serpentTools Documentation

Release 1.0b0+58.g91e7aac.dirty

Andrew Johnson, Dan Kotlyar

Dec 12, 2017

CONTENTS:

CHAPTER

ONE

SERPENT-TOOLS

A suite of parsers designed to make interacting with SERPENT[1] output files simple and flawless.

The SERPENT Monte Carlo code is developed by VTT Technical Research Centre of Finland, Ltd. More information, including distribution and licensing of SERPENT can be found at montecarlo.vtt.fi

1.1 Documentation

A pdf version of the manual can be found in the docs directory.

1. serpent-tools
2. Contributing
3. Examples
4. Developer's Guide
5. Api

1.2 Issues

If you have issues installing the project, find a bug, or want to add a feature, the GitHub issue page is the best place to do that.

1.3 Contributors

Here are all the wonderful people that helped make this project happen

- Andrew Johnson
- Dr. Dan Kotlyar
- Stefano Terlizzi

[1] Leppanen, J. et al. (2015) "The Serpent Monte Carlo code: Status, development and applications in 2013." Ann. Nucl. Energy, 82 (2015) 142-150

1.4 References

The Annals of Nuclear Energy article should be cited for all work <<<<<<< HEAD using `SERPENT`. If you wish to cite this project, please cite as

```
url{@serpentTools
    author = {Andrew Johnson and Dan Kotlyar},
    title = {serpentTools: A suite of parsers designed to make interacting with␣
↪SERPENT outputs simple and flawless},
    url = {https://github.com/CORE-GATECH-GROUP/serpent-tools},
    year = {2017}
}
```

1.5 Installation

The `serpentTools` package can be downloaded either as a git repository or as a zipped file. Both can be obtained through the `Clone or download` option at the serpent-tools GitHub.

Once the repository has been downloaded or extracted from zip, the package can be installed with:

```
cd serpentTools
python setup.py install
python setup.py test
```

Installing with setuptools is preferred over the standard `distutils` module. `setuptools` can be installed with `pip` as:

```
pip install -U setuptools
```

Installing in this manner ensures that the supporting packages, like `numpy` are installed and up to date.

1.6 License

MIT License

Copyright (c) 2017 Andrew Johnson, Dan Kotlyar, GTRC

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER

TWO

CONTRIBUTING

2.1 Contributing

First, thanks for your interest in contributing to this project! This document should help us expedite the process of reviewing issues and pull requests. For a quick look at all the issues that are up for grabs, take a look at the current unclaimed issues. If you claim an issue, use the `Assignees` setting to let us know that you've got it!

2.1.1 Scope

The scope of this project is to simplify and expedite analysis stemming from `SERPENT` outputs. In the future we may expand this project to expand to interacting heavily with input files, but that is currently beyond the scope of this project. Any and all issues, features and pull requests will be examined through this scope.

2.1.2 Issues

The goal for this project is to become the de facto method for processing `SERPENT` outputs and, if you're looking at this, there is some way we can improve. The GitHub issue tracker is the preferred way to post bug reports and feature requests.

Bug Reports

The more information given, the quicker we can reproduce and hopefully resolve the issue. Please see *Issue Template* for a template that should be used for reporting bugs. One of the developers will add a bug label and start moving to resolve the issue posthaste.

Feature Requests

We are very interested in adding functionality from the SERPENT community! Requests can be done through the issue tracker as well. You can create an issue on the issue tracker with `[Feature]` or `[Request]` in the title. Describe what you would like to add, some expected results, and the purpose behind the feature. The development team will apply an enhancement label and proceed accordingly.

2.1.3 Pull Requests

Pull requests are how we review, approve, and incorporate changes into the `develop` and `master` branches. If you have code you want to contribute, please look at the content in the *Developer's Guide* for things like *Pull Request Checklist*, *Coding Style*, and more.

When your content is ready for the pull request, follow the *Pull Request Template* and make a request! Someone of the core development team will review the changes according to the criteria above and make changes and/or approve for merging!

2.2 Issue Template

Summary of issue

Code for reproducing the issue

Actual outcome including console output and error traceback if applicable

Expected outcome - why this is an issue

- Version from `serpentTools.__version__`
- Python version - `python --version`
- IPython or Jupyter version if applicable

2.3 Pull Request Template

Title: Short descriptive title summarizing the request

Body: Directly address the issue that this pull request resolves or addresses:

```
Fixes #99 - some fatal bug that caused things to fail poorly
```

Include a full description of what is included in this PR, what has changed.

CHAPTER

THREE

EXAMPLES

3.1 User Control

The `serpentTools` package is designed to, without intervention, be able to store all the data contained in each of the various output files. However, the `serpentTools.settings` module grants great flexibility to the user over what data is obtained through the rc class. This notebook will provide as an intro into using this class.

3.1.1 Basic Usage

```
>>> import serpentTools
>>> from serpentTools.settings import rc, defaultSettings
INFO    : serpentTools: Using version 1.0b0+24.g23e6eac.dirty
```

Below are the default values for each setting available

```
>>> for setting in sorted(defaultSettings.keys()):
>>>     print(setting)
>>>     for key in defaultSettings[setting]:
>>>         print('\t', key, '-', defaultSettings[setting][key])
depletion.materialVariables
     default - []
     description - Names of variables to store. Empty list -> all variables.
     type - <class 'list'>
depletion.materials
     default - []
     description - Names of materials to store. Empty list -> all materials.
     type - <class 'list'>
depletion.metadataKeys
     default - ['ZAI', 'NAMES', 'DAYS', 'BU']
     description - Non-material data to store, i.e. zai, isotope names, burnup␣
↪schedule, etc.
     type - <class 'list'>
     options - default
depletion.processTotal
     default - True
     description - Option to store the depletion data from the TOT block
     type - <class 'bool'>
serpentVersion
     default - 2.1.29
     description - Version of SERPENT
     type - <class 'str'>
     options - ['2.1.29']
```

```
verbosity
     default - warning
     type - <class 'str'>
     description - Set the level of errors to be shown.
     updater - <function updateLevel at 0x000001B7F3DD6598>
     options - ['critical', 'error', 'warning', 'info', 'debug']
xs.variableExtras
     default - []
     description - Full SERPENT name of variables to be read
     type - <class 'list'>
xs.variableGroups
     default - []
     description - Name of variable groups from variables.yaml to be expanded into␣
↪SERPENT variable to be stored
     type - <class 'list'>
```

Settings such as `depletion.materialVariables` are specific for the `DepletionReader`, while settings that are led with `xs` are sent to the `ResultsReader` and `BranchingReader`, as well as their specific settings. The `rc` class acts as a dictionary, and updating a value is as simple as

```
>> rc['verbosity'] = 'debug'
DEBUG   : serpentTools: Updated setting verbosity to debug
```

The `rc` object automatically checks to make sure the value is of the correct type, and is an allowable option, if given.

```
>>> try:
>>>     rc['depletion.metadataKeys'] = False
>>> except TypeError as te:
>>>     print(te)
Setting depletion.metadataKeys should be of type <class 'list'>, not <class 'bool'>
>>> try:
>>>     rc['serpentVersion'] = '1.2.3'
>>> except KeyError as ke:
>>>     print(ke)
"Setting serpentVersion is
1.2.3
and not one of the allowed options:
['2.1.29']"
```

The `rc` object can also be used inside a context manager to revert changes.

```
>>> with rc:
>>>     rc['depletion.metadataKeys'] = ['ZAI', 'BU']
>>>
>>> rc['depletion.metadataKeys']
>>> rc['verbosity'] = 'info'
DEBUG   : serpentTools: Updated setting depletion.metadataKeys to ['ZAI', 'BU']
DEBUG   : serpentTools: Updated setting depletion.metadataKeys to ['ZAI', 'NAMES',
↪'DAYS', 'BU']
['ZAI', 'NAMES', 'DAYS', 'BU']
```

Group Constant Variables

Two settings control what group constant data and what variables are extracted from the results and coefficient files.

1. `xs.variableExtras`: Full `SERPENT_STYLE` variable names, i.e. `INF_TOT`, `FISSION_PRODUCT_DECAY_HEAT`

2. `xs.variableGroups`: Select keywords that represent blocks of common variables

These variable groups are stored in `serpentTools/variables.yaml` and rely upon the `SERPENT` version to properly expand the groups.

```
>>> rc['serpentVersion']
'2.1.29'
>>> rc['xs.variableGroups'] = ['kinetics', 'xs', 'diffusion']
>>> rc['xs.variableExtras'] = ['XS_DATA_FILE_PATH']
>>> varSet = rc.expandVariables()
>>> print(sorted(varSet))
['ABS', 'ADJ_IFP_ANA_BETA_EFF', 'ADJ_IFP_ANA_LAMBDA', 'ADJ_IFP_GEN_TIME',
 'ADJ_IFP_IMP_BETA_EFF', 'ADJ_IFP_IMP_LAMBDA', 'ADJ_IFP_LIFETIME',
 'ADJ_IFP_ROSSI_ALPHA', 'ADJ_INV_SPD', 'ADJ_MEULEKAMP_BETA_EFF',
 'ADJ_MEULEKAMP_LAMBDA', 'ADJ_NAUCHI_BETA_EFF', 'ADJ_NAUCHI_GEN_TIME',
 'ADJ_NAUCHI_LAMBDA', 'ADJ_NAUCHI_LIFETIME', 'ADJ_PERT_BETA_EFF',
 'ADJ_PERT_GEN_TIME', 'ADJ_PERT_LIFETIME', 'ADJ_PERT_ROSSI_ALPHA', 'CAPT',
 'CHID', 'CHIP', 'CHIT', 'CMM_DIFFCOEF', 'CMM_DIFFCOEF_X', 'CMM_DIFFCOEF_Y',
 'CMM_DIFFCOEF_Z', 'CMM_TRANSPXS', 'CMM_TRANSPXS_X', 'CMM_TRANSPXS_Y',
 'CMM_TRANSPXS_Z', 'DIFFCOEF', 'FISS', 'FWD_ANA_BETA_ZERO',
 'FWD_ANA_LAMBDA', 'INVV', 'KAPPA', 'NSF', 'NUBAR', 'RABSXS', 'REMXS',
 'S0', 'S1', 'S2', 'S3', 'S4', 'S5', 'S6', 'S7', 'SCATT0', 'SCATT1',
 'SCATT2', 'SCATT3', 'SCATT4', 'SCATT5', 'SCATT6', 'SCATT7', 'TOT',
 'TRANSPXS', 'XS_DATA_FILE_PATH']
```

However, one might see that the full group constant cross sections are not present in this set

```
>>> assert 'INF_SCATT3' not in varSet
```

This is because two additional settings instruct the *`BranchingReader`* and *`ResultsReader`* to obtain infinite medium and leakage-corrected cross sections: `xs.getInfXS` and `xs.getB1XS`, respectively. By default, `xs.getInfXS` and `xs.getB1XS` default to True. This, in conjunction with leaving the `xs.variableGroups` and `xs.variableExtras` settings to empty lists, instructs these readers to obtain all the data present in their respective files.

See the *Branching Reader* example for more information on using these settings to control scraped data.

3.2 Branching Reader

This notebook demonstrates the capability of the serpentTools package to read branching coefficient files. The format of these files is structured to iterate over:

1. Branch states, e.g. burnup, material properties
2. Homogenized universes
3. Group constant data

The output files are described in more detail on the SERPENT Wiki

3.2.1 Basic Operation

The simplest way to read these files is using the *`serpentTools.parsers.read()`* function

Note: Without modifying the settings, the *BranchingReader* assumes that all group constant data is presented without the associated uncertainties. See *User Control* for examples on the various ways to control operation

```
>>> import serpentTools
>>> branchFile = 'demo.coe'
INFO    : serpentTools: Using version 1.0b0+58.g91e7aac.dirty
>>> r0 = serpentTools.read(branchFile)
INFO    : serpentTools: Inferred reader for demo.coe: BranchingReader
INFO    : serpentTools: Preparing to read demo.coe
INFO    : serpentTools: Done reading branching file
```

The branches are stored in custom *BranchContainer* objects in the `branches` dictionary

```
>>> r0.branches
{('Fhi', 'Bhi', 'His'):
    <serpentTools.objects.containers.BranchContainer at 0x14d0106c0f0>,
 ('Fhi', 'Blo', 'His'):
    <serpentTools.objects.containers.BranchContainer at 0x14d01063d30>,
 ('Fhi', 'nom', 'His'):
    <serpentTools.objects.containers.BranchContainer at 0x14d0105e908>,
 ('nom', 'Bhi', 'His'):
    <serpentTools.objects.containers.BranchContainer at 0x14d01068748>,
 ('nom', 'Blo', 'His'):
    <serpentTools.objects.containers.BranchContainer at 0x14d01063320>,
 ('nom', 'nom', 'His'):
    <serpentTools.objects.containers.BranchContainer at 0x14d01053ef0>}
```

Here, the keys are tuples of strings indicating what perturbations/branch states were applied for each `SERPENT` solution. Examining a particular case

```
>>> b0 = r0.branches['Fhi', 'Bhi', 'His']
>>> print(b0)
<BranchContainer for Fhi, Bhi, His from demo.coe>
```

`SERPENT` allows the user to define variables for each branch through:

```
var V1_name V1_value
```

cards. These are stored in the `stateData` attribute

```
>>> b0.stateData
{'BOR': '1000',
 'DATE': '17/10/18',
 'TFU': '1200',
 'TIME': '10:26:48',
 'VERSION': '2.1.29'}
```

The keys `'DATE'`, `'TIME'`, and `'VERSION'` are included by default in the output, while the `'BOR'` and `'TFU'` have been defined for this branch. Branch name `'Fhi'` → higher fuel temperature → `'TFU'` = 1200 K

Group Constant Data

Note: Group constants are converted from `SERPENT_STYLE` to `mixedCase` to fit the overall style of the project.

The *BranchContainer* stores group constant data in *HomogUniv* objects in the `universes` dictionary

```
>>> b0.universes
{(0, 0.0, 1): <serpentTools.objects.containers.HomogUniv at 0x14d010689e8>,
 (0, 1.0, 2): <serpentTools.objects.containers.HomogUniv at 0x14d0106c320>,
 (0, 5.0, 3): <serpentTools.objects.containers.HomogUniv at 0x14d0106c4a8>,
 (0, 10.0, 4): <serpentTools.objects.containers.HomogUniv at 0x14d0106c630>,
 (0, 50.0, 5): <serpentTools.objects.containers.HomogUniv at 0x14d0106c668>}
```

The keys here are vectors indicating the universe ID, burnup [MWd/kgU], and burnup index corresponding to the point in the burnup schedule. These universes can be obtained by indexing this dictionary, or by using the *getUniv()* method

```
>>> univ0 = b0.universes[0, 1, 2]
>>> print(univ0)
>>> print(univ0.name)
>>> print(univ0.bu)
>>> print(univ0.step)
>>> print(univ0.day)
<HomogUniv from demo.coe>
0
1.0
2
0
>>> univ1 = b0.getUniv(0, burnup=1)
>>> univ2 = b0.getUniv(0, index=2)
>>> assert univ0 is univ1 is univ2
```

Since the coefficient files do not store the day value of burnup, all *HomogUniv* objects created by the *BranchContainer* default to day zero.

Group constant data is stored in five dictionaries:

1. `infExp`: Expected values for infinite medium group constants
2. `infUnc`: Relative uncertainties for infinite medium group constants
3. `b1Exp`: Expected values for leakge-corrected group constants
4. `b1Unc`: Relative uncertainties for leakge-corrected group constants
5. `metaData`: items that do not fit the in the above categories

```
>>> univ0.infExp
{'infFiss': array([ 0.00286484,  0.0577559 ]),
 'infS0': array([ 0.501168  ,  0.0180394 ,  0.00155388,  1.2875    ]),
 'infS1': array([ 0.247105  ,  0.00535317,  0.00073696,  0.352806  ]),
 'infScatt0': array([ 0.519208,  1.28905 ]),
 'infScatt1': array([ 0.252459,  0.353543]),
 'infTot': array([ 0.529552,  1.38805 ])}
>>> univ0.infUnc
{}
>>> univ0.b1Exp
{}
>>> univ0.metaData
{'macroE': array([], dtype=float64), 'macroNg': array([], dtype=float64)}
```

Group constants and their associated uncertainties can be obtained using the *get()* method.

```
>>> univ0.get('infFiss')
array([ 0.00286484,  0.0577559 ])
>>> try:
>>>     univ0.get('infS0', uncertainty=True)
>>> except KeyError as ke:  # no uncertainties here
>>>     print(str(ke))
'Variable infS0 absent from uncertainty dictionary'
>>> univ0.get('macroE')
array([], dtype=float64)
```

3.2.2 Iteration

The branching reader has a *iterBranches()* method that works to yield branch names and their associated *BranchContainer* objects. This can be used to efficiently iterate over all the branches presented in the file.

```
>>> for names, branch in r0.iterBranches():
>>>    print(names, branch)
('nom', 'nom', 'His') <BranchContainer for nom, nom, His from demo.coe>
('Fhi', 'nom', 'His') <BranchContainer for Fhi, nom, His from demo.coe>
('nom', 'Blo', 'His') <BranchContainer for nom, Blo, His from demo.coe>
('Fhi', 'Blo', 'His') <BranchContainer for Fhi, Blo, His from demo.coe>
('nom', 'Bhi', 'His') <BranchContainer for nom, Bhi, His from demo.coe>
('Fhi', 'Bhi', 'His') <BranchContainer for Fhi, Bhi, His from demo.coe>
```

3.2.3 User Control

The SERPENT set coefpara card already restricts the data present in the coeffient file to user control, and the *BranchingReader* includes similar control. Below are the various settings that the *BranchingReader* uses to read and process coefficient files.

```
>>> import six
>>> from serpentTools.settings import rc
>>> from serpentTools.settings import rc, defaultSettings
>>> for setting in defaultSettings:
>>>     if 'xs' in setting or 'branching' in setting:
>>>         print(setting)
>>>         for k, v in six.iteritems(defaultSettings[setting]):
>>>             print('\t', k+':', v)
branching.areUncsPresent
     default: False
     type: <class 'bool'>
     description: True if the values in the .coe file contain uncertainties
branching.intVariables
     default: []
     description: Name of state data variables to convert to integers for
     each branch
     type: <class 'list'>
branching.floatVariables
     default: []
     description: Names of state data variables to convert to floats for
     each branch
     type: <class 'list'>
xs.getInfXS
     default: True
```

```
     description: If true, store the infinite medium cross sections.
     type: <class 'bool'>
xs.getB1XS
     default: True
     description: If true, store the critical leakage cross sections.
     type: <class 'bool'>
xs.variableGroups
     default: []
     description: Name of variable groups from variables.yaml to be expanded
      into SERPENT variable to be stored
     type: <class 'list'>
xs.variableExtras
     default: []
     description: Full SERPENT name of variables to be read
     type: <class 'list'>
```

In our example above, the `BOR` and `TFU` variables represented boron concentration and fuel temperature, and can easily be cast into numeric values using the `branching.intVariables` and `brancing.floatVariables` settings. From the previous example, we see that the default action is to store all state data variables as strings.

```
>>> assert isinstance(b0.stateData['BOR'], str)
```

As demonstrated in the *Group Constant Variables* example, use of `xs.variableGroups` and `xs.variableExtras` controls what data is stored on the *HomogUniv* objects. By default, all variables present in the coefficient file are stored.

```
>>> rc['branching.floatVariables'] = ['BOR']
>>> rc['branching.intVariables'] = ['TFU']
>>> with rc:
>>>     rc['xs.variableExtras'] = ['INF_TOT', 'INF_SCATT0']
>>>     r1 = serpentTools.read(branchFile)
INFO    : serpentTools: Inferred reader for demo.coe: BranchingReader
INFO    : serpentTools: Preparing to read demo.coe
INFO    : serpentTools: Done reading branching file
>>> b1 = r1.branches['Fhi', 'Bhi', 'His']
>>> b1.stateData
{'BOR': 1000.0,
 'DATE': '17/10/18',
 'TFU': 1200,
 'TIME': '10:26:48',
 'VERSION': '2.1.29'}
>>> assert isinstance(b1.stateData['BOR'], float)
>>> assert isinstance(b1.stateData['TFU'], int)
```

Inspecting the data stored on the homogenized universes reveals only the variables explicitly requested are present

```
>>> univ4 = b1.getUniv(0, 0)
>>> univ4.infExp
{'infScatt0': array([ 0.519337,  1.28894 ]),
 'infTot': array([ 0.529682,  1.38649 ])}
```

3.2.4 Conclusion

The *BranchingReader* is capable of reading coefficient files created by the `SERPENT` automated branching process. The data is stored according to the branch parameters, universe information, and burnup. This reader also

supports user control of the processing by selecting what state parameters should be converted from strings to numeric types, and further down-selection of data.

A more complicated coefficient file, with multiple universes and more varied coefficients, will be coming shortly - Issue #64

3.3 DepletionReader

3.3.1 Basic Operation

SERPENT produces a burned material file, containing the evolution of material properties through burnup for all burned materials present in the problem. The *DepletionReader* is capable of reading this file, and storing the data inside *DepletedMaterial* objects. Each such object has methods and attributes that should ease analysis.

```
>>> import six
>>> import serpentTools
>>> from serpentTools.settings import rc
INFO    : serpentTools: Using version 1.0b0+24.g23e6eac.dirty
>>> depFile = 'demo_dep.m'
>>> dep = serpentTools.read(depFile)
INFO    : serpentTools: Inferred reader for demo_dep.m: DepletionReader
INFO    : serpentTools: Preparing to read demo_dep.m
INFO    : serpentTools: Done reading depletion file
```

The materials read in from the file are stored in the `materials` dictionary, where the keys represent the name of specific materials, and the corresponding values are the depleted material.

```
>>> dep.materials
{'bglass0': <serpentTools.objects.materials.DepletedMaterial at 0x23905154668>,
 'fuel0': <serpentTools.objects.materials.DepletedMaterial at 0x2390578eeb8>,
 'total': <serpentTools.objects.materials.DepletedMaterial at 0x2390579e978>}
```

Metadata, such as the isotopic vector and depletion schedule are also present inside the reader

```
>>> dep.metadata.keys()
dict_keys(['zai', 'burnup', 'names', 'days'])
>>> dep.metadata['burnup']
array([ 0.  ,  0.02,  0.04,  ...,  1.36,  1.38,  1.4 ,  1.42])
>>> dep.metadata['names']
['Xe135', 'I135', 'U234', 'U235', 'U236', 'U238', 'Pu238',
 'Pu239',..., 'lost', 'total']
```

3.3.2 DepletedMaterial

As mentioned before, all the material data is stored inside these *DepletedMaterial* objects. These objects share access to the metadata of the reader as well.

```
>>> fuel = dep.materials['fuel0']
>>> fuel.burnup
array([ 0.  ,  0.02,  0.04,  ...,  1.36,  1.38,  1.4 ,  1.42])
>>> fuel.days is dep.metadata['days']
True
```

All of the variables present in the depletion file for this material are present, stored in the `data` dictionary. A few properties commonly used are accessible as attributes as well.

```
>>> fuel.data.keys()
dict_keys(['a', 'adens', 'burnup', 'gsrc', ..., 'volume'])
>>> fuel.adens
array([[  0.00000000e+00,   2.43591000e-09,   4.03796000e-09, ...,
          4.70133000e-09,   4.70023000e-09,   4.88855000e-09],
       [  0.00000000e+00,   6.06880000e-09,   8.11783000e-09, ...,
          8.05991000e-09,   8.96359000e-09,   9.28554000e-09],
       [  4.48538000e-06,   4.48486000e-06,   4.48432000e-06, ...,
          4.44726000e-06,   4.44668000e-06,   4.44611000e-06],
       ...,
       [  0.00000000e+00,   3.03589000e-11,   7.38022000e-11, ...,
          1.62829000e-09,   1.63566000e-09,   1.64477000e-09],
       [  0.00000000e+00,   1.15541000e-14,   2.38378000e-14, ...,
          8.60736000e-13,   8.73669000e-13,   8.86782000e-13],
       [  6.88332000e-02,   6.88334000e-02,   6.88336000e-02, ...,
          6.88455000e-02,   6.88457000e-02,   6.88459000e-02]])
```

Similar to the original file, the rows of the matrix correspond to positions in the isotopic vector, and the columns correspond to positions in burnup/day vectors.

```
>>> fuel.mdens.shape  # rows, columns
(34, 72)
>>> fuel.burnup.shape
(72,)
>>> len(fuel.names)
34
```

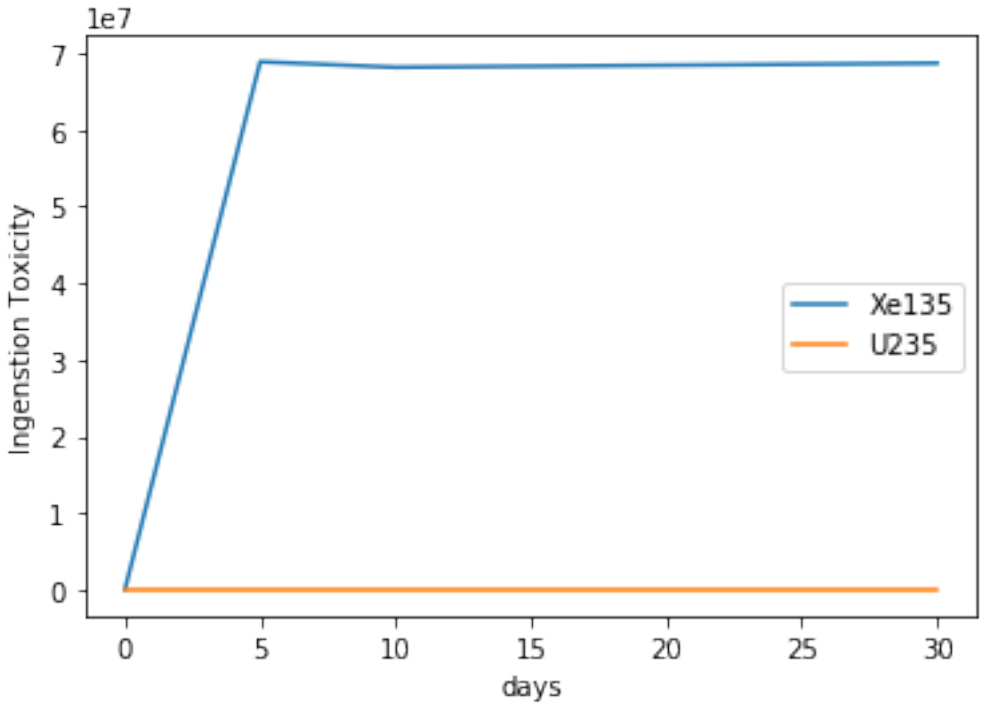
3.3.3 Data Retrieval

At the heart of the *DepletedMaterial* is the *getValues()* method. This method acts as an slicing mechanism that returns data for a select number of isotopes at select points in time.

```
>>> dayPoints = [0, 5, 10, 30]
>>> iso = ['Xe135', 'U235']
>>> vals = fuel.getValues('days', 'a', dayPoints, iso)
>>> print(vals.shape)
(2, 4)
>>> print(vals)
[[  0.00000000e+00   3.28067000e+14   3.24606000e+14   3.27144000e+14]
 [  5.36447000e+07   5.34519000e+07   5.32499000e+07   5.24766000e+07]]
```

The *DepletedMaterial* uses this slicing for the built-in *plot()* method

```
>>> fuel.plot('days', 'ingTox', dayPoints, iso,
              ylabel='Ingenstion Toxicity');
```



3.3.4 Limitations

Currently, the `DepletionReader` cannot catch materials with underscore in the name, due to variables like `ING_TOX` also containing an underscore. Issue #58

3.3.5 Settings

The `DepletionReader` also has a collection of settings to control what data is stored. If none of these settings are modified, the default is to store all the data from the output file.

```
>>> from serpentTools.settings import rc, defaultSettings
>>> for setting in defaultSettings:
>>>     if 'depletion' in setting:
>>>         print(setting)
>>>         for k, v in six.iteritems(defaultSettings[setting]):
>>>             print('\t', k, v)
depletion.materials
     type <class 'list'>
     description Names of materials to store. Empty list -> all materials.
     default []
depletion.processTotal
    type <class 'bool'>
    description Option to store the depletion data from the TOT block
    default True
depletion.materialVariables
    type <class 'list'>
    description Names of variables to store. Empty list -> all variables.
    default []
depletion.metadataKeys
    type <class 'list'>
```

```
    description Non-material data to store, i.e. zai, isotope names, burnup schedule,␣
↪etc.
    options default
    default ['ZAI', 'NAMES', 'DAYS', 'BU']
```

Below is an example of configuring a `DepletionReader` that only stores the burnup days, and atomic density for all materials that begin with `bglass` followed by at least one integer.

Note: Creating the `DepletionReader` in this manner is functionally equivalent to `serpentTools.read(depFile)`

```
>>> rc['depletion.processTotal'] = False
>>> rc['depletion.metadataKeys'] = ['BU']
>>> rc['depletion.materialVariables'] = ['ADENS']
>>> rc['depletion.materials'] = [r'bglass\d+']
>>>
>>> bgReader = serpentTools.parsers.DepletionReader(depFile)
>>> bgReader.read()
INFO    : serpentTools: Preparing to read demo_dep.m
INFO    : serpentTools: Done reading depletion file
>>> bgReader.materials
{'bglass0': <serpentTools.objects.materials.DepletedMaterial at 0x239057dcb00>}
>>> bglass = bgReader.materials['bglass0']
>>> bglass.data
{'adens': array([[ 0.       ,  0.       ,  0.       , ...,  0.       ,  0.       ,
          0.       ],
        [ 0.       ,  0.       ,  0.       , ...,  0.       ,  0.       ,
          0.       ],
        [ 0.       ,  0.       ,  0.       , ...,  0.       ,  0.       ,
          0.       ],
        ...,
        [ 0.       ,  0.       ,  0.       , ...,  0.       ,  0.       ,
          0.       ],
        [ 0.       ,  0.       ,  0.       , ...,  0.       ,  0.       ,
          0.       ],
        [ 0.0715841,  0.0715843,  0.0715845, ...,  0.0715968,  0.0715969,
          0.0715971]])}
>>> bglass.data.keys()
dict_keys(['adens'])
```

3.3.6 Conclusion

The `DepletionReader` is capable of reading and storing all the data from the SERPENT burned materials file. Upon reading, the reader creates custom `DepletedMaterial` objects that are responsible for storing and retrieving the data. These objects also have a handy `plot` method for quick analysis. Use of the `rc` settings control object allows increased control over the data selected from the output file.

3.3.7 References

1. J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho. "The Serpent Monte Carlo code: Status, development and applications in 2013." Ann. Nucl. Energy, 82 (2015) 142-150

CHAPTER

FOUR

DEVELOPER'S GUIDE

4.1 Coding Style

For the most part, this project follows the **PEP 8** standard with a few differences. Some points are included here

- 79 characters per line
- Four spaces per indentation level
- Avoiding extraneous whitespace:

```
Yes: spam(ham[1], {eggs: 2})
No:  spam( ham[ 1 ], { eggs: 2 } )
```

Some of the specific style points for this project are included below

- mixedCase for variables, methods, and functions
- CamelCase for classes:

```
class DemoClass(object):

    def doSomething(self, arg0, longerArgumentName):
        pass
```

- Directly call the __init__ method from a parent class, e.g.:

```
class MyQueue(list):

    def __init__(self, items):
        list.__init__(self)
        self.extend(items)
```

- Arrange imports in the following order:
  1. imports from the standard library: os, sys, etc.
  2. imports from third party code: numpy, matplotlib, etc.
  3. imports from the serpentTools package
- Longer import paths are preferred to shorter:

```
# yes
from really.long.path.to.a import function
function()
# not preferred
```

```
import really
really.long.path.to.a.function()
```

4.2 Documentation

All public functions, methods, and classes should have adequate documentation through docstrings, examples, or inclusion in the appropriate file in the `docs` directory. Good forethought into documenting the code helps resolve issues and, in the case of docstrings, helps produce a full manual.

4.2.1 Docstrings

Docstrings are defined in **PEP 257** and are characterized by `"""triple double quotes"""`. These can be used to reduce the effort in creating a full manual, and can be viewed through python consoles to give the user insight into what is done, what is required, and what is returned from a particular object. This project uses numpy style docstrings, examples of which are given below

Functions and Methods

Below is the *depmtx()* function annotated using short and longer docstrings:

```
def depmtx(filePath):
    """Return t, no, zai, a, and n1 values from the depmtx file."""
```

or:

```
def depmtx(filePath):
    """
    Read the contents of the ``depmtx`` file and return contents

    .. note::

        If ``scipy`` is not installed, matrix ``A`` will be full.
        This can cause some warnings or errors if sparse or
        non-sparse solvers are used.

    Parameters
    ----------
    fileP: str
        Path to depletion matrix file

    Returns
    -------
    t: float
        Length of time
    n0: numpy.ndarray
        Initial isotopic vector
    zai: numpy.array
        String identifiers for each isotope in ``n0`` and ``n1``
    a: numpy.array or scipy.sparse.csc_matrix
        Decay matrix. Will be sparse if scipy is installed
    n1: numpy.array
        Final isotopic vector
    """
```

Both docstrings indicate what the function does, and what is returned. The latter, while more verbose, is preferred in most cases, for the following reasons. First, far more information is yielded to the reader. Second, the types of the inputs and outputs are given and clear. Some IDEs can obtain the expected types from the docstrings and inform the user if they are using an incorrect type.

More content can be added to the docstring, including

- Raises - Errors/warnings raised by this object
- See Also - follow up information that may be useful
- Yields - If this object is a generator. Similar to Returns

Classes

Classes can have a more lengthy docstring, as they often include attributes to which the class has access. Below is an example of the docstring for the `DepletionReader`:

```
"""
Parser responsible for reading and working with depletion files.

Parameters
----------
filePath: str
    path to the depletion file

Attributes
----------
materials: dict
    Dictionary with material names as keys and the corresponding
    :py:class:`~serpentTools.objects.DepletedMaterial` class
    for that material as values
metadata: dict
    Dictionary with file-wide data names as keys and the
    corresponding dataas values, e.g. 'zai': [list of zai numbers]
settings: dict
    names and values of the settings used to control operations
    of this reader

"""
```

Class docstrings can be added to the class signature, or to the `__init__` method, as:

```
class Demo(object):
    """
    Demonstration class

    Parameters
    ----------
    x: str
        Just a string

    Attributes
    ----------
    capX: str
        Capitalized x
    """
```

or:

```
def __init__(self, x):
    """
    Demonstration class

    Parameters
    ----------
    x: str
        Just a string

    Attributes
    ----------
    capX: str
        Capitalized x
    """
```

Deprecation

If an object is deprecated or will be modified in future versions, then the *deprecated()* and *willChange()* decorators should be applied to the object, and a note should be added to the docstring indicating as much.

4.2.2 Examples

When possible, features should be demonstrated, either through Jupyter notebooks in the `examples/` directory, or with an `Examples` section in the docstring. Specifically, all readers should be demonstrated as Jupyter notebooks that detail the typical usage, user control settings, and examples of how the data is stored and accessed.

4.3 Data Model

This project has two key offerings to the `SERPENT` community.

1. Scripting tools that are capable of interpreting the output files
2. Purpose-built containers for storing, accessing, and utilizing the data in the output files

4.3.1 Readers

The readers are the core of this package, and should have a logical and consistent method for structuring data. Each reader contained in this project should:

1. Be capable of reading the intended output file
2. Yield to the user the data in a logical manner
3. Default to storing all the data without user input
4. Receive input from the user regarding what data to restrict/include
5. Report activity at varying levels, including failure, to the user

The third and fourth points refer to use of the `serpentTools.settings` module to receive user input. The final point refers to use of the logging and messaging functions included in *serpentTools.messages*.

For example, the `DepletionReader` stores data in the following manner:

```
Reader
| - filePath
| - fileMetadata
    | - isotope names
    | - isotope ZAIs
    | - depletion schedule in days and burnup
| - materials
# for each material:
    | - DepletedMaterial-n
        | - name
        | - atomic density
        | - mass density
        | - volume
        | etc
```

Here, the reader primarily creates and stores material objects, and the useful data is stored on these objects. Some files, like the fission matrix and depletion matrix files may not have a structure that naturally favors this object-oriented approach.

4.3.2 Containers

The readers are primarily responsible for creating and populating containers, that are responsible for storing and retrieving data. Just like the `DepletedMaterial` has shortcuts for analysis like `getValues()` and `plot()`, each such container should easily and naturally provide access to the data, and also perform some common analysis on the data.

4.4 Pull Request Checklist

Below is the criteria that will be used in the process of reviewing pull requests (PR):

1. The content of the PR fits within the scope of the project - *Scope*
2. The code included in the PR is written in good pythonic fashion, and follows the style of the project - *Coding Style*
3. The code directly resolves a previously raised issue - *Issues*
4. PR does not cause unit tests and builds to fail
5. Changes are reflected in documentation - *Documentation*

CHAPTER

FIVE

API

5.1 Settings

class serpentTools.settings.**UserSettingsLoader**

Class that stores the active user settings.

clear() → None. Remove all items from D.

copy() → a shallow copy of D

expandVariables()

Extend the keyword groups into lists of serpent variables.

Returns Names of all variables to be scraped

Return type set

fromkeys()

Returns a new dict with keys from iterable and values equal to value.

get(*k*[, *d*]) → D[k] if k in D, else d. d defaults to None.

getReaderSettings(*settingsPreffix*)

Get all module-wide and reader-specific settings.

Parameters **settingsPreffix** (*str or list*) – Name of the specific reader. Will look for settings that lead with `readerName`, e.g. `depletion.metadataKeys` or `xs.variables`

Returns Single level dictionary with `settingName: settingValue` pairs

Return type dict

Raises `KeyError` – If the reader name is not located in the `readers` settings dictionary

items() → a set-like object providing a view on D's items

keys() → a set-like object providing a view on D's keys

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v), remove and return some (key, value) pair as a
2-tuple; but raise KeyError if D is empty.

setValue(*name*, *value*)

Set the value of a specific setting.

Parameters

- **name** (*str*) – Full name of the setting

- **value**(*value to be set*) –

Raises

- KeyError – If the value is not one of the allowable options or if the setting does not match an existing setting
- TypeError – If the value is not of the correct type

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

update([*E*], ***F*) → None. Update D from dict/iterable E and F.
If E is present and has a .keys() method, then does: for k in E: D[k] = E[k] If E is present and lacks a .keys() method, then does: for k, v in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → an object providing a view on D's values

5.2 Messages

System-wide methods for producing status update and errors.

See also:

- https://docs.python.org/2/library/logging.html
- https://www.python.org/dev/peps/pep-0391/

exception serpentTools.messages.**SerpentToolsException**
Base-class for all exceptions in this project

with_traceback()
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

serpentTools.messages.**critical**(*message*)
Log that something has gone horribly wrong.

serpentTools.messages.**debug**(*message*)
Log a debug message.

serpentTools.messages.**deprecated**(*useInstead*)
Decorator that warns that different function should be used instead.

serpentTools.messages.**error**(*message*)
Log that something caused an exception but was supressed.

serpentTools.messages.**info**(*message*)
Log an info message, e.g. status update.

serpentTools.messages.**updateLevel**(*level*)
Set the level of the logger.

serpentTools.messages.**warning**(*message*)
Log a warning that something that could go wrong or should be avoided.

serpentTools.messages.**willChange**(*changeMsg*)
Decorator that warns that some functionality may change.

5.3 Parser Module

The main module associated with reading SERPENT files. For information on each individual reader, please see the dedicated file listed below

Reader	Hyperlink
Branching	*Branching Reader*
Bumat	*Bumat Reader*
Depletion	*Depletion Reader*
Detector	*Detector Reader*
FissionMatrix	*Fission Matrix Reader*
Results	*Results Reader*

serpentTools.parsers.**inferReader**(*filePath*)

Attempt to infer the correct reader type.

Parameters **filePath** (*str*) – File to be read.

Raises SerpentToolsException – If a reader cannot be inferred

serpentTools.parsers.**read**(*filePath*, *reader='infer'*)

Simple entry point to read a file and obtain the processed reader.

Note: If you know the type of reader you will be using, it is best to either pass in the string argument, or directly use the appropriate reader class

---

Parameters

- **filePath** (*str*) – Path to the file to be reader
- **reader** (*str or callable*) – Type of reader to use. If a string is given, then the actions described below will happen. If callable, then that function will be used with the file path as the first argument.

String argument	Action
infer	Infer the correct reader based on the file
dep	DepletionReader
branch	BranchingReader
det	DetectorReader
results	ResultsReader
bumat	BumatReader
fission	FissionMatrixReader

Returns Correct subclass corresponding to the file type

Return type serpentTools.objects.readers.BaseReader

Raises

- AttributeError – If the object created by the reader through reader(filePath) does not have a read method.
- SerpentToolsException – If the reader could not be inferred or if the requested reader string is not supported
- NotImplementedError – This has the ability to load in readers that may not be complete, and thus the read method may raise this error.

serpentTools.parsers.**depmtx**(*fileP*)
Read the contents of the depmtx file and return contents

Note: If scipy is not installed, matrix A will be full. This can cause some warnings or errors if sparse or non-sparse solvers are used.

Parameters **fileP** (*str*) – Path to depletion matrix file

Returns

- **t** (*float*) – Length of time
- **n0** (*numpy.ndarray*) – Initial isotopic vector
- **zai** (*numpy.array*) – String identifiers for each isotope in n0 and n1
- **a** (*numpy.array or scipy.sparse.csc_matrix*) – Decay matrix. Will be sparse if scipy is installed
- **n1** (*numpy.array*) – Final isotopic vector

5.4 Parsing Engines

The classes contained in serpentTools/engines.py are part of the drewtils v0.1.9 package and are provided under the following license:

The MIT License (MIT)

Copyright (c) Andrew Johnson, 2017

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

These are designed to facilitate the parsing of files with a regular structure. For example, the depletion files all contain "chunks" of data that are separated by empty lines. Each chunk leads off with either the name of the material and associated variable, or the metadata, e.g. ZAI, DAYS. These parsers help break up these files into more digestible pieces.

Note: For developers, it is not required that these classes be used. These are bundled with this project to eliminate the need to install extra packages. Some of the readers, like the *BranchingReader* are not well suited for this type of parsing.

---

class serpentTools.engines.**KeywordParser**(*filePath*, *keys*, *separators=None*, *eof=''*)
Class for parsing a file for chunks separated by various keywords.

Parameters

- **filePath** (*str*) – Object to be read. Any object with `read` and `close` methods
- **keys** (*Iterable*) – List of keywords/phrases that will indicate the start of a chunk
- **separators** (*Iterable or None*) – List of additional phrases that can separate two chunks. If not given, will default to empty line `'\n'`.
- **eof** (*str*) – String to indicate the end of the file

line
str – Most recently read line

parse()
Parse the file and return a list of keyword blocks.

Returns List of key word argument chunks.

Return type list

seekToTop()
Reset the file pointer to the start of the file.

yieldChunks()
Return each chunk of text as a generator.

Yields *list* – The next chunk in the file.

class serpentTools.engines.**PatternReader**(*filePath*)
Class that can read over a file looking for patterns.

Parameters **filePath** (*str*) – path to the file that is to be read

line
str – Most recently read line

match
regular expression match or None – Match from the most recently read line

searchFor(*pattern*)
Return true if the pattern is found.

Parameters **pattern** (*str or compiled regular expression*) –

Returns **bool**

Return type True if the pattern was found

seekToTop()
Reset the file pointer to the start of the file.

yieldMatches(*pattern*)
Generator that returns all match groups that match pattern.

Parameters **pattern** (*str or compiled regular expression*) – Seek through the file and yield all match groups for lines that contain this patten.

Yields *sequential match groups*

5.5 Containers

Many of the readers utilize custom built objects for storing data in a logical and accessible format. These containers are detailed here.

Note: Variables taken from `SERPENT` will be converted to mixedCase and stored under those names. for example, `INF_TOT` would be stored as `infTot`

5.5.1 Homogenized Universe

class `serpentTools.objects.containers.`**`HomogUniv`**(*container*, *name*, *bu*, *step*, *day*)

Class for storing homogenized universe specifications and retrieving them

Parameters

- **container** (*serpentTools.objects.readers.BaseReader or*) – serpentTools.objects.containers.BranchContainer Object to which this universe is attached
- **name** (*str*) – name of the universe
- **bu** (*float*) – burnup value
- **step** (*float*) – temporal step
- **day** (*float*) – depletion day

name
str – name of the universe

bu
float – burnup value

step
float – temporal step

day
float – depletion day

infExp
dict – Expected values for infinite medium group constants

infUnc
dict – Relative uncertainties for infinite medium group constants

b1Exp
dict – Expected values for leakage corrected group constants

b1Unc
dict – Relative uncertainties for leakage-corrected group constants

metadata
dict – Other values that do not not conform to inf/b1 dictionaries

addData(*variableName*, *variableValue*, *uncertainty=False*)

Sets the value of the variable and, optionally, the associate s.d.

Warning: This method will overwrite data for variables that already exist

Parameters

- **variableName** (*str*) – Variable Name
- **variableValue** – Variable Value

- **uncertainty** (*bool*) – Set to `True` in order to retrieve the uncertainty associated to the expected values

Raises `TypeError` – If the uncertainty flag is not boolean

get(*variableName*, *uncertainty=False*)
Gets the value of the variable VariableName from the dictionaries

Parameters

- **variableName** (*str*) – Variable Name
- **uncertainty** (*bool*) – Boolean Variable- set to True in order to retrieve the uncertainty associated to the expected values

Returns

- *x* – Variable Value
- *dx* – Associated uncertainty

Raises

- `TypeError` – If the uncertainty flag is not boolean
- `KeyError` – If the variable requested is not stored on the object

5.5.2 Branch Containers

class `serpentTools.objects.containers.`**BranchContainer**(*parser*, *branchID*, *branchNames*, *stateData*)
Class that stores data for a single branch.

The *BranchingReader* stores branch variables and branched group constant data inside these container objects. These are used in turn to create *HomogUniv* objects for storing group constant data.

Parameters

- **parser** (*serpentTools.objects.readers.BaseReader*) – Parser that read the file that created this object
- **branchID** (*int*) – Index for the run for this branch
- **branchNames** (*tuple*) – Name of branches provided for this universe
- **stateData** (*dict*) – key: value pairs for branch variables

stateData
dict – Name: value pairs for the variables defined on each branch card

universes
dict – Dictionary storing the homogenized universe objects. Keys are tuples of `(universeID, burnup, burnIndex)`

addUniverse(*univID*, *burnup=0*, *burnIndex=0*, *burnDays=0*)
Add a universe to this branch.

Data for the universes are produced at specific points in time. The additional arguments help track when the data for this universe were created.

Warning: This method will overwrite data for universes that already exist

Parameters

- **univID** (*int or str*) – Identifier for this universe
- **burnup** (*float or int*) – Value of burnup [MWd/kgU]
- **burnIndex** (*int*) – Point in the depletion schedule
- **burnDays** (*int or float*) – Point in time

Returns newUniv

Return type *serpentTools.objects.containers.HomogUniv*

getUniv(*univID*, *burnup=None*, *index=None*)
Return a specific universe given the ID and time of interest

If burnup and index are given, burnup is used to search

Parameters

- **univID** (*int*) – Unique ID for the desired universe
- **burnup** (*float or int*) – Burnup [MWd/kgU] of the desired universe
- **index** (*int*) – Point of interest in the burnup index

Returns univ – Requested Universe

Return type *serpentTools.objects.containers.HomogUniv*

Raises

- KeyError: – If the requested universe could not be found
- SerpentToolsException: – If neither burnup nor index are given

orderedUniv
Universe keys sorted by ID and by burnup

5.5.3 Material Objects

class serpentTools.objects.materials.**DepletedMaterial**(*parser*, *name*)
Class for storing material data from _dep.m files.

Parameters

- **parser** (*DepletionReader*) – Parser that found this material. Used to obtain file metadata like isotope names and burnup
- **name** (*str*) – Name of this material

zai
numpy.array or None – Isotope id's

names
numpy.array or None – Names of isotopes

days
numpy.array or None – Days overwhich the material was depleted

adens
numpy.array or None – Atomic density over time for each nuclide

mdens
numpy.array or None – Mass density over time for each nuclide

burnup
numpy.array or None – Burnup of the material over time

addData(*variable*, *rawData*)
Add data straight from the file onto a variable.

Parameters

- **variable** (*str*) – Name of the variable directly from SERPENT
- **rawData** (*list*) – List of strings corresponding to the raw data from the file

getValues(*xUnits*, *yUnits*, *timePoints=None*, *names=None*)
Return x values for given time, and corresponding isotope values.

Parameters

- **xUnits** (*str*) – name of x value to obtain, e.g. 'days', 'burnup'
- **yUnits** (*str*) – name of y value to return, e.g. 'adens', 'burnup'
- **timePoints** (*list or None*) – If given, select the time points according to those specified here. Otherwise, select all points
- **names** (*list or None*) – If given, return y values corresponding to these isotope names. Otherwise, return values for all isotopes.

Returns Array of values.

Return type numpy.array

Raises

- AttributeError – If the names of the isotopes have not been obtained and specific isotopes have been requested
- KeyError – If at least one of the days requested is not present

getXY(*xUnits*, *yUnits*, *timePoints=None*, *names=None*)
Return x values for given time, and corresponding isotope values.

Deprecated since version 0.1.0: Use getValues() instead.

plot(*xUnits*, *yUnits*, *timePoints=None*, *names=None*, *ax=None*, *legend=True*, *label=True*, *xlabel=None*, *ylabel=None*)
Plot some data as a function of time for some or all isotopes.

Parameters

- **xUnits** (*str*) – name of x value to obtain, e.g. 'days', 'burnup'
- **yUnits** (*str*) – name of y value to return, e.g. 'adens', 'burnup'
- **timePoints** (*list or None*) – If given, select the time points according to those specified here. Otherwise, select all points
- **names** (*list or None*) – If given, return y values corresponding to these isotope names. Otherwise, return values for all isotopes.
- **ax** (None or matplotlib axes) – If given, add the data to this plot. Otherwise, create a new plot
- **legend** (*bool*) – Automatically add the legend
- **label** (*bool*) – Automatically label the axis
- **xlabel** (*None or str*) – If given, use this as the label for the x-axis. Otherwise, use xUnits

- **ylabel** (*None or str*) – If given, use this as the label for the y-axis. Otherwise, use yUnits

Returns Axes corresponding to the figure that was plotted

Return type matplotlib axes

See also:

getValues()

5.6 Branching Reader

The *BranchingReader* is designed to read the files generated by the automated burnup sequence. The output format is described at the SERPENT Wiki.

The reader stores homogenized universe data for each universe at each point in time for each branch in *HomogUniv* objects.

class serpentTools.parsers.branching.**BranchingReader**(*filePath*)
Parser responsible for reading and working with automated branching files.

Parameters **filePath** (*str*) – path to the depletion file

iterBranches()
Iterate over branches yielding paired branch IDs and containers

read()
Read the branching file and store the coefficients.

5.7 Bumat Reader

Warning: This reader is not implemented yet. This merely serves as a placeholder

class serpentTools.parsers.bumat.**BumatReader**(*filePath*, *readerSettingsLevel*)
Parser responsible for reading and working with burned material files.

Parameters **filePath** (*str*) – path to the depletion file

5.8 Depletion Reader

Warning: Does not support depleted materials with underscores, i.e. fuel_1 will not be matched with the current methods

class serpentTools.parsers.depletion.**DepletionReader**(*filePath*)
Parser responsible for reading and working with depletion files.

Parameters **filePath** (*str*) – path to the depletion file

materials
dict – Dictionary with material names as keys and the corresponding DepletedMaterial class for that material as values

metadata
dict – Dictionary with file-wide data names as keys and the corresponding dataas values, e.g. 'zai': [list of zai numbers]

settings
dict – names and values of the settings used to control operations of this reader

read()
Read through the depletion file and store requested data.

5.9 Detector Reader

Warning: This reader is not implemented yet. This merely serves as a placeholder

class serpentTools.parsers.detector.**DetectorReader**(*filePath*, *readerSettingsLevel*)
Parser responsible for reading and working with detector files.

Parameters **filePath** (*str*) – path to the depletion file

read()
Read the file and store the data.

Warning This read function has not been implemented yet

5.10 Fission Matrix Reader

Warning: This reader is not implemented yet. This merely serves as a placeholder

class serpentTools.parsers.fissionMatrix.**FissionMatrixReader**(*filePath*, *readerSettingsLevel*)
Parser responsible for reading and working with fission matrix files.

Parameters **filePath** (*str*) – path to the depletion file

read()
Read the file and store the data.

Warning This read function has not been implemented yet

5.11 Results Reader

Warning: This reader is not implemented yet. This merely serves as a placeholder

class serpentTools.parsers.results.**ResultsReader**(*filePath*, *readerSettingsLevel*)
Parser responsible for reading and working with result files.

Parameters **filePath** (*str*) – path to the depletion file

read()
Read the file and store the data.

Warning This read function has not been implemented yet

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S