pymcxray Documentation

Release 0.1.3

Hendrix Demers

May 23, 2019

Contents

Contents:

CHAPTER 1

pymcxray

Python scripts for using mcxray software

- Free software: Apache Software License 2.0
- Documentation: https://pymcxray.readthedocs.io.

1.1 Features

- TODO

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

Todo: Add pypi installation for this project.

Warning: Right now, the *Stable release* installation does not work.

To install pymcxray, run this command in your terminal:

```
$ pip install pymcxray
```

This is the preferred method to install pymcxray, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 For developer

Clone the public repository:

```
$ git clone git://github.com/drix00/pymcxray
```

Go in the project folder and install it with pip in developer mode:

```
$ cd pymcxray
$ pip install -e .
```

Note: The project use Git LFS for the test data file. Follow the information on Git LFS to get the test data when the repository is pull.

2.3 From sources

The sources for pymcxray can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/drix00/pymcxray
```

Or download the tarball:

```
$ curl  -OL https://github.com/drix00/pymcxray/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use pymcxray in a project look in the *examples* folder.

To run the example, you need to specify the task you want to do.

```
$python simulation_test_maps.py generate
```

Note: On windows use the `py` command and not `python` if you have more than one python version installed.

Currently these tasks are available:

- `generate`
- `check`
- `read`
- `analyze`
- `scheduled_read`

At first, the script may look complex, but starting with a previous script it is easy to create a new script for you simulation.

Each script has two parts:

- **a class that subclass *mcxray._Simulations* were**
  - the simulation parameters are specified
  - results needed extracted from the simulation
  - analysis of the results can be done (this can be also done in another script)
- **a main section were**
  - batch files are generated for the simulation
  - the different task are selected using commend line argument.

Note: These two parts could be separate in two python script (file) without problem. But it seem more logical to kept them together and only have one python script file.

3.1 simulation_test_maps

To explain this example, we are going to start from the main section.

The batch file is define in a run method, which is called by the *__main__*, i.e., only run if the script is called from the command line, but not run if imported.

When the script is ready, these task are run in that order:

1. `generate`
2. `check`
3. `read`
4. `analyze`

```
$python simulation_test_maps.py generate
```

Start the batch files in parallel. It is easier to just select them all in Windows Explorer and right-click and open them.

```
BatchSimulationTestMapsMM2017_1.bat
BatchSimulationTestMapsMM2017_2.bat
```

To check the progress of all simulations, i.e., how many simulations are done and todo

```
$python simulation_test_maps.py check
```

When all simulations are done, extract the results and save it in a hdf5 file.

```
$python simulation_test_maps.py read
```

The specified results can be analysed using the script (from the hdf5 file) or analysed using another script. For example, xray-spectrum-modeling project analyse the hdf5 file generate by this script.

An alternative to use the command line argument for these tasks, it to add them directly in the script and uncommenting the task you want to run in the main section of the script.

```
if __name__ == '__main__': #pragma: no cover
    import sys
    ...
    if len(sys.argv) == 1:
        sys.argv.append(mcxray.ANALYZE_TYPE_GENERATE_INPUT_FILE)
        #sys.argv.append(mcxray.ANALYZE_TYPE_CHECK_PROGRESS)
        #sys.argv.append(mcxray.ANALYZE_TYPE_ANALYZE_RESULTS)
        #sys.argv.append(mcxray.ANALYZE_TYPE_ANALYZE_SCHEDULED_READ)
    ...
```

By default the logging level is set at *logging.WARN* and above, but *pymcxray* gives a lot information at the *logging.INFO* level so it is recommanded to set the logger level as follow

```
...
logging.getLogger().setLevel(logging.INFO)
...
```

The complete *__main__* is given below

```
if __name__ == '__main__': #pragma: no cover
    import sys
    logging.getLogger().setLevel(logging.INFO)
    logging.info(sys.argv)
    if len(sys.argv) == 1:
        sys.argv.append(mcxray.ANALYZE_TYPE_GENERATE_INPUT_FILE)
        #sys.argv.append(mcxray.ANALYZE_TYPE_CHECK_PROGRESS)
        #sys.argv.append(mcxray.ANALYZE_TYPE_ANALYZE_RESULTS)
        #sys.argv.append(mcxray.ANALYZE_TYPE_ANALYZE_SCHEDULED_READ)
    run()
```

The `run` method have three components

- a configuration file, see *Configuration file*
- a batch file object, see *Batch file*
- one or more simulation subclass, see *Simulation subclass*

Here is a example of a complete `run` method

```
def run():
    # import the batch file class.
    from pymcxray.BatchFileConsole import BatchFileConsole

    # Find the configuration file path
    configuration_file_path = get_current_module_path(__file__, "MCXRay_latest.cfg")
    program_name = get_mcxray_program_name(configuration_file_path)

    # Create the batch file object.
    batch_file = BatchFileConsole("BatchSimulationTestMapsMM2017", program_name,␣
↪numberFiles=10)

    # Create the simulation object and add the batch file object to it.
    analyze = SimulationTestMapsMM2017(relativePath=r"mcxray/SimulationTestMapsMM2017
↪",
                                       configurationFilepath=configuration_file_path)
    analyze.run(batch_file)
```

3.2 Configuration file

The configuration file is a ini style configuration file for using pymcxray. It define the paths needed to generate and run the simulations.

```
[Paths]
mcxrayProgramName=console_mcxray_x64.exe
resultsMcGillPath=D:\Dropbox\hdemers\professional\results\simulations
mcxrayArchivePath=D:\Dropbox\hdemers\professional\softwareRelease\mcxray
mcxrayArchiveName=2016-04-11_11h41m28s_MCXRay_v1.6.6.0.zip
```

See the documentation of these functions for more detail on each option

- *pymcxray.get_mcxray_program_name()*
- *pymcxray.get_results_mcgill_path()*
- *pymcxray.get_mcxray_program_path()*
- *pymcxray.get_mcxray_archive_path()*

3.3 Batch file

The batch file is responsible to create the simulation structure with a copy of mcxray program. Batch files are generated to easily run the simulations. One important parameter to set is the *numberFiles*, this is the number of batch files generated and that can be run in parallel. For maximum efficiency it should be set as the number of logical processors minus 1 or 2. For example, on a computer with 12 logical processors, the *numberFiles* should be set at 10.

See *pymcxray.BatchFileConsole.BatchFileConsole* documentation for more information about the other parameters.

3.4 Simulation subclass

The simulation subclass allows to generate a lot of simulations by varying simulations parameters.

The main features are

- **generate input files**
  - regenerate input files of simulations not done
- check the progress of the simulations: done and todo
- **extract results from the completed simulation**
  - save the results in a hdf5 file for easier analysis
- optionally do the analysis of the results

To do that the user need to subclass `pymcxray.mcxray._Simulations` and overwrite these method

- `pymcxray.mcxray._Simulations._initData()` (required)
- `pymcxray.mcxray._Simulations.getAnalysisName()` (required)
- `pymcxray.mcxray._Simulations.createSpecimen()` (required)
- `pymcxray.mcxray._Simulations.read_one_results_hdf5()` (optional)
- `pymcxray.mcxray._Simulations.analyze_results_hdf5()` (optional)

Warning: If any of the required method is modified, the simulation have to be redone completely. It is recommended to just delete the root path for the analysis and generate the input files and do the simulations. For this example, delete *SimulationTestMapsMM2017* folder.

Warning: If `pymcxray.mcxray._Simulations.read_one_results_hdf5()` is modified. In some case, the hdf5 need to be deleted.1 Furthermore, if the results were deleted: *delete_result_files* is *True*, the simulation have to be redone.

Below are given example for each method, for more detail see the method documentation.

3.4.1 Init data

This method is used to specify the options for the analysis and also the parameters used in the simulations.

The most important options for the anslysis are:

- *use_hdf5* to use the recommended hdf5 method. If it is *False* the older serial method will be used.
- *delete_result_files* if it is *True*, the result file are deleted after added in the hdf5 file. Very useful when creating a lot files like for a map.
- *createBackup* if *True* create a backup of the hdf5 file before adding more results to it. Usefull to not loss data in case of error or crash, but you should delete backup file manually as they can take a lot of space.

Warning: If *delete_result_files* is *True*, all results are deleted for a simulation and only the results specified in `pymcxray.mcxray._Simulations.read_one_results_hdf5()` are kept. If `pymcxray.mcxray._Simulations.read_one_results_hdf5()` is modified to extract more results, the simulation have to be simulate again.

This is the recommended values for the options, only change them when you are sure everything is working OK.

```
def _initData(self):
    self.use_hdf5 = True
    self.delete_result_files = False
    self.createBackup = True
```

The simulation parameters are specified in this method.

Note: If not specified, the script use MCXRay default parameters. Start MCXRay program to see the default value of each parameters.

To change simulation parameters, create a *pymcxray.SimulationsParameters.SimulationsParameters* object

```
self._simulationsParameters = SimulationsParameters()
```

Two kinds of parameter can be added:

- varied specified with a list of values
- fixed specified with a single value

The script will automatically generate a simulation for all combination of the varied parameters.

Warning: Adding a lot of varied parameters with a lot of values can generate a lot of simulations. Above 1000 simulations, all tasks of the script will be slow because of the generation or reading of a lot of files. It is recommended to start with 2 or 3 varied parameters and short list of values and test all tasks of the script. When the tests are OK and results make sense, you can increase the list of values. To add more varied parameters, it is receommended to create a new script with again only 2 or 3 varied parameters.

The parameters that can be added are defined as keyword starting with *PARAMETER_* in the module *pymcxray.SimulationsParameters*. If you don't find the parameter you want request an "enhancement" at https://github.com/drix00/pymcxray/issues.

Here is an example how-to add simulation parameters

```
from pymcxray.SimulationsParameters import SimulationsParameters, PARAMETER_INCIDENT_
↪ENERGY_keV, PARAMETER_NUMBER_ELECTRONS, \
PARAMETER_BEAM_POSITION_nm, PARAMETER_NUMBER_XRAYS
...

class SimulationTestMapsMM2017(mcxray._Simulations):
    def _initData(self):
        ...

        # Local variables for value and list if values.
        energy_keV = 30.0
        number_electrons = 10000

        #number_xrays_list = [10, 20, 30, 50, 60, 100, 200, 500, 1000]
        number_xrays_list = [10]
        xs_nm = np.linspace(-5.0e3, 5.0e3, 3)
        probePositions_nm = [tuple(position_nm) for position_nm in np.transpose([np.
↪tile(xs_nm, len(xs_nm)), np.repeat(xs_nm, len(xs_nm))]).tolist()]

        # Simulation parameters
        self._simulationsParameters = SimulationsParameters()

        self._simulationsParameters.addVaried(PARAMETER_NUMBER_XRAYS, number_xrays_
↪list)
        self._simulationsParameters.addVaried(PARAMETER_BEAM_POSITION_nm,␣
↪probePositions_nm)

        self._simulationsParameters.addFixed(PARAMETER_INCIDENT_ENERGY_keV, energy_
↪keV)
        self._simulationsParameters.addFixed(PARAMETER_NUMBER_ELECTRONS, number_
↪electrons)
```

3.4.2 Analysis name

This method specify the name of the analysis or experiment for which the simulation are done. Normally similar to the name of the class and mostly used as basename for the input files and result files. In case of two scripts writing the same path, it allows to differentiate them, but it is not recommended to run two scripts in the same folde.

```
class SimulationTestMapsMM2017(mcxray._Simulations):
    ...
    def getAnalysisName(self):
        return "SimulationTestMapsMM2017"
    ...
```

3.4.3 Create specimen

This method is used to create the specimen for each simulation. The argument of the method contains the option of the specific simulation and can be used to create the specimen. The *pymcxray.Simulation.Simulation* module contains predefined specimen which can be use in this method.

Here an example how-to use the *parameters* argument and the predefined specimen

```
def createSpecimen(self, parameters):
    weightFractions = parameters[PARAMETER_WEIGHT_FRACTIONS]

    elements = [(self.atomicNumberA, weightFractions[0]),
                (self.atomicNumberB, weightFractions[1])]
    specimen = Simulation.createAlloyBulkSample(elements)
    return specimen
```

A more complex example, where each region are specified is given below

```
def createSpecimen(self, parameters):
    # Create the specimen with a name and number of regions.
    specimen = Specimen.Specimen()
    specimen.name = "Maps01"
    specimen.numberRegions = 10

    # Region 0
    region = Region.Region()
    region.numberElements = 0
    region.regionType = RegionType.REGION_TYPE_BOX
    parameters = [-10000000000.0, 10000000000.0, -10000000000.0, 10000000000.0, 0.0,␣
↪20000000000.0]
    region.regionDimensions = RegionDimensions.RegionDimensionsBox(parameters)
    specimen.regions.append(region)

    # Region 1
    region = Region.Region()
    region.numberElements = 2
    region.elements = [Element.Element(27, massFraction=0.01), Element.Element(26,␣
↪massFraction=0.99)]
    region.regionType = RegionType.REGION_TYPE_BOX
    parameters = [-7.5e4, -2.5e4, -7.5e4, -2.5e4, 0.0, 0.2e4]
    region.regionDimensions = RegionDimensions.RegionDimensionsBox(parameters)
    specimen.regions.append(region)
    ...
```

Warning: Creating a specimen in MCXRay is complicate as sometime you need to create a empty region 0. Look at the predefined specimen in *pymcxray.Simulation.Simulation* for help. Drawing the trajectory with the `FileFormat.Results.ElectronTrajectoriesResults` will help debug the specimen. You can also request a "help wanted" at https://github.com/drix00/pymcxray/issues.

3.4.4 Read one simulation results

This method extract results from one complete simulation and added them in a hdf5 group for this simulation. The result that can be extracted are in the package *pymcxray.FileFormat.Results* and only the class implementing `write_hdf5()` can be extracted. If the desired results does not implement the `write_hdf5()` method, request an "enhancement" at https://github.com/drix00/pymcxray/issues.

Note: The format of the hdf5 file is not well documented. Check the implementation of the `write_hdf5()` method and request an "enhancement" at https://github.com/drix00/pymcxray/issues for the documentation.

Note: The program HDFView is useful to look at the hdf5 file. See https://support.hdfgroup.org/products/java/hdfview/.

Here is an example how-to extract the electron results (BSE, TE, ...)

```
def read_one_results_hdf5(self, simulation, hdf5_group):
    electronResults = ElectronResults.ElectronResults()
    electronResults.path = self.getSimulationsPath()
    electronResults.basename = simulation.resultsBasename
    electronResults.read()
    electronResults.write_hdf5(hdf5_group)
```

So far this class are implemented with hdf5 support

- *pymcxray.FileFormat.Results.ElectronResults.ElectronResults*
- *pymcxray.FileFormat.Results.PhirhozEmittedCharacteristic.PhirhozEmittedCharacteristic*
- *pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristic.PhirhozGeneratedCharacteristic*
- *pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristicThinFilm.PhirhozGeneratedCharacteristicThinFilm*
- *pymcxray.FileFormat.Results.XrayIntensities.XrayIntensities*
- *pymcxray.FileFormat.Results.XraySpectraRegionsEmitted.XraySpectraRegionsEmitted*
- *pymcxray.FileFormat.Results.XraySpectraSpecimenEmittedDetected.XraySpectraSpecimenEmittedDetected*

3.4.5 Analyze all simulations

This method is only needed for the task `analyze`.

Often the method start by calling `mcxray._Simulations.readResults()` to read all new results and add them in the hdf5 file.

The example below shows how-to open the hdf5 file in memory for somewhat fast analysis simulation. The file is read only at the beginning and stored in memory.

```
def analyze_results_hdf5(self): #pragma: no cover
    self.readResults()

    file_path = self.get_hdf5_file_path()
    with h5py.File(file_path, 'r', driver='core') as hdf5_file:
        hdf5_group = self.get_hdf5_group(hdf5_file)
        logging.info(hdf5_group.name)
```

Todo: Document *pymcxray.mcxray*

Todo: Document *pymcxray.SimulationsParameters*

Todo: Document *pymcxray.Simulation*

Todo: Document *pymcxray.FileFormat.Results.ElectronResults.ElectronResults*

Todo: Document *pymcxray.FileFormat.Results.PhirhozEmittedCharacteristic.PhirhozEmittedCharacteristic*

Todo: Document *pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristic.PhirhozGeneratedCharacteristic*

Todo: Document *pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristicThinFilm.PhirhozGeneratedCharacteristicThinFilm*

Todo: Document *pymcxray.FileFormat.Results.XrayIntensities.XrayIntensities*

Todo: Document *pymcxray.FileFormat.Results.XraySpectraRegionsEmitted.XraySpectraRegionsEmitted*

Todo: Document *pymcxray.FileFormat.Results.XraySpectraSpecimenEmittedDetected.XraySpectraSpecimenEmittedDetected*

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/drix00/pymcxray/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

pymcxray could always use more documentation, whether as part of the official pymcxray docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/drix00/pymcxray/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *pymcxray* for local development.

1. Fork the *pymcxray* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pymcxray.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv pymcxray
$ cd pymcxray/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pymcxray tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/drix00/pymcxray/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_pymcxray
```

CHAPTER 5

Credits

5.1 Development Lead

- Hendrix Demers <hendrix.demers@mail.mcgill.ca>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.2 (2017-06-26)

- Add create multi-layers specimen.
- Add number of layers X, Y, and Z as parameters.

6.2 0.1.0 (2017-03-07)

- First release on PyPI.

CHAPTER 7

ToDo

Todo: Add units.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/pymcxray/ElementPrope of pymcxray.ElementProperties.g_FermiEnergy, line 6.)

Todo: Add units.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/pymcxray/ElementPrope of pymcxray.ElementProperties.g_kFermi, line 6.)

Todo: Add units.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/pymcxray/ElementPrope of pymcxray.ElementProperties.g_plasmonEnergy, line 6.)

Todo: Add pypi installation for this project.

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/installation.rst, line 13.)

Todo: Document *pymcxray.mcxray*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 436.)

Todo: Document *pymcxray.SimulationsParameters*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 437.)

Todo: Document *pymcxray.Simulation*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 438.)

Todo: Document *pymcxray.FileFormat.Results.ElectronResults.ElectronResults*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 439.)

Todo: Document *pymcxray.FileFormat.Results.PhirhozEmittedCharacteristic.PhirhozEmittedCharacteristic*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 440.)

Todo: Document *pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristic.PhirhozGeneratedCharacteristic*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 441.)

Todo: Document *pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristicThinFilm.PhirhozGeneratedCharacteristicThinFilm*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 442.)

Todo: Document *pymcxray.FileFormat.Results.XrayIntensities.XrayIntensities*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 443.)

Todo: Document *pymcxray.FileFormat.Results.XraySpectraRegionsEmitted.XraySpectraRegionsEmitted*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 444.)

Todo: Document *pymcxray.FileFormat.Results.XraySpectraSpecimenEmittedDetected.XraySpectraSpecimenEmittedDetected*

(The original entry is located in /home/docs/checkouts/readthedocs.org/user_builds/pymcxray/checkouts/stable/docs/usage.rst, line 445.)

CHAPTER 8

pymcxray

8.1 pymcxray package

8.1.1 Subpackages

pymcxray.FileFormat package

Subpackages

pymcxray.FileFormat.Results package

Subpackages

pymcxray.FileFormat.Results.exported package

Submodules

pymcxray.FileFormat.Results.exported.DataMap module

Read data map exported by MCXRay.

class pymcxray.FileFormat.Results.exported.DataMap.**DataMap**(*filepath*)

Bases: object

imageName

pixels

read()

saveImage(*imageFilepath=None*)

showImage()

size

pymcxray.FileFormat.Results.exported.DataMap.**run**()

pymcxray.FileFormat.Results.exported.XrayIntensityXY module

Read x-ray intensity distribution in XY exported from mcxray GUI.

class pymcxray.FileFormat.Results.exported.XrayIntensityXY.**XrayIntensityXY**
Bases: object

readData(*filepath*)

pymcxray.FileFormat.Results.exported.test_DataMap module

Tests for the module *DataMap*.

class pymcxray.FileFormat.Results.exported.test_DataMap.**TestDataMap**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *DataMap*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_init()
Tests for method *init*.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.exported.test_XrayIntensityXY module

Tests for the module XrayIntensityXY.

class pymcxray.FileFormat.Results.exported.test_XrayIntensityXY.**TestXrayIntensityXY**(*methodNa*
Bases: unittest.case.TestCase

TestCase class for the module *XrayIntensityXY*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testOpenFile()
Test if the test data file can be open.

testSkeleton()
First test to check if the testcase is working with the testing framework.

pymcxray.FileFormat.Results.exported.tests module

Module contents

Submodules

pymcxray.FileFormat.Results.BaseResults module

BaseResults

class pymcxray.FileFormat.Results.BaseResults.**BaseResults**(*path=''*, *basename='MCXRay'*)
Bases: object

basename

filepath

path

pymcxray.FileFormat.Results.BeamParameters module

MCXRay beam parameters from results file.

class pymcxray.FileFormat.Results.BeamParameters.**BeamParameters**
Bases: object

acquisitionTime_s

current_A

diameter90_A

gaussianMean

gaussianSigma

incidentEnergy_keV

readFromLines(*lines*)

tiltAngle_deg

pymcxray.FileFormat.Results.DetectorParameters module

MCXRay detector parameters from results file.

class pymcxray.FileFormat.Results.DetectorParameters.**DetectorParameters**
Bases: object

angleBetweenDetectorSpecimenNormal_deg

angleBetweenDetectorXAxis_deg

beamDetectorDistance_cm

crystalDensity_g_cm3

crystalName

crystalRadius_cm

crystalThickness_cm

deadLayerThickness_A

diffusionLength_A

noiseEdsDetector_eV

readFromLines(*lines*)

solidAngle_deg

surfaceQualityFactor

takeoffAngleEffective_deg

takeoffAngleNormalIncidence_deg

thicknessAir_um

thicknessAlWindow_um

thicknessBeWindow_um

thicknessH2O_um

thicknessMoxtek_um

thicknessOil_um

thicknessTiWindow_um

pymcxray.FileFormat.Results.Dump module

MCXRay dump results file.

class pymcxray.FileFormat.Results.Dump.**Dump**
Bases: object

read(*filepath*)

pymcxray.FileFormat.Results.ElectronExistResults module

Read electron exit results simulated with MCXRay.

class pymcxray.FileFormat.Results.ElectronExistResults.**ElectronDetector**
Bases: object

azimuthalAngleRange_deg

detectElectrons(*data*)

energyRange_keV

maximumAzimuthalAngle_deg

maximumEnergy_keV

maximumPolarAngle_deg

minimumAzimuthalAngle_deg

minimumEnergy_keV

minimumPolarAngle_deg

polarAngleRange_deg

class pymcxray.FileFormat.Results.ElectronExistResults.**ElectronExistResults**(**args*, ***kargs*)

Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

data

getEnergyDistribution(*numberBins=10*)

numberData

read()

pymcxray.FileFormat.Results.ElectronParameters module

MCXRay electron parameters results file.

class pymcxray.FileFormat.Results.ElectronParameters.**ElectronParameters**

Bases: object

backscatteredRatio

eRatio

internalRatio

meanAzimuthalAngleCollision_deg

meanDistanceBetweenCollisions_A

meanNumberCollisionPerElectrons

meanPolarAngleCollision_deg

numberSimulatedElectrons

readFromLines(*lines*)

skirtRatio

throughRatio

pymcxray.FileFormat.Results.ElectronResults module

Read ElectronResults MCXRay results file.

class pymcxray.FileFormat.Results.ElectronResults.**ElectronResults**

Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

fieldNames

fractionBackscatteredElectrons

fractionInternalElectrons

fractionSkirtedElectrons

fractionTransmittedElectrons

numberBackscatteredElectrons

numberElectronCollisions

numberInternalElectrons

numberSimulatedElectrons

numberSkirtedElectrons

numberTransmittedElectrons

read()

write_hdf5(*hdf5_group*)

pymcxray.FileFormat.Results.ElectronTrajectoriesResults module

Read MCXray electron trajectories results file.

class pymcxray.FileFormat.Results.ElectronTrajectoriesResults.**Collision**
Bases: object

collisionType

correctedX_A

correctedY_A

correctedZ_A

energy_keV

indexRegion

x_A

y_A

z_A

class pymcxray.FileFormat.Results.ElectronTrajectoriesResults.**ElectronTrajectoriesResults**(*f*
Bases: object

drawXY(*title=''*, *corrected=False*, *x_limit=None*, *y_limit=None*)

drawXZ(*title=''*, *corrected=False*, *theta_deg=0.0*, *colorType='colorTrajectoryType'*, *trajectoryIndexes=None*, *x_limit=None*, *y_limit=None*)

drawYZ(*title=''*, *corrected=False*, *x_limit=None*, *y_limit=None*)

getElectronGunPositions_nm()

read(*filepath*)

write_hdf5(*hdf5_group*)

class pymcxray.FileFormat.Results.ElectronTrajectoriesResults.**Trajectory**
Bases: object

addCollision(*collision*)

collisions

index

trajectoryType

pymcxray.FileFormat.Results.ElectronTrajectoriesResults.**run**()

pymcxray.FileFormat.Results.ElectronTrajectoriesResults.**runAuCThinFilm**()

pymcxray.FileFormat.Results.ElectronTrajectoriesResults.**runFogging**()

pymcxray.FileFormat.Results.ElementParameters module

MCXRay element parameters result file.

class pymcxray.FileFormat.Results.ElementParameters.**ElementParameters**
Bases: object

pymcxray.FileFormat.Results.Intersections module

MCXRay intersections results file.

class pymcxray.FileFormat.Results.Intersections.**Intersections**
Bases: object

extractFromLines(*lines*)

pymcxray.FileFormat.Results.MicroscopeParameters module

MCXRay microscope parameter in results file.

class pymcxray.FileFormat.Results.MicroscopeParameters.**MicroscopeParameters**
Bases: object

beamParameters

detectorParameters

readFromLines(*lines*)

pymcxray.FileFormat.Results.ModelParameters module

MCXRay model parameters from results file.

class pymcxray.FileFormat.Results.ModelParameters.**ModelParameters**
Bases: object

atomCollisionModel

atomCollisionScreeningModel

atomCrossSectionModel

atomCrossSectionScreeningModel

atomElectronRangeModel

atomEnergyLossModel

atomMeanIonizationPotentialModel

atomScreeningModel

bremsstrahlungCrossSectionModel

characterisitcCrossSectionModel

readFromLines(*lines*)

regionEnergyLossModel

pymcxray.FileFormat.Results.Phirhoz module

MCXRay phirhoz result file.

class pymcxray.FileFormat.Results.Phirhoz.**Phirhoz**(*symbol*, *shell*)
Bases: object

depths_A

intensity

readFromLines(*lines*)

shell

symbol

values

pymcxray.FileFormat.Results.PhirhozElement module

MCXRay result file phirhoz element.

class pymcxray.FileFormat.Results.PhirhozElement.**PhirhozElement**
Bases: object

isIonizationShell_K

isIonizationShell_L

isIonizationShell_M

readFromLine(*line*)

symbol

weightFraction

pymcxray.FileFormat.Results.PhirhozEmittedCharacteristic module

Read PhirhozEmittedCharacteristic file from MCXRay.

class pymcxray.FileFormat.Results.PhirhozEmittedCharacteristic.**PhirhozEmittedCharacteristi**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

depth_A

depth_nm

fieldNames

phirhozs

read(*regionID=0*)

write_hdf5(*hdf5_group*)

pymcxray.FileFormat.Results.PhirhozEmittedCharacteristicThinFilm module

Read mcxray phirhoz thin film emitted results.

class pymcxray.FileFormat.Results.PhirhozEmittedCharacteristicThinFilm.**PhirhozEmittedCharac**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

fieldNames

getIntensity(*regionID*, *atomicSymbol*, *xrayLine*)

intensities

numberRegions

read()

pymcxray.FileFormat.Results.PhirhozGenerated module

MCXRay phirhoz generated result file.

class pymcxray.FileFormat.Results.PhirhozGenerated.**PhirhozGenerated**
Bases: object

electronParameters

getCharacteristicPhiRhoZ(*regionID*)

microscopeParameters

modelParameters

numberRegions

read(*filepath*)

readElectron(*lines*)

readMicroscope(*lines*)

readRegions(*lines*)

readSimulationParameters(*lines*)

readSolidSimulationModels(*lines*)

simulationParameters

pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristic module

Read PhirhozGeneratedCharacteristic file from MCXRay program.

class pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristic.**PhirhozGeneratedCharacter**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

depth_A

depth_nm

fieldNames

phirhozs

read(*regionID=0*)

write_hdf5(*hdf5_group*)

pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristicThinFilm module

Read mcxray phirhoz thin film generated results.

class pymcxray.FileFormat.Results.PhirhozGeneratedCharacteristicThinFilm.**PhirhozGeneratedCl**

Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

fieldNames

getIntensity(*regionID*, *atomicSymbol*, *xrayLine*)

get_subshells()

get_symbols()

intensities

numberRegions

read()

write_hdf5(*hdf5_group*)

pymcxray.FileFormat.Results.PhirhozRegion module

MCXRay phirhoz results file for a region.

class pymcxray.FileFormat.Results.PhirhozRegion.**PhirhozRegion**(*numberEnergyWindows*, *numberLayersZ*)

Bases: object

readBackgroundPhirhozs(*lines*)

readCharacteristicPhirhozs(*lines*)

readElements(*lines*)

readFromLines(*lines*)

readPhirhozDistributions(*lines*)

readRegionVolume(*lines*)

regionID

pymcxray.FileFormat.Results.RegionParameters module

MCXRay region parameters result file.

class pymcxray.FileFormat.Results.RegionParameters.**RegionParameters**

Bases: object

elements

layerThickness_A

numberElements

regionID

pymcxray.FileFormat.Results.RegionVolume module

MCXRay result file region volume.

class pymcxray.FileFormat.Results.RegionVolume.**RegionVolume**
Bases: object

readFromLines(*lines*)

regionID

pymcxray.FileFormat.Results.SimulationParameters module

MCXRay simulation parameters results file.

class pymcxray.FileFormat.Results.SimulationParameters.**SimulationParameters**
Bases: object

edsMaximumEnergy_keV

generalizedWalk

interpolationType

maximumLiveTime_s

numberChannels

numberElectrons

numberEnergyWindows

numberLayersX

numberLayersY

numberLayersZ

numberPhotons

readFromLines(*lines*)

useLiveTime_s

pymcxray.FileFormat.Results.Spectra module

MCXRay spectra result file.

class pymcxray.FileFormat.Results.Spectra.**Spectra**
Bases: object

getElementSpectra()

getElementSpectrum(*regionID*, *elementName*)

getRegionParameters(*regionID*)

getRegionSpectrum(*regionID*)

getSpecimenSpectrum()

numberRegions

read(*filepath*)

readRegion(*lines*)

readRegions(*lines*)

readSpecimen(*lines*)

pymcxray.FileFormat.Results.SpectraEDS module

Read MCXRay spectra EDS results file.

class pymcxray.FileFormat.Results.SpectraEDS.**SpectraEDS**
Bases: object

characteristicProbability

elements

numberCharateristicPeaks

numberElements

numberSimulatedPhotons

readFileObject(*inputFile*)

readFilepath(*filepath*)

readPartialSpectraReferenceSection(*lines*)

readRegionSpectraSection(*lines*)

readTestInputSection(*lines*)

regionID

pymcxray.FileFormat.Results.SpectraEDS.**runExample**()

pymcxray.FileFormat.Results.Spectrum module

MCXRay spectrum result file.

class pymcxray.FileFormat.Results.Spectrum.**Spectrum**
Bases: object

backgroundIntensities

characteristicIntensities

energies_keV

intensities

pymcxray.FileFormat.Results.SpectrumEDS module

Read MCXRay EDS spectrm from results file.

class pymcxray.FileFormat.Results.SpectrumEDS.**SpectrumEDS**(*lines=None*)
Bases: object

channels

countsList

enegies_keV

readLines(*lines*)

pymcxray.FileFormat.Results.Tags module

MCXRay tags used in output files.

exception pymcxray.FileFormat.Results.Tags.**TagNotFoundError**
Bases: ValueError

pymcxray.FileFormat.Results.Tags.**findAllTag**(*tag*, *lines*, *contains=None*)

pymcxray.FileFormat.Results.Tags.**findTag**(*tag*, *lines*)

pymcxray.FileFormat.Results.XrayIntensities module

Xray intensities result file from MCXRay.

class pymcxray.FileFormat.Results.XrayIntensities.**XrayIntensities**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

fieldNames

getAtomicNumberLineEnergySets()

getDetectedIntensity(*atomicNumber*, *xrayLine*)

getIntensity(*data_type*, *region_id*, *atomic_number*, *xray_line*)

getIntensityEmitted(*atomicNumber*, *xrayLine*, *total=True*)

getIntensityEmittedDetected(*atomicNumber*, *xrayLine*, *total=True*)

getIntensityGenerated(*atomicNumber*, *xraySubshell*, *total=False*)

getIntensityGeneratedDetected(*atomicNumber*, *xraySubshell*)

get_result_types()

get_xray_lines()

intensities

numberIntensities

read()

write_hdf5(*hdf5_group*)

pymcxray.FileFormat.Results.XraySimulatedSpectraRegion module

description

class pymcxray.FileFormat.Results.XraySimulatedSpectraRegion.**XraySimulatedSpectraRegion**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

channelNumbers

detectedIntensities

eNetPeak

energiesReference_keV

energies_keV

fieldNames

peakToBackgrpound

peakToBackgrpoundAverage

read(*regionID=0*)

simulatedIntensities

pymcxray.FileFormat.Results.XraySimulatedSpectraSpecimen module

description

class pymcxray.FileFormat.Results.XraySimulatedSpectraSpecimen.**XraySimulatedSpectraSpecimen**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

energies_keV

fieldNames

read()

totals

pymcxray.FileFormat.Results.XraySpectraAtomEmittedDetectedLines module

Read MCXRay XraySpectraAtomEmittedDetectedLines file.

class pymcxray.FileFormat.Results.XraySpectraAtomEmittedDetectedLines.**XraySpectraAtomEmitte**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

characteristics

energies_keV

fieldNames

read(*regionID*)

pymcxray.FileFormat.Results.XraySpectraRegionEmitted module

Read XraySpectraRegionEmitted mcxray result file.

class pymcxray.FileFormat.Results.XraySpectraRegionEmitted.**XraySpectraRegionEmitted**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

bremsstrahlungValue_1_ekeVsr(*energy_keV*)

bremsstrahlung_1_ekeVsr

characteristicValue_1_ekeVsr(*energy_keV*)

characteristic_1_ekeVsr

energies_keV

fieldnames

read(*regionID=0*)

totalValue_1_ekeVsr(*energy_keV*)

total_1_ekeVsr

pymcxray.FileFormat.Results.XraySpectraRegionEmitted.**run**()

pymcxray.FileFormat.Results.XraySpectraRegionsEmitted module

Read all XraySpectraRegionsEmitted mcxray result files for all regions.

class pymcxray.FileFormat.Results.XraySpectraRegionsEmitted.**XraySpectraRegionsEmitted**
Bases: *pymcxray.FileFormat.Results.XraySpectraRegionEmitted.XraySpectraRegionEmitted*

read()

write_hdf5(*hdf5_group*)

pymcxray.FileFormat.Results.XraySpectraRegionsEmitted.**run**()

pymcxray.FileFormat.Results.XraySpectraSpecimen module

Read XraySpectraSpecimen MCXRay results file.

class pymcxray.FileFormat.Results.XraySpectraSpecimen.**XraySpectraSpecimen**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

backgrounds

characteristics

energies_keV

fieldNames

read()

totals

write_hdf5(*hdf5_group*)

pymcxray.FileFormat.Results.XraySpectraSpecimenEmittedDetected module

Read XraySpectraSpecimenEmittedDetected MCXRay results file.

class pymcxray.FileFormat.Results.XraySpectraSpecimenEmittedDetected.**XraySpectraSpecimenEm**
Bases: *pymcxray.FileFormat.Results.BaseResults.BaseResults*

backgrounds

characteristics

energies_keV

fieldNames

read()

totals

write_hdf5(*hdf5_group*)

pymcxray.FileFormat.Results.test_BaseResults module

Tests for the module *BaseResults*.

class pymcxray.FileFormat.Results.test_BaseResults.**TestBaseResults**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *BaseResults*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_basename()
Tests for method *basename*.

test_init()
Tests for method *init*.

test_path()
Tests for method *path*.

pymcxray.FileFormat.Results.test_BeamParameters module

Tests for module *BeamParameters*.

class pymcxray.FileFormat.Results.test_BeamParameters.**TestBeamParameters**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *BeamParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLines()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_DetectorParameters module

Tests for module *DetectorParameters*.

class pymcxray.FileFormat.Results.test_DetectorParameters.**TestDetectorParameters**(*methodName=*
Bases: unittest.case.TestCase

TestCase class for the module *DetectorParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLines()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_Dump module

Tests for module *Dump*

class pymcxray.FileFormat.Results.test_Dump.**TestDump**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Dump*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_ElectronExistResults module

pymcxray.FileFormat.Results.test_ElectronParameters module

Tests for module *ElectronParameters*.

class pymcxray.FileFormat.Results.test_ElectronParameters.**TestElectronParameters**(*methodName=*
Bases: unittest.case.TestCase

TestCase class for the module *ElectronParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLines()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_ElectronParameters.**getLinesAndReference**()

pymcxray.FileFormat.Results.test_ElectronResults module

Tests for the module *XraySpectraSpecimen*.

class pymcxray.FileFormat.Results.test_ElectronResults.**TestElectronResults**(*methodName='runTest*
Bases: unittest.case.TestCase

TestCase class for the module *ElectronResults*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_ElementParameters module

Tests for the module *ElementParameters*.

class pymcxray.FileFormat.Results.test_ElementParameters.**TestElementParameters**(*methodName='r*
Bases: unittest.case.TestCase

TestCase class for the module *ElementParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

pymcxray.FileFormat.Results.test_Intersections module

Tests for module *Intersections*.

class pymcxray.FileFormat.Results.test_Intersections.**TestIntersections**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Intersections*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_extractFromLines()
Tests for method *extractFromLines*.

pymcxray.FileFormat.Results.test_MicroscopeParameters module

Tests for module *MicroscopeParameters*.

class pymcxray.FileFormat.Results.test_MicroscopeParameters.**TestMicroscopeParameters**(*methodl*
Bases: unittest.case.TestCase

TestCase class for the module *MicroscopeParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLines()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_MicroscopeParameters.**getLinesAndReference**()

pymcxray.FileFormat.Results.test_ModelParameters module

ModelParameters

class pymcxray.FileFormat.Results.test_ModelParameters.**TestModelParameters**(*methodName='runTest*
Bases: unittest.case.TestCase

TestCase class for the module *ModelParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLines()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_ModelParameters.**getLinesAndReference**()

pymcxray.FileFormat.Results.test_Phirhoz module

Tests for the module *Phirhoz*.

class pymcxray.FileFormat.Results.test_Phirhoz.**TestPhirhoz**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Phirhoz*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLines()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_Phirhoz.**getLinesAndReference**(*path*)

pymcxray.FileFormat.Results.test_PhirhozElement module

Tests for the module *PhirhozElement*.

class pymcxray.FileFormat.Results.test_PhirhozElement.**TestPhirhozElement**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *PhirhozElement*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLine()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_PhirhozElement.**getLineAndReference**()

pymcxray.FileFormat.Results.test_PhirhozEmittedCharacteristic module

Tests for the module *PhirhozEmittedCharacteristic*.

class pymcxray.FileFormat.Results.test_PhirhozEmittedCharacteristic.**TestPhirhozEmittedChara**
Bases: unittest.case.TestCase

TestCase class for the module *PhirhozEmittedCharacteristic*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_PhirhozEmittedCharacteristicThinFilm module

Tests for module *PhirhozEmittedCharacteristicThinFilm*.

class pymcxray.FileFormat.Results.test_PhirhozEmittedCharacteristicThinFilm.**TestPhirhozEmit**
Bases: unittest.case.TestCase

TestCase class for the module *PhirhozEmittedCharacteristicThinFilm*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_getIntensity()
Tests for method *getIntensity*.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_PhirhozGenerated module

Tests for the module *PhirhozGenerated*.#

class pymcxray.FileFormat.Results.test_PhirhozGenerated.**TestPhirhozGenerated**(*methodName='runT*
Bases: unittest.case.TestCase

TestCase class for the module *moduleName*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_PhirhozGeneratedCharacteristic module

Tests for the module *PhirhozGeneratedCharacteristic*.

class pymcxray.FileFormat.Results.test_PhirhozGeneratedCharacteristic.**TestPhirhozGenerated**
Bases: unittest.case.TestCase

TestCase class for the module *PhirhozGeneratedCharacteristic*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_PhirhozGeneratedCharacteristicThinFilm module

Tests for module *PhirhozGeneratedCharacteristicThinFilm*.

class pymcxray.FileFormat.Results.test_PhirhozGeneratedCharacteristicThinFilm.**TestPhirhozGe**
Bases: unittest.case.TestCase

TestCase class for the module *PhirhozGeneratedCharacteristicThinFilm*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_getIntensity()
Tests for method *getIntensity*.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_PhirhozRegion module

Tests for the module *PhirhozRegion*.

class pymcxray.FileFormat.Results.test_PhirhozRegion.**TestPhirhozRegion**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *PhirhozRegion*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLines()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_PhirhozRegion.**getLinesAndReference**(*path*)

pymcxray.FileFormat.Results.test_RegionParameters module

Tests for the module *RegionParameters*.

class pymcxray.FileFormat.Results.test_RegionParameters.**TestRegionParameters**(*methodName='runT*
Bases: unittest.case.TestCase

TestCase class for the module *RegionParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

pymcxray.FileFormat.Results.test_RegionVolume module

Tests for the module *RegionVolume*.

class pymcxray.FileFormat.Results.test_RegionVolume.**TestRegionVolume**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *RegionVolume*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

NOtest_readFromLines()
Tests for method *readFromLines*.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

pymcxray.FileFormat.Results.test_RegionVolume.**getLinesAndReference**()

pymcxray.FileFormat.Results.test_SimulationParameters module

Tests for the module *SimulationParameters*.

class pymcxray.FileFormat.Results.test_SimulationParameters.**TestSimulationParameters**(*methodl*
Bases: unittest.case.TestCase

TestCase class for the module *SimulationParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_readFromLines()
Tests for method *readFromLines*.

pymcxray.FileFormat.Results.test_SimulationParameters.**getLinesAndReference**()

pymcxray.FileFormat.Results.test_Spectra module

Tests for the module *Spectra*.

class pymcxray.FileFormat.Results.test_Spectra.**TestSpectra**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Spectra*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test__extractElementHeader()
Tests for method *_extractElementHeader*.

test__extractRegionHeader()
Tests for method *_extractRegionHeader*.

test_read()
Tests for method *read*.

test_readRegion()
Tests for method *readRegion*.

test_readSpecimen()
Tests for method *readSpecimen*.

pymcxray.FileFormat.Results.test_SpectraEDS module

Tests for module *SpectraEDS*.

class pymcxray.FileFormat.Results.test_SpectraEDS.**TestSpectraEDS**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *SpectraEDS*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_FindTestData()
Tests for method *FindTestData*.

test__extractTotalCounts()
Tests for method *_extractTotalCounts*.

test__isPartialSpectraReferenceSection()
Tests for method *_isTestInputSection*.

test__isRegionSpectraSection()
Tests for method *_isRegionSpectraSection*.

test__isTestInputSection()
Tests for method *_isTestInputSection*.

test_readPartialSpectraReferenceSection()
Tests for method *readTestInputSection*.

test_readRegionSpectraSection()
Tests for method *readRegionSpectraSection*.

test_readTestInputSection()
Tests for method *readTestInputSection*.

pymcxray.FileFormat.Results.test_Spectrum module

Tests for the module *Spectrum*

class pymcxray.FileFormat.Results.test_Spectrum.**TestSpectrum**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Spectrum*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

pymcxray.FileFormat.Results.test_Tags module

Tests for the modules *Tags*.

class pymcxray.FileFormat.Results.test_Tags.**TestTags**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Tags*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

getLines()

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_findTag()
Tests for method *findTag*.

pymcxray.FileFormat.Results.test_XrayIntensities module

Tests for the module *XrayIntensities*.

class pymcxray.FileFormat.Results.test_XrayIntensities.**TestXrayIntensities**(*methodName='runTest*
Bases: unittest.case.TestCase

TestCase class for the module *XrayIntensities*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_XraySimulatedSpectraRegion module

description

class pymcxray.FileFormat.Results.test_XraySimulatedSpectraRegion.**TestXraySimulatedSpectra**
Bases: unittest.case.TestCase

TestCase class for the module *moduleName*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_XraySimulatedSpectraSpecimen module

class pymcxray.FileFormat.Results.test_XraySimulatedSpectraSpecimen.**TestXraySimulatedSpectr**
Bases: unittest.case.TestCase

TestCase class for the module *XraySimulatedSpectraSpecimen*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_XraySpectraAtomEmittedDetectedLines module

Tests for the module *XraySpectraAtomEmittedDetectedLines*.

class pymcxray.FileFormat.Results.test_XraySpectraAtomEmittedDetectedLines.**TestXraySpectraA**
Bases: unittest.case.TestCase

TestCase class for the module *XraySpectraAtomEmittedDetectedLines*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_XraySpectraRegionEmitted module

pymcxray.FileFormat.Results.test_XraySpectraRegionsEmitted module

pymcxray.FileFormat.Results.test_XraySpectraSpecimen module

Tests for the module *XraySpectraSpecimen*.

class pymcxray.FileFormat.Results.test_XraySpectraSpecimen.**TestXraySpectraSpecimen**(*methodNa*
Bases: unittest.case.TestCase

TestCase class for the module *XraySpectraSpecimen*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.test_XraySpectraSpecimenEmittedDetected module

Tests for the module *XraySpectraSpecimenEmittedDetected*.

class pymcxray.FileFormat.Results.test_XraySpectraSpecimenEmittedDetected.**TestXraySpectraSp**
Bases: unittest.case.TestCase

TestCase class for the module *XraySpectraSpecimenEmittedDetected*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.Results.tests module

Module contents

Submodules

pymcxray.FileFormat.Element module

MCXRay element input file.

class pymcxray.FileFormat.Element.**Element**(*atomicNumber=0*, *massFraction=1.0*)
Bases: object

atomicNumber

createLineOldVersion()

createLinesWithKey()

extractFromLineOldVersion(*line*)

extractFromLinesWithKey(*lines*)

massFraction

name

pymcxray.FileFormat.ExportedSpectrum module

Read and write exported spectrum from McXRay.

class pymcxray.FileFormat.ExportedSpectrum.**ExportedSpectrum**
Bases: object

getData()

getSpectrumType()

read(*filepath*)

pymcxray.FileFormat.FileReaderWriterTools module

description

pymcxray.FileFormat.FileReaderWriterTools.**reduceAfterDot**(*value*)

pymcxray.FileFormat.MCXRayModel module

Model type used in MCXRay.

class pymcxray.FileFormat.MCXRayModel.**AtomCollisionModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_BROWNING = 1

TYPE_GAUVIN = 2

TYPE_RUTHERFORD = 0

class pymcxray.FileFormat.MCXRayModel.**AtomCollisionScreeningModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_HENOC_MAURICE = 0

class pymcxray.FileFormat.MCXRayModel.**AtomCrossSectionModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_BROWNING = 0

TYPE_GAUVIN_DROUIN = 1

class pymcxray.FileFormat.MCXRayModel.**AtomCrossSectionScreeningModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_HENOC_MAURICE = 0

class pymcxray.FileFormat.MCXRayModel.**AtomElectronRangeModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_KANAYA_OKAYAMA = 0

class pymcxray.FileFormat.MCXRayModel.**AtomEnergyLossModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_BETHE = 0

class pymcxray.FileFormat.MCXRayModel.**AtomMeanIonizationPotentialModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_JOY_LUO = 0

class pymcxray.FileFormat.MCXRayModel.**AtomScreeningModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_HENOC_MAURICE = 0

class pymcxray.FileFormat.MCXRayModel.**MCXRayModel**(*currentModel=None*)
Bases: object

getModel()

setModel(*modelType*)

setModelFromString(*text*)

class pymcxray.FileFormat.MCXRayModel.**MassAbsorptionCoefficientModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_CHANTLER2005 = 3

TYPE_HEINRICH_DATA = 1

TYPE_HEINRICH_PARAMETERIZATION = 2

TYPE_HENKE = 0

class pymcxray.FileFormat.MCXRayModel.**RegionEnergyLossModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_BETHE = 1

TYPE_BETHE_JOY_LUO = 0

TYPE_BETHE_RELATIVISTIC = 2

TYPE_JOY_LUO_KGAUVIN = 3

TYPE_JOY_LUO_MONSEL = 4

class pymcxray.FileFormat.MCXRayModel.**SampleEnergyLossModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_BETHE_JOY_LUO = 0

class pymcxray.FileFormat.MCXRayModel.**SpectrumInterpolationModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_COPY = 0

TYPE_LINEAR = 1

TYPE_LINEAR_DOUBLE = 2

TYPE_SPLINE = 3

TYPE_SPLINE_BATCH = 4

TYPE_SPLINE_POINT = 5

class pymcxray.FileFormat.MCXRayModel.**XRayCSBremsstrahlungModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_BETHE_HEITLER = 0

TYPE_DING = 2

TYPE_GAUVIN = 3

TYPE_KIRKPATRICK_WIEDMAN = 1

class pymcxray.FileFormat.MCXRayModel.**XRayCSCharacteristicModel**(*currentModel=None*)
Bases: *pymcxray.FileFormat.MCXRayModel.MCXRayModel*

TYPE_BOTE2009 = 1

TYPE_CASTANI1982 = 0

pymcxray.FileFormat.MicroscopeParameters module

MCXRay microscope parameters input file.

class pymcxray.FileFormat.MicroscopeParameters.**MicroscopeParameters**
Bases: object

beamCurrent_A

beamDiameter_A

beamEnergy_keV

beamPositionX_A

beamPositionY_A

beamStandardDeviation_A

beamTilt_deg

defaultValues()

detectorAzimuthalAngle_deg

detectorBFHigh_rad

detectorBFLow_rad

detectorChannelWidth_eV

detectorCrystalAtomSymbol

detectorCrystalDistance_cm

detectorCrystalRadius_cm

detectorCrystalThickness_cm

detectorDFHigh_rad

detectorDFLow_rad

detectorDeadLayer_A

detectorDiffusionLenght_A

detectorHAADFHigh_rad

detectorHAADFLow_rad

detectorNoise_eV

detectorPitch_deg

detectorSurfaceQuality

detectorTOA_deg

read(*filepath*)

time_s

version

write(*filepath*)

pymcxray.FileFormat.Models module

MCXRay models file.

class pymcxray.FileFormat.Models.**Models**
Bases: object

getModelList()

modelAtomCollision

modelAtomCrossSection

modelAtomMac

modelSampleEnergyLoss

modelXrayBremsstrahlung

modelXrayCharacteristic

read(*filepath*)

version

write(*filepath*)

pymcxray.FileFormat.Region module

MCXRay region input file.

class pymcxray.FileFormat.Region.**Region**
Bases: object

clear()

createLinesWithVersion()

createLinesWithoutVersion()

extractFromLinesWithVersion(*lines*)

extractFromLinesWithoutVersion(*lines*)

pymcxray.FileFormat.RegionDimensions module

MCXRay region dimensions input file.

class pymcxray.FileFormat.RegionDimensions.**RegionDimensions**(*parameters=None*)
Bases: object

createLineOldVersion()

createLineWithKey()

extractFromLineOldVersion(*line*)

extractFromLinesWithKey(*line*)

getLineFormat()

class pymcxray.FileFormat.RegionDimensions.**RegionDimensionsBox**(*parameters=None*)
Bases: *pymcxray.FileFormat.RegionDimensions.RegionDimensions*

getLineFormat()

maximumX

maximumY

maximumZ

minimumX

minimumY

minimumZ

class pymcxray.FileFormat.RegionDimensions.**RegionDimensionsCylinder**(*parameters=None*)
Bases: *pymcxray.FileFormat.RegionDimensions.RegionDimensionsSphere*

directionX

directionY

directionZ

getLineFormat()

length

class pymcxray.FileFormat.RegionDimensions.**RegionDimensionsSphere**(*parameters=None*)
Bases: *pymcxray.FileFormat.RegionDimensions.RegionDimensions*

getLineFormat()

positionX

positionY

positionZ

radius

pymcxray.FileFormat.RegionDimensions.**createRegionDimensions**(*regionType*)

pymcxray.FileFormat.RegionType module

MCXRay region type input file.

pymcxray.FileFormat.ResultsParameters module

MCXRay ResultsParameters input file.

class pymcxray.FileFormat.ResultsParameters.**ResultsParameters**
Bases: object

defaultValues()

isComputeXrayBremsstrahlung

isComputeXrayCharacteristic

isComputeXrayPhirhoz

isComputeXraySimulatedSpectrum

read(*filepath*)

version

write(*filepath*)

pymcxray.FileFormat.SimulationInputs module

MCXRay simulation inputs file.

class pymcxray.FileFormat.SimulationInputs.**SimulationInputs**
Bases: object

getExtension(*key*)

mapFilename

microsopeFilename

modelFilename

read(*filepath*)

resultParametersFilename

simulationParametersFilename

snrFilename

specimenFilename

title

version

write(*filepath*)

pymcxray.FileFormat.SimulationParameters module

MCXRay simulation parameters input file.

class pymcxray.FileFormat.SimulationParameters.**SimulationParameters**
Bases: object

baseFilename

defaultValues()

elasticCrossSectionScalingFactor

energyChannelWidth_eV

energyLossScalingFactor

numberChannels

numberElectrons

numberFilmsX

numberFilmsY

numberFilmsZ

numberPhotons

numberWindows

read(*filepath*)

spectrumInterpolationModel

version

voxelSimplification

write(*filepath*)

pymcxray.FileFormat.SnrParameters module

MCXRay Snr input file.

class pymcxray.FileFormat.SnrParameters.**SnrParameters**
Bases: object

backgroundEnergyWindowsSize

defaultValues()

energyEnd_keV

energyStart_keV

numberEnergySteps

read(*filepath*)

snrType

spectrumEnergyWindowsSize

write(*filepath*)

pymcxray.FileFormat.Specimen module

MCXRay specimen input file.

class pymcxray.FileFormat.Specimen.**Specimen**
Bases: object

clear()

name

numberRegions

read(*filepath*)

regions

version

write(*filepath*)

pymcxray.FileFormat.Version module

MCXray version information.

class pymcxray.FileFormat.Version.**Version**(*major*, *minor*, *revision*)
Bases: object

fromString(*versionString*)

key = 'Version'

major

minor

readFromFile(*filepath*)

revision

toString()

writeLine(*outputFile*)

pymcxray.FileFormat.testUtilities module

Utilities used for testing this package.

pymcxray.FileFormat.testUtilities.**createTempDataPath**(*path*)

pymcxray.FileFormat.testUtilities.**getSimulationTitles**()

pymcxray.FileFormat.testUtilities.**removeTempDataPath**(*path*)

pymcxray.FileFormat.test_Element module

Tests for module *Element*.

class pymcxray.FileFormat.test_Element.**TestElement**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Element*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_createLineOldVersion()
Tests for method *createLineOldVersion*.

test_createLineWithKey()
Tests for method *createLineOldVersion*.

test_extractFromLineOldVersion()
Tests for method *extractFromLineOldVersion*.

test_extractFromLineWithKey()
Tests for method *extractFromLinesWithKey*.

pymcxray.FileFormat.test_ExportedSpectrum module

Tests for module *ExportedSpectrum*.

class pymcxray.FileFormat.test_ExportedSpectrum.**TestExportedSpectrum**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *moduleName*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_read()
Tests for method *read*.

pymcxray.FileFormat.test_FileReaderWriterTools module

Tests for the module *FileReaderWriterTools*.

class pymcxray.FileFormat.test_FileReaderWriterTools.**TestFileReaderWriterTools**(*methodName='r*
Bases: unittest.case.TestCase

TestCase class for the module *FileReaderWriterTools*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test__reduceAfterDot()
Tests for method *reduceAfterDot*.

pymcxray.FileFormat.test_MCXRayModel module

Tests for module *MCXRayModel*.

class pymcxray.FileFormat.test_MCXRayModel.**TestMCXRayModel**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *MCXRayModel*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_MCXRayModel()
Tests for method *MCXRayModel*.

pymcxray.FileFormat.test_MicroscopeParameters module

pymcxray.FileFormat.test_Models module

pymcxray.FileFormat.test_Region module

Tests for module *Region*.

class pymcxray.FileFormat.test_Region.**TestRegion**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *moduleName*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

getTestRegionLinesWithVersion(*title*, *userMassDensity=True*)

getTestRegionLinesWithoutVersion(*title*, *userMassDensity=True*)

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_createLinesWithVersion()
Tests for method *createLinesWithVersion*.

test_createLinesWithoutVersion()
Tests for method *createLinesWithoutVersion*.

test_extractFromLinesWithVersion()
Tests for method *read*.

test_extractFromLinesWithoutVersion()
Tests for method *read*.

pymcxray.FileFormat.test_RegionDimensions module

Tests for module *RegionDimensions*.

class pymcxray.FileFormat.test_RegionDimensions.**TestRegionDimensions**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *RegionDimensions*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_RegionDimensionsBox_createLineOldVersion()
Tests for class *RegionDimensionsBox*.

test_RegionDimensionsBox_createLineWithKey()
Tests for class *RegionDimensionsBox*.

test_RegionDimensionsBox_extractFromLineOldVersion()
Tests for class *RegionDimensionsBox*.

test_RegionDimensionsBox_extractFromLinesWithKey()
Tests for class *RegionDimensionsBox*.

test_RegionDimensionsCylinder_createLineOldVersion()
Tests for class *RegionDimensionsCylinder*.

test_RegionDimensionsCylinder_createLineWithKey()
Tests for class *RegionDimensionsCylinder*.

test_RegionDimensionsCylinder_extractFromLineOldVersion()
Tests for class *RegionDimensionsCylinder*.

test_RegionDimensionsCylinder_extractFromLinesWithKey()
Tests for class *RegionDimensionsCylinder*.

test_RegionDimensionsSphere_createLineOldVersion()
Tests for class *RegionDimensionsSphere*.

test_RegionDimensionsSphere_createLineWithKey()
Tests for class *RegionDimensionsSphere*.

test_RegionDimensionsSphere_extractFromLineOldVersion()
Tests for class *RegionDimensionsSphere*.

test_RegionDimensionsSphere_extractFromLinesWithKey()
Tests for class *RegionDimensionsSphere*.

test_createRegionDimensions()
Tests for method *createRegionDimensions*.

pymcxray.FileFormat.test_RegionType module

Test for module *RegionType*.

class pymcxray.FileFormat.test_RegionType.**TestRegionType**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *RegionType*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_Constants()
Tests for method *Constants*.

pymcxray.FileFormat.test_ResultsParameters module

pymcxray.FileFormat.test_SimulationInputs module

pymcxray.FileFormat.test_SimulationParameters module

pymcxray.FileFormat.test_SnrParameters module

Tests for the module *SnrParameters*.

class pymcxray.FileFormat.test_SnrParameters.**TestSnrParameters**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *SnrParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

getSnrParametersReference(*title*)

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test__createKeys()
Tests for method *_createKeys*.

test_read()
Tests for method *read*.

test_write()
Tests for method *write*.

pymcxray.FileFormat.test_Specimen module

pymcxray.FileFormat.test_Version module

Tests for module *Version*.

class pymcxray.FileFormat.test_Version.**TestVersion**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Version*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_VersionConstants()
Tests for method *VersionConstants*.

test_comparison()
Test comparison operation on Version class.

test_fromString()
Tests for method *fromString*.

test_readFromFile()
Tests for method *readFromFile*.

test_readFromFile_BadFile()
Tests for method *readFromFile*.

test_toString()
Tests for method *toString*.

test_writeLine()
Tests for method *writeLine*.

pymcxray.FileFormat.tests module

Module contents

pymcxray.serialization package

Submodules

pymcxray.serialization.SerializationH5py module

class pymcxray.serialization.SerializationH5py.**SerializationNumpy**(*filename=None*, *verbose=True*)
Bases: pymcxray.serialization._Serialization._Serialization

load()

save(*serializedData*)

pymcxray.serialization.SerializationNumpy module

class pymcxray.serialization.SerializationNumpy.**SerializationNumpy**(*filename=None*, *verbose=True*)

Bases: pymcxray.serialization._Serialization._Serialization

load()

save(*serializedData*)

class pymcxray.serialization.SerializationNumpy.**SerializationNumpyNPY**(*filename=None*, *verbose=True*)

Bases: *pymcxray.serialization.SerializationNumpy.SerializationNumpy*

load()

save(*data*)

class pymcxray.serialization.SerializationNumpy.**SerializationNumpyNPZ**(*filename=None*, *verbose=True*)

Bases: *pymcxray.serialization.SerializationNumpy.SerializationNumpy*

load()

save(*data*)

class pymcxray.serialization.SerializationNumpy.**SerializationNumpyTxt**(*filename=None*, *verbose=True*)

Bases: *pymcxray.serialization.SerializationNumpy.SerializationNumpy*

load()

save(*data*)

class pymcxray.serialization.SerializationNumpy.**SerializationNumpyTxtGz**(*filename=None*, *verbose=True*)

Bases: *pymcxray.serialization.SerializationNumpy.SerializationNumpyTxt*

pymcxray.serialization.SerializationPickle module

class pymcxray.serialization.SerializationPickle.**SerializationPickle**(*filename=None*, *verbose=True*)

Bases: pymcxray.serialization._Serialization._Serialization

KEY_FILE_VERSION = 'fileVersion'

KEY_SERIALIZED_DATA = 'serializedData'

load()

save(*serializedData*)

pymcxray.serialization.test_Serialization module

pymcxray.serialization.test_SerializationH5py module

class pymcxray.serialization.test_SerializationH5py.**TestSerializationH5py**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

testSkeleton()

pymcxray.serialization.test_SerializationNumpy module

class pymcxray.serialization.test_SerializationNumpy.**TestSerializationNumpy**(*methodName='runTe*
Bases: unittest.case.TestCase

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

testSkeleton()

test_loadSaveSerializationNumpy()

test_loadSaveSerializationNumpyNPY()

test_loadSaveSerializationNumpyNPZ()

test_loadSaveSerializationNumpyTxt()

test_loadSaveSerializationNumpyTxtGz()

pymcxray.serialization.test_SerializationPickle module

class pymcxray.serialization.test_SerializationPickle.**TestSerialization**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

testSkeleton()

test_loadSave()

test_version()

pymcxray.serialization.tests module

Regression testing for the project.

Module contents

pymcxray.tests package

Submodules

pymcxray.tests.test_pymcxray module

test_pymcxray

Tests for *pymcxray* module.

class pymcxray.tests.test_pymcxray.**TestPymcxray**(*methodName='runTest'*)
Bases: unittest.case.TestCase

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Hook method for setting up the test fixture before exercising it.

tearDown()
Hook method for deconstructing the test fixture after testing it.

test_000_something()

Module contents

8.1.2 Submodules

8.1.3 pymcxray.AnalyzeNumberBackgroundWindows module

Analyze the number of background windows on the x-ray spectrum.

class pymcxray.AnalyzeNumberBackgroundWindows.**AnalyzeNumberBackgroundWindows**
Bases: object

plotData()

plotDifference()

readData()

pymcxray.AnalyzeNumberBackgroundWindows.**run**()

8.1.4 pymcxray.AtomData module

MCXRay atom data.

pymcxray.AtomData.**getAtomSymbol**(*atomic_number*)

pymcxray.AtomData.**getAtomicNumber**(*symbol*)

pymcxray.AtomData.**getIonizationEnergy_keV**(*shell*, *element*)

pymcxray.AtomData.**getMassDensity_g_cm3**(*symbol*)

pymcxray.AtomData.**getShellList**()

pymcxray.AtomData.**getXRayEnergy_keV**(*line*, *element*)

pymcxray.AtomData.**get_atomic_weight_g_mol**(*atomic_number*)

pymcxray.AtomData.**run**()

8.1.5 pymcxray.BatchFile module

MCXRay batch file creator.

class pymcxray.BatchFile.**BatchFile**(*name*, *numberFiles=1*)

Bases: object

addSimulationName(*simulationFilename*)

removePreviousFiles(*path*)

write(*path*)

8.1.6 pymcxray.BatchFileConsole module

MCXRay console batch file creator.

class pymcxray.BatchFileConsole.**BatchFileConsole**(*name*, *programName*, *numberFiles=1*)

Bases: object

The batch file is responsible to create the simulation structure with a copy of mcxray program.

One important parameter to set is the *numberFiles*, this is the number of batch files generated and that can be run in parallel. For maximum efficiency it should be set as the number of logical processors minus 1 or 2. For example, on a computer with 12 logical processors, the *numberFiles* should be set at 10.

Parameters

- **name** (*str*) – Basename used for the batch files
- **programName** (*str*) – Name of the executable to add in the batch file
- **numberFiles** (*int*) – Number of batch files to generate and possibly to run in parallel

addSimulationName(*simulationFilename*)

Add a simulation in the simulation list.

Parameters **simulationFilename** (*str*) – File path of the simulation added

write(*path*)

Write the batch files for all simulations in the simulation list.

Parameters **path** (*str*) – Path where the batch files are written.

8.1.7 pymcxray.ComparisonModels module

Comparison of the models used by MCXray.

class pymcxray.ComparisonModels.**ComparisonModels**(*dataPath*)
Bases: object

graphicsEnergyLoss()

graphicsIonizationCrossSection()

graphicsXrayCrossSectionBremstrahlung()

graphicsXrayMassAbsorptionCoefficient()

pymcxray.ComparisonModels.**runVersion1_2_3**()

pymcxray.ComparisonModels.**runVersion1_4_0**()

pymcxray.ComparisonModels.**runVersion1_4_1**()

8.1.8 pymcxray.DebugSimulatedSpectrum module

Debug the simulated spectrum implementation in mcxray.

class pymcxray.DebugSimulatedSpectrum.**DebugSimulatedSpectrum**
Bases: object

runRegion()

runSpecimen()

pymcxray.DebugSimulatedSpectrum.**run**()

8.1.9 pymcxray.ElementProperties module

pymcxray.ElementProperties.**computeAtomicDensity_atom_cm3**(*massDensity_g_cm3*, *atomicMass_g_mol*)

Compute the atomic density.

$$n_i = \frac{N_A \rho_i}{A_i}$$

where

- n_i is the atomic density in atoms/cm^3
- N_A is the Avogadro number in $\text{atoms}/mole$
- ρ_i is the mass density in g/cm^3
- A_i is the atomic mass in $g/mole$

Parameters

- **massDensity_g_cm3** (*float*) –
- **atomicMass_g_mol** (*float*) –

pymcxray.ElementProperties.**g_FermiEnergy = [1.0, 1.0, 4.7, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1**
Fermi energy of element in atomic number order.

For element H to Lr (1-103). From: CASINO source code, DOS version.

Todo: Add units.

pymcxray.ElementProperties.**g_atomicMass_g_mol = [1.0079, 4.0026, 6.941, 9.01218, 10.81, 12**
Atomic weight of element in atomic number order.

For element H to Sg (1-106).

Unit $g/mole$.

From: Tableau periodique des elements, Sargent-Welch scientifique Canada Limitee.

pymcxray.ElementProperties.**g_kFermi = [70000000.0, 70000000.0, 110000000.0, 70000000.0, 700**
Fermi wavelength of element in atomic number order.

For element H to Lr (1–103). From: CASINO source code, DOS version.

Todo: Add units.

pymcxray.ElementProperties.**g_massDensity_g_cm3 = [0.0899, 0.1787, 0.53, 1.85, 2.34, 2.62, 1**
Mass density of element in atomic number order.

For element H to Cm (1-96).

In $g/cm3$.

From: Tableau periodique des elements, Sargent-Welch scientifique Canada Limitee.

Note: Element Z = 85 and 87 set to 1 for the calculation.

pymcxray.ElementProperties.**g_plasmonEnergy = [15.0, 15.0, 7.1, 18.7, 22.7, 15.0, 15.0, 15.0**
Plasmon energy of element in atomic number order.

For element H to Lr (1-103). From: CASINO source code, DOS version.

Todo: Add units.

pymcxray.ElementProperties.**getAtomicMass_g_mol**(*atomicNumber*)

pymcxray.ElementProperties.**getAtomicNumber**(*atomicNumber=None*, *name=None*, *symbol=None*)

pymcxray.ElementProperties.**getAtomicNumberByName**(*name*)

pymcxray.ElementProperties.**getAtomicNumberBySymbol**(*symbol*)

pymcxray.ElementProperties.**getFermiEnergy_eV**(*atomicNumber*)

pymcxray.ElementProperties.**getKFermi_eV**(*atomicNumber*)

pymcxray.ElementProperties.**getKRatioCorrection**(*atomicNumber*)
Get the constant k ratio correction needed by the mean ionization potential from the atomic number.

Parameters **atomic_number** (*int*) – Atomic number

pymcxray.ElementProperties.**getKRatioCorrectionMonsel**(*atomicNumber*, *workFunction_keV*)
/// K value as defined by Monsel. /// Used in DE/DS calculation. Casino uses K Gauvin,but for low energy, /// JR Lowney says that this one is more appropriate (and by experience, /// it is effectively better for the secondary yield). /// <p> NOTE : Depends on J (ionisation potential). So it must already be calculated before. /// @param element Element for whom we want to calculate the K value. /// @return The K value of the element passed in argument

pymcxray.ElementProperties.**getMassDensity_g_cm3**(*atomicNumber*)

pymcxray.ElementProperties.**getMeanIonizationEnergy_eV**(*atomic_number*)
Get the mean ionization potential from the atomic number.

In eV.

Parameters **atomic_number** (*int*) – Atomic number

pymcxray.ElementProperties.**getName**(*atomicNumber*)

pymcxray.ElementProperties.**getPlasmonEnergy_eV**(*atomicNumber*)

pymcxray.ElementProperties.**getSymbol**(*atomicNumber*)

pymcxray.ElementProperties.**run**()

pymcxray.ElementProperties.**runAtomicNumberSymbol**()

8.1.10 pymcxray.Simulation module

MCXRay simulation parameters.

class pymcxray.Simulation.**Layer**(*elements*, *thickness_nm*, *mass_density_g_cm3=None*)
Bases: object

class pymcxray.Simulation.**Simulation**(*overwrite=True*)
Bases: object

basename

beamDiameter_nm

beamPosition_nm

beamTilt_deg

createSimulationFiles(*path*, *simulationPath*, *hdf5_group*)

current_A

detectorAzimuthalAngle_deg

detectorChannelWidth_eV

detectorCrystalDistance_cm

detectorCrystalRadius_cm

detectorCrystalThickness_cm

detectorNoise_eV

elasticCrossSectionScalingFactor

energyLossScalingFactor

energy_keV

filename

generateBaseFilename()

getFilenameSuffixes()

getParameters()

getProgramVersionFilepath(*simulationPath*)

isDone(*simulationPath*, *hdf5_group=None*)

modelXrayBremsstrahlung

name

numberContinuumWindows

numberElectrons

numberEnergyWindows

numberLayersX

numberLayersY

numberLayersZ

numberPhotons

removeInputsFiles()

resultsBasename

setParameters(*parameters*)

solidAngle_sr

spectrumInterpolationModel

takeOffAngle_deg

time_s

pymcxray.Simulation.**computeWeightFraction**(*atomicNumberRef*, *atomicWeights*)

pymcxray.Simulation.**createAlloyBoxInSubstrate**(*elementsParticle*, *elementsSubstrate*, *boxParameters_nm*)

pymcxray.Simulation.**createAlloyBoxInThinFilm**(*elementsParticle*, *atomicNumberSubstrate*, *boxParameters_nm*, *filmThickness_nm*)

pymcxray.Simulation.**createAlloyBoxInVaccuum**(*elements*, *boxParameters_nm*)

pymcxray.Simulation.**createAlloyBulkSample**(*elements*, *sampleName=None*)

pymcxray.Simulation.**createAlloyFilmOverSubstrate**(*film_elements*, *substrate_elements*, *film_thickness_nm=10.0*, *film_mass_density_g_cm3=None*, *substrate_mass_density_g_cm3=None*)

pymcxray.Simulation.**createAlloyMultiVerticalLayer**(*elementsLayers*, *layerWidths_nm*)

pymcxray.Simulation.**createAlloyParticleInSubstrate**(*elementsParticle*, *atomicNumberSubstrate*, *particleRadius_nm*, *particlePositionZ_nm=None*)

pymcxray.Simulation.**createAlloyParticleInThinFilm**(*elementsParticle*, *atomicNumberSubstrate*, *particleRadius_nm*, *filmThickness_nm*, *particlePositionZ_nm=None*)

pymcxray.Simulation.**createAlloyThinFilm**(*elements*, *filmThickness_nm*)

pymcxray.Simulation.**createAlloyThinFilm2**(*elements*, *filmThickness_nm*)

pymcxray.Simulation.**createBoxFeatureInSubstrate**(*feature_elements*, *substrate_elements*, *depth_nm*, *width_nm*)

pymcxray.Simulation.**createFilmInSubstrate**(*atomicNumberFilm*, *atomicNumberSubstrate*, *filmThickness_nm*, *filmTopPositionZ_nm*)

pymcxray.Simulation.**createFilmOverSubstrate**(*atomicNumberFilm*, *atomicNumberSubstrate*, *filmThickness_nm=10.0*)

pymcxray.Simulation.**createParticleInSubstrate**(*atomicNumberParticle*, *atomicNumberSubstrate*, *particleRadius_nm*, *particlePositionZ_nm=None*)

pymcxray.Simulation.**createParticleOnFilm**(*atomicNumberParticle*, *atomicNumberSubstrate*, *particleDiameter_nm*, *filmThiskness_nm*)

pymcxray.Simulation.**createParticleOnSubstrate**(*atomicNumberParticle*, *atomicNumberSubstrate*, *particleDiameter_nm*)

pymcxray.Simulation.**createPhirhozSpecimens**(*atomicNumberTracer*, *atomicNumberMatrix*, *tracerThickness_nm*, *maximumThickness_nm*)

pymcxray.Simulation.**createPureBulkSample**(*atomicNumber*)

pymcxray.Simulation.**create_cnt_sample**(*body_elements*, *cnt_length_nm=1000.0*, *cnt_outside_diameter_nm=100.0*, *cnt_inside_diameter_nm=50.0*, *particle_diameter_nm=5.0*)

pymcxray.Simulation.**create_multi_horizontal_layer**(*substrate_elements*, *layers*, *substrate_mass_density_g_cm3=None*)

Create a horizontal multi layer sample.

The substrate is the first region created. The other region are each of the element in the *layers*.

Parameters

- **substrate_elements** – list of atomic number and weight fraction pair for the composition of the substrate
- **layers** –
- **substrate_mass_density_g_cm3** –

Returns

pymcxray.Simulation.**create_weight_fractions**(*weight_fraction_step*, *number_elements*)

8.1.11 pymcxray.SimulationsParameters module

Simulations parameters

class pymcxray.SimulationsParameters.**SimulationsParameters**

Bases: dict

addCompute(*parameterKey*)

addFixed(*parameterKey*, *value*)

addVaried(*parameterKey*, *values*)

computeNumberXrays(*experiment*)

computeNumberXraysFilepath

fixedParameters

getAllSimulationParameters()

getVariedParameterLabels()

variedParameters

class pymcxray.SimulationsParameters.**SimulationsParametersFixed**

Bases: object

addExperiment(*experiment*)

getAllSimulationParameters()

8.1.12 pymcxray.Testings module

8.1.13 pymcxray.mcxray module

Base module to create and analuyze MCXRay simulations.

8.1.14 pymcxray.multipleloop module

This module provides a tool for handling computer experiments with of a set of input parameters, where each input parameter is varied in a prescribed fashion.

In short, the parameters are held in a dictionary where the keys are the names of the parameters and the values are the numerical, string or other values of the parameters. The value can take on multiple values: e.g., an integer parameter 'a' can have values -1, 1 and 10. Similarly, a string parameter 'method' can have values 'Newton' and 'Bisection'. The module will generate all combination of all parameters and values, which in the mentioned example will be (-1, 'Newton'), (1, 'Newton'), (10, 'Newton'), (-1, 'Bisection'), (1, 'Bisection'), and (10, 'Bisection'). Particular combination of values can easily be removed.

The usage and implementation of the module are documented in the book "Python Scripting for Computational Science" (H. P. Langtangen, Springer, 2009), Chapter 12.1.

pymcxray.multipleloop.**combine**(*prm_values*)

Compute the combination of all parameter values in the prm_values (nested) list. Main function in this module.

param prm_values: nested list `(parameter_name, list_of_parameter_values)` or dictionary `prm_values[parameter_name] = list_of_parameter_values`. return: (all, names, varied) where

- all contains all combinations (experiments) all[i] is the list of individual parameter values in experiment no i
- names contains a list of all parameter names
- varied holds a list of parameter names that are varied (i.e. where there is more than one value of the parameter, the rest of the parameters have fixed values)

Code example:

```
>>> dx = array([1.0/2**k for k in range(2,5)])
>>> dt = 3*dx;  dt = dt[:-1]
>>> p = {'dx': dx, 'dt': dt}
>>> p
{'dt': [ 0.75 , 0.375,], 'dx': [ 0.25  , 0.125 , 0.0625,]}
>>> all, names, varied = combine(p)
>>> all
[[0.75, 0.25], [0.375, 0.25], [0.75, 0.125], [0.375, 0.125],
 [0.75, 0.0625], [0.375, 0.0625]]
```

8.1.15 pymcxray.pymcxray module

8.1.16 pymcxray.test_AtomData module

Tests for the module *AtomData*.

class pymcxray.test_AtomData.**TestAtomData**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *AtomData*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testConstants()
First test to check if the testcase is working with the testing framework.

testSkeleton()
First test to check if the testcase is working with the testing framework.

8.1.17 pymcxray.test_BatchFileConsole module

Tests for the module *BatchFileConsole*.

class pymcxray.test_BatchFileConsole.**TestBatchFileConsole**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *BatchFileConsole*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

8.1.18 pymcxray.test_ComparisonModels module

Tests for the module *ComparisonModels*.

class pymcxray.test_ComparisonModels.**TestComparisonModels**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *ComparisonModels*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

8.1.19 pymcxray.test_Simulation module

Tests for the module *Simulation*.

class pymcxray.test_Simulation.**TestSimulation**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *Simulation*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

test_create_multi_horizontal_layer()
Test the *create_multi_horizontal_layer* method.

test_create_weight_fractions()
First test to check if the testcase is working with the testing framework.

8.1.20 pymcxray.test_SimulationsParameters module

Tests for the module *SimulationsParameters*.

class pymcxray.test_SimulationsParameters.**TestSimulationsParameters**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *SimulationsParameters*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

8.1.21 pymcxray.test_mcxray module

Tests for the module *mcxray*.

class pymcxray.test_mcxray.**Testmcxray**(*methodName='runTest'*)
Bases: unittest.case.TestCase

TestCase class for the module *mcxray*.

Create an instance of the class that will use the named test method when executed. Raises a ValueError if the instance does not have a method with the specified name.

setUp()
Setup method.

tearDown()
Teardown method.

testSkeleton()
First test to check if the testcase is working with the testing framework.

8.1.22 pymcxray.tests module

8.1.23 Module contents

pymcxray.**create_path**(*path*)
Create a path from the input string if does not exists.

Does not try to distinct between file and directory in the input string. path = "dir1/filename.ext" => "dir1/filename.ext/" where the new directory "filename.ext" is created.

Parameters **path** (*str*) – The path input string.

Returns The path with the path separator at the end

Return type str

pymcxray.**find_all_files**(*root*, *patterns='*'*, *ignore_path_patterns=''*, *ignore_name_patterns=''*, *single_level=False*, *yield_folders=False*)
Find all files in a root folder.

From Python Cookbook section 2.16 pages 88–90

Parameters

- **root** –
- **patterns** –
- **ignore_path_patterns** –
- **ignore_name_patterns** –

- **single_level** –
- **yield_folders** –

Returns

pymcxray.**get_current_module_path**(*module_path*, *relative_path=''*)

Extract the current module path and combine it with the relative path and return it.

Parameters

- **module_path** (*str*) – Pass the *__FILE__* python keyword for this parameter
- **relative_path** (*str*) – The relative path to combine with the module path

Returns The path obtained when combine the module path and relative path

Return type str

pymcxray.**get_mcxray_archive_name**(*configuration_file_path*, *default=None*)

Read the MCXRay archive name in the configuration file. This option allows to choose which version of MCXRay to use for the simulations.

The configuration file need to have this entry in the section [Paths]:

```
[Paths]
mcxrayArchiveName=2016-04-11_11h41m28s_MCXRay_v1.6.6.0.zip
```

Parameters

- **configuration_file_path** (*str*) – The file path of the configuration file
- **default** (*str*) – Default value to use if the entry is not found

Returns The MCXRay archive name

Return type str

pymcxray.**get_mcxray_archive_path**(*configuration_file_path*, *relative_path=''*)

Read the MCXRay archive path in the configuration file.

The configuration file need to have this entry in the section [Paths]:

```
[Paths]
mcxrayArchivePath=D:\Dropbox\hdemers\professional\softwareRelease\mcxray
```

Parameters

- **configuration_file_path** (*str*) – The file path of the configuration file
- **relative_path** (*str*) – Relative path to add to the path read in the configuration file

Returns Path where the mcxray archive can be found.

Return type str

pymcxray.**get_mcxray_program_name**(*configuration_file_path*, *default=None*)

Read the MCXRay program name in the configuration file.

This option specify which executable to use in the script. The *console_mcxray_x64.exe* should be OK for most situation. If you have a 32-bit system, you have to use *console_mcxray.exe* (32-bit version).

The configuration file need to have this entry in the section [Paths]:

```
[Paths]
mcxrayProgramName=console_mcxray_x64.exe
```

Parameters

- **configuration_file_path** (*str*) – The fule path of the configuration file
- **default** (*str*) – Default value to use if the entry is not found

Returns The MCXRay program name

Return type str

pymcxray.**get_mcxray_program_path**(*configuration_file_path*, *relative_path=''*)

Read the MCXRay program path in the configuration file.

The configuration file need to have this entry in the section [Paths]:

```
[Paths]
mcxrayProgramPath=C:\hdemers\codings\devcasino
```

Parameters

- **configuration_file_path** (*str*) – The file path of the configuration file
- **relative_path** (*str*) – Relative path to add to the path read in the configuration file

Returns Path where the mcxray program can be found.

Return type str

Deprecated since version 0.1.

See also:

function *pymcxray.get_mcxray_archive_path()*

pymcxray.**get_results_mcgill_path**(*configuration_file_path*, *relative_path=''*)

Read the results path for McGill in the configuration file. The results path read in the configuration file is combine with the *relative_path* and return.

The configuration file need to have this entry in the section [Paths]:

```
[Paths]
resultsMcGillPath=D:\Dropbox\hdemers\professional\results\simulations
```

Parameters

- **configuration_file_path** (*str*) – The file path of the configuration file
- **relative_path** (*str*) – Relative path to add to the path read in the configuration file

Returns Path where the simulation input and results will be written

Return type str

pymcxray.**read_value_from_configuration_file**(*configuration_file*, *section_name*, *key_name*, *default=None*)

Read a value from an entry in a section from a configuration file.

Parameters

- **configuration_file** (*str*) – The file path of the configuration file
- **section_name** (*str*) – Name of the section
- **key_name** (*str*) – Name of the entry to read
- **default** – Default value of the entry if not found

Returns The value read or default value

Return type str

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

A

B

C

D

E

F

G

I

K

L

M

N

P

R

S

T

Y

Z