The MCNPTools Package: Installation and Use

Clell J. (CJ) Solomon, Cameron Bates, and Joel Kulesza

LANL, XCP-3

March 30, 2017

Contents

1 Installation

If the user has installed MCNP 6.2.0 from the installation DVDs, then prebuilt versions of the MCNPTools binaries (see below) and the MCNPTools libraries are available under the respective `bin` and `lib` directories of the installation. As with MCNP itself, the standalone MCNPTools binaries have been placed in the user's path for use.

If the user desires to build the MCNPTools utilities and C++ bindings or the user desires the Python bindings, the following sections will describe how to build and install the MCNPTools binaries, libraries, and Python bindings. After the MCNP 6.2.0 installation process, the MCNPTools source code can be found under `MCNP_CODE/MCNP620/Utilities/MCNPTOOLS/Source` and the prebuilt Python wheels can be found under `MCNP_CODE/MCNP620/Utilities/MCNPTOOLS/wheels`.

1.1 Overview and Requirements

MCNPTools[1] 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 3.1 or greater

Currently, the following compilers are supported:

- GCC 5.3.0 and above on Linux and Mac OS X
- MSVC 19.0 on Windows
- Apple Clang 7.3.0 and above on Mac OS X

Additionally, one must have Python installed to build the Python bindings. CMake is not required should one desire to build only the Python.

1.2 Building the MCNPTools C++ Library and Utilities

To begin building MCNPTools' C++ library and utilities, open a command-line interface and create a build directory. On Mac OS X or Linux, execute the following instructions:

```
cmake -DCMAKE_INSTALL_PREFIX=/install/path /path/to/mcnptools/libmcnptools
make
make test
```

[1] MCNP® and Monte Carlo N-Particle® are registered trademarks owned by Los Alamos National Security, LLC, manager and operator of Los Alamos National Laboratory. Any third party use of such registered marks should be properly attributed to Los Alamos National Security, LLC, including the use of the ® designation as appropriate. Any questions regarding licensing, proper use, and/or proper attribution of Los Alamos National Security, LLC marks should be directed to trademarks@lanl.gov.

```
make install
```

where "/install/path" should be replaced with the desired installation directory on the system and "/path/to/mcnptools/libmcnptools" is the path to the "libmcnptools" source directory. Execution of "make test" is recommended but optional and will run the MCNPTools unit tests.

On Windows, only the "Visual Studio 14 2015" build tools are currently supported. To build on Windows issue the following commands at a Windows "Developer Command Prompt for VS2015" (it is assumed the `cmake` command is in your `PATH`):

```
cmake -G "Visual Studio 14 2015 Win64" -DCMAKE_INSTALL_PREFIX=C:\install\path^
    C:\path\to\mcnptools\libmcnptools
cmake --build . --config RelWithDebInfo
ctest -C RelWithDebInfo
cmake --build . --config RelWithDebInfo --target install
```

1.3 Building the MCNPTools Python2 and Python3 Extensions

Building and installing the MCNPTools' Python extensions is performed with Python's "setuptools" package. A "setup.py" file is provided in the "mcnptools/python" directory. The Python extensions can be built with the following commands at the command-line interface:

```
python setup.py build_ext
python setup.py test
python setup.py install --prefix=/path/to/install/dir
```

where `--prefix=/path/to/install/dir` specifies an optional installation location different from the Python installation's default location. If not installing to the Python installation's default install location, one will likely be required to set the `PYTHONPATH` environment variable to include the path to the location where the `mcnptools` package is installed—for Python version X.X this is typically `/path/to/install/dir/lib/pythonX.X/site-packages`.

Depending on the users Python installation, it is possible that minor tweaks, e.g., altering some compile or link flags, to the "setup.py" file will be required. Builds of the Python bindings have been extensively tested with the Anaconda Python distribution (https://www.continuum.io/downloads), but have been cursorily tested with other distributions as well.

1.4 Installing the Python3 Extensions with `pip`

The MCNPTools release also ships with "Python Wheel" files to directly install pre-built Python3 bindings. The wheels were assembled using Anaconda 4.3.0 (https://www.continuum.io/downloads) which is based on Python 3.6. The wheel files can be installed with `pip` using the following command:

```
pip install --prefix /path/to/install/dir mcnptools-3.8.0-XXXXXX.whl
```

Above, `/path/to/install/dir` is the location where the MCNPTools package should be installed, and, if it is omitted, defaults to the install location of the Python installation. The `XXXXXX` is a placeholder for information about the system the for which the specific wheel file is built, e.g., `win_amd64`.

2 The MCNPTools Utilities

MCNPTools comes with binary utilities to facilitate common tasks or query MCNP output files. This section provides information regarding the usage 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 The `mergemctals` Utility

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.

```
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
```

2.2 The `mergemeshtals` Utility

The `mergemeshtals` utility statistically merges the results in multiple MCNP MESHTAL (type B/FMESH) files and produces a single resulting MESHTAL file. `mergemeshtals` **only** operates on the column formatted version of the 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.

```
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
```

2.3 The `mctal2rad` Utility

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

```
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
```

2.4 The `l3dinfo` Utility

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.

```
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
```

2.5 The `l3dcoarsen` Utility

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.

```
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 number 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)
```

2.6 The `l3dscale` Utility

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

```
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]
```

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's output formats. Currently, three of MCNP's outputs 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 The `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 following table defines the public methods available for the `Mctal` class:

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

Method	Description
`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
`GetKcode()`	Returns a `MctalKcode` class instance of the kcode calculation data

The most commonly used methods to access data in the MCTAL file are `GetTallyList` and `GetTally` for tally data and `GetKcode` for kcode 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 follows:

```python
# 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 The `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 as follows:

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

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 following public class 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

Method	Description
`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)`

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 following Python code 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 in class `mctal`):

```python
# 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) )
```

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.

3.1.3 The `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 following 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

Method	Description
`GetValue(QUANTITY, CYCLE)`	return the value of `QUANTITY` at the specified `CYCLE` (default last)

The `QUANTITY` value that is handed 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:

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

The following Python code 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 class `mctal`):

```python
# 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)
```

3.2 Accessing MESHTAL Data with MCNPTools

MCNP MESHTAL (type B, a.k.a, MCNP5 stype mesh tallies) data is accessed through the following classes:

`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 The `Meshtal` Class

To construct (create) and 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 following table defines the public methods available for the `Meshtal` class:

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`

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

```python
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
```

3.2.2 The `MeshtalTally` Class

The `MeshtalTally` provides accessors for a tally in a MESHTAL file. The public methods of the `MeshtalTally` class are as follows:

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/theta 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/theta bin centers
`GetEBins()`	return a list/vector of the energy bins

Method	Description
`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 following Python code 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`.)

```
# get the tally to process (say 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))
```

3.3 Accessing PTRAC Data with MCNPTools

MCNP's PTRAC data is organized such that the PTRAC file contains histories and each history contains events—i.e., thing 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 acces 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
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 The Ptrac Class

The `Ptrac` class opens and manages MCNP PTRAC files and supports both binary and ASCII formatted PTRAC files. To construct the PTRAC file class, simply pass the PTRAC file name to the `Ptrac` constructor with the file type (binary or ASCII). For example, in Python one would use

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

to open a binary PTRAC file and

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

to open an ASCII PTRAC file. If the file type is omitted, 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. Typical use of reading histories in Python looks like the following:

```
# open the ptrac file (assuming 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 somehting with the history data

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

3.3.2 The PtracHistory Class

The `PtracHistory` class provides access to the events within the history. The public class methods are

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 looks like the following in Python (it is assumed that a `PtracHistory` exists in the variable `hist`):

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

    # now do something with the event
```

3.3.3 The PtracNPS Class

The `PtracNPS` class contains information about the history. The public methods in the `PtracNPS` class are the following:

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)

3.3.4 The 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 as follows:

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 of 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 presented below:

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

Data Type	Description
TIME	particle time

The following Python code 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):

```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))
```

The following table lists the PTRAC bank type variable specifiers (with associated ID numbers) that are part of the `Ptrac` name space:

Bank 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	Photonuclear 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

Bank Type	Description
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

The following table lists the PTRAC termination types (with associated ID numbers) that are members of the `Ptrac` name space:

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 would like to acknowledge Mike Rising, David Dixon, and Jeff Bull for their review of MCNPTools' documentation and testing.

5 C++ Examples

5.1 Mctal Example 1

This example opens a MCTAL file and extracts the energy bins and energy-bin tally values for tally 4.

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

int main() {

  // construct the mctal class from mctal file "my_mctal"
  mcnptools::Mctal m("my_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;
}
```

5.2 Mctal Example 2

This example extracts the k_{eff} value and standard deviation for the active cycles, i.e., from the last settle cycle through the last active cycle.

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

int main() {
```

```

  // construct the mctal class from the mctal file "my_mctal"
  mcnptools::Mctal m("my_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;
}
```

5.3 Meshtal Example

This example reads tally 4 from MESHTAL file `my_meshtal` and prints the values at a slice through the z index 5 (using 0 indexing).

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

int main() {

  // construct the meshtal class from meshtal file "my_meshtal"
  mcnptools::Meshtal m("my_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;
}
```

5.4 Ptrac Example 1

This example opens the binary PTRAC file `my_ptrac` and prints the (x, y, z) location and energy of bank events.

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

int main() {

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

  // explicitly open the file as a binary ptrac
  mcnptools::Ptrac p("my_ptrac", mcnptools::Ptrac::BIN);

  // 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;
}
```

5.5 Ptrac Example 2

This example opens binary PTRAC file `my_ptrac` and prints the (x, y, z) location and angle of surface crossings.

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

int main() {

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

  // explicitly open the file as a binary ptrac
  mcnptools::Ptrac p("my_ptrac", mcnptools::Ptrac::BIN);

  // 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;
}
```

6 Python Examples

6.1 Mctal Example 1

This example opens a MCTAL file and extracts the energy bins and energy-bin tally values for tally 4.

```python
from mcnptools import Mctal, MctalTally

# construct the mctal class from mctal file "my_mctal"
m = Mctal("my_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];
```

6.2 Mctal Example 2

This example extracts the k_{eff} value and standard deviation for the active cycles, i.e., from the last settle cycle through the last active cycle.

```python
from mcnptools import Mctal, MctalKcode

# construct the mctal class from the mctal file "my_mctal"
m = Mctal("my_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)
```

6.3 Meshtal Example

This example reads tally 4 from MESHTAL file `my_meshtal` and prints the values at a slice through the z index 5 (using 0 indexing).

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

# construct the meshtal class from meshtal file "my_meshtal"
m = Meshtal("my_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")
```

6.4 Ptrac Example 1

This example opens the binary PTRAC file `my_ptrac` and prints the (x, y, z) location and energy of bank events.

```
from mcnptools import Ptrac
from sys import stdout

# explicitly open the file as a binary ptrac
p = Ptrac("my_ptrac", Ptrac.BIN)

# 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)
```

6.5 Ptrac Example 2

This example opens binary PTRAC file `my_ptrac` and prints the (x, y, z) location and angle of surface crossings.

```
from mcnptools import Ptrac
from sys import stdout

# explicitly open the file as a binary ptrac
p = Ptrac("my_ptrac", Ptrac.BIN)

# 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)
```