IDS and Scipp McStas Intro 2022
March 17, 2022
ESS/DMSC

MCPL
Monte Carlo Particle Lists

Introduction to MCPL

Presentation: T. Kittelmann
Practical: P. Willendrup

EUROPEAN SPALLATION SOURCE

brightness

Acknowledgements:
E. Klinkby[1] E.B. Knudsen[1]
P. Willendrup[12] K. Kanaki[2]
D. Di Julio[2] X.-X. Cai[12→3]
1: DTU 2: ESS 3: CSNS

SINE 2020

DTU Technical University of Denmark

Key MCPL features

MCPL: Monte Carlo Particle Lists

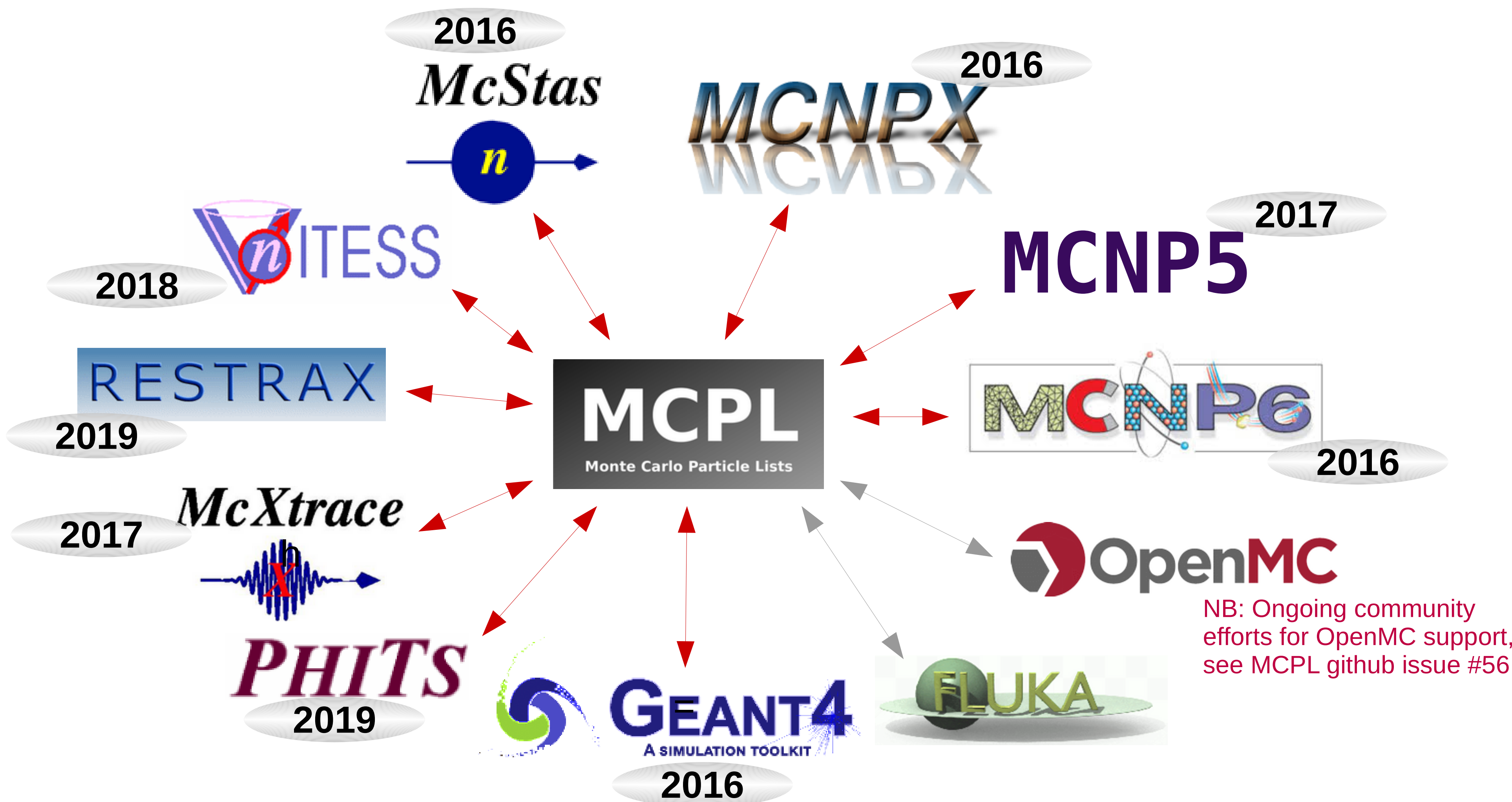
- It is a ***simple*** binary file-format. Each file contains a list of MC particles with enough info to seed simulations.
- MCPL files can contain ***meta-data***. This makes it possible to tell what data is in a file, where it came from, how it should be interpreted.
- The format is ***flexible***: can contain a lot of information if needed, or can contain only minimal information if small file-size is important. Can be gzip'ed.
- It is ***easy*** to make code dealing with MCPL, so it is easy to make plugins & converters for the various Monte Carlo frameworks.
  → End-users will simply use those converters.
- MCPL comes with ***tools and APIs***, such as for inspecting or editing contents.
- ***Well-defined*** versioned format, focus on backwards compatibility.

MCPL background/philosophy

MCPL: Monte Carlo Particle Lists

- In principle simple: just a “**bag of Monte Carlo particles**”, with properties such as particle type, energy, position, direction, weights, ...
- Goal of the MCPL project: Make this a new **standard particle-exchange format**.
  - Original motivation: we needed to chain MCNP→McStas→Geant4
- To achieve this, we tried to make it **attractive to use**:
  - Have custom hooks for most major Monte Carlo particle codes.
  - Have cmdline tools and easy to use C/C++/Python API
- .. and we tried hard to avoid:
  - Annoying dependencies.
  - *“MCPL is too bloated/slow for my usecase, I’ll roll my own custom solution”*
- So it must be **flexible in what is actually on-disk** (e.g. don’t need “type” field in McStas output since it is always neutrons)
- But should **always look the same when opening the files** to make reading files trivial.

Codes with MCPL support



Certainly have critical mass by now! :-)

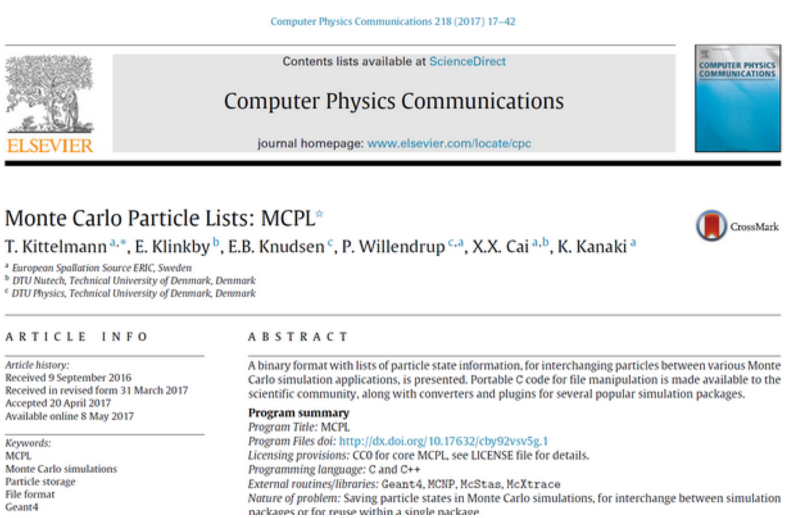
Available Missing

… focus on availability:

- Extremely liberal license (CC0) encourage bundling.
- API for C/C++/Python code (all versions).
- “fat” single-file versions of all C code (even embedding zlib)
- Can “pip install” Python API+pymcpltool.

Download, follow, and report issues @GitHub

… and documentation:


Computer Physics Communications 218 (2017) 17–42
Contents lists available at ScienceDirect
Computer Physics Communications
journal homepage: www.elsevier.com/locate/cpc

Monte Carlo Particle Lists: MCPL

T. Kittelmann [a,*], E. Klinkby [b], E.B. Knudsen [c], P. Willendrup [c,a], X.X. Cai [a,b], K. Kanaki [a]
[a] European Spallation Source ERIC, Sweden
[b] DTU Nutech, Technical University of Denmark, Denmark
[c] DTU Physics, Technical University of Denmark, Denmark

ARTICLE INFO

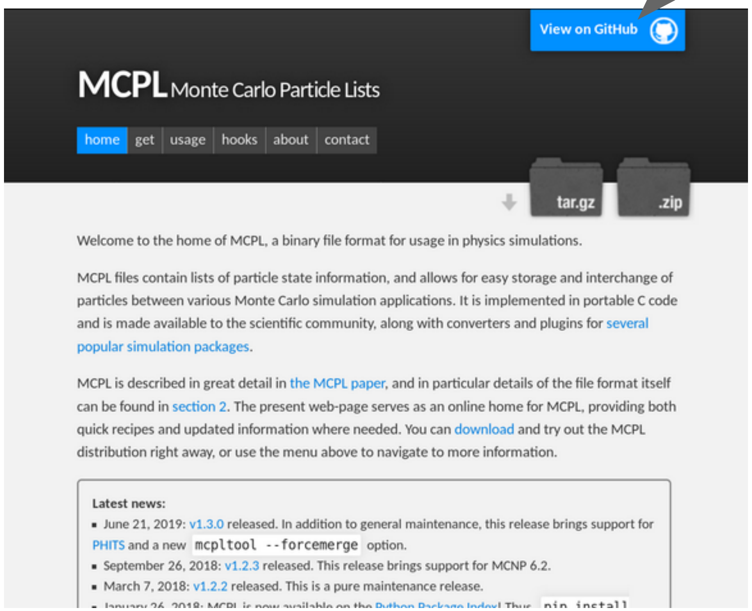
Article history:
Received 9 September 2016
Received in revised form 31 March 2017
Accepted 20 April 2017
Available online 8 May 2017

Keywords:
MCPL
Monte Carlo simulations
Particle storage
File format
Geant4

ABSTRACT

A binary format with lists of particle state information, for interchanging particles between various Monte Carlo simulation applications, is presented. Portable C code for file manipulation is made available to the scientific community, along with converters and plugins for several popular simulation packages.

Program summary
Program Title: MCPL
Program Files doi: http://dx.doi.org/10.17632/cby92vsv5g.1
Licensing provisions: CC0 for core MCPL, see LICENSE file for details.
Programming language: C and C++
External routines/libraries: Geant4, MCNP, McStas, McXtrace
Nature of problem: Saving particle states in Monte Carlo simulations, for interchange between simulation



- Detailed paper for release 1.1.0: (DOI 10.1016/j.cpc.2017.04.012)
- Online docs with recipes (https://mctools.github.io/mcpl/)

What form does MCPL support take?

- Built-in support in instrument simulation codes:
  - McStas, McXtrace, VITESS, RESTRAX/SIMRES
  - Batteries included → great for users!

Most work done by developers of these applications!

- C++ helper classes for particle capture or event seeding available for Geant4 (in line with how most Geant4 users work)

T. Kittelmann

- MCNP support relies on inbuilt ability to dump particles to/seed from “SSW” files.
  - We provide **ssw2mcpl** and **mcpl2ssw** tools.
  - Somewhat high maintenance burden due to plethora of MCNP flavours + closed nature of programme.
  - Complication is that particles need “surface ID”. Can be provided as MCPL userflags or via global setting.
  - **mcpl2ssw** must be provided with sample SSW files from target setup.

T. Kittelmann+E. Klinkby

- PHITS support: Like MCNP, but simpler. More details later.

T. Kittelmann+D. Di Julio

Data in MCPL files

All generic parameters always Available to reading code, no matter source of MCPL file.

Particle state information	
Field	**Description**
PDG code	32 bit integer indicating particle type.
Position	Vector, values in centimetres.
Direction	Unit vector along the particle momentum.
Kinetic energy	Value in MeV.
Time	Value in milliseconds.
Weight	Weight or intensity.
Polarisation	Vector.
User-flags	32 bit integer with custom info.

Flexibility in how this is actually stored!

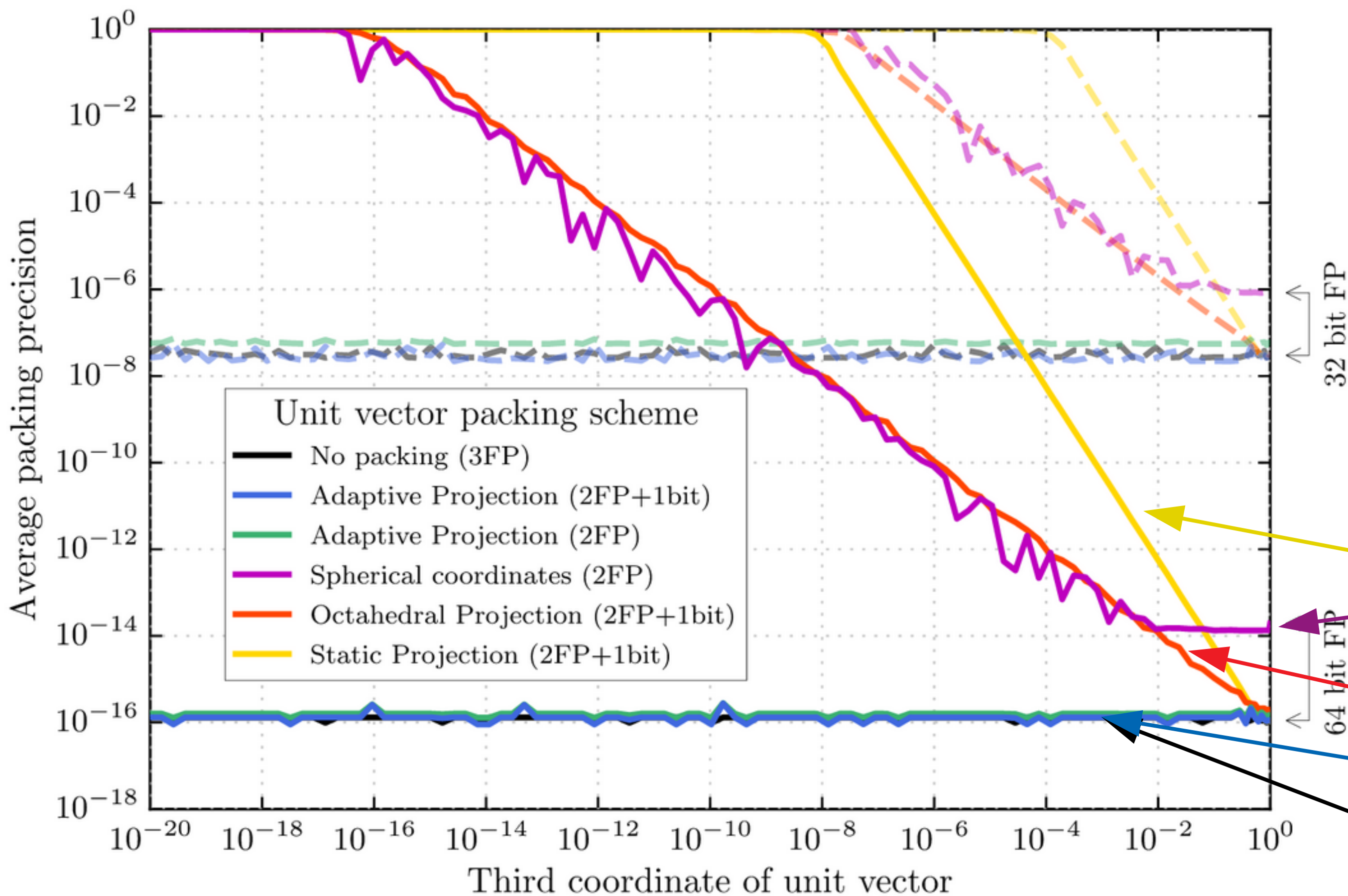
Detailed layout of the data associated with each particle in an MCPL file.

FP can be single (32bit) or double precision (64bit)

Particle data layout		
Presence	**Count & type**	**Description**
OPTIONAL	$3 \times$ FP	Polarisation vector (if enabled in file).
ALWAYS	$3 \times$ FP	Position vector
ALWAYS	$3 \times$ FP	Packed direction vector and kinetic energy.
ALWAYS	$1 \times$ FP	Time.
OPTIONAL	$1 \times$ FP	Weight (if file does not have universal weight).
OPTIONAL	$1 \times$ INT32	PDG code (if file does not have universal PDG code).
OPTIONAL	$1 \times$ UINT32	User-flags (if enabled in file).

This implies from 28 to 96 bytes/particle. Already good, but most files are gzip'ed (by MCPL or user) and consume less. (NB: MCPL code can read .mcpl.gz files directly)

Novel packing of direction vectors: Optimal 2xFP storage size without precision loss!



MCNP/SSW files
Spherical coords
MCPL (2016)
MCPL (2017+)
Ideal (unpacked)

Breakdown of the Adaptive Projection Packing method, in which a unit vector, (u_x, u_y, u_z) is stored into two floating point numbers, FP1 and FP2, and one extra bit of information.

Adaptive Projection Packing				
Scenario	FP1	FP2	+1 bit	Packed signature
$\lvert u_x\rvert$ largest	$1/u_z$	u_y	sign(u_x)	$\lvert FP1\rvert > 1, \lvert FP2\rvert < 1$
$\lvert u_y\rvert$ largest	u_x	$1/u_z$	sign(u_y)	$\lvert FP1\rvert < 1, \lvert FP2\rvert > 1$
$\lvert u_z\rvert$ largest	u_x	u_y	sign(u_z)	$\lvert FP1\rvert < 1, \lvert FP2\rvert < 1$

Example file

Inspected with (py)mcpltool

```
Opened MCPL file recordfwd.mcpl.gz:

  Basic info
    Format             : MCPL-3
    No. of particles   : 542199
    Header storage     : 826 bytes
    Data storage       : 17350368 bytes

  Custom meta data
    Source             : "Geant4"
    Number of comments : 8
          -> comment 0 : "Created with the Geant4 MCPLWriter in the ESS/dgcod
          -> comment 1 : "MPCLWriter volumes considered : ['RecordFwd']"
          -> comment 2 : "MPCLWriter steps considered : <at-volume-exit>"
          -> comment 3 : "MPCLWriter write filter : <unfiltered>"
          -> comment 4 : "MPCLWriter user flags : <disabled>"
          -> comment 5 : "MPCLWriter track kill strategy : <none>"
          -> comment 6 : "ESS/dgcode geometry module : G4StdGeometries/GeoSla
          -> comment 7 : "ESS/dgcode generator module : G4StdGenerators/Simpl
    Number of blobs    : 2
          -> 74 bytes of data with key "ESS/dgcode_geopars"
          -> 231 bytes of data with key "ESS/dgcode_genpars"

  Particle data format
    User flags         : no
    Polarisation info  : no
    Fixed part. type   : no
    Fixed part. weight : yes (weight 1)
    FP precision       : single
    Endianness         : little
    Storage            : 32 bytes/particle

index     pdgcode   ekin[MeV]       x[cm]       y[cm]       z[cm]          ux          uy          uz    time[ms]
    0          22      1.2238     -13.327      3.5344          40    -0.43426   -0.036564     0.90005     0.14113
    1          22     0.12059     -15.976      14.788          40    -0.63971    0.082934     0.76413     0.14113
    2          22     0.10212     -22.452     -7.1864          40    -0.58735    -0.35527     0.72718     0.14113
    3          22       7.695      12.547      36.899          40     0.19775     0.47066     0.85987     0.20354
    4        2112     2.5e-08           0           0          40           0           0           1      0.1829
    5          22    0.077251     -33.171      15.428          40    -0.81854     0.33885     0.46387   0.0047377
                                                                          666     0.38747     0.91761   0.0047367
                                                                          866   -0.075343     0.97717     0.12339
                                                                            0           0           1      0.1829
    9        2112     2.5e-08           0           0          40           0           0           1      0.1829
```

Custom meta-data
- This file is from ESS-DG Geant4
- Comments reminding us of setup used to create file
- Binary "blobs" keep more complete configuration details, here ESS-DG geo/gen parameters. Could be McStas instrument file, input deck from MCNP/PHITS, etc.

NB: compresses to 19.2bytes/particle

Columns of particle data
In this file: No *userflags* or *polarisation*

PDG codes: 2112 = neutron, 22 = gamma
More at http://pdg.lbl.gov/2015/reviews/rpp2015-rev-monte-carlo-numbering.pdf

C API

- Stable C API for reading/creating/editing MCPL
- Use to create most application-specific hooks
- Some users use it to analyse or tailor MCPL files

```
#include "mcpl.h"
void read_example()
{
  mcpl_file_t f = mcpl_open_file("myfile.mcpl");
  const mcpl_particle_t* prtcl;
  while ( ( prtcl = mcpl_read(f) ) ) {
    //<Access here: prtcl->ekin, prtcl->time, ...>
  }
  mcpl_close_file(f);
}
```

C not C++ to support more apps (C is “lingua franca” of SW)

Despite being C, interface is “object oriented” and hopefully easy.

```
#include "mcpl.h"
void create_example()
{
  mcpl_outfile_t f = mcpl_create_outfile("myfile.mcpl");
  mcpl_hdr_set_srcname(f,"Custom C code");
  mcpl_hdr_add_comment(f,"Just an example.");
  mcpl_enable_doubleprec(f);
  int i;
  mcpl_particle_t * prtcl = mcpl_get_empty_particle(f);
  for ( i = 0; i < 1000; ++i ) {
    //<Set here: prtcl->ekin, prtcl->time, ...>
    mcpl_add_particle(f,prtcl);
  }
  mcpl_close_outfile(f);
}
```

Custom filtering via C API

Filtering files with custom code in very few lines:

mcpl_transfer_metadata does all the hard work of configuring output file

```
#include "mcpl.h"
void filter_example()
{
  mcpl_file_t fi = mcpl_open_file("all.mcpl");
  mcpl_outfile_t fo = mcpl_create_outfile("lowEneutrons.mcpl");
  mcpl_transfer_metadata(fi, fo);
  mcpl_hdr_add_comment(fo,"Only neutrons, ekin<0.1MeV");
  const mcpl_particle_t* prtcl;
  while ( ( prtcl = mcpl_read(fi) ) ) {
    if ( prtcl->pdgcode == 2112 && prtcl->ekin < 0.1 )
      mcpl_transfer_last_read_particle(fi,fo);
  }
  mcpl_close_outfile(fo);
  mcpl_close_file(fi);
}
```

mcpl_transfer_last_read_particle from MCPL v1.3.0 prevents lossy unpacking+repacking of data. If need to edit particles fields, replace with:
mcpl_add_particle(fo,prtcl);

Python API (readonly)

To enable MCPL Python module, download mcpl.py or do
`python -mpip install mcpl`
(this incidently also installs the pymcpltool…)

Technical details:
- Pure Python, does not use mcpl.c
- Usage of Numpy for efficiency.
- Works with both Python 2 and 3.
- Readonly access for now.

```
import mcpl
myfile = mcpl.MCPLFile("myfile.mcpl")
for p in myfile.particles:
   print( p.x, p.y, p.z, p.ekin )
```

Accessing particles is straight-forward

Can also process blocks of N particles at a time, for increased efficency.

```
for p in myfile.particle_blocks:
   print( p.x, p.y, p.z, p.ekin )
```

Numpy arrays of length N

```
print( myfile.sourcename,
       myfile.nparticles,
       myfile.opt_singleprec )
for cmt in myfile.comments:
    print( 'Comment: "%s"' % cmt )
```

Can of course access meta data as well.

Command-line tools

- **mcpltool** and **pymcpltool**, both can:
  - **Inspect** files, extract binary blobs metadata to stdout
  - Convert MCPL to (inefficient) **ASCII** files for interoperability with software lacking MCPL support.
  - Show all options with **--help**
- The **mcpltool**:
  - Compiled executable with C compiler (from “fat” or proper linked code)
  - Can edit files:
    - **Merge** files
    - **Extract** subset of particles to smaller file (select by type or file idx)
    - **Repair** files leftover by crashed jobs
- The **pymcpltool**:
  - Built upon Python API (fast because of Numpy)
  - Download 1 file + run, or “pip install mcpl”
  - Can provide **statistics** (see next slide)

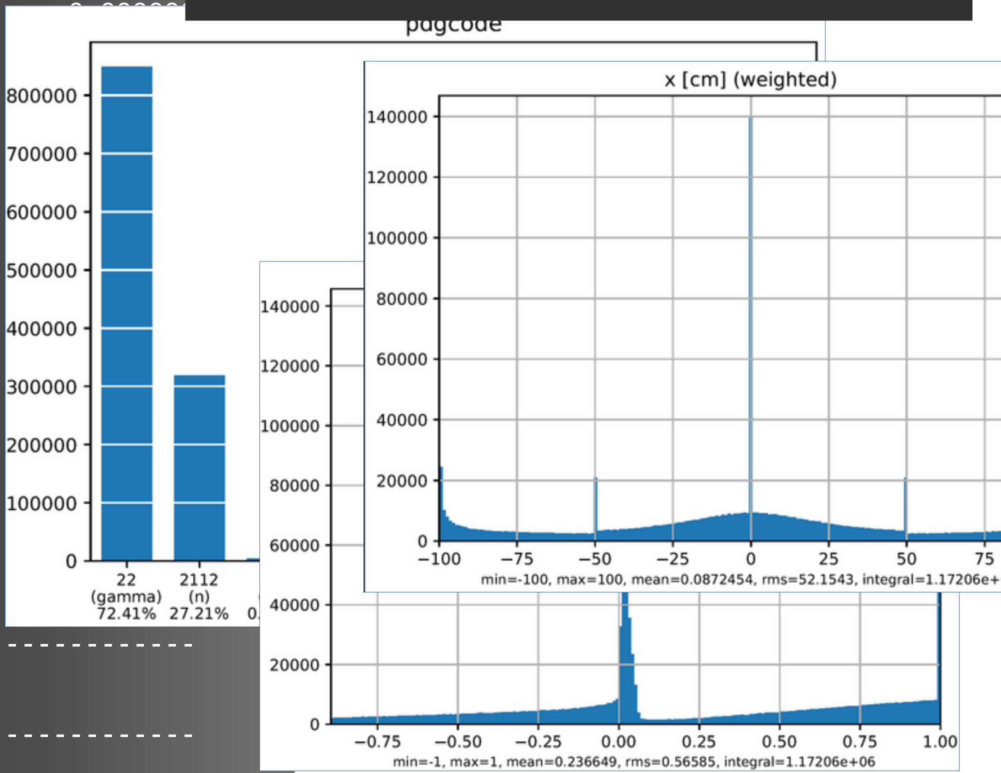
File statistics with pymcpltool

```
pymcpltool --stats <filename>
```

```
---------------------------------------------------------------------------------
nparticles    : 1172044
sum(weights) : 1.17206e+06
---------------------------------------------------------------------------------
              :          mean            rms            min              ma
---------------------------------------------------------------------------------
ekin   [MeV] :        0.68247        14.1939      9.7657e-11          1889.4
x       [cm] :      0.0872454        52.1543            -100             10
y       [cm] :      0.0192493        52.1484            -100             10
z       [cm] :        98.2832        78.6334    -5.55112e-17             25
ux           :   -0.000322662       0.558483              -1
uy           :    6.59925e-05       0.558487       -0.999998
uz           :       0.236649        0.56585              -1
time    [ms] :          24658     4.3971e+06       1.462e-06
weight       :        1.00001     0.00483571        0.654834
polx         :    0.000415962      0.0178829               0
poly         :    0.000166385     0.00715315               0
polz         :    0.000499154      0.0214595               0
---------------------------------------------------------------------
pdgcode      :          22 (gamma)              848745 (72.41%)
                      2112 (n)                  318868 (27.21%)
                        11 (e-)                   3922 ( 0.33%)
                       -11 (e+)                    431 ( 0.04%)
                      2212 (p)                      80 ( 0.01%)
                       211 (pi+)                     5 ( 0.00%)
                       -12 (nu_e-bar)                4 ( 0.00%)
                1000010030 (T)                       2 ( 0.00%)
                        14 (nu_mu)                   2 ( 0.00%)
                1000020040 (alpha)                   1 ( 0.00%)
                      -211 (pi-)                     1 ( 0.00%)
                     [ values ]              [ weighted counts ]
---------------------------------------------------------------------------------
userflags    :          0 (0x00000000)  1.17206e+06 (100.00%)
                     [ values ]              [ weighted counts ]
---------------------------------------------------------------------------------
```

```
pymcpltool --stats --gui <filename>
```

```
pymcpltool --stats --pdf <filename>
```



Merging files

- Ability to merge files is crucial for collecting output of concurrent simulations.
  - But other use-cases exists for combining files.
- Done via “**mcpltool --merge**” or “**mcpl_merge_files(..)**“ in the C API.
- As a quality concern, MCPL is conservative about not producing files with misleading meta-data.
- All meta-data must be identical and will be transferred to the newly created file.
- On some occasions this restriction has caused problems...

Introduced “mcpltool --forcemerge” in release 1.3.0

- Can always merge, but will ***throw away all meta-data***.
  - Should be considered as a last resort only!
- ***Particle data format options adapted*** to accommodate particles from all input files.
  - Options concerning FP prec., polarisation, fixed pdg/weight set as needed.
  - Discards userflags by default, since these are normally documented in metadata [override with --keepuserflags]
- Loss-less particle data transfer whenever possible.

```
Opened MCPL file forcemerged.mcpl:

  Basic info
    Format             : MCPL-3
    No. of particles   : 1170823
    Header storage     : 9[illegible] bytes
    Data storage       : 79615964 bytes

  Custom meta data
    Source             : "mcpl_forcemerge_files (from MCPL v1.3.0)"
    Number of comments : 0
    Number of blobs    : 0

  Particle data format
    User flags         : no
    Polarisation info  : no
    Fixed part. type   : no
    Fixed part. weight : no
    FP precision       : double
    Endianness         : little
    Storage            : 68 bytes/particle

index     pdgcode   ekin[MeV]       x[cm]       y[cm]       z[cm]
    0          22    0.040287      59.118      67.828         250
    1          22    0.048627      19.774     -98.025      197.92
    2          22    0.044083     -81.242      58.308      71.342
    3          22    0.042855      70.895     -70.526      8.9938
    4          22        0.05     -68.413     -72.936      160.88
    5          22    0.049592     -95.998     -28.005      223.32
    6          22    0.042521       84.72     -7.3153         250
    7          22     0.04898     -52.851     -84.892       26.98
    8          22    0.045358      66.239     -74.916      127.78
    9          22     0.04368     -98.073      19.537      219.24
```

How to use MCPL in McStas

- There's not much to it! Ships with components for **input**:

```
COMPONENT vin = MCPL_input( filename="myfile.mcpl" )
AT(0,0,0) RELATIVE Origin
```

  - Ignores non-neutrons.
  - See mcdoc MCPL_input for options controlling max energy of neutrons, or smearing of input particle properties.

- And the **output** component:

```
COMPONENT mcplout = MCPL_output( filename="myoutput.mcpl" )
AT(0,0,0) RELATIVE PREVIOUS
```

  - See mcdoc MCPL_output for options controlling e.g. floating point precision.
  - Also possible for advanced users to set MCPL userflags.

- Good to be aware of a few design decisions by Peter & Erik in how McStas deals with MCPL files:
  - Will always use the full MCPL file (so ignores -n flag to mcrun). Use the repeat_count option of MCPL_input allows replaying the file multiple times.
  - Concurrent modes (MPI/GPU) will replay the full MCPL file in each process, possibly with smearing.

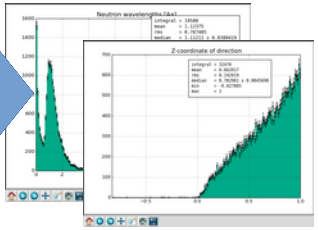
Outlook / wishful thinking

Funding missing

- Github issue 6: Mergeable statistics? E.g. “NEvtsSimulated” which would be added when files are merged. Would allow easier book-keeping.
- Github issue 44: In ESS Detector Group we have internal C++-based enhanced tools for working with MCPL files, based on our ExpressionParser and histogram classes:

```
mcplfilterfile in.mcpl.gz  out.mcpl.gz “time<2ms and is_neutron and neutron_wl>2.2Aa”
```

```
mcplbrowse in.mcpl.gz where “pdgcode!=11 and ekin<10keV”
```



GEANT4 A SIMULATION TOOLKIT

```
gen.input_file = "myfile.mcpl.gz"
gen.input_filter = "ekin>1keV && sqrt(x^2+y^2) < 10 cm"
```

  - It would be great to export these tools to the greater community, but needs significant work to disentangle and prepare.
- IMHO if the Python API would not be read-only, we could easily build and easily distribute a lot of great new tools (e.g. GUI for editing). It would also be easy for people to compose/filter their own MCPL files from cmdline or code.
- Nice-to-have: An actual modern C++ interface with all the safety and convenience guarantees that entails.