pyPENELOPEtools Documentation

Release 0+untagged.56.ga5324f8

Philippe Pinard and Hendrix Demers

Dec 26, 2018

Contents

pyPENELOPEtools is an open-source software to facilitate the use of the Monte Carlo code PENELOPE and its main programs such as PENEPMA. It is a programming interface to setup, run and analyze Monte Carlo simulations. Most of the code was adapted from pyPENELOPE, but with the goal to facilitate the integration with pyMonteCarlo.

Warning: **pyPENELOPEtools** does not contain executables of PENELOPE or any of its main programs. It also does not provide any code to execute these programs. It provides tools to create the input file(s) (e.g. `.in`) for the main programs and to parse the output file(s) from the simulations (e.g. `.dat`).

Contents:

CHAPTER 1

Installation

1.1 Stable release

Requires Python 3.5+.

```
$ pip install pypenelopetools
```

1.2 Development

Clone the Github repository, either directly or after forking:

```
$ git clone https://github.com/pymontecarlo/pypenelopetools.git
```

Install the project in editable mode:

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

Run the unit tests to make sure everything works properly:

```
$ python setup.py nosetests
```

CHAPTER 2

Tutorial

2.1 PENEPMA

In this tutorial, we will show how to create the 2nd example distributed with PENEPMA (`epma2.in`). It consists of a couple of copper on one side and iron on the other. The materials, geometry (`.geo`) and input file (`.in`) will all be created and the results will be analyzed using **pyPENELOPEtools**. The complete code can also be found in the unit tests.

The tutorial assumes that:

- **pyPENELOPEtools** is installed.
- PENELOPE and PENEPMA are installed,
- PENELOPE is located in a folder called `/penelope` and the material executable in `/penelope/pendbase/material`,
- PENEPMA is located in a folder called `/penepma` and the executable in `/penepma/bin/penepma`, and
- Simulation files are placed in the folder `/simulation/epma2`.

2.1.1 Materials

First, let's create two *`Material`* definitions.

```
from pypenelopetools.material import Material
material_cu = Material('Cu', {29: 1.0}, 8.9)
material_fe = Material('Fe', {26: 1.0}, 7.874)
```

To create the actual material files (`.mat`), we first need to export the input file for the `material` executable.

```
with open('/penelope/pendbase/Cu.mat.in', 'w') as fp:
    material_cu.write_input(fp)
```

(continues on next page)

(continued from previous page)

```
with open('/penelope/pendbase/Fe.mat.in', 'w') as fp:
    material_fe.write_input(fp)
```

From a command prompt in the folder `/penelope/pendbase`, run the following commands to create the two material files (`Cu.mat` and `Fe.mat`):

```
./material < Cu.mat.in
./material < Fe.mat.in
```

The material files can now be copied in `/simulation/epma2`.

2.1.2 Geometry

PENGEOM geometries consist of surfaces and modules. A *SurfaceImplicit* can be arbitrarily defined using quadratic equations, but **pyPENELOPEtools** also provides utility functions for simple cases. A *Module* group different surfaces and other modules together forming an object with an associated *Material*.

For this example, the couple geometry consists of a cylinder with three planes: one defining the top surface, one for the bottom surface and a dividing plane in the middle. Let's create these surfaces using the utility functions:

```
from pypenelopetools.pengeom.surface import xplane, zplane, cylinder
surface_top = zplane(0.0)
surface_bottom = zplane(-0.1)
surface_cylinder = cylinder(1.0)
surface_divider = xplane(0.0)
```

All the dimensions are in centimeters, so the code above creates a cylinder with a radius of 1cm and a depth of 100mm. The couple is divided with a plane perpendicular to the x-axis.

Let's now construct our two modules, corresponding to the right and left halves of the couple geometry:

```
from pypenelopetools.pengeom.module import Module, SidePointer
module_right = Module(material_cu, 'Right half of the sample')
module_right.add_surface(surface_top, SidePointer.NEGATIVE)
module_right.add_surface(surface_bottom, SidePointer.POSITIVE)
module_right.add_surface(surface_cylinder, SidePointer.NEGATIVE)
module_right.add_surface(surface_divider, SidePointer.POSITIVE)

module_left = Module(material_fe, 'Left half of the sample')
module_left.add_surface(surface_top, SidePointer.NEGATIVE)
module_left.add_surface(surface_bottom, SidePointer.POSITIVE)
module_left.add_surface(surface_cylinder, SidePointer.NEGATIVE)
module_left.add_module(module_right)
```

The side pointer specifies which side of the surface forms the enclosed module. Note that the right module is added to the left module. This tells PENGEOM that the two modules are touching each other and share a common interface.

The two modules are put together into a *Geometry*:

```
geometry = Geometry('Cylindrical homogeneous foil')
geometry.add_module(module_right)
geometry.add_module(module_left)
```

Finally, the geometry can be saved as a `.geo` file. The filename should be remembered since it will be needed to construct the input file.

```
geofilename = 'epma2.geo'
with open('/simulation/epma2/' + geofilename, 'w') as fp:
    index_lookup = geometry.write(fp)
```

The `write()` method returns an important value, a lookup table with the indexes that were associated with the modules and materials. These indexes will be needed to construct the input file.

Important: PENELOPE, PENGEOM and PENEPMA rely on indexes and properly ordered lines in the input file to identify the right materials or modules. This strategy does not work very well with object oriented programming (i.e. classes, objects, etc.), so **pyPENELOPEtools** uses an *index lookup* to try to link the two approaches. This way the user does not have to remember all the indexes, although the index-based approach can still be used if needed.

2.1.3 Input

Let's now create the input file (`.in`) for the simulation. All the simulation parameters are stored in a *PenepmaInput <pypenelopetools.penepma.input.PenepmaInput>* object.

```
from pypenelopetools.penepma.input import PenepmaInput
input = PenepmaInput()
```

The *PenepmaInput <pypenelopetools.penepma.input.PenepmaInput>* object contains all the keywords available for a PENEPMA simulation. Have a look at the documentation to see which ones are available and what are their parameters.

First, we setup the title and electron beam definition: a 15kV electron beam with an initial position at `x=20um` and `z=1cm` pointing downwards.

```
input.TITLE.set('A CU-Fe couple')
input.SENERG.set(15e3)
input.SPOSIT.set(2e-5, 0.0, 1.0)
input.SDIREC.set(180, 0.0)
input.SAPERT.set(0.0)
```

Secondly, the materials and their simulation parameters. We use the *index_lookup* to find the index of the materials and the *filename* property to find the filename of each material.

```
input.materials.add(index_lookup[material_cu], material_cu.filename, 1e3, 1e3, 1e3, 0.
↪2, 0.2, 1e3, 1e3)
input.materials.add(index_lookup[material_fe], material_fe.filename, 1e3, 1e3, 1e3, 0.
↪2, 0.2, 1e3, 1e3)
```

Thirdly, the geometry definition and the maximum step length parameters. Again here we use the *index_lookup* to find the index of the modules.

```
input.GEOMFN.set(geofilename)
input.DSMAX.add(index_lookup[module_right], 1e-4)
input.DSMAX.add(index_lookup[module_left], 1e-4)
```

Fourthly, the interaction forcings and splitting parameters. This is a copy of the parameters in the PENEPMA example.

```
input.IFORCE.add(index_lookup[module_right], 1, 4, -5, 0.9, 1.0)
input.IFORCE.add(index_lookup[module_right], 1, 5, -250, 0.9, 1.0)
input.IFORCE.add(index_lookup[module_right], 2, 2, 10, 1e-3, 1.0)
```

(continues on next page)

(continued from previous page)

```
input.IFORCE.add(index_lookup[module_right], 2, 3, 10, 1e-3, 1.0)
input.IFORCE.add(index_lookup[module_left], 1, 4, -5, 0.9, 1.0)
input.IFORCE.add(index_lookup[module_left], 1, 5, -7, 0.9, 1.0)
input.IFORCE.add(index_lookup[module_left], 2, 2, 10, 1e-3, 1.0)
input.IFORCE.add(index_lookup[module_left], 2, 3, 10, 1e-3, 1.0)

input.IBRSPL.add(index_lookup[module_right], 2)
input.IBRSPL.add(index_lookup[module_left], 2)

input.IXRSPL.add(index_lookup[module_right], 2)
input.IXRSPL.add(index_lookup[module_left], 2)
```

Fifthly, the emerging particle distributions, photon detectors and spatial distribution.

```
import pyxray
from pypenelopetools.penepma.utils import convert_xrayline_to_izs1s200

input.NBE.set(0, 0, 300)
input.NBANGL.set(45, 30)

input.photon_detectors.add(0, 90, 0, 360, 0, 0.0, 0.0, 1000)
input.photon_detectors.add(5, 15, 0, 360, 0, 0.0, 0.0, 1000)
input.photon_detectors.add(15, 25, 0, 360, 0, 0.0, 0.0, 1000)
input.photon_detectors.add(25, 35, 0, 360, 0, 0.0, 0.0, 1000)
input.photon_detectors.add(35, 45, 0, 360, 0, 0.0, 0.0, 1000)
input.photon_detectors.add(45, 55, 0, 360, 0, 0.0, 0.0, 1000)
input.photon_detectors.add(55, 65, 0, 360, 0, 0.0, 0.0, 1000)
input.photon_detectors.add(65, 75, 0, 360, 0, 0.0, 0.0, 1000)
input.photon_detectors.add(75, 85, 0, 360, 0, 0.0, 0.0, 1000)

input.GRIDX.set(-1e-5, 5e-5, 60)
input.GRIDY.set(-3e-5, 3e-5, 60)
input.GRIDZ.set(-6e-5, 0.0, 60)
input.XRLINE.add(convert_xrayline_to_izs1s200(pyxray.xray_line(26, 'Ka2')))
input.XRLINE.add(convert_xrayline_to_izs1s200(pyxray.xray_line(29, 'Ka2')))
```

Important: The theta angles of a photon detector are defined as angles from the positive z-axis. This is different than the take-off angle usually used in microanalysis. For a take-off angle of 30deg, theta would be 60deg.

Hint: Use `convert_xrayline_to_izs1s200` to convert `XrayLine` from pyxray library to PENELOPE's ILB(4) notation.

Finally, the job properties. The first random seed is negative to force the generation of a random seed every time a simulation is started. We also use the conversion function from `XrayLine` to ILB(4) notation for the relative uncertainty termination `REFLIN`. The simulation will terminate if the relative uncertainty (3-sigma) on the Fe Ka2 total characteristic intensity detected by the first detector is less than 0.15%.

```
input.RESUME.set('dump2.dat')
input.DUMPTO.set('dump2.dat')
input.DUMPP.set(60)

input.RSEED.set(-10, 1)
```

(continues on next page)

(continued from previous page)

```
input.REFLIN.set(convert_xrayline_to_izs1s200(pyxray.xray_line(26, 'Ka2')), 1, 1.5e-3)
input.NSIMSH.set(2e9)
input.TIME.set(2e9)
```

That completes all the parameters for the simulation. The last step is to save them as a `.in` file.

```
with open('/simulation/epma2/epma2.in', 'w') as fp:
    input.write(fp)
```

From the `/simulation/epma2` folder, the simulation can be run using the PENEPMA main program.

```
penepma < epma2.in
```

2.1.4 Results

After the simulation has terminated or even after the first dump, the results can be read using **pyPENELOPEtools**. Each result has its own class that follows the same interface. The next lines describe how to read the backscatter electron coefficient and plot the photon spectrum of the first detector. For more information about the results, please refer to the *API* section.

To read the simulation time, we create a *`PenepmaResult`* object and call its `read_directory()` method.

```
from pypenelopetools.penepma.results import PenepmaResult
result = PenepmaResult()
result.read_directory('/simulation/epma2')
```

The backscatter electron coefficient is stored in the attribute `upbound_fraction` which returns a `ufloat` object. `ufloat` objects are from the library uncertainties which is designed to handle values and their uncertainties. Here is how to extract the nominal and standard deviation (1-sigma) of the backscatter electron coefficient:

```
bse = result.upbound_fraction.nominal_value
bse = result.upbound_fraction.n # alternative
unc_bse = result.upbound_fraction.std_dev
unc_bse = result.upbound_fraction.s # alternative
```

Important: All results in **pyPENELOPEtools** are stored for consistency using `ufloat`, even those where the uncertainty is always 0.0 (e.g. total simulation time).

To plot the photon spectrum, we first create a *`PenepmaSpectrumResult`* object. The class takes one argument, the index of the detector to read. The first detector has an index of 1.

```
from pypenelopetools.penepma.results import PenepmaSpectrumResult
result = PenepmaSpectrumResult(1)
result.read_directory('/simulation/epma2')
```

To plot the spectrum, we use the library matplotlib. The x-axis (energy axis) is stored in the attribute `energies_eV` whereas the intensities in 1/(sr.electron) in the attribute `intensities_1_per_sr_electron`.

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1, 1)
ax.plot(result.energies_eV, result.intensities_1_per_sr_electron, '-')
plt.show()
```

This completes the PENEPMA tutorial.

CHAPTER 3

API

3.1 Material

Material definition.

Example

Create a material definition for copper:

```
mat = Material('copper', {29: 1.0}, 8.9)
with open('copper.mat.in', 'w') as fp:
    mat.write_input(fp)
```

Using the `material` program of PENELOPE in the `pendbase` folder, run:

```
material.exe < copper.mat.in
```

There should now be a `copper.mat` in the `pendbase` folder.

class pypenelopetools.material.**Material**(*name*, *composition*, *density_g_per_cm3*, *mean_excitation_energy_eV=None*, *oscillator_strength_fcb=None*, *plasmon_energy_wcb_eV=None*)

Bases: `object`

Creates a new material.

Parameters

- **name** (*str*) – Name of material.
- **composition** (*dict*) – Composition in weight fraction. The composition is specified by a dictionary. The keys are atomic numbers and the values are weight fraction between]0.0, 1.0].

- **density_g_per_cm3** (*float*) – Material's density in g/cm3.
- **mean_excitation_energy_eV** (*float, optional*) – mean excitation energy. If `None`, it will be calculated by PENELOPE.
- **oscillator_strength_fcb** (*float, optional*) – oscillator strength of plasmon excitation. If `None`, it will be estimated by PENELOPE.
- **plasmon_energy_wcb_eV** (*float, optional*) – energy of plasmon excitation in eV. If `None`, it will be estimated by PENELOPE.

name
str – Name of material.

composition
dict – Composition in weight fraction. The composition is specified by a dictionary. The keys are atomic numbers and the values are weight fraction between]0.0, 1.0].

density_g_per_cm3
float – Material's density in g/cm3.

mean_excitation_energy_eV
float, optional – mean excitation energy. If `None`, it will be calculated by PENELOPE.

oscillator_strength_fcb
float, optional – oscillator strength of plasmon excitation. If `None`, it will be estimated by PENELOPE.

plasmon_energy_wcb_eV
float, optional – energy of plasmon excitation in eV. If `None`, it will be estimated by PENELOPE.

filename

classmethod read_input(*fileobj*)
Reads the input file created by this class (see *write_input*).

Parameters **fileobj** (*file object*) – file object opened with read access.

Returns new material.

Return type *Material*

classmethod read_material(*fileobj*)
Reads a PENELOPE generated material file (.mat).

Parameters **fileobj** (*file object*) – file object opened with read access.

Returns new material.

Return type *Material*

write_input(*fileobj*)
Writes the input file to create this material. The material program should be called with this input file as standard input:

```
material.exe < material.in
```

Parameters **fileobj** (*file object*) – file object opened with write access.

pypenelopetools.material.**VACUUM = <Material(Vacuum)>**
Material representing vacuum.

3.2 PENCYL

To be written...

3.3 PENELOPE

3.3.1 Base classes

Definition of base classes.

class pypenelopetools.penelope.base.**InputLineBase**
Bases: object

Base class to parse and read PENELOPE text files.

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 44

_abc_registry = <_weakrefset.WeakSet object>

_create_line(*name*, *values*, *comment=''*)
Creates an input line of this keyword from the specified text. The white space between the items is automatically adjusted to fit the line size. The keyword and the total length of the line is checked not to exceed their respective maximum size.

Parameters

- **name** (*str*) – 6-character keyword.
- **values** (*tuple or list*) – Values.
- **comment** (*str, optional*) – Comment associated with the line.

Returns Formatted line.

Return type str

_parse_line(*line*)
Extracts the keyword, the values and the comment of an input line. The values are returned as a list.

Parameters **line** (*str*) – Input line.

Returns Keyword, values, comment.

Return type tuple(str, tuple(str), str)

_peek_next_line(*fileobj*)
Returns the next line without advancing the current position.

Parameters **fileobj** (*file object*) – File object opened with read access.

Returns Next line, stripped of all trailing white spaces.

Return type str

_read_next_line(*fileobj*)
Returns the next line and advances the current position. Comment line (line starting with 7 spaces) are automatically skipped.

Parameters **fileobj** (*file object*) – File object opened with read access.

Returns Next line, stripped of all trailing white spaces.

Return type str

read(*fileobj*)
Reads a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with read access.

write(*fileobj*)
Writes to a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with write access.

Definition of base keyword classes.

class pypenelopetools.penelope.keyword.**KeywordBase**
Bases: *pypenelopetools.penelope.base.InputLineBase*

Base of all PENELOPE keywords.

copy()

Returns Deep copy of this keyword.

Return type _KeyboardBase

get()

Returns Value(s) of this keyword.

Return type tuple

name
str – Name of keyword.

class pypenelopetools.penelope.keyword.**KeywordGroupBase**
Bases: *pypenelopetools.penelope.keyword.KeywordBase*

Group of keywords, keywords that should always be defined together.

copy()
Returns: _KeyboardBase: Deep copy of this keyword.

get()

Returns Value(s) of all keywords.

Return type tuple

get_keywords()

Returns Keywords apart of this group.

Return type tuple

name
str – Name of keyword.

read(*fileobj*)
Reads a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with read access.

write(*fileobj*)
Writes to a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with write access.

class pypenelopetools.penelope.keyword.**KeywordSequence**(*keyword*, *maxlength*)
Bases: *pypenelopetools.penelope.keyword.KeywordBase*

Sequence of keywords, keywords that can be defined multiple times.

Parameters

- **keyword** (KeywordBase) – Base keyword.
- **maxlength** (*int*) – Maximum number of keywords that can be added.

add(**args*)
Adds a new keyword definition. This internally creates a new keyword based on the base keyword, sets the value(s) and add it to a list.

Parameters ***args** – Value(s).

clear()
Clears all added keywords.

copy()
Returns: _KeyboardBase: Deep copy of this keyword.

get()

Returns Value(s) of all keywords.

Return type tuple

name
str – Name of keyword.

pop(*index*)
Removes a keyword.

Parameters **index** (*int*) – Index of the keyword to be removed.

read(*fileobj*)
Reads a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with read access.

write(*fileobj*)
Writes to a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with write access.

class pypenelopetools.penelope.keyword.**TypeKeyword**(*name*, *types*, *comment=''*)
Bases: *pypenelopetools.penelope.keyword.KeywordBase*

Keyword where the type of the values are checked.

Parameters

- **name** (*str*) – Name of keyword.
- **types** (*tuple(type)*) – Type of each value of the keyword.
- **comment** (*str, optional*) – Comment.

comment
str – Comment.

copy()
Returns: _KeyboardBase: Deep copy of this keyword.

get()
Returns: tuple: Value(s) of this keyword.

name
str – Name of keyword.

read(*fileobj*)
Reads a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with read access.

set(**args*)
Sets the value(s) of the keyword. Each value is checked if it matches the defined type.

Parameters ***args** – Value(s).

Raises TypeError – If one value does not match its defined type.

validate(**args*)

write(*fileobj*)
Writes to a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with write access.

Definition of base separator classes.

class pypenelopetools.penelope.separator.**Separator**(*text*, *name=''*)
Bases: *pypenelopetools.penelope.base.InputLineBase*

Base of all PENELOPE separators.

Parameters

- **text** (*str*) – Comment of the separator
- **name** (*str, optional*) – name of separator keyword (e.g. END)

text
str – Comment of the separator

name
str, optional – name of separator keyword (e.g. END)

read(*fileobj*)
Reads a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with read access.

write(*fileobj*)
Writes to a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with write access.

Definition of base input classes.

class pypenelopetools.penelope.input.**PenelopeInputBase**
Bases: *pypenelopetools.penelope.base.InputLineBase*

Base input class.

get_keywords()

Returns Sorted list of all keywords in the input.

Return type list

read (*fileobj*)
Reads an input file (i.e. `.in`).

Parameters **fileobj** (*file object*) – File object opened with read access.

write (*fileobj*)
Writes to an input file (i.e. `.in`).

Parameters **fileobj** (*file object*) – File object opened with write access.

Definition of base result classes.

class pypenelopetools.penelope.result.**PenelopeResultBase**
Bases: object

Base class representing a type of result.

read (*fileobj*)
Reads a result file.

Parameters **fileobj** (*file object*) – File object opened with read access.

read_directory (*dirpath*)
Read a result file from a directory.

Parameters **dirpath** (*str*) – Path of a directory.

3.3.2 Keywords

Keywords used across different PENELOPE main programs.

class pypenelopetools.penelope.keywords.**DSMAX** (*maxlength=5000*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Maximum step length of electrons and positrons in body.

Note: This parameter is important only for thin bodies; it should be given a value of the order of one tenth of the body thickness or less.

---

add (*kb*, *dsmax*)
Sets maximum step length.

Parameters

- **kb** (*int*) – Index of body.
- **dsmax** – Maximum step length in cm.

class pypenelopetools.penelope.keywords.**DUMPP**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Dump interval.

When the DUMPTO option is activated, simulation results are written in the output files every DUMPP seconds. This option is useful to check the progress of long simulations. It also allows the program to be run with a long execution time and to be stopped when the required statistical uncertainty has been reached.

set (*dumpp*)
Sets interval.

Parameters **dumpp** (*int*) – Dump interval in seconds.

class pypenelopetools.penelope.keywords.**DUMPTO**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Name of dump file.

Generate a dump file named 'dump2.dmp' after completing the simulation run. This allows the simulation to be resumed later on to improve statistics.

Note: If the file 'dump2.dmp' already exists, it is overwritten.

---

set (*filename*)
Sets filename.

Parameters **filename** (*str*) – File name of dump file (up to 20 characters).

class pypenelopetools.penelope.keywords.**EABSB**(*maxlength=5000*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Local absorption energies EABSB(KPAR,KB) of particles of type KPAR in body KB.

These values must be larger than EABS(KPAR,M), where M is the material of body KB. When the particle is moving within body KB, the absorption energy EABS(KPAR,M) is temporarily set equal to EABSB(KPAR,KB). Thus, the simulation of the particle history is discontinued when the energy becomes less than EABSB(KPAR,KB). This feature can be used, e.g., to reduce the simulation work in regions of lesser interest.

add (*kb*, *eabs1*, *eabs2*, *eabs3*)
Sets local absorption energies.

Parameters

- **kb** (*int*) – Index of body.
- **eabs1** (*float*) – Absorption energy of electrons in eV.
- **eabs2** (*float*) – Absorption energy of photons in eV.
- **eabs3** (*float*) – Absorption energy of positrons in eV.

class pypenelopetools.penelope.keywords.**EDSPC**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Name of the output spectrum file.

set (*filename*)
Sets filename.

Parameters **filename** (*str*) – File name of output spectrum file (up to 20 characters).

class pypenelopetools.penelope.keywords.**ENDETC**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Definition of an energy-deposition detector.

set (*el*, *eu*, *nbe*)
Sets energy limits.

Parameters

- **el** (*float*) – Lower limit in eV.
- **eu** (*float*) – Upper limit in eV.

- **nbe** (*int*) – Number of bins in the output energy distribution. Should be less than 1000. If NBE is positive, energy bins have uniform width, DE=(EU-EL)/NBE. When NBE is negative, the bin width increases geometrically with the energy, i.e., the energy bins have uniform width on a logarithmic scale.

class pypenelopetools.penelope.keywords.**GEOMFN**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Name of geometry definition file.

The bodies in the material structure are normally identified by the sequential labels assigned by PENGEOM. For complex geometries, however, it may be more practical to employ user labels, i.e., the four-character strings that identify the body in the geometry definition file. In PENMAIN (and only in the parts of the code that follow the definition of the geometry), a body can be specified by giving either its PENGEOM numerical label or its user label enclosed in a pair of apostrophes (e.g., 'BOD1'). However, bodies that result from the cloning of modules (as well as those defined in an INCLUDEd geometry file) do not have a user label and only the PENGEOM numerical label is acceptable.

set (*filename*)
Sets filename.

Parameters **filename** (*str*) – File name of material file (up to 20 characters).

class pypenelopetools.penelope.keywords.**GRIDX**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Definition of x-coordinates of the vertices of the dose box.

set (*xl*, *xu*, *ndbx*)
Sets dimensions.

Parameters

- **xl** (*float*) – Lower limit of the dose box along the x-axis in cm.
- **xu** (*float*) – Upper limit of the dose box along the x-axis in cm.
- **ndbx** (*int*) – Number of bins (i.e. voxels) along the x-axis.

class pypenelopetools.penelope.keywords.**GRIDY**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Definition of y-coordinates of the vertices of the dose box.

set (*yl*, *yu*, *ndby*)
Sets dimensions.

Parameters

- **yl** (*float*) – Lower limit of the dose box along the y-axis in cm.
- **yu** (*float*) – Upper limit of the dose box along the y-axis in cm.
- **ndby** (*int*) – Number of bins (i.e. voxels) along the y-axis.

class pypenelopetools.penelope.keywords.**GRIDZ**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Definition of z-coordinates of the vertices of the dose box.

set (*zl*, *zu*, *ndbz*)
Sets dimensions.

Parameters

- **zl** (*float*) – Lower limit of the dose box along the z-axis in cm.

- **zu** (*float*) – Upper limit of the dose box along the z-axis in cm.
- **ndbz** (*int*) – Number of bins (i.e. voxels) along the z-axis.

class pypenelopetools.penelope.keywords.**IBRSPL**(*maxlength=5000*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Bremsstrahlung splitting for electrons and positrons.

Note: Note that bremsstrahlung splitting is applied in combination with interaction forcing and, consequently, it is activated only in those bodies where interaction forcing is active.

---

add(*kb*, *ibrspl*)
Add Bremsstrahlung splitting.

Parameters

- **kb** (*int*) – Index of body.
- **ibrspl** (*int*) – Splitting factor.

class pypenelopetools.penelope.keywords.**IXRSPL**(*maxlength=5000*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Splitting of characteristic x rays emitted.

Each unsplit x ray with ILB(2)=2 (i.e., of the second generation) when extracted from the secondary stack is split into IXRSPL quanta. The new, lighter, quanta are assigned random directions distributed isotropically.

add(*kb*, *ixrspl*)
Add characteristic x rays splitting.

Parameters

- **kb** (*int*) – Index of body.
- **ixrspl** (*int*) – Splitting factor.

class pypenelopetools.penelope.keywords.**InteractionForcings**(*maxlength=120000*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Forcing of interactions.

FORCER is the forcing factor, which must be larger than unity. WLOW and WHIG are the lower and upper limits of the pweight window where interaction forcing is applied. When several interaction mechanisms are forced in the same body, the effective weight window is set equal to the intersection of the windows for these mechanisms.

If the mean free path for real interactions of type ICOL is MFP, the program will simulate interactions of this type (real or forced) with an effective mean free path equal to MFP/FORCER.

Hint: A negative input value of FORCER, -FN, is assumed to mean that a particle with energy E=EPMAX should interact, on average, +FN times in the course of its slowing down to rest, for electrons and positrons, or along a mean free path, for photons. This is very useful, e.g., to generate x-ray spectra from bulk samples.

---

add(*kb*, *kpar*, *icol*, *forcer*, *wlow*, *whig*)
Adds forcing for an interaction.

Parameters

- **kb** (*int*) – Index of body.

- **kparp** (KPAR) – Type of primary particles

class pypenelopetools.penelope.keywords.**MFNAME**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Name of a PENELOPE input material data file.

This file must be generated in advance by running the program MATERIAL.

set (*filename*)
Sets filename.

Parameters **filename** (*str*) – File name of material file (up to 20 characters).

class pypenelopetools.penelope.keywords.**MSIMPA**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Set of simulation parameters for this material

- absorption energies, EABS(1:3,M),
- elastic scattering parameters, C1(M) and C2(M), and
- cutoff energy losses for inelastic collisions and Bremsstrahlung emission, WCC(M) and WCR(M).

set (*eabs1*, *eabs2*, *eabs3*, *c1*, *c2*, *wcc*, *wcr*)
Sets parameters.

Parameters

- **eabs1** (*float*) – Absorption energy of electrons in eV.
- **eabs2** (*float*) – Absorption energy of photons in eV.
- **eabs3** (*float*) – Absorption energy of positrons in eV.
- **c1** (*float*) – Elastic scattering coefficient.
- **c2** (*float*) – Elastic scattering coefficient.
- **wcc** (*float*) – Cutoff energy losses for inelastic collisions in eV.
- **wcr** (*float*) – Cutoff energy losses for Bremsstrahlung emission in eV.

class pypenelopetools.penelope.keywords.**MaterialGroup**
Bases: *pypenelopetools.penelope.keyword.KeywordGroupBase*

Group to define both material file name and its simulation parameters.

get_keywords()
Returns: tuple: Keywords apart of this group.

set (*filename*, *eabs1*, *eabs2*, *eabs3*, *c1*, *c2*, *wcc*, *wcr*, *index=None*)
Sets material file name and simulation parameters.

Parameters

- **filename** (*str*) – File name of material file (up to 20 characters).
- **eabs1** (*float*) – Absorption energy of electrons in eV.
- **eabs2** (*float*) – Absorption energy of photons in eV.
- **eabs3** (*float*) – Absorption energy of positrons in eV.
- **c1** (*float*) – Elastic scattering coefficient.
- **c2** (*float*) – Elastic scattering coefficient.

- **wcc** (*float*) – Cutoff energy losses for inelastic collisions in eV.
- **wcr** (*float*) – Cutoff energy losses for Bremsstrahlung emission in eV.
- **index** (*int, optional*) – Index of this material in the geometry

class pypenelopetools.penelope.keywords.**Materials**(*maxlength=10*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Definition of materials.

add(*index*, *filename*, *eabs1*, *eabs2*, *eabs3*, *c1*, *c2*, *wcc*, *wcr*)
Adds a new material.

Parameters

- **index** (*int*) – Index of this material in the geometry
- **filename** (*str*) – File name of material file (up to 20 characters).
- **eabs1** (*float*) – Absorption energy of electrons in eV.
- **eabs2** (*float*) – Absorption energy of photons in eV.
- **eabs3** (*float*) – Absorption energy of positrons in eV.
- **c1** (*float*) – Elastic scattering coefficient.
- **c2** (*float*) – Elastic scattering coefficient.
- **wcc** (*float*) – Cutoff energy losses for inelastic collisions in eV.
- **wcr** (*float*) – Cutoff energy losses for Bremsstrahlung emission in eV.

get()
Returns: tuple: Value(s) of all keywords.

write(*fileobj*)
Writes to a PENELOPE-type file.

Parameters **fileobj** (*file object*) – File object opened with write access.

class pypenelopetools.penelope.keywords.**NBANGL**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Definition of angular distributions of emerging particles.

Note: In the output files, the terms 'upbound' and 'downbound' are used to denote particles that leave the material system moving upwards (W>0) and downwards (W<0), respectively.

---

set(*nbth*, *nbph*)
Sets angular distributions.

Parameters

- **nbth** (*int*) – Numbers of bins for the polar angle THETA. Should be less than 3600. If NBTH is positive, angular bins have uniform width, DTH=180./NBTHE. When NBTH is negative, the bin width increases geometrically with THETA, i.e., the bins have uniform width on a logarithmic scale.
- **nbph** (*int*) – Number of bins for the azimuthal angle PHI Should be less than 180.

class pypenelopetools.penelope.keywords.**NBE**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Definition of energy distributions of emerging particles.

set (*el*, *eu*, *nbe*)
Sets energy distributions.

Parameters

- **el** (*float*) – Lower limit in eV.
- **eu** (*float*) – Upper limit in eV.
- **nbe** (*int*) – Number of bins in the output energy distribution. Should be less than 1000. If NBE is positive, energy bins have uniform width, DE=(EU-EL)/NBE. When NBE is negative, the bin width increases geometrically with the energy, i.e., the energy bins have uniform width on a logarithmic scale.

class pypenelopetools.penelope.keywords.**NSIMSH**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Desired number of simulated showers.

set (*dshn*)
Sets showers.

Parameters **dshn** (*int*) – Number of simulated showers.

class pypenelopetools.penelope.keywords.**RESUME**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Name of the resume file.

The program will read the dump file named `dump1.dmp` and resume the simulation from the point where it was left.

> **Warning:** Use this option very, **VERY** carefully. Make sure that the input data file is fully consistent with the one used to generate the dump file.

set (*filename*)
Sets filename.

Parameters **filename** (*str*) – File name of resume file (up to 20 characters).

class pypenelopetools.penelope.keywords.**RSEED**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Seeds of the random-number generator.

set (*iseed1*, *iseed2*)
Sets seeds.

Parameters

- **iseed1** (*int*) – First seed. When ISEED1 is equal to a negative integer, -N, the seeds are set by calling subroutine RAND0(N) with the input argument equal to N. This ensures that sequences of random numbers used in different runs of the program (with different values of N) are truly independent.
- **iseed2** (*int*) – Second seed.

class pypenelopetools.penelope.keywords.**SCONE**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Initial direction of primary particles is sampled uniformly within a conical beam.

Conical source beam. Polar and azimuthal angles of the beam axis direction, THETA and PHI, and angular aperture, ALPHA, in deg.

The case ALPHA=0 defines a monodirectional source, and ALPHA =180 deg corresponds to an isotropic source.

set (*theta*, *phi*, *alpha*)
Sets angles.

Parameters

- **theta** (*float*) – Polar angle of the beam axis direction in deg.
- **phi** (*float*) – Azimuthal angle of the beam axis direction in deg.
- **alpha** (*float*) – Angular aperture in deg.

class pypenelopetools.penelope.keywords.**SENERG**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

For a monoenergetic source, initial energy SE0 of primary particles.

set (*se0*)
Sets value.

Parameters **se0** (*float*) – Initial energy of primary particles in eV

class pypenelopetools.penelope.keywords.**SGPOL**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Activates the simulation of polarisation effects in the scattering of photons.

This line activates the simulation of polarisation effects in the scattering of photons (electrons and positrons are assumed to be unpolarised). SP1, SP2, SP3 are the Stokes parameters of primary photons, which define the degrees of linear polarisation at 45 deg azimuth, of circular polarisation, and of linear polarisation at zero azimuth, respectively. It is assumed that secondary photons are emitted with null polarisation (SP1=SP2=SP3=0).

set (*sp1*, *sp2*, *sp3*)
Sets Stokes polarisation parameters.

Parameters

- **sp1** (*float*) – Degrees of linear polarisation at 45 deg azimuth
- **sp2** (*float*) – Degrees of circular polarisation
- **sp3** (*float*) – Degrees of linear polarisation at 0 deg azimuth

class pypenelopetools.penelope.keywords.**SKPAR**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Type of primary particle KPARP (1=electrons, 2=photons or 3=positrons).

If KPARP=0, the initial states of primary particles are set by subroutine SOURCE, to be provided by the user. An example of that subroutine, corresponding to a 60-Co source (two gamma rays in each nuclear deexcitation), is included in the PENMAIN package (file 'source.f').

set (*kparp*)
Sets value.

Parameters **kparp** (KPAR) – Type of primary particles

class pypenelopetools.penelope.keywords.**SPECTR**(*maxlength=1000*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Define a source with continuous (stepwise constant) spectrum.

For a source with continuous (stepwise constant) spectrum, each 'SPECTR' line gives the lower end-point of an energy bin of the source spectrum (Ei) and the associated relative probability (Pi), integrated over the bin. Up to NSEM=1000 lines, in arbitrary order. The upper end of the spectrum is defined by entering a line with Ei equal to the upper energy end point and with a negative Pi value.

add(*ei*, *pi*)
Adds a step in the spectrum.

Parameters

- **ei** (*float*) – Lower end-point of an energy bin of the source spectrum in eV
- **pi** (*float*) – Associated relative probability

class pypenelopetools.penelope.keywords.**SPOSIT**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Coordinates of the source centre.

set(*sx0*, *sy0*, *sz0*)
Sets coordinates.

Parameters

- **sx0** (*float*) – x-coordinate in cm.
- **sy0** (*float*) – y-coordinate in cm.
- **sz0** (*float*) – z-coordinate in cm.

class pypenelopetools.penelope.keywords.**SRECTA**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Initial direction of primary particles is sampled uniformly within a rectangular beam.

Rectangular source beam. Limiting polar and azimuthal angles of the source beam window, (THETAL,THETAU)x(PHIL,PHIU), in deg.

The case THETAL=THETAU, PHIL=PHIU defines a monodirectional source. To define an isotropic source, set THETAL=0, THETAU= 180, PHIL=0 and PHIU=360.

set(*thetal*, *thetau*, *phil*, *phiu*)
Sets angles.

Parameters

- **thetal** (*float*) – Lower limit polar angle in deg.
- **thetau** (*float*) – Upper limit polar angle in deg.
- **phil** (*float*) – Lower limit azimuthal angle in deg.
- **phiu** (*float*) – Upper limit azimuthal angle in deg.

class pypenelopetools.penelope.keywords.**TIME**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Allotted simulation time.

set(*timea*)
Sets simulation time.

Parameters **timea** (*int*) – Allotted simulation time in seconds.

class pypenelopetools.penelope.keywords.**TITLE**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Title of the job.

The TITLE string is used to mark dump files. To prevent the improper use of wrong resuming files, change the title each time you modify basic parameters of your problem. The code will then be able to identify the inconsistency and to print an error message before stopping.

set (*title*)
Sets value.

Parameters **title** (*str*) – Title of the job (up to 65 characters).

3.3.3 Separators

Separators used across different PENELOPE main programs.

pypenelopetools.penelope.separators.**BREMSSTRAHLUNG_SPLITTING = <pypenelopetools.penelope.se**
Section for Bremsstralung splitting.

pypenelopetools.penelope.separators.**DOT = <pypenelopetools.penelope.separator.Separator obj**
Section separator.

pypenelopetools.penelope.separators.**EMERGING_PARTICLES = <pypenelopetools.penelope.separato**
Section for emerging particles.

pypenelopetools.penelope.separators.**END = <pypenelopetools.penelope.separator.Separator obj**
End separator

pypenelopetools.penelope.separators.**ENERGY_DEPOSITON_DETECTORS = <pypenelopetools.penelope**
Section for energy deposition detectors.

pypenelopetools.penelope.separators.**INTERACTION_FORCING = <pypenelopetools.penelope.separat**
Section for interaction forcing(s).

pypenelopetools.penelope.separators.**JOB_PROPERTIES = <pypenelopetools.penelope.separator.Se**
Section for job properties.

pypenelopetools.penelope.separators.**MATERIAL = <pypenelopetools.penelope.separator.Separato**
Section for material(s).

pypenelopetools.penelope.separators.**SOURCE_DEFINITION = <pypenelopetools.penelope.separator**
Section for source definition.

pypenelopetools.penelope.separators.**XRAY_SPLITTING = <pypenelopetools.penelope.separator.Se**
Section for characteristic x ray splitting.

3.4 PENEPMA

3.4.1 Input

Input of PENEPMA simulation.

class pypenelopetools.penepma.input.**PenepmaInput**
Bases: *pypenelopetools.penelope.input.PenelopeInputBase*

Creates an object representing a PENEPMA input file.

TITLE
TITLE – Title of the job (up to 65 characters).

SKPAR
SKPAR – Type of primary particle KPARP (1=electrons, 2=photons or 3=positrons).

SENERG
SENERG – Initial energy SE0 of primary particles.

SPOSIT
SPOSIT – Coordinates of the source centre.

SRADI
SRADI – Initial position of the particle is sampled randomly within a circle.

SDIREC
SDIREC – Polar and azimuthal angles of the electron beam axis direction.

SAPERT
SAPERT – Angular aperture of the electron beam.

materials
Materials – Definition of materials.

GEOMFN
GEOMFN – Name of geometry definition file.

DSMAX
DSMAX – Maximum step length of electrons and positrons in body.

IFORCE
IFORCE – Forcing of interactions.

IBRSPL
IBRSPL – Bremsstrahlung splitting for electrons and positrons.

IXRSPL
IXRSPL – Splitting of characteristic x rays emitted.

NBE
NBE – Definition of energy distributions of emerging particles.

NBANGL
NBANGL – Definition of angular distributions of emerging particles.

photon_detectors
PhotonDetectors – Definition of the photon detectors.

GRIDX
GRIDX – Definition of x-coordinates of the vertices of the dose box.

GRIDY
GRIDY – Definition of y-coordinates of the vertices of the dose box.

GRIDZ
GRIDZ – Definition of z-coordinates of the vertices of the dose box.

XRAYE
XRAYE – Space distribution of emission sites of x rays within an energy interval.

XRLINE
XRLINE – Space distribution of emission sites of x rays.

RESUME
RESUME – Name of the resume file.

DUMPTO
DUMPTO – Name of dump file.

DUMPP
DUMPP – Dump interval.

RSEED
RSEED – Seeds of the random-number generator.

REFLIN
REFLIN – Termination of simulation based on relative statistical uncertainty of the intensity of line.

NSIMSH
NSIMSH – Desired number of simulated showers.

TIME
TIME – Allotted simulation time.

get_keywords()
Returns: list: Sorted list of all keywords in the input.

read(*fileobj*)
Reads an input file (i.e. `.in`).

Parameters **fileobj** (*file object*) – File object opened with read access.

write(*fileobj*)
Writes to an input file (i.e. `.in`).

Parameters **fileobj** (*file object*) – File object opened with write access.

3.4.2 Results

Results of PENEPMA simulation.

class pypenelopetools.penepma.results.**PenepmaIntensityResult**(*detector_index*)
Bases: *pypenelopetools.penepma.results.PenepmaPhotonDetectorResultBase*

Results from `pe-intens-XX.dat`, where `XX` is the index of the detector. The intensities are given for each characteristic x-ray detected by the detector.

Note: The characteristic x-rays are expressed using `XrayLine` object of the pyxray package. `XrayLine` objects define the atomic number and x-ray transition of characteristic x-rays.

The intensities are expressed using the uncertainties package. The nominal value can be accessed with the property `nominal_value` or the abbreviation `n`, whereas the standard deviation (1-sigma), with the property `std_dev` or `s`.

For example:

```
import pyxray
x = pyxray.XrayLine(29, 'Ka1')
result.total_intensities_1_per_sr_electron[x].n #-> 8.56e-10
result.total_intensities_1_per_sr_electron[x].s #-> 0.15e-10
```

Parameters **detector_index** (*int*) – Index of the detector to read the results from.

detector_index
int – Index of detector

theta1_deg
ufloat – Lower limit polar angle in deg.

theta2_deg
ufloat – Upper limit polar angle in deg.

phi1_deg
ufloat – Lower limit azimuthal angle in deg.

phi2_deg
ufloat – Upper limit azimuthal angle in deg.

primary_intensities_1_per_sr_electron
dict(XrayLine, ufloat) – Intensities of characteristic x-rays generated by primary electrons and measured by the detector.

characteristic_fluorescence_intensities_1_per_sr_electron
dict(XrayLine, ufloat) – Intensities of characteristic x-rays generated by the fluorescence of characteristic x-rays and measured by the detector.

bremsstrahlung_fluorescence_intensities_1_per_sr_electron
dict(XrayLine, ufloat) – Intensities of characteristic x-rays generated by the fluorescence of Bremsstrahlung x-rays and measured by the detector.

total_fluorescence_intensities_1_per_sr_electron
dict(XrayLine, ufloat) – Intensities of characteristic x-rays generated by fluorescence (characteristic and Bremsstrahlung) and measured by the detector. Intensities are equal to the sum of *characteristic_fluorescence_intensities_1_per_sr_electron* and *bremsstrahlung_fluorescence_intensities_1_per_sr_electron*.

total_intensities_1_per_sr_electron
dict(XrayLine, ufloat) – Intensities of characteristic x-rays generated and measured by the detector. Intensities are equal to the sum of *primary_intensities_1_per_sr_electron* and *total_fluorescence_intensities_1_per_sr_electron*.

read(*fileobj*)
Reads a result file.

Parameters **fileobj** (*file object*) – File object opened with read access.

read_directory(*dirpath*)
Read a result file from a directory.

Parameters **dirpath** (*str*) – Path of a directory.

class pypenelopetools.penepma.results.**PenepmaPhotonDetectorResultBase**(*detector_index*)
Bases: *pypenelopetools.penelope.result.PenelopeResultBase*

Base result associated with a photon detector.

Parameters **detector_index** (*int*) – Index of the detector to read the results from.

detector_index
int – Index of detector

theta1_deg
ufloat – Lower limit polar angle in deg.

theta2_deg
ufloat – Upper limit polar angle in deg.

phi1_deg
ufloat – Lower limit azimuthal angle in deg.

phi2_deg
ufloat – Upper limit azimuthal angle in deg.

read(*fileobj*)
Reads a result file.

Parameters **fileobj** (*file object*) – File object opened with read access.

read_directory(*dirpath*)
Read a result file from a directory.

Parameters **dirpath** (*str*) – Path of a directory.

class pypenelopetools.penepma.results.**PenepmaResult**
Bases: *pypenelopetools.penelope.result.PenelopeResultBase*

Results from `penepma-res.dat`.

Note: All results are expressed using the uncertainties package. The nominal value can be accessed with the property `nominal_value` or the abbreviation `n`, whereas the standard deviation (1-sigma), with the property `std_dev` or `s`. For example:

```
result.simulation_time_s.n #-> 100.0
result.simulation_time_s.s #-> 0.0
```

---

simulation_time_s
ufloat – Simulation time in seconds.

simulation_speed_1_per_s
ufloat – Simulation speed in simulation per second.

simulated_primary_showers
ufloat – Number of primary showers simulated.

upbound_primary_particles
ufloat – Number of primary particles that exited the geometry upwards.

downbound_primary_particles
ufloat – Number of primary particles that exited the geometry downwards.

absorbed_primary_particles
ufloat – Number of primary particles that were absorbed within the geometry.

upbound_fraction
ufloat – Fraction of primary particles that exited the geometry upwards.

downbound_fraction
ufloat – Fraction of primary particles that exited the geometry downwards.

absorbed_fraction
ufloat – Fraction of primary particles that were absorbed within the geometry.

upbound_secondary_electron_generation_probabilities
ufloat – Probability of second generation electrons exited the geometry upwards.

downbound_secondary_electron_generation_probabilities
ufloat – Probability of second generation electrons exited the geometry downwards.

absorbed_secondary_electron_generation_probabilities
ufloat – Probability of second generation electrons absorbed within the geometry.

upbound_secondary_photon_generation_probabilities
ufloat – Probability of second generation photons exited the geometry upwards.

downbound_secondary_photon_generation_probabilities
ufloat – Probability of second generation photons exited the geometry downwards.

absorbed_secondary_photon_generation_probabilities
ufloat – Probability of second generation photons absorbed within the geometry.

upbound_secondary_positron_generation_probabilities
ufloat – Probability of second generation positrons exited the geometry upwards.

downbound_secondary_positron_generation_probabilities
ufloat – Probability of second generation positrons exited the geometry downwards.

absorbed_secondary_positron_generation_probabilities
ufloat – Probability of second generation positrons absorbed within the geometry.

average_deposited_energy_eV
dict(int, ufloat) – Average deposited energy in each body. Dictionary where keys are indexes of body and values, the average deposited energy in eV.

average_photon_energy_eV
dict(int, ufloat) – Average photon energy in each detector. Dictionary where keys are indexes of detector and values, the average energy in eV.

last_random_seed1
ufloat – Last first seed of the random number generator.

last_random_seed2
ufloat – Last second seed of the random number generator.

reference_line_uncertainty
ufloat – Relative uncertainty of the x-ray line used as a termination condition

read(*fileobj*)
Reads a result file.

Parameters **fileobj** (*file object*) – File object opened with read access.

read_directory(*dirpath*)
Read a result file from a directory.

Parameters **dirpath** (*str*) – Path of a directory.

class pypenelopetools.penepma.results.**PenepmaSpectrumResult**(*detector_index*)
Bases: *pypenelopetools.penepma.results.PenepmaPhotonDetectorResultBase*

Results from `pe-spect-XX.dat`, where `XX` is the index of the detector. The spectrum is stored as a numpy array where the first column contains the energies in eV and the second the intensities in 1/(sr.electron).

Note: The energies and intensities are expressed using the uncertainties package. The nominal value can be accessed with the property `nominal_value` or the abbreviation `n`, whereas the standard deviation (1-sigma), with the property `std_dev` or `s`.

For example:

```
from uncertainty import unumpy
energies_eV = unumpy.nominal_values(result.spectrum[:,0])
intensities = unumpy.nominal_values(result.spectrum[:,1])
```

Parameters **detector_index** (*int*) – Index of the detector to read the results from.

detector_index
int – Index of detector

theta1_deg
ufloat – Lower limit polar angle in deg.

theta2_deg
ufloat – Upper limit polar angle in deg.

phi1_deg
ufloat – Lower limit azimuthal angle in deg.

phi2_deg
ufloat – Upper limit azimuthal angle in deg.

energy_window_start_eV
ufloat – Energy of first window in eV.

energy_window_end_eV
ufloat – Energy of last window in eV.

channel_width_eV
ufloat – Width of one channel in eV.

spectrum
unumpy.uarray – Array where the first column contains the energies in eV and the second the intensities measured by the detector in 1/(sr.electron).

energies_eV
numpy array – Nominal values of the energy axis in eV.

intensities_1_per_sr_electron
numpy array – Nominal values of the intensity axis in 1/(sr.electron).

read(*fileobj*)
Reads a result file.

Parameters **fileobj** (*file object*) – File object opened with read access.

read_directory(*dirpath*)
Read a result file from a directory.

Parameters **dirpath** (*str*) – Path of a directory.

3.4.3 Keywords

Keywords used specifically for PENEPMA.

class pypenelopetools.penepma.keywords.**PhotonDetectorGroup**
Bases: *pypenelopetools.penelope.keyword.KeywordGroupBase*

Definition of photon detector.

Each detector collects photons that leave the sample with directions within a *rectangle(on the unit sphere, limited by the *parallels* THETA1 and THETA2 and the *meridians* PHI1 and PHI2. The output spectrum is the energy distribution of photons that emerge within the acceptance solid angle of the detector with energies in the interval from EDEL to EDEU, recorded using NCHE channels. Notice that the spectrum is given in absolute units (per incident electron, per eV and per unit solid angle).

get_keywords()
Returns: tuple: Keywords apart of this group.

set(*theta1*, *theta2*, *phi1*, *phi2*, *ipsf*, *edel*, *edeu*, *nche*, *emission_filename=None*)
Sets parameters of detector.

Note: `phi1` and `phi2` must be both either in the interval (0,360) or in the interval (-180,180).

Parameters

- **theta1** (*float*) – Lower limit polar angle in deg.
- **theta2** (*float*) – Upper limit polar angle in deg.
- **phi1** (*float*) – Lower limit azimuthal angle in deg.
- **phi2** (*float*) – Upper limit azimuthal angle in deg.
- **ipsf** (*int*) – Flag to activate the creation of a phase-space file (psf), which contains the state variables and weights of particles that enter the detector. Use this option with care, because psf's may grow very fast.
  - `ipsf=0`: the psf is not created.
  - `ipsf=1`: a psf is created.
  - `ipsf>1`: a psf is created, but contains only state variables of detected photons that have ILB(4)=IPSF (used for studying angular distributions of x rays).

  Generating the psf is useful for tuning interaction forcing, which requires knowing the weights of detected particles.
- **edel** (*float*) – Lower limits of the energy window covered by the detector in eV.
- **edeu** (*float*) – Upper limits of the energy window covered by the detector in eV.
- **nche** (*int*) – Number of energy channels in the output spectrum. Should be less than 1000.
- **emission_filename** (*str, optional*) – File name of the generation file. Specifying a file name activates the generation of a file with the position coordinates of the emission sites of the photons that reach the detector. Notice that the file may grow very fast, so use this option only in short runs. The output file is overwritten when a simulation is resumed.

class pypenelopetools.penepma.keywords.**PhotonDetectors**(*maxlength=25*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Definition of the photon detectors.

Up to 25 different detectors can be defined.

add(*theta1*, *theta2*, *phi1*, *phi2*, *ipsf*, *edel*, *edeu*, *nche*, *emission_filename=None*)
Add a new photon detector.

Note: `phi1` and `phi2` must be both either in the interval (0,360) or in the interval (-180,180).

Important: The theta angles are defined as angles from the positive z-axis. This is different than the take-off angle usually used in microanalysis. For a take-off angle of 30deg, theta would be 60deg.

Parameters

- **theta1** (*float*) – Lower limit polar angle in deg.
- **theta2** (*float*) – Upper limit polar angle in deg.
- **phi1** (*float*) – Lower limit azimuthal angle in deg.
- **phi2** (*float*) – Upper limit azimuthal angle in deg.
- **ipsf** (*int*) – Flag to activate the creation of a phase-space file (psf), which contains the state variables and weights of particles that enter the detector. Use this option with care, because psf's may grow very fast.
  - `ipsf=0`: the psf is not created.
  - `ipsf=1`: a psf is created.
  - `ipsf>1`: a psf is created, but contains only state variables of detected photons that have ILB(4)=IPSF (used for studying angular distributions of x rays).

  Generating the psf is useful for tuning interaction forcing, which requires knowing the weights of detected particles.
- **edel** (*float*) – Lower limits of the energy window covered by the detector in eV.
- **edeu** (*float*) – Upper limits of the energy window covered by the detector in eV.
- **nche** (*int*) – Number of energy channels in the output spectrum. Should be less than 1000.
- **emission_filename** (*str, optional*) – File name of the generation file. Specifying a file name activates the generation of a file with the position coordinates of the emission sites of the photons that reach the detector. Notice that the file may grow very fast, so use this option only in short runs. The output file is overwritten when a simulation is resumed.

class pypenelopetools.penepma.keywords.**REFLIN**

Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Termination of simulation based on relative statistical uncertainty of the intensity of line.

set (*izs1s200*, *idet*, *tol*)

Sets termination.

Hint: Use `convert_xrayline_to_izs1s200` to convert `XrayLine` to ILB(4) notation.

Parameters

- **izs1s200** – x ray identification. ILB(4) notation, note the final double zero (IZ*1e6+S1*1e4+S2*1e2).
- **idet** (*int*) – Index of detector.
- **tol** (*float*) – Relative statistical uncertainty (3*sigma) of the intensity of line

class pypenelopetools.penepma.keywords.**SAPERT**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Angular aperture of the electron beam.

set(*salpha*)
Sets angle.

Parameters **salpha** (*float*) – Angular aperture in deg.

class pypenelopetools.penepma.keywords.**SDIREC**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Polar and azimuthal angles of the electron beam axis direction.

set(*stheta*, *sphi*)
Sets angles.

Parameters

- **stheta** (*float*) – Polar angle in deg.
- **sphi** (*float*) – Azimuthal angle in deg.

class pypenelopetools.penepma.keywords.**SRADI**
Bases: *pypenelopetools.penelope.keyword.TypeKeyword*

Initial position of the particle is sampled randomly within a circle.

The circle is centered at (SX0,SY0,SZ0) and perpendicular to the beam axis direction.

set(*sradius*)
Sets circle.

Parameters **sradius** (*float*) – Radius of the circle in cm.

class pypenelopetools.penepma.keywords.**XRAYE**(*maxlength=10*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Space distribution of emission sites of x rays within an energy interval.

add(*emin*, *emax*)
Add an energy interval.

Parameters

- **emin** (*float*) – Lower limit in eV.
- **emax** (*float*) – Upper limit in eV.

class pypenelopetools.penepma.keywords.**XRLINE**(*maxlength=10*)
Bases: *pypenelopetools.penelope.keyword.KeywordSequence*

Space distribution of emission sites of x rays.

add(*izs1s200*)
Adds a characteristic x ray.

Hint: Use `convert_xrayline_to_izs1s200` to convert `XrayLine` to ILB(4) notation.

Parameters **izs1s200** (*int*) – x ray identification. ILB(4) notation, note the final double zero (IZ*1e6+S1*1e4+S2*1e2).

3.5 PENGEOM

3.5.1 Base classes

Definition of base PENGEOM classes.

class pypenelopetools.pengeom.base.**GeometryBase**
Bases: object

Base class for geometry objects.

_abc_cache = <_weakrefset.WeakSet object>

_abc_negative_cache = <_weakrefset.WeakSet object>

_abc_negative_cache_version = 48

_abc_registry = <_weakrefset.WeakSet object>

_create_expline(*keyword*, *value*, *termination=''*)
Creates a exponent line. This type of line is characterised by a keyword, a value express as an exponent (see *_toexponent()*) and a termination string. The keyword and the total length of the line is checked not to exceed their respective maximum size.

Parameters

- **keyword** (*str*) – 8-character keyword.
- **value** (*float*) – value of the keyword.
- **termination** (*str, optional*) – comment associated with the line.

_create_line(*keyword*, *text*, *termination=''*)
Creates an input line from the specified keyword, text and comment. The white space between the items is automatically adjusted to fit the line size. The keyword and the total length of the line is checked not to exceed their respective maximum size.

Parameters

- **keyword** (*str*) – 8-character keyword.
- **text** (*str*) – value of the keyword.
- **termination** (*str, optional*) – comment associated with the line.

_parse_expline(*line*)

_parse_line(*line*)

_peek_next_line(*fileobj*)
Returns the next line without advancing the current position.

Parameters **fileobj** (*file object*) – File object opened with read access.

Returns Next line, stripped of all trailing white spaces.

Return type str

_read(*fileobj*, *material_lookup*, *surface_lookup*, *module_lookup*)
Reads file object.

Parameters

- **fileobj** (*file object*) – File object opened with read access.

- **material_lookup** (dict(int, *Material*)) – A lookup table for the materials used in the geometry. Dictionary where the keys are material indexes in the geometry file and the values, *Material* instances.
- **surface_lookup** (dict(int, Surface)) – A lookup table for surfaces used in the geometry. Dictionary where the keys are surface indexes in the geometry file and the values, Surface instances.
- **module_lookup** (dict(int, *Module*)) – A lookup table for modules used in the geometry. Dictionary where the keys are module indexes in the geometry file and the values, *Module* instances.

_read_next_line(*fileobj*)
Returns the next line and advances the current position.

Parameters **fileobj** (*file object*) – File object opened with read access.

Returns Next line, stripped of all trailing white spaces.

Return type str

_write(*fileobj*, *index_lookup*)
Writes to file object.

Parameters

- **fileobj** (*file object*) – File object opened with write access.
- **index_lookup** (dict(*GeometryBase*, int)) – A lookup table for the surfaces, modules and materials of the associated geometry. Each component is assigned an index by the method *indexify*. Dictionary where the keys are surfaces, modules and materials instances, and the values, an integer index.

pypenelopetools.pengeom.base.**_toexponent**(*number*)
Formats exponent to PENELOPE format (E22.15)

Parameters **number** (*Float*) – Number to format

Returns number to PENELOPE format (E22.15)

Return type str

Mixins used to create geometry objects.

class pypenelopetools.pengeom.mixin.**DescriptionMixin**
Bases: object

Mixin that adds a description property.

description
str – Description of the geometry object.

class pypenelopetools.pengeom.mixin.**ModuleMixin**
Bases: object

Mixin that adds methods to add, pop and clear modules.

add_module(*module*)
Adds a module.

Parameters **module** (*Module*) – Module to add.

clear_modules()
Clear all modules.

get_modules()

Returns All modules.

Return type tuple

pop_module(*module*)
Removes a module.

Parameters **module** (*Module*) – Module to remove.

Definition of geometrical transformations.

class pypenelopetools.pengeom.transformation.**Rotation**(*omega_deg=0.0*, *theta_deg=0.0*, *phi_deg=0.0*)
Bases: *pypenelopetools.pengeom.base.GeometryBase*

Represents a rotation using 3 Euler angles (YZY).

Parameters

- **omega_deg** (*float*) – Rotation around the z-axis (deg).
- **theta_deg** (*float*) – Rotation around the y-axis (deg).
- **phi_deg** (*float*) – Rotation around the new z-axis (deg).

omega_deg
float – Rotation around the z-axis (deg). The value must be between 0 and 360.

phi_deg
float – Rotation around the new z-axis (deg). The new z-axis refer to the axis after the omega and theta rotation were applied on the original coordinate system. The value must be between 0 and 360.

theta_deg
float – Rotation around the y-axis (deg). The value must be between 0 and 360.

class pypenelopetools.pengeom.transformation.**Scale**(*x=1.0*, *y=1.0*, *z=1.0*)
Bases: *pypenelopetools.pengeom.base.GeometryBase*

Represents scaling.

Parameters

- **x** (*float*) – Scaling along the x direction.
- **y** (*float*) – Scaling along the y direction.
- **z** (*float*) – Scaling along the z direction.

x
float – Scaling along the x direction. The value cannot be 0.

y
float – Scaling along the y direction. The value cannot be 0.

z
float – Scaling along the z direction. The value cannot be 0.

class pypenelopetools.pengeom.transformation.**Shift**(*x_cm=0.0*, *y_cm=0.0*, *z_cm=0.0*)
Bases: *pypenelopetools.pengeom.base.GeometryBase*

Represents a translation in space.

Parameters

- **x_cm** (*float*) – Translation along the x direction (cm).
- **y_cm** (*float*) – Translation along the y direction (cm).

- **z_cm** (*float*) – Translation along the z direction (cm).

x_cm
float – Translation along the x direction (cm).

y_cm
float – Translation along the y direction (cm).

z_cm
float – Translation along the z direction (cm).

3.5.2 Geometry

Geometry definition for PENGEOM.

class pypenelopetools.pengeom.geometry.**Geometry**(*title='Untitled'*, *tilt_deg=0.0*, *rotation_deg=0.0*)

Bases: *pypenelopetools.pengeom.mixin.ModuleMixin*, *pypenelopetools.pengeom.base.GeometryBase*

Creates a new PENELOPE geometry.

Parameters

- **title** (*str, optional*) – Title of the geometry.
- **tilt_deg** (*float, optional*) – Specimen tilt in degrees along the x-axis.
- **rotation_deg** (*float, optional*) – Specimen rotation in degrees along the z-axis

add_module(*module*)
Adds a module.

Parameters **module** (*Module*) – Module to add.

clear_modules()
Clear all modules.

get_materials()
Returns all materials in this geometry.

get_modules()

Returns All modules.

Return type tuple

get_surfaces()
Returns all surfaces in this geometry.

indexify()
Returns a lookup table which associates the surfaces, modules and materials of this geometry to their index used in the geometry file. The lookup table is a dictionary where the keys are surfaces, modules and materials instances, and the values, an integer index.

pop_module(*module*)
Removes a module.

Parameters **module** (*Module*) – Module to remove.

read(*fileobj*, *material_lookup*)
Reads a geometry file (`.geo`).

Parameters

- **fileobj** (*file object*) – File object opened with read access.
- **material_lookup** (dict(int, *Material*)) – A lookup table for the materials used in the geometry. Dictionary where the keys are material indexes in the geometry file and the values, *Material* instances.

rotation_deg
Specimen rotation in degrees along the z-axis

tilt_deg
Specimen tilt in degrees along the x-axis

title
Title of the geometry. The title must have less than 61 characters.

write (*fileobj*, *index_lookup=None*)
Writes the geometry file (`.geo`) to create this geometry.

Parameters

- **fileobj** (*file object*) – File object opened with write access.
- **index_lookup** (dict(*GeometryBase*, int), optional) – A lookup table for the surfaces, modules and materials of this geometry. If `None`, the index lookup is generated by the method *indexify*.

Returns lookup table

Return type dict(*GeometryBase*, int)

3.5.3 Surfaces

Definition of surfaces.

class pypenelopetools.pengeom.surface.**SurfaceImplicit**(*coefficients=None*, *description=''*)
Bases: pypenelopetools.pengeom.surface.SurfaceBase

Definition of an implicit surface.

Parameters

- **coefficients** (*dict(str, float) or list(float)*) – Coefficients for the implicit form of the quadratic equation. If the argument is a `dict`, the keys are the names of coefficient (e.g. `xx`) and the values the coefficient values. If the argument is a `list`, the list must contain 10 values, one for each coefficient.
- **description** (*str*) – Description of the surface

coefficients
(dict(str, float) or list(float)) – Coefficients for the implicit form of the quadratic equation. If the value is a `dict`, the keys are the names of coefficient (e.g. `xx`) and the values the coefficient values. If the argument is a `list`, the list must contain 10 values, one for each coefficient.

Examples

```
>>> s = Surface()
>>> s.coefficients = {'xx': 0.0, 'xy': 0.0, 'xz': 0.0, 'yy': 0.0, 'yz': 0.0,
↪'zz': 0.0, 'x': 0.0, 'y': 0.0, 'z': 0.0, '0': 0.0}
```

(continues on next page)

(continued from previous page)

```
>>> s.coefficients = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
>>> s.coefficients = {'xx': 1.0, 'xy': 1.0}
```

description
str – Description of the geometry object.

rotation
Rotation – Rotation of the surface.

shift
Shift – Shift/translation of the surface.

class pypenelopetools.pengeom.surface.**SurfaceReduced**(*indices=None*, *description=''*)
Bases: pypenelopetools.pengeom.surface.SurfaceBase

Definition of a reduced/explicit surface.

Parameters

- **indices** (*tuple(int)*) – Indices for the explicit form of the quadratic equation. Indices are 5 integers (-1, 0 or 1) defining the surface.
- **description** (*str*) – Description of the surface

description
str – Description of the geometry object.

indices
(tuple(int)) – Indices for the explicit form of the quadratic equation. Indices are 5 integers (-1, 0 or 1) defining the surface.

rotation
Rotation – Rotation of the surface.

scale
Scale – Scaling of the surface.

shift
Shift – Shift/translation of the surface.

pypenelopetools.pengeom.surface.**xplane**(*x_cm*)
Returns a surface for a plane X=x.

Parameters **x_cm** (*float*) – Intercept on the x-axis (in cm).

Returns *SurfaceReduced*

pypenelopetools.pengeom.surface.**yplane**(*y_cm*)
Returns a surface for a plane Y=y.

Parameters **y_cm** (*float*) – Intercept on the y-axis (in cm).

Returns *SurfaceReduced*

pypenelopetools.pengeom.surface.**zplane**(*z_cm*)
Returns a surface for a plane Z=z.

Parameters **z_cm** (*float*) – Intercept on the z-axis (in cm).

Returns *SurfaceReduced*

pypenelopetools.pengeom.surface.**cylinder**(*radius_cm*, *axis='x'*)
Returns a surface for a cylinder along *axis* with *radius*.

Parameters

- **radius_cm** (*float*) – Radius of the cylinder (in cm).
- **axis** (*str*) – Axis of the cylinder, either `x`, `y` or `z`.

Returns *SurfaceReduced*

pypenelopetools.pengeom.surface.**sphere**(*radius_cm*)

Returns a surface for a sphere or *radius*.

Parameters **radius_cm** (*float*) – Radius of the sphere (in cm).

Returns *SurfaceReduced*

3.5.4 Module

Definition of module.

class pypenelopetools.pengeom.module.**Module**(*material=None*, *description=''*)

Bases: *pypenelopetools.pengeom.mixin.DescriptionMixin*, *pypenelopetools.pengeom.mixin.ModuleMixin*, *pypenelopetools.pengeom.base.GeometryBase*

Definition of a module.

Parameters

- **material** (*Material*, optional) – Material associated with this module. If `None`, the material is set to *VACUUM*.
- **description** (*str*) – Description of the module

add_surface(*surface*, *pointer*)

Adds a surface.

Parameters

- **surface** (*SurfaceImplicit* or *SurfaceReduced*) – Surface to add.
- **pointer** (*SidePointer*) – Whether the surface is pointing in the positive or negative direction.

clear_surfaces()

Clear all surfaces.

get_surface_pointer(*surface*)

Returns the surface pointer for the specified surface.

Parameters **surface** (*SurfaceImplicit* or *SurfaceReduced*) – Surface of interest.

Returns Side pointer.

Return type *SidePointer*

get_surfaces()

Returns All surfaces.

Return type tuple

pop_surface(*surface*)

Removes a surface.

Parameters **surface** (*SurfaceImplicit* or *SurfaceReduced*) – Surface to remove.

rotation
Rotation – Rotation of the module.

shift
Shift – Shift/translation of the module.

class pypenelopetools.pengeom.module.**SidePointer**
Bases: enum.IntEnum

Whether the surface is pointing in the positive or negative direction.

NEGATIVE = -1
Negative direction.

POSITIVE = 1
Positive direction.

3.6 PENMAIN

To be written…

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

Symbols

A

B

C

D

E

F

G

I

J

K

L

M

N

O

P

R

S