LA-UR-22-28935

Approved for public release; distribution is unlimited.

Title: The MCNPTools Package: Installation and Use

Author(s): Bates, Cameron Russell
Bolding, Simon R.
Josey, Colin James
Kulesza, Joel A.
Solomon, Clell Jeffrey Jr.
Zukaitis, Anthony J.

Intended for: Report

Issued: 2022-08-25

Los Alamos NATIONAL LABORATORY

The MCNPTools Package: Installation and Use

Cameron R. Bates, Simon R. Bolding, Colin J. Josey, Joel A. Kulesza,
Clell J. (CJ) Solomon Jr., Anthony J. Zukaitis

August 2022

1 Introduction

MCNPTools is a C++ software library bound to Python (2 & 3) via the Simplified Wrapper and Interface Generator (SWIG version 3.0.7). The minimum requirements to build MCNPTools as a C++ library are the following:

- a C++ compiler supporting C++11 features
- the CMake tool set version 3.21 or above
- HDF5 version 1.10.2 or above

Currently, the following compiler options are tested and supported:

- GCC 8.3.0 and above on Linux and macOS
- MSVC 19.0 on Windows
- Apple Clang 7.3.0 and above on macOS
- Intel C++ Classic Compiler 18.0.5 and above

For the Python bindings, the following must be installed:

- Python 2.7 or newer
- Setuptools
- Pip

Builds of the Python bindings have been extensively tested with the Anaconda Python distribution (https://www.anaconda.com/products/individual), but have been cursorily tested with other distributions as well.

1.1 Installing MCNPTools from a Wheel

If you would like to install MCNPTools without building it yourself, you can do so by downloading a wheel for your operating system. Then, run:

```
python -m pip install mcnptools-X.Y.Z-??????.whl
```

The `??????` is a placeholder for information about the system for which the specified wheel file is built, and can include your OS and Python version. One can add the `-user` command to install in your user Python modules if you do not wish to install system-wide, or `-prefix [path]` to select an installation directory.

Note that MCNPTools will need to be re-installed whenever you upgrade your Python major version, e.g., from 3.9.X to 3.10.X.

1.2 Building MCNPTools

Once your build environment is set up (see Section 1.2.1 for tips for getting HDF5 working), create a directory to build MCNPTools. Within the directory, run the following commands:

```
cmake -D CMAKE_INSTALL_PREFIX=[path to install] -D mcnptools.python_install=User [
  path to MCNPTools source directory]
cmake --build . --config Release
ctest --build-config Release
cmake --install . --config Release
```

This will configure, build, test, and install the MCNPTools library, utilities, and Python bindings. Testing is optional but recommended. One should confirm all tests pass prior to installation.

The two CMake variables `CMAKE_INSTALL_PREFIX` and `mcnptools.python_install` control where components of MCNPTools are installed. The location for the library and the utilities is controlled by the variable `CMAKE_INSTALL_PREFIX`. The Python bindings will be placed at `CMAKE_INSTALL_PREFIX/lib` and the utilities will be placed at `CMAKE_INSTALL_PREFIX/bin`.

The Python binding install location is controlled by `mcnptools.python_install`, which has three options:

`Global` This will install in the current global Python module directory, and is most useful for system-wide installs or for Python virtual environments. (Default)

`User` This will will install in the current user's Python module directory. This is most useful for installing without administration privileges.

`Prefix` This will install within `CMAKE_INSTALL_PREFIX/lib`, which is most useful for packaging and maintaining multiple versions. The precise location is OS-dependent, but on Linux, the location will likely be `CMAKE_INSTALL_PREFIX/lib/pythonX.X/site-packages`, where `X.X` corresponds to the specific Python version used to build MCNPTools. In this case, you will have to add the `site-packages` path to the `PYTHONPATH` environment variable for Python to find the bindings.

Note that MCNPTools will need to be rebuilt and re-installed whenever you upgrade your Python version, e.g., from 3.9.X to 3.10.X.

1.2.1 Setting Up HDF5

Sometimes it is difficult for CMake to find a working HDF5 installation, and if it does, it may not load all the necessary libraries.

CMake can find HDF5 in 3 different ways, in order of most reliable to least reliable:

1. By setting the `HDF5_DIR` environment variable to HDF5's own CMake folder, located at `<path to HDF5 install>/share/cmake/hdf5`. This folder may not be present if HDF5 was not built using CMake.

2. Through finding the program `h5cc` in the current environment's path.

3. By setting the `HDF5_ROOT` environment variable to `<path to HDF5 install>`.

2 MCNPTools Utilities

MCNPTools releases include binary utilities that facilitate common tasks or querying MCNP output files. This section provides information regarding the use of these utilities. The usage information presented can be obtained from all utilities by running the utility with the `-h` or `--help` options specified.

2.1 `lnk3dnt` Utilities

2.1.1 `l3d2vtk`

The `l3d2vtk` utility converts LNK3DNT files to XML-based StructuredGrid VTK (.vts) files. This can be particularly useful to MCNP users because a LNK3DNT file can be produced as MCNP output that represents a discretized representation of the MCNP CSG, which can then visualized interactively in a 3-D application.

By default, `l3d2vtk` produces no standard output and writes a `lnk3dnt.vts` file. If the `--verbose` option is given, then status is output periodically as the conversion proceeds.

This utility functions for (x) (Cartesian), (r) (cylindrical), (r) (spherical), (x, y), (r, z), (r, θ), (x, y, z), (r, z, θ) geometries. For large LNK3DNT files, this utility can become sensitive to the computer's stack size. However, large ($\sim$ 100 million zone) 3-D Cartesian files have been successfully converted and visualized interactively.

The execution options given via the help message is given in Listing 9.

2.1.2 `l3dcoarsen`

The `l3dcoarsen` utility coarsens a LNK3DNT file and produces a new LNK3DNT file. By default, the resulting LNK3DNT file with have preserved material boundaries and the same number of mixed-material zones as the original; however, the user may keep more or less mixed-materials in a zone if desired.

The execution options given via the help message is given in Listing 10.

2.1.3 `l3dinfo`

The `l3dinfo` utility reports information about LNK3DNT files. By default, `l3dinfo` reports only basic information about the LNK3DNT file: geometry, extents, etc. If the `--full` option is given, then the material information will be read and reported in addition to the basic information.

The execution options given via the help message is given in Listing 11.

2.1.4 `l3dscale`

The `l3dscale` utility linearly scales the dimensions of a LNK3DNT file by a user-specified factor and produces a new LNK3DNT file.

The execution options given via the help message is given in Listing 12.

2.2 `mctal` Utilities

2.2.1 `mctal2rad`

The `mctal2rad` utility converts MCNP image tally results (e.g., `FIR`, `FIP`, etc.) in a MCTAL file into TIFF images. Accordingly, `mctal2rad` depends on libtiff being installed and available during compilation. The output images can be created from only the direct detector contributions and the results can be transposed and/or scaled logarithmically.

The execution options given via the help message is given in Listing 13.

2.2.2 `mergemctals`

The `mergemctals` utility statistically merges the results in multiple MCNP MCTAL files and produces a single resulting MCTAL file.

`mergemctals` can also be compiled using Boost MPI so that MCTAL files can be merged in parallel. All machines (e.g., back-end nodes of a cluster) performing parallel operations must have access to the files to be merged.

The execution options given via the help message is given in Listing 14.

2.3 `meshtal` Utilities

2.3.1 `mergemeshtals`

The `mergemeshtals` utility statistically merges the results in multiple MCNP Type-B MESHTAL files (i.e., those created with an `fmesh` card) and produces a single resulting MESHTAL file. `mergemeshtals` *only* operates on column-formatted MESHTAL files.

`mergemeshtals` can also be compiled using Boost MPI so that the MESHTAL files can be merged in parallel, though all machines (e.g., back-end nodes of a cluster) performing parallel operations must have access to the files to be merged.

The execution options given via the help message is given in Listing 15.

2.3.2 `meshtal2vtk`

The `meshtal2vtk` utility converts MCNP XYZ (Cartesian) and/or RZT (cylindrical) MCNP mesh tally results in a MESHTAL file into XML-formatted StructuredGrid VTK (.vts) files. These files can then be viewed in scientific visualization applications such as ParaView or VisIt.

Data series are logically named according to any binning that exists, or if no binning, as Tally_Value and Tally_Error. The user has the option of selecting only certain tallies with the TALLY parameter shown below. If left unspecified, all tallies are processed. Each tally is given its own .vts file.

The execution options given via the help message is given in Listing 16.

3 Description of the MCNPTools Library

The true power of MCNPTools is in the ability for users to write their own custom tools and process MCNP outputs without the need to parse MCNP output formats. Currently, three MCNP output files can be read by MCNPTools and accessed in an object-oriented manner:

MCTAL files accessed via the `Mctal` class which in turn provides access to the `MctalTally` and `MctalKcode` classes.

MESHTAL files accessed via the `Meshtal` class which in turn provides access to the `MeshtalTally` class

PTRAC files accessed via the `Ptrac` class which in turn provides access to the `PtracHistory` class which provides access to the `PtracEvent` class

Each of these three outputs will be discussed in more detail in the following subsections.

3.1 Accessing MCTAL Data with MCNPTools

MCNP MCTAL file data is accessed via three of MCNPTools' classes:

`Mctal` class Provides object-oriented access to a MCTAL file.

`MctalTally` class Provides object-oriented access to a tally in a MCTAL file

`MctalKcode` class Provides object-oriented access to kcode outputs in a MCTAL file

Each class will be discussed in the following sections.

3.1.1 `Mctal` Class

To construct (create) an instance of the `Mctal` class, one simply passes the name of a MCTAL file to the `Mctal` constructor, e.g.,

```
Mctal("mymctal")
```

The public methods available in the `Mctal` class are given in Table 1.

The most commonly used methods to access data in the MCTAL file are `GetTallyList` and `GetTally` for tally data and `GetKcode` for k-eigenvalue data. With `GetTallyList` and `GetTally`, loops over the tallies in the MCTAL file can be created to perform analyses. A Python example of such a loop structure is given in Listing 1.

Table 1: `Mctal` Class Public Methods

Method	Description
`GetCode()`	Returns a string of the generating code name
`GetVersion()`	Returns a string of the code version
`GetProbid()`	Returns a string of the problem identification
`GetDump()`	Returns an integer of the corresponding restart dump number
`GetNps()`	Returns an integer of the number of histories used in the normalization
`GetRandoms()`	Returns an integer the number of random numbers used
`GetTallyList()`	Returns a list/vector of tally numbers available in in the the MCTAL file
`GetTally(NUM)`	Returns a `MctalTally` class instance of tally number NUM

Listing 1: `Mctal` Class Use Example

```
# open the mctal file "mymctal"
mctal = mcnptools.Mctal("mymctal")

# loop over tallies
for tallynum in mctal.GetTallyList():
    tally = mctal.GetTally(tallynum)

    # now do something with the tally
```

3.1.2 `MctalTally` Class

The `MctalTally` class should only be created through calls to the `GetTally` method of the `Mctal` class. The `MctalTally` class will provide information about the tally and the values of data contained within the tally.

A Note on MCNP Tallies: MCNP tallies are essentially a nine-dimensional array with each index of the array describing a bin structure of the tally. These bin structures are given in Table 2.

With these bin structures, the values and errors in a tally are uniquely identified by the indices `(f,d,u,s,m,c,e,t,pert)`.

The `MctalTally` class has the public class methods given in Table 3.

Often it is desirable to interrogate a tally value at the *Tally Fluctuation Chart* (TFC) bin—the bin on which statistical analyses are performed. MCNPTools provides a defined constant `TFC` member of the `MctalTally` class that can be used in place of a bin index for any of the `(f,d,u,s,m,c,e,t)` bins. The Python code in Listing 2 illustrates how one would fill a list with tally values by iterating over the energy bins of a tally (for brevity it is assumed the MCTAL file has been opened as object `mctal`).

Note that the `pert` index has been omitted from the example above. The `GetValue` and `GetError` methods will default to the unperturbed tally quantities if `pert` is omitted.

Table 2: MCNP Tally Array Indices

Name	Identifier	Description
facet	`f`	The facet of the tally, cell, surface, point number
direct/flagged	`d`	The flagged/unflagged contribution for cell/surface tallies *or* the direct/scattered contribution for point detectors (this dimension never exceeds 2)
user	`u`	The user bins established by use of an `FT` tally input or by use of a `TALLYX` routine
segment	`s`	The segmenting bins established by use of an `FS` tally input
multiplier	`m`	The multiplier bins established by use of an `FM` tally input
cosine	`c`	The cosine bins established by use of an `C` tally input
energy	`e`	The energy bins established by use of an `E` tally input
time	`t`	The time bins established by use of a `T` tally input
perturbation	`pert`	The perturbation number established by use of `PERT` inputs

Table 3: `MctalTally` Class Public Methods

Method	Description
`ID()`	Return the integer tally number
`GetFBins()`	Return a list/vector of the "facet" bins of the tally
`GetDBins()`	Return a list/vector of the "direct/flagged" bins of the tally
`GetUBins()`	Return a list/vector of the "user" bins of the tally
`GetSBins()`	Return a list/vector of the "segment" bins of the tally
`GetMBins()`	Return a list/vector of the "multiplier" bins of the tally
`GetCBins()`	Return a list/vector of the "cosine" bins of the tally
`GetEBins()`	Return a list/vector of the "energy" bins of the tally
`GetTBins()`	Return a list/vector of the "time" bins of the tally
`GetValue(f,d,u,s,m,c,e,t,pert)`	Return the tally value identified by the indices `(f,d,u,s,m,c,e,t,pert)`
`GetError(f,d,u,s,m,c,e,t,pert)`	Return the tally *relative* error identified by the indices `(f,d,u,s,m,c,e,t,pert)`

Listing 2: `MctalTally` Class Use Example

```
# get the tally of interest (say tally 834)
tally = mctal.GetTally(834)

# create an alias for the TFC bin
TFC = tally.TFC

# get the energy bins
ebins = tally.GetEBins()

#create lists for tally values and errors
values = list()
errors = list()

# iterate over the energy bins
for i in range( len(ebins) ):
    #                              f    d    u    s    m    c  e    t
    values.append( tally.GetValue(TFC, TFC, TFC, TFC, TFC, TFC, i, TFC) )
    errror.append( tally.GetError(TFC, TFC, TFC, TFC, TFC, TFC, i, TFC) )
```

Table 4: `MctalKcode` Class Public Methods

Method	Description
`GetCycles()`	return the integer number of total kcode cycles
`GetSettle()`	return the integer number of inactive kcode cycles
`GetNdat()`	return the integer number of data elements in a kcode entry
`GetValue(QUANTITY, CYCLE)`	return the value of `QUANTITY` at the specified `CYCLE` (default last)

3.1.3 `MctalKcode` Class

The `MctalKcode` class should be obtained only through calls to `GetKcode()` method of the `Mctal` class. The `MctalKcode` class will provide information about the k_{eff} calculation as a function of cycle. The `MctalKcode` class has the public methods given in Table 4.

The `QUANTITY` value that is passed into the `GetValue` method is a parameterized member constant of the `MctalKcode` class. `QUANTITY` must be one of the following defined parameters within the `MctalKcode` class namespace as given in Table 5.

The Python code in Listing 3 illustrates how to get the combined (collision/absorption/track-length) value of k_{eff} and its standard deviation (for brevity it is assumed the MCTAL file has been opened in object `mctal`).

3.2 Accessing MESHTAL Data with MCNPTools

MCNP column-formatted MESHTAL (type B, a.k.a, MCNP5 style mesh tallies from the `fmesh` card) data is accessed through the following classes:

Table 5: `MctalKcode` Quantity Values

Method	Description
`COLLSION_KEFF`	estimated collision k_{eff} for this cycle
`ABSORPTION_KEFF`	estimated absorption k_{eff} for this cycle
`TRACKLENGTH_KEFF`	estimated track-length k_{eff} for this cycle
`COLLISION_PRLT`	estimated collision prompt-removal lifetime for this cycle
`ABSORPTION_PRLT`	estimated absorption prompt-removal lifetime for this cycle
`AVG_COLLSION_KEFF`	average collision k_{eff} to this cycle
`AVG_COLLSION_KEFF_STD`	standard deviation in the collision k_{eff} to this cycle
`AVG_ABSORPTION_KEFF`	average absorption k_{eff} to this cycle
`AVG_ABSORPTION_KEFF_STD`	standard deviation in the absorption k_{eff} to this cycle
`AVG_TRACKLENGTH_KEFF`	average track-length k_{eff} to this cycle
`AVG_TRACKLENGTH_KEFF_STD`	standard deviation in the track-length k_{eff} to this cycle
`AVG_COMBINED_KEFF`	average combined k_{eff} to this cycle
`AVG_COMBINED_KEFF_STD`	standard deviation in the combined k_{eff} to this cycle
`AVG_COMBINED_KEFF_BCS`	average combined k_{eff} by cycles skipped
`AVG_COMBINED_KEFF_BCS_STD`	standard deviation in the combined k_{eff} by cycles skipped
`COMBINED_PRLT`	average combined prompt-removal lifetime
`COMBINED_PRLT_STD`	standard deviation in the combined prompt-removal lifetime
`CYCLE_NPS`	number of histories used in each cycle
`AVG_COMBINED_FOM`	combined figure of merit

Listing 3: `MctalKcode` Class Use Example

```
# get the kcode data from the mctal file
kcode = mctal.GetKcode()

# get the average combined keff from the last cycle
keff = kcode.GetValue(MctalKcode.AVG_COMBINED_KEFF)

# get the standard deviation in combined keff
keff = kcode.GetValue(MctalKcode.AVG_COMBINED_KEFF_STD)
```

Table 6: `Meshtal` Class Public Methods

Method	Description
`GetCode()`	return a string of the generating code name
`GetVersion()`	return a string the code version
`GetProbid()`	return a string the problem id number
`GetComment()`	return a string of the problem comment
`GetNps()`	return the number of histories to which values are normalized
`GetTallyList()`	return a list/vector of tallies in the file
`GetTally(NUM)`	return a `MeshtalTally` class instance for tally `NUM`

Listing 4: `Meshtal` Class Use Example

```
import mcnptools

# load the meshtal file mymeshtal
meshtal = mcnptools.Meshtal("mymeshtal")

# loop over all the tallies in the file
for tallynum in meshtal.GetTallyList():
    # obtain the tally data
    tally = meshtal.GetTally(tallynum)

    # now do something with the tally
```

`Meshtal` provides object-oriented access to the MESHTAL file

`MeshtalTally` provides object-oriented access to tally data

Each class will be discussed in the following sections.

3.2.1 `Meshtal` Class

To construct (create) an instance of the `Meshtal` class, one simply passes the name of a MESHTAL (type B) file to the `Meshtal` constructor, e.g.,

```
Meshtal("mymeshtal")
```

The public methods available for the `Meshtal` class are given in Table 6.

The most commonly used methods of the `Meshtal` class are `GetTallyList()` and `GetTally`. The Python code in Listing 4 illustrates how to open a MESHTAL file with the `Meshtal` class, loop over the tallies, and obtain the tally data

3.2.2 `MeshtalTally` Class

The `MeshtalTally` provides accessors for a tally in a MESHTAL file. The public methods of the `MeshtalTally` class are given in Table 7.

Table 7: `MeshtalTally` Class Public Methods

Method	Description
`ID()`	return a list/vector of the tally id (number)
`GetXRBounds()`	return a list/vector of the x/r bin boundaries
`GetYZBounds()`	return a list/vector of the y/z bin boundaries
`GetZTBounds()`	return a list/vector of the z/θ bin boundaries
`GetEBounds()`	return a list/vector of the energy bin boundaries
`GetTBounds()`	return a list/vector of the time bin boundaries
`GetXRBins()`	return a list/vector of the x/r bin centers
`GetYZBins()`	return a list/vector of the y/z bin centers
`GetZTBins()`	return a list/vector of the z/θ bin centers
`GetEBins()`	return a list/vector of the energy bins
`GetTBins()`	return a list/vector of the time bins
`GetVolume(I,J,K)`	return the volume of element at index `(I,J,K)`
`GetValue(I,J,K,E,T)`	return the value at index `(I,J,K)` and optionally energy index `E` and time index `T`
`GetError(I,J,K,E,T)`	return the *relative* error at index `(I,J,K)` and optionally energy index `E` and time index `T`

If the energy-bin index is omitted from the `GetValue` or `GetError` method calls, then the total bin will be used if present. Otherwise, the largest energy bin will be used. Similarly, if the time-bin index is omitted from the `GetValue` and `GetError` method calls then the total bin will be used if present. Otherwise the last time bin will be used.

The Python code in Listing 5 illustrates how to loop through spatial elements of a `MeshtalTally` and query the values and errors at each element. For brevity it is assumed the MESHTAL file has already been loaded into `meshtal`.

3.3 Accessing PTRAC Data with MCNPTools

MCNP particle track (PTRAC) data are organized such that the PTRAC file contains histories and each history contains events—i.e., things that actually happened to particles. PTRAC data can be read and processed with MCNPTools by use of the following classes:

`Ptrac` provides object-oriented access to PTRAC files and accesses `PtracHistory` classes

`PtracHistory` provides object-oriented access to histories within the PTRAC file and accesses `PtracEvents`

`PtracNPS` provides object-oriented access to NPS information in a `PtracHistory`

`PtracEvent` provides object-oriented access to events and their data within a `PtracHistory`

The typical workflow when processing PTRAC files with MCNPTools is as follows:

1. Open the PTRAC file with the `Ptrac` class
2. Obtain histories in `PtracHistory` objects from the `Ptrac` class

Listing 5: `MeshtalTally` Class Use Example

```python
# get the tally to process (e.g., tally 324)
tally = meshtal.GetTally(324)

xrbins = tally.GetXRBins()
yzbins = tally.GetYZBins()
ztbins = tally.GetZTBins()

# loop over xrbins
for i in range(len(xrbins)):
    # loop over yzbins
    for j in range(len(yzbins)):
        # loop over ztbins
        for k in range(len(ztbins)):
            # print the value and error
            print(i,j,k,meshtal.GetValue(i,j,k),meshtal.GetError(i,j,k))
```

3. Iterate over the events in `PtracEvent` objects from the `PtracHistory` class

Each of these classes is discussed in the sections that follow.

3.3.1 `Ptrac` Class

The `Ptrac` class opens and manages MCNP PTRAC files and supports legacy binary, ASCII, and HDF5-formatted[1] PTRAC files. To construct the PTRAC file class, simply pass the PTRAC file name to the `Ptrac` constructor with the file type. For example, in Python one would use

```python
Ptrac("myptrac", Ptrac.BIN_PTRAC)
```

to open a legacy binary PTRAC file,

```python
Ptrac("myptrac", Ptrac.ASC_PTRAC)
```

to open an ASCII PTRAC file, and

```python
Ptrac("myptrac", Ptrac.HDF5_PTRAC)
```

to open an HDF5-formatted PTRAC file.

If the file type is omitted, legacy binary is assumed.

The `Ptrac` class has only one method `ReadHistories(NUM)` which returns a list/vector of histories. If `NUM` is omitted, then all the histories in the PTRAC file are read—this can be quite time consuming and is generally not recommended. A typical to reading histories in Python is shown in Listing 6.

3.3.2 `PtracHistory` Class

The `PtracHistory` class provides access to the events within the history. The public class methods are given in Table 8.

[1]HDF5-formatted PTRAC files are anticipated to be available in the next public release of the MCNP code.

Listing 6: `Ptrac` Class Use Example

```
# open the ptrac file (assuming legacy binary)
ptrac = mcnptools.Ptrac("myptrac")

# read history data in batches of 10000 histories
histories = ptrac.ReadHistories(10000)

# while histories has something in it
while histories:

    # iterate over the histories
    for h in histories:
        # do something with the history data

    # read in more histories, again a batch of 10000
    histories = ptrac.ReadHistories(10000)
```

Table 8: `PtracHistory` Class Public Methods

Method	Description
`GetNPS()`	returns a `PtracNPS` class with NPS information
`GetNumEvents()`	returns the number of events in the history
`GetEvent(I)`	returns the `I`th event in the history

A typical use of the `PtracHistory` class to obtain its events using Python is shown in Listing 7, where it is assumed that a `PtracHistory` exists in the variable `hist`):

3.3.3 `PtracNPS` Class

The `PtracNPS` class contains information about the history. The public methods in the `PtracNPS` class are given in Table 9.

For an HDF5 PTRAC file, the filtering cell, surface, tally, and value are not recorded in the PTRAC file. Please contact an MCNP developer at mcnp_help@lanl.gov if this limitation proves prohibitive.

3.3.4 `PtracEvent` Class

The `PtracEvent` class contains information about the event. Different event types contain different information about the event. The `PtracEvent` public class methods are given in Table 10.

Listing 7: `PtracHistory` Class Use Example

```
for i in range(hist.GetNumEvents()):
    event = hist.GetEvent(i)

    # now do something with the event
```

Table 9: `PtracNPS` Class Public Methods

Method	Description
`NPS()`	return the history number
`Cell()`	return the filtering cell from CELL keyword (if present)
`Surface()`	return the filtering surface from SURFACE keyword (if present)
`Tally()`	return the filtering tally from TALLY keyword (if present)
`Value()`	return the tally score from TALLY keyword (if present)

Table 10: `PtracEvent` Class Public Methods

Method	Description
`Type()`	returns the event type: one of `Ptrac::SRC` (source), `Ptrac::BNK` (bank), `Ptrac::COL` (collision), `Ptrac::SUR` (surface crossing), or `Ptrac::TER` (termination)
`BankType()`	returns the bank event type (only for `Ptrac::BNK` events)
`Has(DATA)`	returns a Boolean indicating whether or not the data type DATA is contained within the event
`Get(DATA)`	returns the value of the requested data type DATA

The DATA types available for the `Has` and `Get` methods are part of the `Ptrac` name space and are given in Table 12.

The Python code given in Listing 8 demonstrates how to find all collision events in a history and print the energy (for brevity a `PtracHistory` instance is assumed to be in the `hist` variable).

The PTRAC bank type variable specifiers that are part of the `Ptrac` name space are listed in Table 12.

The PTRAC termination types that are members of the `Ptrac` name space are listed in Table 13.

Listing 8: `PtracEvent` Class Use Example

```python
#iterate over all events in the history
for i in range(hist.GetNumEvents()):
    event = hist.GetEvent()

    # check if the event is a collision event
    if( event.Type() == Ptrac.COL ):
        # print the energy
        print(event.Get(Ptrac.ENERGY))
```

Table 11: PtracEvent Data Types

Data Type	Description
NODE	node number
ZAID	ZAID the particle interacts with
RXN	reaction type (MT number)
SURFACE	surface number
ANGLE	angle of particle crossing the surface
TERMINATION_TYPE	termination type
PARTICLE	particle type
CELL	cell number
MATERIAL	material number
COLLISION_NUMBER	collision number
X	particle x coordinate
Y	particle y coordinate
Z	particle z coordinate
U	particle direction cosine with respect to the x axis
V	particle direction cosine with respect to the y axis
W	particle direction cosine with respect to the z axis
ENERGY	particle energy
WEIGHT	particle weight
TIME	particle time

Table 12: PtracEvent Data Types

Data Type	Description
BNK_DXT_TRACK	DXTRAN particle
BNK_ERG_TME_SPLIT	Energy or Time splitting
BNK_WWS_SPLIT	Weight-window surface crossing
BNK_WWC_SPLIT	Weight-window collision
BNK_UNC_TRACK	Forced-collision uncollided part
BNK_IMP_SPLIT	Importance splitting
BNK_N_XN_F	Neutrons from fission
BNK_N_XG	Gammas from neutron production
BNK_FLUORESCENCE	Fluorescence x-rays
BNK_ANNIHILATION	Annihilation photons
BNK_PHOTO_ELECTRON	Photo electrons
BNK_COMPT_ELECTRON	Compton electrons
BNK_PAIR_ELECTRON	Pair-production electron
BNK_AUGER_ELECTRON	Auger electrons
BNK_PAIR_POSITRON	Pair-production positron
BNK_BREMSSTRAHLUNG	Bremsstrahlung production
BNK_KNOCK_ON	Knock-on electron
BNK_K_X_RAY	K-shell x-ray production
BNK_N_XG_MG	Multigroup (n,xγ)
BNK_N_XF_MG	Multigroup (n,f)
BNK_N_XN_MG	Multigroup (n,xn)
BNK_G_XG_MG	Multigroup (γ,xγ)
BNK_ADJ_SPLIT	Multigroup adjoint splitting
BNK_WWT_SPLIT	Weight-window mean-free-path split
BNK_PHOTONUCLEAR	Photo-nuclear production
BNK_DECAY	Radioactive decay
BNK_NUCLEAR_INT	Nuclear interaction
BNK_RECOIL	Recoil nucleus
BNK_DXTRAN_ANNIHIL	DXTRAN annihilation photon from pulse-height tally variance reduction
BNK_N_CHARGED_PART	Light ions from neutrons
BNK_H_CHARGED_PART	Light ions from protons
BNK_N_TO_TABULAR	Library neutrons from model neutrons
BNK_MODEL_UPDAT1	Secondary particles from inelastic nuclear interactions
BNK_MODEL_UPDATE	Secondary particles from elastic nuclear interactions
BNK_DELAYED_NEUTRON	Delayed neutron from radioactive decay
BNK_DELAYED_PHOTON	Delayed photon from radioactive decay
BNK_DELAYED_BETA	Delayed β^- from radioactive decay
BNK_DELAYED_ALPHA	Delayed α from radioactive decay
BNK_DELAYED_POSITRN	Delayed β^+ from radioactive decay

Table 13: PtracEvent Termination Types

Termination Type	Description
TER_ESCAPE	Escape
TER_ENERGY_CUTOFF	Energy cutoff
TER_TIME_CUTOFF	Time cutoff
TER_WEIGHT_WINDOW	Weight-window roulette
TER_CELL_IMPORTANCE	Cell importance roulette
TER_WEIGHT_CUTOFF	Weight-cutoff roulette
TER_ENERGY_IMPORTANCE	Energy-importance roulette
TER_DXTRAN	DXTRAN roulette
TER_FORCED_COLLISION	Forced-collision
TER_EXPONENTIAL_TRANSFORM	Exponential-transform
TER_N_DOWNSCATTERING	Neutron downscattering
TER_N_CAPTURE	Neutron capture
TER_N_N_XN	Loss to (n,xn)
TER_N_FISSION	Loss to fission
TER_N_NUCLEAR_INTERACTION	Nuclear interactions
TER_N_PARTICLE_DECAY	Particle decay
TER_N_TABULAR_BOUNDARY	Tabular boundary
TER_P_COMPTON_SCATTER	Photon Compton scattering
TER_P_CAPTURE	Photon capture
TER_P_PAIR_PRODUCTION	Photon pair production
TER_P_PHOTONUCLEAR	Photonuclear reaction
TER_E_SCATTER	Electron scatter
TER_E_BREMSSTRAHLUNG	Bremsstrahlung
TER_E_INTERACTION_DECAY	Interaction or decay
TER_GENNEUT_NUCLEAR_INTERACTION	Generic neutral-particle nuclear interactions
TER_GENNEUT_ELASTIC_SCATTER	Generic neutral-particle elastic scatter
TER_GENNEUT_DECAY	Generic neutral-particle particle decay
TER_GENCHAR_MULTIPLE_SCATTER	Generic charged-particle multiple scatter
TER_GENCHAR_BREMSSTRAHLUNG	Generic charged-particle bremsstrahlung
TER_GENCHAR_NUCLEAR_INTERACTION	Generic charged-particle nuclear interactions
TER_GENCHAR_ELASTIC_SCATTER	Generic charged-particle elastic scatter
TER_GENCHAR_DECAY	Generic charged-particle particle decay
TER_GENCHAR_CAPTURE	Generic charged-particle capture
TER_GENCHAR_TABULAR_SAMPLING	Generic charged-particle tabular sampling

4 Acknowledgments

The authors acknowledge Mike Rising, David Dixon, and Jeff Bull for their review of MCNPTools documentation and for testing it. Finally, the authors are grateful to the support provided by the Advanced Simulation and Computing (ASC) Program to develop, maintain, and release MCNPTools.

A Help Messages for MCNPTools Utilities

Listing 9: `l3d2vtk` Help Message Output

```
USAGE: l3d2vtk [--version] [--verbose] <LNK3DNT> [OUTPUT]

DESCRIPTION:

l3d2vtk converts a LNK3DNT file into an XML-formatted StructuredGrid (.vts) VTK
file.

OPTIONS:

--version, -v     : Print version and exit

--verbose, -V     : Produce standard output giving status (Default: False)

LNK3DNT           : LNK3DNT file name to convert

OUTPUT            : Converted LNK3DNT output name (Default: lnk3dnt.vts)

AUTHOR: Joel A. Kulesza [jkulesza@lanl.gov]
```

Listing 10: `l3dcoarsen` Help Message Output

```
USAGE: l3dcoarsen [--version] [--novoid] [--ifact ifact] [--jfact jfact]
       [--kfact kfact] [--maxmats maxmats] <LNK3DNT> [OUTPUT]

DESCRIPTION:

l3dcoarsen coarsens a LNK3DNT file mesh by specified factors

OPTIONS:

--version, -v     : Print version and exit

--novoid, -n      : Make voids material ’0’ rather than the assumed material
                    ’1’ (not recommended)

--ifact, -i       : Factor by which to coarsen in the first mesh dimension

--jfact, -j       : Factor by which to coarsen in the second mesh dimension
                    (if applicable)

--kfact, -k       : Factor by which to coarsen in the third mesh dimension (if
                    applicable)

--maxmats, -m     : Maximum umber of materials to keep include on the
                    coarsened LNK3DNT file (default: same as original)

LNK3DNT           : LNK3DNT file name to coarsen

OUTPUT            : coarsened LNK3DNT output name (Default: lnk3dnt.coarse)

AUTHOR: Clell J. (CJ) Solomon [csolomon@lanl.gov]
```

Listing 11: `l3dinfo` Help Message Output

```
USAGE: l3dinfo [--version] [--full] <LNK3DNT [LNK3DNT ... ]>

DESCRIPTION:

l3dinfo produces information about LNK3DNT files to stdout

OPTIONS:

--version, -v     : Print version and exit

--full, -f        : Produce a full listing of the LNK3DNT contents (can
                    greatly increase runtime)

LNK3DNT           : LNK3DNT files about which to produce information

AUTHOR: Clell J. (CJ) Solomon [csolomon@lanl.gov]
```

Listing 12: `l3dscale` Help Message Output

```
USAGE: l3dscale [--version] <LNK3DNT> <FACTOR> [OUTPUT]

DESCRIPTION:

l3dscale scales the dimensions of a LNK3DNT file

OPTIONS:

--version, -v     : Print version and exit

LNK3DNT           : LNK3DNT file to be scaled

FACTOR            : Scaling factor to be applied to the file

OUTPUT            : Output LNK3DNT file name [Default: LNK3DNT.scaled]

AUTHOR: Clell J. (CJ) Solomon [csolomon@lanl.gov]
```

Listing 13: `mctal2rad` Help Message Output

```
USAGE: mctal2rad [--version] [--log] [--direct] [--transpose] <MCTAL>
       [TALLY [TALLY ... ]]

DESCRIPTION:

mctal2rad converts an image tally from an MCNP MCTAL file into a TIFF image

OPTIONS:

--version, -v     : Print version and exit

--log, -l         : Produce an image of the log of the MCTAL values

--direct, -d      : Produce an image of the direct contribution

--transpose, -t   : Transpose the image

MCTAL             : MCTAL file containing one or more image tallies

TALLY             : Tally number for which to produce the images

AUTHOR: Clell J. (CJ) Solomon [csolomon@lanl.gov]
```

Listing 14: `mergemctals` Help Message Output

```
USAGE: mergemctals [--version] [--verbose] [--output output]
       <MCTAL [MCTAL ... ]>

DESCRIPTION:

mergemctals statistically merges multiple MCNP MCTAL files into a single MCTAL
file.

OPTIONS:

--version         : Print version and exit

--verbose, -v     : Increase output verbosity

--output, -o      : Output MCTAL file name [Default: mergemctals.out]

MCTAL             : MCTAL file names to be merged

AUTHOR: Clell J. (CJ) Solomon [csolomon@lanl.gov]
```

Listing 15: `mergemeshtals` Help Message Output

```
USAGE: mergemeshtals [--version] [--verbose] [--output output]
       <MESHTAL [MESHTAL ... ]>

DESCRIPTION:

mergemeshtals statistically merges multiple MCNP MESHTAL files into a single
MESHTAL file.

OPTIONS:

--version         : Print version and exit

--verbose, -v     : Increase output verbosity

--output, -o      : Output MESHTAL file name [Default: mergemeshtals.out]

MESHTAL           : MESHTAL file names to be merged

AUTHOR: Clell J. (CJ) Solomon [csolomon@lanl.gov]
```

Listing 16: `meshtal2vtk` Help Message Output

```
USAGE: meshtal2vtk [--version] <MESHTAL> [TALLY [TALLY ... ]]

DESCRIPTION:

meshtal2vtk converts mesh tallies from an MCNP MESHTAL file into XML-formatted
StructuredGrid (.vts) VTK files. This utility only works for XYZ (Cartesian)
and RZT (cylindrical) geometries.

OPTIONS:

--version, -v     : Print version and exit

MESHTAL           : MESHTAL file containing one or more mesh tallies

TALLY             : Tally number for which to produce the VTK files

AUTHOR: Joel A. Kulesza [jkulesza@lanl.gov]
```

B C++ Examples

B.1 Mctal Example 1

Listing 17 opens the MCTAL file `example_mctal_1.mcnp.mctal` and extracts the energy bins and energy-bin tally values for tally 4.

Listing 17: C++ Mctal Example 1

```cpp
#include "mcnptools/McnpTools.hpp"
#include <iostream>
#include <vector>

int main() {

    // construct the mctal class from mctal file "example_mctal_1.mcnp.mctal"
    mcnptools::Mctal m("example_mctal_1.mcnp.mctal");

    int tfc = mcnptools::MctalTally::TFC; // alias for -1

    // get tally 4 from the mctal file
    mcnptools::MctalTally t4 = m.GetTally(4);

    // get the energy bins of tally 4
    std::vector<double> t4_e = t4.GetEBins();

    // loop over energy bin indices to store and print tally bin value
    // using the TFC bin for all other bins
    std::vector<double> t4_evals(t4_e.size()); // storage for tally values
    for (unsigned int i = 0; i < t4_e.size(); i++) {
        //                          f   d   u   s   m   c  e  t
        t4_evals[i] = t4.GetValue(tfc, tfc, tfc, tfc, tfc, tfc, i, tfc);
        std::cout << t4_evals.at(i) << std::endl;
    }

    return 0;
}
```

B.2 Mctal Example 2

Listing 18 opens the MCTAL file `example_mctal_2.mcnp.mctal` and extracts the k_{eff} value and standard deviation for the active cycles, i.e., from the last settle cycle through the last active cycle.

Listing 18: C++ Mctal Example 2

```cpp
#include "mcnptools/McnpTools.hpp"
#include <iostream>

int main() {

    // construct the mctal class from the mctal file
    // "example_mctal_2.mcnp.mctal"
    mcnptools::Mctal m("example_mctal_2.mcnp.mctal");

    // get the kcode data
    mcnptools::MctalKcode kc = m.GetKcode();

    // alias for average combined keff
    unsigned int keff = mcnptools::MctalKcode::AVG_COMBINED_KEFF;
    // alias for average combined keff standard deviation
    unsigned int keff_std = mcnptools::MctalKcode::AVG_COMBINED_KEFF_STD;

    // loop over ACTIVE cycles and print
    for (unsigned int i = kc.GetSettle(); i < kc.GetCycles(); i++) {
        std::cout << i << "  " << kc.GetValue(keff, i) << "  " << kc.GetValue(keff_std, i)
                  << std::endl;
    }

    return 0;
}
```

B.3 Meshtal Example

Listing 19 reads tally 4 from MESHTAL file `example_meshtal.mcnp.meshtal` and prints the values at a slice through the z index 5 (using 0 indexing).

Listing 19: C++ Meshtal Example

```cpp
#include "mcnptools/McnpTools.hpp"
#include <iomanip>
#include <iostream>
#include <vector>

int main() {

    // construct the meshtal class from meshtal file
    // "example_meshtal.mcnp.meshtal"
    mcnptools::Meshtal m("example_meshtal.mcnp.meshtal");

    // get tally 4 from the meshtal file
    mcnptools::MeshtalTally t4 = m.GetTally(4);

    // get the x and y bin centers
    std::vector<double> x = t4.GetXRBins();
    std::vector<double> y = t4.GetYZBins();

    // loop over x and y bins indices and print the tally value for
    // z index of 5
    std::cout << std::scientific << std::setprecision(5);
    for (unsigned int i = 0; i < x.size(); i++) {
        for (unsigned int j = 0; j < y.size(); j++) {
            std::cout << std::setw(12) << t4.GetValue(i, j, 5);
        }
        std::cout << std::endl;
    }

    return 0;
}
```

B.4 Ptrac Example 1

Listing 20 opens the binary PTRAC file `example_ptrac_1.mcnp.ptrac` and prints the (x, y, z) location and energy of bank events.

Listing 20: C++ Ptrac Example 1

```
#include "mcnptools/McnpTools.hpp"
#include <iomanip>
#include <iostream>
#include <vector>

int main() {

    std::cout << std::scientific << std::setprecision(5);

    // explicitly open the file as a binary ptrac
    mcnptools::Ptrac p("example_ptrac_1.mcnp.ptrac", mcnptools::Ptrac::BIN_PTRAC);

    // initialize counter
    unsigned int cnt = 0;

    // read histories in batches of 10000
    std::vector<mcnptools::PtracHistory> hists = p.ReadHistories(10000);
    while (hists.size() > 0) {

        // loop over all histories
        for (unsigned int h = 0; h < hists.size(); h++) {
            // loop over all events in the history
            for (unsigned int e = 0; e < hists.at(h).GetNumEvents(); e++) {

                mcnptools::PtracEvent event = hists.at(h).GetEvent(e);

                if (event.Type() == mcnptools::Ptrac::BNK) {
                    cnt += 1;
                    std::cout << std::setw(13) << cnt << std::setw(13)
                              << event.Get(mcnptools::Ptrac::X) << std::setw(13)
                              << event.Get(mcnptools::Ptrac::Y) << std::setw(13)
                              << event.Get(mcnptools::Ptrac::Z) << std::setw(13)
                              << event.Get(mcnptools::Ptrac::ENERGY) << std::endl;
                }
            }
        }
        hists = p.ReadHistories(10000);
    }
    return 0;
}
```

B.5 Ptrac Example 2

Listing 21 opens binary PTRAC file `example_ptrac_2.mcnp.ptrac` and prints the (x, y, z) location and angle of surface crossings.

Listing 21: C++ Ptrac Example 2

```cpp
#include "mcnptools/McnpTools.hpp"
#include <iomanip>
#include <iostream>
#include <vector>

int main() {

    std::cout << std::scientific << std::setprecision(5);

    // explicitly open the file as a binary ptrac
    mcnptools::Ptrac p("example_ptrac_2.mcnp.ptrac", mcnptools::Ptrac::BIN_PTRAC);

    // read histories in batches of 10000
    std::vector<mcnptools::PtracHistory> hists = p.ReadHistories(10000);
    while (hists.size() > 0) {

        // loop over all histories
        for (unsigned int h = 0; h < hists.size(); h++) {
            // loop over all events in the history
            for (unsigned int e = 0; e < hists.at(h).GetNumEvents(); e++) {

                mcnptools::PtracEvent event = hists.at(h).GetEvent(e);

                if (event.Type() == mcnptools::Ptrac::SUR) {
                    std::cout << std::setw(13) << event.Get(mcnptools::Ptrac::X) << std::setw(13)
                              << event.Get(mcnptools::Ptrac::Y) << std::setw(13)
                              << event.Get(mcnptools::Ptrac::Z) << std::setw(13)
                              << event.Get(mcnptools::Ptrac::ANGLE) << std::endl;
                }
            }
        }

        hists = p.ReadHistories(10000);
    }

    return 0;
}
```

C Python Examples

C.1 Mctal Example 1

Listing 22 opens the MCTAL file `example_mctal_1.mcnp.mctal` and extracts the energy bins and energy-bin tally values for tally 4.

Listing 22: Python Mctal Example 1

```python
from mcnptools import Mctal, MctalTally

# construct the mctal class from mctal file "python_example_mctal_1.mcnp.mctal"
m = Mctal("example_mctal_1.mcnp.mctal")

tfc = MctalTally.TFC
# alias for -1

# get tally 4 from the mctal file
t4 = m.GetTally(4)

# get the energy bins of tally 4
t4_e = t4.GetEBins()

# loop over energy bin indices to store and print tally bin value
# using the TFC bin for all other bins

# store the tally values with list comprehension
#                        f    d    u    s    m    c   e   t
t4_evals = [t4.GetValue(tfc, tfc, tfc, tfc, tfc, tfc, i, tfc) for i in range(len(t4_e))]

# print the tally values
for i in range(len(t4_evals)):
    print(t4_evals[i])
```

C.2 Mctal Example 2

Listing 23 opens the MCTAL file `example_mctal_2.mcnp.mctal` and extracts the k_{eff} value and standard deviation for the active cycles, i.e., from the last settle cycle through the last active cycle.

Listing 23: Python Mctal Example 2

```python
from mcnptools import Mctal, MctalKcode

# construct the mctal class from the mctal file "python_example_mctal_2.mcnp.mctal"
m = Mctal("example_mctal_2.mcnp.mctal")

# get the kcode data
kc = m.GetKcode()

# alias for average combined keff
keff = MctalKcode.AVG_COMBINED_KEFF
# alias for average combined keff standard deviation
keff_std = MctalKcode.AVG_COMBINED_KEFF_STD

# loop over active cycles and print
for i in range(kc.GetSettle(), kc.GetCycles()):
    print(i, "   ", kc.GetValue(keff, i), "   ", kc.GetValue(keff_std, i))
```

C.3 Meshtal Example

Listing 24 reads tally 4 from MESHTAL file `example_meshtal.mcnp.meshtal` and prints the values at a slice through the z index 5 (using 0 indexing).

Listing 24: Python Meshtal Example

```
from mcnptools import Meshtal, MeshtalTally
from sys import stdout

# construct the meshtal class from meshtal file "python_example_meshtal.mcnp.meshtal"
m = Meshtal("example_meshtal.mcnp.meshtal")

# get tally 4 from the meshtal file
t4 = m.GetTally(4)

# get the x and y bin centers
x = t4.GetXRBins()
y = t4.GetYZBins()

# loop over x and y bins indices and print the tally value for
# z index of 5
for i in range(len(x)):
    for j in range(len(y)):
        stdout.write("{:12.5e}".format(t4.GetValue(i, j, 5)))
    stdout.write("\n")
```

C.4 Ptrac Example 1

Listing 25 opens the legacy binary PTRAC file `example_ptrac_1.mcnp.ptrac` and prints the (x, y, z) location and energy of bank events.

Listing 25: Python Ptrac Example 1

```python
from mcnptools import Ptrac
from sys import stdout

# explicitly open the file as a binary ptrac
p = Ptrac("example_ptrac_1.mcnp.ptrac", Ptrac.BIN_PTRAC)

# initialize counter
cnt = 0

# read histories in batches of 10000
hists = p.ReadHistories(10000)
while hists:

    # loop over all histories
    for h in hists:
        # loop over all events in the history
        for e in range(h.GetNumEvents()):

            event = h.GetEvent(e)

            if event.Type() == Ptrac.BNK:
                cnt += 1

                stdout.write(
                    "{:13d}{:13.5e}{:13.5e}{:13.5e}{:13.5e}\n".format(
                        cnt,
                        event.Get(Ptrac.X),
                        event.Get(Ptrac.Y),
                        event.Get(Ptrac.Z),
                        event.Get(Ptrac.ENERGY),
                    )
                )

    hists = p.ReadHistories(10000)
```

C.5 Ptrac Example 2

Listing 26 opens legacy binary PTRAC file `example_ptrac_2.mcnp.ptrac` and prints the (x, y, z) location and angle of surface crossings.

Listing 26: Python Ptrac Example 2

```python
from mcnptools import Ptrac
from sys import stdout

# explicitly open the file as a legacy binary ptrac
p = Ptrac("example_ptrac_2.mcnp.ptrac", Ptrac.BIN_PTRAC)

# read histories in batches of 10000
hists = p.ReadHistories(10000)

while hists:

    # loop over all histories
    for h in hists:
        # loop over all events in the history
        for e in range(h.GetNumEvents()):

            event = h.GetEvent(e)

            if event.Type() == Ptrac.SUR:
                stdout.write(
                    "{:13.5e}{:13.5e}{:13.5e}{:13.5e}\n".format(
                        event.Get(Ptrac.X),
                        event.Get(Ptrac.Y),
                        event.Get(Ptrac.Z),
                        event.Get(Ptrac.ANGLE),
                    )
                )

    hists = p.ReadHistories(10000)
```

C.6 Ptrac Example 3

Listing 27 opens HDF5 PTRAC file `example_ptrac_3.mcnp.ptrac.h5` and prints information about surface-crossing and termination events.

Listing 27: Python Ptrac Example 3

```python
from mcnptools import Ptrac
from sys import stdout

# explicitly open the file as an HDF5 ptrack
p = Ptrac("example_ptrac_3.mcnp.ptrac.h5", Ptrac.HDF5_PTRAC)

# read histories in batches of 10000
hists = p.ReadHistories(10000)

while hists:

    # loop over all histories
    for h in hists:
        print("History: ", h.GetNPS().NPS())
        # loop over all events in the history
        for e in range(h.GetNumEvents()):

            event = h.GetEvent(e)

            if event.Type() == Ptrac.SUR:
                stdout.write(
                    "SUR: {:13.5e}{:13.5e}{:13.5e}{:13.5e}\n".format(
                        event.Get(Ptrac.X),
                        event.Get(Ptrac.Y),
                        event.Get(Ptrac.Z),
                        event.Get(Ptrac.ANGLE),
                    )
                )
            elif event.Type() == Ptrac.TER:
                stdout.write(
                    "TER: {:13.5e}{:13.5e}{:13.5e}{:13.5e}\n".format(
                        event.Get(Ptrac.X),
                        event.Get(Ptrac.Y),
                        event.Get(Ptrac.Z),
                        event.Get(Ptrac.TERMINATION_TYPE),
                    )
                )

    hists = p.ReadHistories(10000)
```