml.common_functions*), 326

[NoUniquePathFound](#), 265, 406

[nspins](#) (*maschi_tools.tools.greensfunction.GreensFunction* property), 216

[num_plots](#) (*maschi_tools.vis.parameters.Plotter* property), 166

[number_nodes\(\)](#) (*maschi_tools.io.fleur_xml._EvalContext* method), 277

O

[occupation\(\)](#) (*maschi_tools.tools.greensfunction.GreensFunction* method), 216

[onsite](#) (*maschi_tools.tools.greensfunction.GreensfElement* attribute), 217

[open_general\(\)](#) (in module *maschi_tools.io.common_functions*), 391

[orbital](#) (*maschi_tools.util.xml.xml_setters_nmmpmat.LDAUElement* attribute), 368

[out_version](#) (*maschi_tools.io.parsers.fleur_schema.OutputSchemaDict* property), 409

[out_version](#) (*maschi_tools.io.parsers.fleur_schema.schema_dict.OutputSchemaDict* property), 268

[OutParserLogAdapter](#) (class in *maschi_tools.util.logging_util*), 393

[OUTPUT_FILE](#) *maschi_tools-convert-inpgen* command line option, 309

[OutputSchemaDict](#) (class in *maschi_tools.io.parsers.fleur_schema*), 406

[OutputSchemaDict](#) (class in *maschi_tools.io.parsers.fleur_schema.schema_dict*), 265

[outxml_parser\(\)](#) (in module *maschi_tools.io.parsers.fleur*), 227

P

[parent_attributes\(\)](#) (*maschi_tools.io.fleur_xml._EvalContext* method), 278

[parse_array_float\(\)](#) (in module *maschi_tools.io.parsers.kkrparser_functions*), 225

[parse_kkr_outputfile\(\)](#) (in module *maschi_tools.io.parsers.kkrparser_functions*), 225

[parse_kkrimp_outputfile\(\)](#) (*maschi_tools.io.parsers.kkrimp_parser_functions.KkrimpParserFunctions* method), 226

[parse_voronoi_output\(\)](#) (in module *maschi_tools.io.parsers.voroparser_functions*), 226

[path](#) (*maschi_tools.util.xml.xpathbuilder.XPathBuilder* property), 334

[PDF](#) (class in *maschi_tools.vis.plot_methods*), 183

[performIntegration\(\)](#) (*maschi_tools.tools.cf_calculation.CFCalculation*

- method*), 211
- `periodic_elements()` (in module *masci_tools.io.parsers.hdf5.transforms*), 291
- `periodic_table_plot()` (in module *masci_tools.vis.bokeh_plots*), 206
- `plot_bands()` (in module *masci_tools.vis.plot_methods*), 188
- `plot_colortable()` (in module *masci_tools.vis.plot_methods*), 189
- `plot_convergence()` (in module *masci_tools.vis.bokeh_plots*), 207
- `plot_convergence()` (in module *masci_tools.vis.plot_methods*), 189
- `plot_convergence_results()` (in module *masci_tools.vis.bokeh_plots*), 208
- `plot_convergence_results()` (in module *masci_tools.vis.plot_methods*), 190
- `plot_convergence_results_m()` (in module *masci_tools.vis.bokeh_plots*), 208
- `plot_convergence_results_m()` (in module *masci_tools.vis.plot_methods*), 190
- `plot_convex_hull2d()` (in module *masci_tools.vis.plot_methods*), 191
- `plot_corelevel_spectra()` (in module *masci_tools.vis.plot_methods*), 191
- `plot_corelevels()` (in module *masci_tools.vis.plot_methods*), 192
- `plot_crystal_field_calculation()` (in module *masci_tools.tools.cf_calculation*), 213
- `plot_crystal_field_potential()` (in module *masci_tools.tools.cf_calculation*), 214
- `plot_dos()` (in module *masci_tools.vis.plot_methods*), 192
- `plot_fleur_bands()` (in module *masci_tools.vis.fleur*), 159
- `plot_fleur_bands_characterize()` (in module *masci_tools.vis.fleur*), 159
- `plot_fleur_dos()` (in module *masci_tools.vis.fleur*), 160
- `plot_kwargs()` (*masci_tools.vis.matplotlib_plotter.MatplotlibPlotter* method), 181
- `plot_kwargs()` (*masci_tools.vis.parameters.Plotter* method), 166
- `plot_lattice_constant()` (in module *masci_tools.vis.bokeh_plots*), 208
- `plot_lattice_constant()` (in module *masci_tools.vis.plot_methods*), 192
- `plot_one_element_corelv()` (in module *masci_tools.vis.plot_methods*), 193
- `plot_relaxation_results()` (in module *masci_tools.vis.plot_methods*), 193
- `plot_residuen()` (in module *masci_tools.vis.plot_methods*), 193
- `plot_shapefun()` (in module *masci_tools.vis.kkr_plot_shapefun*), 162
- `plot_spectral_function()` (in module *masci_tools.vis.plot_methods*), 194
- `plot_spinpol_bands()` (in module *masci_tools.vis.plot_methods*), 194
- `plot_spinpol_dos()` (in module *masci_tools.vis.plot_methods*), 195
- `PlotBackend` (class in *masci_tools.vis.common*), 175
- `PlotData` (class in *masci_tools.vis.data*), 168
- `PlotDataIterator` (class in *masci_tools.vis.data*), 172
- `Plotter` (class in *masci_tools.vis.parameters*), 163
- `pop()` (*masci_tools.util.lockable_containers.LockableList* method), 321
- `position` (*masci_tools.io.common_functions.AtomSiteProperties* attribute), 387
- `prepare_figure()` (*masci_tools.vis.bokeh_plotter.BokehPlotter* method), 200
- `prepare_plot()` (*masci_tools.vis.matplotlib_plotter.MatplotlibPlotter* method), 182
- `printElements()` (in module *masci_tools.tools.greensfunction*), 219
- `process()` (*masci_tools.util.logging_util.OutParserLogAdapter* method), 393
- `process_condition()` (*masci_tools.util.xml.xpathbuilder.XPathBuilder* method), 334
- `process_data_arguments()` (in module *masci_tools.vis.data*), 173
- `process_xpath_argument()` (in module *masci_tools.util.xml.common_functions*), 326
- `pseudo_voigt_profile()` (in module *masci_tools.vis.plot_methods*), 196
- ## R
- `read()` (*masci_tools.io.parsers.hdf5.reader.HDF5Reader* method), 285
- `read_charge_density()` (*masci_tools.tools.cf_calculation.CFCalculation* method), 212
- `read_constants()` (in module *masci_tools.util.schema_dict_util*), 400
- `read_groups()` (in module *masci_tools.io.hdf5_util*), 392
- `read_hdf_simple()` (in module *masci_tools.io.hdf5_util*), 392
- `read_inpgen_file()` (in module *masci_tools.io.fleur_inpgen*), 229
- `read_keywords_from_inputcard()` (*masci_tools.io.kkr_params.kkrparams* method), 224
- `read_nmmpmat_block()` (in module *masci_tools.io.io_nmmpmat*), 282

[read_potential\(\)](#) (*masci_tools.tools.cf_calculation.CFCalculation*[292](#)
method), [212](#)

[read_shapefun\(\)](#) (*in module masci_tools.io.parsers.hdf5.transforms*),
masci_tools.io.kkr_read_shapefun_info), [292](#)
[224](#)

[readCDN\(\)](#) (*masci_tools.tools.cf_calculation.CFCalculation*
method), [212](#)

[readd_inpgen_comments\(\)](#) (*in module masci_tools.util.xml.common_functions*),
[327](#)

[readPot\(\)](#) (*masci_tools.tools.cf_calculation.CFCalculation*
method), [212](#)

[register\(\)](#) (*masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDictDispatch.common*), [176](#)
method), [272](#)

[register_migration\(\)](#) (*in module masci_tools.io.parsers.fleur*), [228](#)

[rek_econ\(\)](#) (*in module masci_tools.util.econfig*), [405](#)

[rel_to_abs\(\)](#) (*in module masci_tools.io.common_functions*), [391](#)

[rel_to_abs_f\(\)](#) (*in module masci_tools.io.common_functions*), [391](#)

[relative_attrib_xpath\(\)](#)
(masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDictUtilDispatch.common), [400](#)
method), [270](#)

[relative_attrib_xpath\(\)](#)
(masci_tools.io.parsers.fleur_schema.SchemaDictUtilDispatch.common), [327](#)
method), [411](#)

[relative_iteration_attrib_xpath\(\)](#)
(masci_tools.io.parsers.fleur_schema.OutputSchemaDictUtilDispatch.common), [409](#)
method), [409](#)

[relative_iteration_attrib_xpath\(\)](#)
(masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDictUtilDispatch.common), [268](#)
method), [268](#)

[relative_iteration_tag_xpath\(\)](#)
(masci_tools.io.parsers.fleur_schema.OutputSchemaDictUtilDispatch.common), [410](#)
method), [410](#)

[relative_iteration_tag_xpath\(\)](#)
(masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDictUtilDispatch.common), [269](#)
method), [269](#)

[relative_tag_xpath\(\)](#)
(masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDictUtilDispatch.common), [271](#)
method), [271](#)

[relative_tag_xpath\(\)](#)
(masci_tools.io.parsers.fleur_schema.SchemaDictUtilDispatch.common), [412](#)
method), [412](#)

[remove\(\)](#) (*masci_tools.util.lockable_containers.LockableList*
method), [321](#)

[remove_added_parameters\(\)](#)
(masci_tools.vis.parameters.Plotter method),
[166](#)

[remove_value\(\)](#) (*masci_tools.io.kkr_params.kkrparams*
method), [224](#)

[repeat_array\(\)](#) (*in module masci_tools.io.parsers.hdf5.transforms*),
[292](#)

[repeat_array_by_attribute\(\)](#) (*in module masci_tools.io.parsers.hdf5.transforms*),
[292](#)

[replace_tag\(\)](#) (*in module masci_tools.util.xml.xml_setters_names*),
[343](#)

[replace_tag\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier*
method), [240](#)

[reset_bokeh_plot_defaults\(\)](#) (*in module masci_tools.vis.bokeh_plots*), [209](#)

[reset_defaults\(\)](#) (*in module masci_tools.vis.parameters.Plotter*
method), [166](#)

[reset_mpl_plot_defaults\(\)](#) (*in module masci_tools.vis.plot_methods*), [196](#)

[reset_parameters\(\)](#) (*masci_tools.vis.parameters.Plotter*
method), [167](#)

[reverse\(\)](#) (*masci_tools.util.lockable_containers.LockableList*
method), [322](#)

[reverse_xinclude\(\)](#) (*in module masci_tools.util.schema_dict_util*), [400](#)

[reverse_xinclude\(\)](#) (*in module masci_tools.util.xml.common_functions*),
[327](#)

[rotate_nmmpmat\(\)](#) (*in module masci_tools.util.xml.xml_setters_nmmpmat*),
[369](#)

[rotate_nmmpmat\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier*
method), [241](#)

[route_schema_block\(\)](#) (*in module masci_tools.io.io_nmmpmat*), [282](#)

[save_bokeh_defaults\(\)](#) (*in module masci_tools.vis.bokeh_plots*), [209](#)

[save_defaults\(\)](#) (*in module masci_tools.vis.common*),
[166](#)

[save_defaults\(\)](#) (*masci_tools.vis.parameters.Plotter*
method), [167](#)

[save_mpl_defaults\(\)](#) (*in module masci_tools.vis.plot_methods*), [196](#)

[save_plot\(\)](#) (*masci_tools.vis.bokeh_plotter.BokehPlotter*
method), [200](#)

[save_plot\(\)](#) (*masci_tools.vis.matplotlib_plotter.MatplotlibPlotter*
method), [182](#)

[scatter\(\)](#) (*in module masci_tools.vis.common*), [177](#)

[schema_dict_version_dispatch\(\)](#) (*in module masci_tools.io.parsers.fleur_schema*), [413](#)

[schema_dict_version_dispatch\(\)](#) (*in module masci_tools.io.parsers.fleur_schema.schema_dict*),
[413](#)

- 272
SCHEMA_FILE
 masci_tools-fleur-schema-add command
 line option, 309
- SchemaDict (class in masci_tools.io.parsers.fleur_schema), 410
- SchemaDict (class in masci_tools.io.parsers.fleur_schema.schema_dict), 269
- SchemaDictDispatch (class in masci_tools.io.parsers.fleur_schema.schema_dict), 272
- select_element_indices() (in module masci_tools.tools.greensfunction), 219
- select_elements() (in module masci_tools.tools.greensfunction), 220
- select_elements_from_file() (in module masci_tools.tools.greensfunction), 220
- serialize_xml_objects() (in module masci_tools.util.xml.common_functions), 327
- set_atomgroup() (in module masci_tools.util.xml.xml_setters_names), 344
- set_atomgroup() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 241
- set_atomgroup_label() (in module masci_tools.util.xml.xml_setters_names), 345
- set_atomgroup_label() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 242
- set_attrib_value() (in module masci_tools.util.xml.xml_setters_names), 346
- set_attrib_value() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 243
- set_bokeh_plot_defaults() (in module masci_tools.vis.bokeh_plots), 209
- set_color_palette_by_num_plots() (masci_tools.vis.bokeh_plotter.BokehPlotter method), 201
- set_complex_tag() (in module masci_tools.util.xml.xml_setters_names), 348
- set_complex_tag() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 244
- set_default_backend() (in module masci_tools.vis.common), 177
- set_defaults() (in module masci_tools.vis.common), 177
- set_defaults() (masci_tools.vis.parameters.Plotter method), 167
- set_first_attrib_value() (in module masci_tools.util.xml.xml_setters_names), 349
- set_first_attrib_value() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 245
- set_first_text() (in module masci_tools.util.xml.xml_setters_names), 350
- set_first_text() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 246
- set_inpchanges() (in module masci_tools.util.xml.xml_setters_names), 351
- set_inpchanges() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 248
- set_kpath() (in module masci_tools.util.xml.xml_setters_names), 352
- set_kpath() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 249
- set_kpath_max4() (in module masci_tools.util.xml.xml_setters_names), 353
- set_kpointlist() (in module masci_tools.util.xml.xml_setters_names), 353
- set_kpointlist() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 249
- set_kpointlist_max4() (in module masci_tools.util.xml.xml_setters_names), 355
- set_kpointmesh() (in module masci_tools.util.xml.xml_setters_names), 355
- set_kpointmesh() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 250
- set_kpointpath() (in module masci_tools.util.xml.xml_setters_names), 356
- set_kpointpath() (masci_tools.io.fleurxmlmodifier.FleurXMLModifier method), 251
- set_legend() (masci_tools.vis.bokeh_plotter.BokehPlotter method), 201
- set_limits() (masci_tools.vis.bokeh_plotter.BokehPlotter method), 201
- set_limits() (masci_tools.vis.matplotlib_plotter.MatplotlibPlotter method), 182
- set_mpl_plot_defaults() (in module masci_tools.vis.plot_methods), 196
- set_multiple_values() (masci_tools.io.kkr_params.kkrparams method), 224
- set_nkpts() (in module masci_tools.util.xml.xml_setters_names), 357

[set_nkpts\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 251
[set_nkpts_max4\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 357
[set_nmmpmat\(\)](#) (in module *masci_tools.util.xml.xml_setters_nmmpmat*), 370
[set_nmmpmat\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 252
[set_parameters\(\)](#) (*masci_tools.vis.parameters.Plotter* method), 167
[set_scale\(\)](#) (*masci_tools.vis.matplotlib_plotter.MatplotlibPlotter* method), 182
[set_simple_tag\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 358
[set_simple_tag\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 252
[set_single_default\(\)](#) (*masci_tools.vis.parameters.Plotter* method), 167
[set_species\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 359
[set_species\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 254
[set_species_label\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 361
[set_species_label\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 256
[set_text\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 362
[set_text\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 256
[set_value\(\)](#) (*masci_tools.io.kkr_params.kkrparams* method), 224
[set_xcfunctional\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 363
[set_xcfunctional\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 258
[setup\(\)](#) (in module *masci_tools.util.sphinxext*), 444
[shift_by_attribute\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 292
[shift_data\(\)](#) (*masci_tools.vis.data.PlotData* method), 171
[shift_dataset\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 292
[shift_value\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 364
[shift_value\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 258
[shift_value_species_label\(\)](#) (in module *masci_tools.util.xml.xml_setters_names*), 365
[shift_value_species_label\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 259
[show_bokeh_plot_defaults\(\)](#) (in module *masci_tools.vis.bokeh_plots*), 209
[show_colorbar\(\)](#) (*masci_tools.vis.matplotlib_plotter.MatplotlibPlotter* method), 182
[show_defaults\(\)](#) (in module *masci_tools.vis.common*), 177
[show_defaults\(\)](#) (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 252
[show_mpl_plot_defaults\(\)](#) (in module *masci_tools.vis.plot_methods*), 196
[simple_xpath\(\)](#) (*masci_tools.io.fleur_xml._EvalContext* method), 278
[single_plot](#) (*masci_tools.vis.parameters.Plotter* property), 168
[single_scatterplot\(\)](#) (in module *masci_tools.vis.plot_methods*), 196
[single_value\(\)](#) (*masci_tools.io.fleur_xml._EvalContext* method), 279
[skipHeader\(\)](#) (in module *masci_tools.io.common_functions*), 391
[slice_dataset\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 292
[sort_data\(\)](#) (*masci_tools.vis.data.PlotData* method), 172
[species](#) (*masci_tools.util.xml.xml_setters_nmmpmat.LDAUElement* attribute), 368
[sphavg](#) (*masci_tools.tools.greensfunction.GreensfElement* attribute), 218
[spin_down](#) (*masci_tools.tools.cf_calculation.CFCoefficient* attribute), 213
[spin_polarized](#) (*masci_tools.tools.cf_calculation.CFCalculation* attribute), 212
[spin_up](#) (*masci_tools.tools.cf_calculation.CFCoefficient* attribute), 213
[spinpol_bands\(\)](#) (in module *masci_tools.vis.common*), 177
[spinpol_dos\(\)](#) (in module *masci_tools.vis.common*), 178
[split_array\(\)](#) (in module *masci_tools.io.parsers.hdf5.transforms*), 293
[split_kkr_options\(\)](#)

`(masci_tools.io.kkr_params.kkrparams static method)`, 224
`split_off_attrib()` (in module `masci_tools.util.xml.common_functions`), 328
`split_off_tag()` (in module `masci_tools.util.xml.common_functions`), 328
`stack_datasets()` (in module `masci_tools.io.parsers.hdf5.transforms`), 293
`stevens_prefactor()` (`masci_tools.tools.cf_calculation.CFCalculation` method), 212
`strip_off_tag()` (`masci_tools.util.xml.xpathbuilder.XPathBuilder` method), 335
`sum_over_dict_entries()` (in module `masci_tools.io.parsers.hdf5.transforms`), 293
`sum_weights_over_atoms()` (in module `masci_tools.vis.fleur`), 161
`surface_plot()` (in module `masci_tools.vis.plot_methods`), 197
`switch_kpointset()` (in module `masci_tools.util.xml.xml_setters_names`), 366
`switch_kpointset()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 260
`switch_kpointset_max4()` (in module `masci_tools.util.xml.xml_setters_names`), 366
`switch_species()` (in module `masci_tools.util.xml.xml_setters_names`), 367
`switch_species()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 260
`switch_species_label()` (in module `masci_tools.util.xml.xml_setters_names`), 368
`switch_species_label()` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` method), 261
`symbol` (`masci_tools.io.common_functions.AtomSiteProperties` attribute), 387
`symmetric_difference()` (`masci_tools.util.case_insensitive_dict.CaseInsensitiveFrozenDict` method), 323

T
`T` (in module `masci_tools.util.lockable_containers`), 322
`T` (in module `masci_tools.util.xml.common_functions`), 324
`T_co` (in module `masci_tools.util.case_insensitive_dict`), 323
`T_co` (in module `masci_tools.util.lockable_containers`), 322
`tag_exists()` (in module `masci_tools.util.schema_dict_util`), 401
`tag_exists()` (`masci_tools.io.fleur_xml._EvalContext` method), 280
`tag_info()` (`masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDict` method), 271
`tag_info()` (`masci_tools.io.parsers.fleur_schema.SchemaDict` method), 412
`tag_xpath()` (`masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDict` method), 271
`tag_xpath()` (`masci_tools.io.parsers.fleur_schema.SchemaDict` method), 412
`TagLike` (class in `masci_tools.io.parsers.fleur_schema.fleur_schema_parser`), 414
`task_list` (`masci_tools.io.fleurxmlmodifier.FleurXMLModifier` property), 262
`text()` (`masci_tools.io.fleur_xml._EvalContext` method), 280
`tile_array()` (in module `masci_tools.io.parsers.hdf5.transforms`), 293
`tile_array_by_attribute()` (in module `masci_tools.io.parsers.hdf5.transforms`), 293
`to_global_frame()` (`masci_tools.tools.greensfunction.GreensFunction` method), 216
`to_local_frame()` (`masci_tools.tools.greensfunction.GreensFunction` method), 216
`to_m_index()` (`masci_tools.tools.greensfunction.GreensFunction` static method), 216
`to_spin_indices()` (`masci_tools.tools.greensfunction.GreensFunction` static method), 217

TOOL VERSION
`masci_tools-inpxml-convert` command line option, 311
`masci_tools-inpxml-generate-conversion` command line option, 312
`masci_tools-inpxml-show-conversion` command line option, 312
`trace_energy_dependence()` (`masci_tools.tools.greensfunction.GreensFunction` method), 217
`Transformation` (class in `masci_tools.io.parsers.hdf5.reader`), 285
`truncate_colormap()` (`masci_tools.vis.matplotlib_plotter.MatplotlibPlotter` static method), 182
`TXPathLike` (in module `masci_tools.util.typing`), 323
`type_order()` (in module `masci_tools.io.parsers.fleur_schema.fleur_schema_parser_function`), 417

U

`undo()` (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 262

`union()` (*masci_tools.util.case_insensitive_dict.CaseInsensitiveFrozenSet* method), 323

`unit` (*masci_tools.tools.cf_calculation.CFCoefficient* attribute), 213

`update_to_kkrimp()` (*masci_tools.io.kkr_params.kkrparams* method), 224

`update_to_voronoi()` (*masci_tools.io.kkr_params.kkrparams* method), 224

`use_BdG()` (in module *masci_tools.io.parsers.kkrparser_functions*), 226

`use_newsosol()` (in module *masci_tools.io.parsers.kkrparser_functions*), 226

`xml_add_number_to_first_attrb()` (in module *masci_tools.util.xml.xml_setters_xpaths*), 372

`xml_create_tag()` (in module *masci_tools.util.xml.xml_setters_basic*), 377

`xml_create_tag()` (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 262

`xml_create_tag_schema_dict()` (in module *masci_tools.util.xml.xml_setters_xpaths*), 373

`xml_delete_att()` (in module *masci_tools.util.xml.xml_setters_basic*), 378

`xml_delete_att()` (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 262

`xml_delete_tag()` (in module *masci_tools.util.xml.xml_setters_basic*), 378

`xml_delete_tag()` (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier* method), 263

XML_FILE

V

`validate()` (*masci_tools.io.parsers.fleur_schema.schema_dict.SchemaDict* method), 272

`validate()` (*masci_tools.io.parsers.fleur_schema.SchemaDict* method), 413

`validate()` (*masci_tools.tools.cf_calculation.CFCalculation* method), 213

`validate_nmmpmat()` (in module *masci_tools.util.xml.xml_setters_nmmpmat*), 371

`validate_xml()` (in module *masci_tools.util.xml.common_functions*), 328

`values()` (*masci_tools.vis.data.PlotData* method), 172

`vec_to_angles()` (in module *masci_tools.io.common_functions*), 392

`voigt_profile()` (in module *masci_tools.vis.plot_methods*), 197

masci_tools-fleur-schema-validate-input command line option, 310

masci_tools-fleur-schema-validate-output command line option, 311

masci_tools-inpxml-convert command line option, 311

masci_tools-parse-all-attrs command line option, 313

masci_tools-parse-attrb command line option, 313

masci_tools-parse-cell command line option, 314

masci_tools-parse-constants command line option, 314

masci_tools-parse-fleur-modes command line option, 314

masci_tools-parse-inp-file command line option, 314

masci_tools-parse-kpoints command line option, 315

masci_tools-parse-nkpts command line option, 315

masci_tools-parse-number-nodes command line option, 315

masci_tools-parse-out-file command line option, 316

masci_tools-parse-parameters command line option, 316

masci_tools-parse-parent-attrs command line option, 317

masci_tools-parse-relaxation command line option, 317

masci_tools-parse-structure command line option, 317

masci_tools-parse-symmetry command line option, 317

W

`waterfall_plot()` (in module *masci_tools.vis.plot_methods*), 197

`write_inpgen_file()` (in module *masci_tools.io.fleur_inpgen*), 229

`write_nmmpmat()` (in module *masci_tools.io.io_nmmpmat*), 283

`write_nmmpmat_from_orbitals()` (in module *masci_tools.io.io_nmmpmat*), 283

`write_nmmpmat_from_states()` (in module *masci_tools.io.io_nmmpmat*), 283

X

`xml_add_number_to_attrb()` (in module *masci_tools.util.xml.xml_setters_xpaths*), 371

masci_tools-parse-tag-exists command
 line option, 318
masci_tools-parse-text command line
 option, 318
xml_replace_tag() (in module
masci_tools.util.xml.xml_setters_basic), 378
xml_replace_tag() (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier*
 method), 263
xml_set_attrib_value() (in module
masci_tools.util.xml.xml_setters_xpaths),
 373
xml_set_attrib_value_no_create() (in module
masci_tools.util.xml.xml_setters_basic), 378
xml_set_attrib_value_no_create()
 (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier*
 method), 263
xml_set_complex_tag() (in module
masci_tools.util.xml.xml_setters_xpaths),
 374
xml_set_first_attrib_value() (in module
masci_tools.util.xml.xml_setters_xpaths),
 375
xml_set_first_text() (in module
masci_tools.util.xml.xml_setters_xpaths),
 375
xml_set_simple_tag() (in module
masci_tools.util.xml.xml_setters_xpaths),
 376
xml_set_text() (in module
masci_tools.util.xml.xml_setters_xpaths),
 376
xml_set_text_no_create() (in module
masci_tools.util.xml.xml_setters_basic), 379
xml_set_text_no_create()
 (*masci_tools.io.fleurxmlmodifier.FleurXMLModifier*
 method), 263
XMLFileLike (in module *masci_tools.util.typing*), 324
XMLLike (in module *masci_tools.util.typing*), 324
XPathBuilder (class in
masci_tools.util.xml.xpathbuilder), 332
XPathLike (in module *masci_tools.util.typing*), 324
Z
zoom_in() (in module
masci_tools.vis.kkr_plot_shapefun), 162