BDSIM Documentation

Release 1.3.2

BDSIM Collaboration

Apr 20, 2019
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Licence &amp; Disclaimer</td>
<td>3</td>
</tr>
<tr>
<td>1.1</td>
<td>Licence</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Referencing &amp; Citation</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>Geant4 Acknowledgement</td>
<td>3</td>
</tr>
<tr>
<td>1.4</td>
<td>Funding Acknowledgements</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>Authorship</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Current Authors</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Past Authors</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Introduction</td>
<td>7</td>
</tr>
<tr>
<td>3.1</td>
<td>Purpose of BDSIM</td>
<td>7</td>
</tr>
<tr>
<td>3.2</td>
<td>Capabilities</td>
<td>8</td>
</tr>
<tr>
<td>3.3</td>
<td>Simulation Procedure</td>
<td>8</td>
</tr>
<tr>
<td>3.4</td>
<td>How BDSIM Works</td>
<td>8</td>
</tr>
<tr>
<td>3.5</td>
<td>Tracking</td>
<td>11</td>
</tr>
<tr>
<td>3.6</td>
<td>Limits</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>Installation</td>
<td>13</td>
</tr>
<tr>
<td>4.1</td>
<td>Supported Systems</td>
<td>13</td>
</tr>
<tr>
<td>4.2</td>
<td>Obtaining BDSIM</td>
<td>13</td>
</tr>
<tr>
<td>4.3</td>
<td>Requirements &amp; Environment</td>
<td>14</td>
</tr>
<tr>
<td>4.4</td>
<td>Setting Up</td>
<td>15</td>
</tr>
<tr>
<td>4.5</td>
<td>Upgrading BDSIM</td>
<td>22</td>
</tr>
<tr>
<td>4.6</td>
<td>Troubleshooting</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Running BDSIM</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>Basic Operation</td>
<td>27</td>
</tr>
<tr>
<td>5.2</td>
<td>Executable Options</td>
<td>27</td>
</tr>
<tr>
<td>5.3</td>
<td>Interactively</td>
<td>29</td>
</tr>
<tr>
<td>5.4</td>
<td>In Batch Mode</td>
<td>29</td>
</tr>
<tr>
<td>5.5</td>
<td>Recreate Mode</td>
<td>31</td>
</tr>
<tr>
<td>6</td>
<td>Model Description - Input Syntax</td>
<td>33</td>
</tr>
<tr>
<td>6.1</td>
<td>Lattice Description</td>
<td>34</td>
</tr>
<tr>
<td>6.2</td>
<td>GMAD Syntax</td>
<td>34</td>
</tr>
<tr>
<td>6.3</td>
<td>Coordinates &amp; Units</td>
<td>36</td>
</tr>
<tr>
<td>6.4</td>
<td>Circular Machines</td>
<td>37</td>
</tr>
<tr>
<td>6.5</td>
<td>Beamline Elements</td>
<td>38</td>
</tr>
<tr>
<td>6.6</td>
<td>Aperture Parameters</td>
<td>63</td>
</tr>
<tr>
<td>6.7</td>
<td>Magnet Geometry Parameters</td>
<td>64</td>
</tr>
<tr>
<td>6.8</td>
<td>Offsets &amp; Tilts - Component Misalignment</td>
<td>74</td>
</tr>
<tr>
<td>6.9</td>
<td>Cavity Geometry Parameters</td>
<td>74</td>
</tr>
<tr>
<td>6.10</td>
<td>Fields</td>
<td>76</td>
</tr>
<tr>
<td>6.11</td>
<td>Externally Provided Geometry</td>
<td>80</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>6.12 Placements</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>6.13 Lattice Sequence</td>
<td>83</td>
<td></td>
</tr>
<tr>
<td>6.14 Output at a Plane - Samplers</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>6.15 Physics Processes</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>6.16 Physics Biasing</td>
<td>94</td>
<td></td>
</tr>
<tr>
<td>6.17 Options</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>6.18 Beam Parameters</td>
<td>108</td>
<td></td>
</tr>
<tr>
<td>6.19 Tunnel Geometry</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>6.20 Materials and Atoms</td>
<td>120</td>
<td></td>
</tr>
<tr>
<td>6.21 Crystals</td>
<td>124</td>
<td></td>
</tr>
<tr>
<td>6.22 Regions</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>6.23 Bends</td>
<td>126</td>
<td></td>
</tr>
<tr>
<td>6.24 Colours</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>6.25 Controlling Simulation Speed</td>
<td>129</td>
<td></td>
</tr>
<tr>
<td>6.26 Multiple Beam Lines</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>7 External Geometry Formats</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>7.1 GDML</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>7.2 GDML Preparation</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>7.3 ggmad</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>7.4 Mokka</td>
<td>138</td>
<td></td>
</tr>
<tr>
<td>8 Model Conversion</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>8.1 Manual Preparation</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>8.2 MAD-X Conversion</td>
<td>145</td>
<td></td>
</tr>
<tr>
<td>8.3 MAD8 Conversion</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>8.4 Python Builder</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>8.5 Optical Validation</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>9 Output</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>9.1 File Writing Policy</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>9.2 Output Information</td>
<td>153</td>
<td></td>
</tr>
<tr>
<td>9.3 Particle Identification</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>9.4 Output Data Selection &amp; Reduction</td>
<td>154</td>
<td></td>
</tr>
<tr>
<td>9.5 Output Files</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>9.6 Basic Data Inspection</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>9.7 Structure Of Output</td>
<td>156</td>
<td></td>
</tr>
<tr>
<td>10 Output Analysis</td>
<td>175</td>
<td></td>
</tr>
<tr>
<td>10.1 Setup</td>
<td>175</td>
<td></td>
</tr>
<tr>
<td>10.2 Quick Recipes</td>
<td>175</td>
<td></td>
</tr>
<tr>
<td>10.3 rebdsim - General Analysis Tool</td>
<td>176</td>
<td></td>
</tr>
<tr>
<td>10.4 rebdsimCombine - Output Combination</td>
<td>179</td>
<td></td>
</tr>
<tr>
<td>10.5 rebdsimHistoMerge - Simple Histogram Merging</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>10.6 rebdsimOptics - Optical Functions</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>10.7 Speed &amp; Efficiency</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>10.8 User Analysis</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>10.9 Numerical Methods</td>
<td>188</td>
<td></td>
</tr>
<tr>
<td>11 Python Utilities</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>11.1 Installing Python Utilities</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>11.2 Documentation</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>11.3 Geometry Preparation</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>12 Visualisation</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>12.1 Colour Coding</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>12.2 Common Useful Commands</td>
<td>193</td>
<td></td>
</tr>
<tr>
<td>12.3 Visualisation Features</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td>12.4 Default and Custom Visualisers</td>
<td>199</td>
<td></td>
</tr>
</tbody>
</table>
13 Interfacing
  13.1 Compiling Against BDSIM ........................ 201
  13.2 Custom Beam Line Component ....................... 202

14 Examples & Tests
  14.1 Basic Examples .................................. 207
  14.2 Worked Examples ................................. 216
  14.3 Specific Machines ............................... 254
  14.4 Features ....................................... 257

15 Support
  15.1 BDSIM News Letter ................................ 285
  15.2 Trouble Running BDSIM ............................ 285
  15.3 Feature Request .................................. 285

16 Version History
  16.1 V1.3.2 - 2019/04/20 ............................. 287
  16.2 V1.3.1 - 2019/03/05 ............................. 288
  16.3 V1.3 - 2019/02/27 ................................ 288
  16.4 V1.2 - 2018/08/26 ................................ 294
  16.5 V1.1 - 2018/05/23 ................................ 298
  16.6 V1.0 - 2018/04/10 ................................ 298
  16.7 V0.993 - 2017/12/11 .............................. 301
  16.8 V0.992 - 2017/07/26 ................................ 302
  16.9 V0.991 - 2017/07/04 ................................ 303
  16.10 V0.99 - 2017/06/09 ................................ 304
  16.11 V0.95 - 2016/11/07 ................................ 306
  16.12 V0.94 - 2016/09/13 ................................ 307
  16.13 V0.93 - 2016/08/24 ................................ 308
  16.14 V0.92 - 2016/03/29 ................................ 310
  16.15 V0.91 - 2015/12/17 .............................. 312
  16.16 V0.9 - 2015/11/10 ............................... 313
  16.17 V0.8 - 2015/08/10 ............................... 315
  16.18 V0.702 2015/07/28 - Hotfix ...................... 316
  16.19 V0.701 2015/07/02 - Hotfix ...................... 316
  16.20 V0.7 - 2015/06/30 ................................ 317
  16.21 V0.65 - 2015/04/10 ................................ 317
  16.22 V0.64 - 2015/02/16 ................................ 318
  16.23 V0.63 - 2015/02/06 ................................ 318
  16.24 V0.62 - 2014/08/07 ................................ 318
  16.25 V0.61 - 2014/08/05 ................................ 318
  16.26 V0.6 - 2013/12/02 ................................ 318
  16.27 V0.5 - 2008/11/08 ................................ 318
  16.28 V0.4 - 2008/02/26 ................................ 318
  16.29 V0.3 - 2007/01/26 ................................ 318
  16.30 V0.2 - 2006/05/18 ................................ 318
  16.31 V0.1 - 2006/02/22 ................................ 318
  16.32 beta - 2005/05/01 ................................ 318

17 Developer Documentation
  17.1 Purpose of Developer Documentation .................. 319
  17.2 Style Guide ........................................ 319
  17.3 Release Checklist ................................... 321
  17.4 Program Layout ...................................... 322
  17.5 Build System & Testing .............................. 323
  17.6 Parser ............................................... 323
  17.7 Geant4 User Action Classes .......................... 326
17.8 Geometry .................................................. 328
17.9 Tracking Algorithms ........................................... 331
17.10 Fields .......................................................... 347
17.11 Beam Generation ............................................. 373
17.12 Sensitivity, Output & Analysis .............................. 374
17.13 Analysis Suite .................................................. 376

18 Indices and tables ........................................ 379
Contents:
1.1 Licence

Beam Delivery Simulation (BDSIM) Copyright (C) Royal Holloway, University of London 2001 - 2019.
BDSIM is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation version 3 of the License.
BDSIM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
You should have received a copy of the GNU General Public License along with BDSIM. If not, see http://www.gnu.org/licenses/.

1.2 Referencing & Citation

Any publications including simulations made using BDSIM should cite:
Please cite BDSIM as well as Geant4.

1.3 Geant4 Acknowledgement

This software includes software developed by Members of the Geant4 Collaboration (http://cern.ch/geant4).

1.4 Funding Acknowledgements

BDSIM development has received funding from the following sources:

- John Adams Institute
- STFC
BDSIM was originally started by G.A. Blair around 2001 and has since been developed and maintained by a group based at Royal Holloway, University of London.

We also welcome contributions and additions to BDSIM. Please contact us to get involved!

2.1 Current Authors

- Laurie Nevay (RHUL, lead developer)
- Stewart Boogert (RHUL, lead developer)
- Andrey Abramov (RHUL)
- Joshua Albrecht (RHUL)
- Siobhan Alden (RHUL)
- Hector Garcia Morales (RHUL, CERN)
- Stephen Gibson (RHUL)
- Helena Pikhartova (RHUL)
- William Shields (RHUL)
- Jochem Snuverink (RHUL, PSI)
- Stuart Walker (RHUL)

2.2 Past Authors

- Grahame Blair (RHUL)
- Lawrence Deacon (RHUL, CERN, UCL)
- Regina Kwee-Hinzmann (RHUL, CERN)
- Jaime Van Oers (RHUL)
- Will Parker (RHUL)
- Yngve Levinsen (CERN)
- Giovanni Marchiori (INFN)
- Olivier Dadoun (LAL)
- Helmut Burkhardt (CERN)
- Daniel Brewer (RHUL)
- Robert Ainsworth (RHUL)
• Stephen Malton (RHUL)
• John Carter (RHUL)
• Ilya Agapov (RHUL)
3.1 Purpose of BDSIM

Beam Delivery Simulation (BDSIM) is a C++ program that utilises the Geant4 toolkit to simulate both the transport of particles in an accelerator and their interaction with the accelerator material. BDSIM is capable of simulating a wide variety of accelerator components and magnets with Geant4 geometry dynamically built based on a text input file. Thick lens accelerator tracking routines are provided for fast accurate tracking in a vacuum.

3.1.1 What BDSIM is suitable for

- Single particle Monte-Carlo simulations of particle accelerators
- Simulating beam loss in a particle accelerator
- Simulating particle transport in an accelerator where matter interaction is expected, such as transport in air.
- Simulating detector backgrounds from halo and machine background sources

3.1.2 What BDSIM is not intended for

- Long-term tracking studies
- Simulating collective effects
- Lattice optical design and optimisation
- A replacement for tracking codes like MAD-X, SixTrack or PTC

3.1.3 Example Applications

- Detector background from the accelerator
- Beam transport in air
- Beam interaction with vacuum gas
- Losses in beam extraction
- Collimation system efficiency and secondary radiation
- Transport of products from target through accelerator
- LHC beam loss and energy deposition
- CLIC muon background from the accelerator
- Laserwire detector signal to background ratio
- ILC collimator efficiency study and detector backgrounds

See *Worked Examples* for walk through examples with exact commands and input required and example plots.
3.2 Capabilities

- BDSIM uses ASCII text input with a syntax designed to be very similar to MAD8 / MAD-X.
- Convert MAD-X / MAD8 / TRANSPORT model to a 3D model in minutes
- Generate beam distribution according to Twiss parameters of a beam
- Track beam distribution and records particle distribution after each component
- Simulate energy deposition in all components along beam line
- Calculate beam distribution and Twiss optical functions from particle distribution
- Use the full set of physics processes available in Geant4
- Adjust cross-sections of processes of interest
- Use externally provided geometry and field maps for a fully customised model
- Interactively visualise a model in 3D as well as particle tracks
- Analyse history and origin of radiation produced in accelerator with analysis suite
- Strong reproducibility - recreate any event again exactly

3.3 Simulation Procedure

1) Create a text input .gmad lattice for BDSIM by converting a MAD-X or MAD8 Twiss file or writing your own manually.
2) Run BDSIM with core beam distribution for validation of optics and therefore model preparation.
3) Run BDSIM with desired input distribution and physics processes with low statistics to verify desired application.
4) Repeat 3) with greater statistics either as a single instance or on a computing cluster.
5) Analyse output data as desired.

3.4 How BDSIM Works

- BDSIM builds a complete Geant4 model and runs it as a Geant4 model.
- BDSIM does not link to another particle tracking code.
- BDSIM does not pass information back and forth between two codes.
- Thick lens tracking routines are used in place of normal 4th order Runge-Kutta integrators.
- BDSIM provides the required transforms between Cartesian and Curvilinear coordinate systems for accelerator tracking routines.

In a Geant4 program, code is written in C++ to construct a 3D model of the object to be simulated. A Geant4 example is shown below being interactively visualised.

This is labour-intensive and inflexible for different accelerator models or optics. As accelerators typically consist of a standard set of components these can be made reasonably generic. BDSIM provides a library of geometries and fields that allow simple optical descriptions to be made into 3D models. Example screenshots are shown below.

Apart from the 3D geometry, a crucial component of a model is the electromagnetic fields. Fields in Geant4 may be specified through a developer-provided C++ class that returns the field vector as a function of global Cartesian variables x, y, z and t. BDSIM provides classes to describe the magnetic fields found for each type of accelerator magnet as well as the transforms so that they can be described locally with respect to a particular magnet.
Fig. 1: Example Geant4 program being visualised with events displayed.

Fig. 2: “beamLoss” example of four quadrupoles in a small tunnel section.

3.4. How BDSIM Works
Fig. 3: Accelerator Test Facility 2 in KEK, Japan with tunnel model.

Fig. 4: Accelerator Test Facility 2 in KEK, Japan with tunnel model.
To calculate the motion of charged particle in a field, Geant4 uses a numerical integrator such as a 4th Order Runge-Kutta algorithm. This is the most general solution for a varying field but in an accelerator the specific fields have specific analytical solutions that can be used for improved accuracy and computational efficiency. BDSIM provides these tracking routines for “thick lens” tracking.

These ‘integrators’ are typically constructed with a strength that represents the field (such as $kI$ for a quadrupole) and the field vector $\vec{B}$ is ignored. Of course, in a full radiation transport simulation, there can be many different types of particles in all directions (even backwards). The thick lens tracking routines do not work for particles travelling backwards or perpendicular, so we resort back to a numerical integrator (typically 4th order Runge-Kutta) in these cases. The thick lens routines are used for paraxial particles only.

Thick lens tracking routines typically work in a curvilinear coordinate system that follows the reference trajectory, whereas Geant4 must work in global Cartesian coordinates. BDSIM bridges these two systems with an automatically created parallel geometry of simple cylinders that follow the beam line. Using the coordinate system of this parallel geometry creates transforms between coordinate systems.

### 3.5 Tracking

There are a variety of particle tracking routines and BDSIM supplies several sets. The set “bdsimmatrix” issued *out-of-the-box* uses thick lens tracking and provides agreement with MAD-X and PTC tracking codes.

A second set of routines called “bdsimtwo” is similar but differs in the way dipoles magnets are treated. In this case, a constant pure dipole field is used to calculate the motion of the particle (using a Rodrigues rotation in global Cartesian coordinates). The field is a ‘hard-edge’ field - it exists inside the volume at the same strength everywhere and is zero outside. Whilst the tracking algorithm is accurate, such a model does not agree with MAD-X or PTC when the dipoles have angled pole faces. This integrator set is computationally more efficient than the “bdsimmatrix” set, as no transforms between Cartesian and curvilinear coordinate systems are required for dipoles. In the case of a high-energy accelerator with no pole face angles or low angle bends, “bdsimtwo” may safely be used for accurate results and increased performance.

**Note:** With the “bdsimmatrix” routines, the tracking associated with the pole face angle is handled not by the physical shape of the magnet but by the thick lens matrix. Therefore, no pole face angles are physically constructed. The tracking however does represent the pole faces. Developments underway will allow both correct tracking with the thick lens matrix and the physical angled pole face.

### 3.6 Limits

#### 3.6.1 Energy

The user must understand the validity of the Geant4 models used and the applicability of the physics processes / models at their energy regime. Most Geant4 high-energy processes will not work above (and including) 50 TeV for a single particle. In Geant4.10.4, limits have been raised to 100 TeV.

#### 3.6.2 Model Physical Size

BDSIM uses a small padding distance between all surfaces and in addition, Geant4 treats the intersection with every surface of every solid with a certain tolerance. Specifying a tolerance like this avoids infinite recursion (or at least costly recursion) to ascertain the intersection of a curved track with a surface. This tolerance is by default $10^{-9}$ mm. BDSIM and Geant4 use double floating point precision throughout providing approximately 15 to 16 significant figures. Therefore, a maximum size of a model while still maintaining tracking precision is $10^9$ mm. This leads us to conclude that a model of order the size of the LHC is a practical maximum. Developments are underway to dynamically adjust this tolerance so as to increase this size. Please contact us for advice (see Feature Request). However, Geant4 and CLHEP are not fully templated (yet) to allow the use of higher precision numbers.
4.1 Supported Systems

BDSIM is developed and used on Mac OSX and Linux.

Tested systems:

- Mac OSX 10.14.4 (Mojave), XCode 10.2.1 (Apple LLVM version 10.0.1 (clang-1001.0.46.4)), Geant4.10.5.p01, ROOT 6.16/00, CLHEP 2.4.1.0, Qt5.12.1
- Mac OSX 10.14.3 (Mojave), XCode 10.1, Geant4.10.5, ROOT 6.16/00, CLHEP 2.4.1.0, Qt5.12.0
- Mac OSX 10.13.3 (High Sierra), XCode 10.1, Geant4.10.4.p02, ROOT 6.12/06, CLHEP 2.3.4.4, Qt5.12.0
- SLC6, GCC 4.9.3, Geant4.10.5.1, ROOT 6.10/08, CLHEP 2.3.3.0, Qt5.7.0
- SLC6 as above with Geant4.10.4.p02, Geant4.10.3.p03, Geant4.10.2.p03, Geant4.10.1.p03

4.2 Obtaining BDSIM

BDSIM may be obtained either from the BDSIM website or the git repository may be cloned. The user must compile it on their system and must have Geant4, CLHEP and ROOT already present (or access to AFS).

Obtaining via the git repository allows easier updates in future as the user can ‘pull’ the latest version and then recompile without having to create a separate copy.

4.2.1 From the GIT Repository

To download the source from the git repository, use the command:

```
git clone --recursive https://bitbucket.org/jairhul/bdsim
```

This will create a directory called bdsim, inside which all the code, examples and documentation is provided. Also, the python utilities that come with BDSIM will be present when the --recursive option is used.

4.2.2 Download

BDSIM versions can be downloaded from the git repository website:

https://bitbucket.org/jairhul/bdsim/downloads/?tab=tags

4.2.3 From Precompiled Sources

BDSIM may also be downloaded from precompiled sources. These are available on: http://www.pp.rhul.ac.uk/bdsim/download
4.3 Requirements & Environment

1) A recent compiler with full C++11 support. Proven compiler versions are GCC 4.9 or higher, or clang 6 or higher.
2) **CMake** 2.8.12 or higher (Geant4.10.2 onward requires **CMake** 3.3 or higher).
3) **CLHEP** 2.1.3.1 or higher, see also **CLHEP Installation Guide**. Latest recommended.
4) **Optional** - Python 2.7 series for python utilities and easy data loading with ROOT.
5) **ROOT** 6.0 or higher, for output & analysis compiled with python 2.7 support (default is 3 series).
6) **Optional** - Qt5 libraries for best Geant4 visualiser.
7) **Optional** - Xerces-C++ 3.2 XML library for GDML geometry file loading in Geant4.
8) **Geant4** installed or access to **AFS**\(^1\). Version 4.10 or higher (latest patch of that release). See **Geant4 Installation Guide**
9) Flex 2.5.37 or higher.
10) Bison 2.3 or higher.

**Note:** These are listed in the correct order of installation / requirement.

For nice analysis and use of **pybdsim** for model conversion, we recommend Python2.7 with matplotlib and numpy. ROOT should be installed with Python support in this case.

4.3.1 Geant4 and ROOT Versions

We have found some problems with certain versions of software and these should be avoided. Generally, we recommend the latest patch version of Geant4. These are the problems we have found:

- Geant4.10.3.0 - excessively long overlap checking - 15mins per solid vs normal 40ms.
- Geant4.10.3.pX - generic biasing has no effect - same code works in every other version.
- Geant4.10.4.0 - crash within constructor of G4ExtrudedSolid used extensively in BDSIM.
- Geant4.10.5.0 - the cashkarp integrator for fields will always crash.

4.3.2 Mac OS X Issues

- Mac OSX Mojave - OpenGL visualisations in Geant4 appear to be missing in a grey screen or worse, bits of the interface double size. The user must use Qt 5.12.1 or greater for these issues to be resolved. This issue is documented here: https://bugzilla-geant4.kek.jp/show_bug.cgi?id=2104

4.3.3 Geant4 Environment

Note: even though installed, the Geant4 environmental variables must be available. You can test this in a terminal with:

```
> echo $G4 <tab>
$G4ABLADATA $G4NEUTRONHPDATA $G4RADIOACTIVE DATA
$G4L EDATA $G4NEUTRONXSDATA $G4REAL SURFACEDATA
$G4LEVELGAMMA DATA $G4PIIDATA $G4SAIDXSDATA
```

\(^1\) **Note:** the use of **AFS** with the Mac OS build of BDSIM is not supported, as there is no compatible version of Geant4 available on AFS.
If these do not exist, please source the Geant4 environmental script before installing BDSIM and each time before using BDSIM. It is common to add this to your `.bashrc` or profile so that it’s loaded automatically every time:

```
source path/to/geant4/installation/bin/geant4.sh
```

## 4.4 Setting Up

The following sections detail the setup process for different operating systems.

- **Mac OSX**
- **Linux**

### 4.4.1 Mac OSX

- XCode should be installed.
- XCode command lines should be installed (xcode-select –install).
- XQuartz should be installed - see https://www.xquartz.org.
- The `make` command is available in the terminal.

We recommend obtaining `Requirements & Environment` using the MacPorts package manager, although they can be obtained both through other package managers and by manually downloading, compiling and installing the source for each.

For MacPorts you can do:

```
sudo port install root6 +python27
sudo port install xercesc3 flex bison clhep qt5
sudo port install py27-matplotlib py27-numpy
```

- It is best to install Geant4 manually to ensure you use the system CLHEP option (required by BDSIM for strong reproducibility) as well as visualiser choices and GDML geometry loading.

After this, Building can be started.

**Warning:** For Mac OSX Mojave, see [Mac OS X Issues](#).

### 4.4.2 Linux

Install the `Requirements & Environment` preferably with a package manager.

Older versions of Geant4 can be downloaded from their archive. For Scientific Linux 6 or modern Linux versions, we recommend the latest version of Geant4 (currently 4.10.2). Note: the required compiler version (GCC 4.9) is more modern than the default one (GCC 4.4) on SL6. You can check the compiler version with:

```
gcc --version
```

With AFS access version 4.9 can be found here:

```
source /afs/cern.ch/sw/lcg/external/gcc/4.9/x86_64-slc6-gcc49-opt/setup.sh
```

After this, Building can be started.
4.4.3 Building

Once ready, make a directory outside the BDSIM source directory to build BDSIM in:

```
> ls
bdsim
> mkdir bdsim-build
> ls
bdsim bdsim-build
```

It is important that the build directory be outside the source directory, otherwise trouble may be encountered when receiving further updates from the git repository. From this directory use the following CMake command to configure the BDSIM installation:

```
> cd bdsim-build
> cmake ../bdsim
```

This typically produces the following output, which is slightly different on each computer:

```
-- The C compiler identification is AppleClang 8.0.0.8000042
-- The CXX compiler identification is AppleClang 8.0.0.8000042
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/
    Toolchains/XcodeDefault.xctoolchain/usr/bin/cc
-- Check for working C compiler: /Applications/Xcode.app/Contents/Developer/
    Toolchains/XcodeDefault.xctoolchain/usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/
    Toolchains/XcodeDefault.xctoolchain/usr/bin/c++
-- Check for working CXX compiler: /Applications/Xcode.app/Contents/Developer/
    Toolchains/XcodeDefault.xctoolchain/usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring BDSIM 1.2.develop
-- Build Type RelWithDebInfo
-- Compiler supports C++14
-- Looking for CLHEP... - found
-- Found CLHEP 2.4.1.0 in /opt/local/lib/CLHEP-2.4.1.0/.../.../include
-- Use ROOTSYS from environment: /Users/nevay/physics/packages/root-6.10.08-install
-- Looking for ROOT...
-- Found ROOT 6.10/08 in /Users/nevay/physics/packages/root-6.10.08-install
-- GDML support ON
-- Geant4 Use File: /Users/nevay/physics/packages/geant4.10.04.p02-install/lib/
    Geant4-10.4.2/UseGeant4.cmake
-- Geant4 Definitions: -DG4_STORE_TRAJECTORY;-DG4VERBOSE;-DG4UI_USE;-DG4VIS_USE;-
    -DG4UI_USE_TCSH;-DG4INTY_USE_XT;-DG4VIS_USE_RAYTRACERX;-DG4INTY_USE_QT;-DG4UI_USE_-
    -QT;-DG4VIS_USE_OPENGLQT;-DG4UI_USE_XM;-DG4VIS_USE_OPENGLXM;-DG4VIS_USE_OPENGLX;-
    -DG4VIS_USE_OPENGL
-- G4_VERSION: 10.4.2
-- Found Doxygen: /opt/local/bin/doxygen (found version "1.8.14")
-- components: doxygen dot
-- Found BISON: /opt/local/bin/bison (found suitable version "3.2.2", minimum
-- required is "2.4")
-- Found FLEX: /opt/local/bin/flex (found version "2.6.4")
-- Looking for zlib
-- Using Geant4 built in zlib
-- Performing Test COMPILER_HAS_HIDDEN_VISIBILITY
-- Performing Test COMPILER_HAS_HIDDEN_VISIBILITY - Success
```

(continues on next page)
CMake will search your system for the required dependencies. In the above example, this proceeded without any errors. In the case where a required dependency cannot be found, an error will be shown and CMake will stop. Please see Configuring the BDSIM Build with CMake for further details on how to fix this and further configure the BDSIM installation.

You can then compile BDSIM with:

```bash
> make
```

BDSIM can then be installed (default directory /usr/local) for access from anywhere on the system with:

```bash
> sudo make install
```

To change the installation directory, see Configuring the BDSIM Build with CMake. From any directory on your computer, bdsim should be available.

At this point, BDSIM itself will work, but more environmental variables must be set to use the analysis tools (this is a requirement of ROOT). These can be set by sourcing the bdsim.sh shell script in the installation directory:

```bash
source <bdsim-install-dir>/bin/bdsim.sh
```

This can be added to your .profile or .bashrc file. The user should adapt this if they use a C-shell.

- Re-source your profile (or restart the terminal).
- You should be able to execute bdsim --help or rebdsim

If the analysis will be regularly used interactively, it is worth automating the library loading in root by finding and editing the rootlogon.C in your <root-install-dir>/macros/ directory. Example text would be:

```c
cout << "Loading rebdsim libraries" << endl;
gSystem->Load("librebdsimLib");
gSystem->Load("libbdsimRootEvent");
```

**Note**: The file extension is omitted on purpose.

The absolute path is not necessary, as the above environmental variables are used by ROOT to find the library.
From the build directory you can verify your installation using a series of tests included with BDSIM (excluding long running tests):

```
> ctest -LE LONG
```

### Environmental Variables

These variables are required by ROOT to access the BDSIM classes and not by BDSIM itself. These variables are set in the `<bdsim-install-dir>/bin/bdsim.sh` provided shell script, but are also described here manually.

```bash
export BDSIM=<bdsim-install-dir>
export PATH=$PATH:$BDSIM/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$BDSIM/lib (Linux only)
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$BDSIM/lib (mac only)
export ROOT_INCLUDE_PATH=$BDSIM/include/bdsim/:$BDSIM/include/bdsim/analysis/:--$BDSIM/include/bdsim/parser
```

These can of course be manually added to your `.profile` or `.bashrc` file.

### 4.4.4 Python Utilities

- Quick setup: simply run `make` from the `bdsim/utils` directory.

BDSIM includes copies of our accompanying Python utilities (pytransport, pymad8, pymadx and pybdsim) that can now be installed. These all exist in separate git repositories in the following locations:

- https://bitbucket.org/jairhul/pybdsim
- https://bitbucket.org/jairhul/pymadx
- https://bitbucket.org/jairhul/pymad8
- https://bitbucket.org/jairhul/pytransport

These can all be set up separately, or alternatively the user can install all at once with the MakeFile added for convenience (running make command). The Python package installer (“PIP”) is required for this.

**Note:** ROOT should be compiled with Python2.7 support for the full functionality of pybdsim data loading to be exploited.

To set up all utilities at once:

```
cd bdsim/utils
make
```

The utilities should now be available through Python:

```
>>> import pybdsim
>>> import pymadx
>>> import pymad8
>>> import pytransport
```

**Note:** If it’s required to edit these utilities, please do not edit the copy in bdsim/utils, as this will cause problems with git and pulling changes. It is strongly recommended to clone each utility separately outside the BDSIM source directory and edit that version, leaving the included one untouched.
4.4.5 Configuring the BDSIM Build with CMake

To either enter paths to dependencies manually, or edit the configuration, the following command will give you an interface to CMake (from `bdsim-build` directory):

```
> ccmake .
```

You can then use `up` and `down` arrows to select the desired parameter and `enter` to edit it. If the parameter is a path, press `enter` again after entering the path to confirm.

Once the parameter has been edited, you can proceed by pressing `c` to run the configuration and if successful, follow this by `g` to generate the build. After configuring the installation, you should run:

```
> make
> sudo make install
```

Note, `sudo` is used here as the default installation directory will be a system folder. You can however, specify a different directory in the above `ccmake` configuration and that won’t require the `sudo` command. The installation directory can be specified by editing the `CMAKE_INSTALL_PREFIX` variable.

4.4.6 Making the Manual

The manual is available online at http://www.pp.rhul.ac.uk/bdsim/manual and included as a pdf in the source directory, but if desired the user can compile the manual in both HTML and pdflatex from the build directory using the following command to make the HTML manual in the folder `manual/html`:

```
> make manual
```

Similarly:

```
> make manual-pdf
```

will make the pdf Manual in the folder `manual/latex`.
4.4.7 Making Doxygen Code Documentation

Doxygen code documentation is available online at http://www.pp.rhul.ac.uk/bdsim/doxygen/

If desired the user can create this from the build directory using the following command to make the Doxygen documentation in a folder called Doxygen:

> make doc

Note: This requires the Doxygen documentation system to be installed.

4.4.8 CLHEP Installation Guide

If not installed with a package manager, download CLHEP-2.3.1.1 or a newer version from the CLHEP website.

Move and unpack to a suitable place:

> tar -xzf clhep-2.3.1.1.tgz
> cd 2.3.1.1

Make build directory:

> mkdir build
> cd build
> cmake ../CLHEP

Adapt parameters if needed with:

> ccmake .

Make and install:

> make
> sudo make install

4.4.9 Geant4 Installation Guide

BDSIM builds with most recent versions of Geant4 (version 4.10 onwards). You can usually get Geant4 through a package manager such as MacPorts or Brew, but often a manual installation is more flexible to allow choice of visualiser and use of GDML (necessary for external geometry). For manual installation, download the latest patch version 4.10.2 from the Geant website. Move and unpack to a suitable place

> tar -xzf geant4.10.5.tar.gz
> ls
geant4.10.5

Make a build and installation directory outside that directory

> mkdir geant4.10.5-build
> mkdir geant4.10.5-install

Configure Geant4 using CMake
At this point it’s useful to define the installation directory for Geant4 by modifying the CMake configuration as generally described in *Configuring the BDSIM Build with CMake.*

> ccmake .

It is useful to change a few options with Geant4 for practical purposes.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>CMAKE_INSTALL_PREFIX</code></td>
<td>Useful to specify a known folder to install to.</td>
</tr>
<tr>
<td><code>GEANT4_BUILD_CXXSTD</code></td>
<td>14 - For ROOT version 6 (and gcc compiler).</td>
</tr>
<tr>
<td><code>GEANT4_BUILD_MULTITHREADED</code></td>
<td>OFF - BDSIM does not support this yet.</td>
</tr>
<tr>
<td><code>GEANT4_INSTALL_DATA</code></td>
<td>ON - otherwise Geant will try to download data dynamically, as it’s required during the simulation and it may not be possible to run offline.</td>
</tr>
<tr>
<td><code>GEANT4_INSTALL_DATADIR</code></td>
<td>Useful to specify to a known folder you make. Typically any <code>CMAKE_INSTALL_PREFIX</code> / data.</td>
</tr>
<tr>
<td><code>GEANT4_USE_GDML</code></td>
<td>ON - for external geometry import.</td>
</tr>
<tr>
<td><code>GEANT4_USE_OPENGL_X11</code></td>
<td>ON - basic visualiser.</td>
</tr>
<tr>
<td><code>GEANT4_USE_QT</code></td>
<td>ON - the best and most interactive visualiser. Needs Qt to be installed.</td>
</tr>
<tr>
<td><code>GEANT4_USE_SYSTEM_CLHEP</code></td>
<td>ON - must be on so both Geant4 and BDSIM use the same CLHEP library. Therefore, there’s only one random number generator and simulations have strong reproducibility.</td>
</tr>
<tr>
<td><code>GEANT4_USE_SYSTEM_ZLIB</code></td>
<td>OFF - easier if we use the Geant4 internal version.</td>
</tr>
<tr>
<td><code>GEANT4_USE_RAYTRACER_X11</code></td>
<td>ON - The most accurate visualiser, but relatively slow and not interactive. Useful for promotional materials.</td>
</tr>
<tr>
<td><code>GEANT4_USE_XM</code></td>
<td>ON - similar to Qt and the one to use if Qt isn’t available. Needs motif to be installed.</td>
</tr>
</tbody>
</table>
**Warning:** Make sure GEANT4_BUILD_MULTITHREADED is off since this is currently not supported.

**Note:** The CLHEP option is required. The GDML and QT options are strongly recommended. Others are to the user’s preference.

Once the installation directory is set, press `c` to run the configuration process, and when complete, press `g` to generate the build. If `g` is not an available option, then continue to press `c` until it becomes available. This typically takes two or three times - it is due to dependencies being dependent on other dependencies. Geant4 can then be compiled

```
> make
```

Note: Geant4 can take around 20 minutes to compile on a typical computer. If your computer has multiple cores, you can significantly decrease the time required to compile by using extra cores

```
> make -jN
```

where \( N \) is the number of cores on your computer\(^2\). Geant4 should then be installed

```
> make install
```

Note: if you’ve specified the directory to install, you will not need the `sudo` command. However, if you’ve left the settings as default, it’ll be installed in a folder that requires `sudo` permissions such as `/usr/local/`.

**IMPORTANT** - you should source the Geant4 environment each time before running BDSIM, as this is required for the physics models of Geant4. This can be done using

```
> source path/to/geant4.10.5-install/bin/geant4.sh
```

It may be useful to add this command to your `.bashrc` or profile script.

### 4.5 Upgrading BDSIM

To update BDSIM when a new release is made, we recommend receiving updates through the git repository. To receive the latest version of the software, the user must ‘pull’ the changes from the git repository and then update the build.

**Note:** Assuming you have a BDSIM source directory (“bdsim”) that is a clone of the git repository and a separate build directory (“bdsim-build”) that is outside the source directory.

```
cd bdsim
git pull
git submodule update
```

You then have two options: 1) make a clean build or 2) update the current build. The first option is generally more robust and we recommend that. Both are described for completeness.

#### 4.5.1 Clean Build

\(^2\) If your computer supports hyper-threading, you can use twice the number of cores with the `make -jN` command (i.e. a computer has 4 cores and supports hyper-threading, can support up to `make -j8`). Exceeding this number will result in slower than normal compilation.
If custom locations for various dependencies had to be specified with CMake for the initial configuration and compilation of BDSIM, these will have to be repeated (see Configuring the BDSIM Build with CMake for details on using ccmake to do this).

### 4.5.2 Updated Existing Build

```bash
cd ../bdsim-build
cmake ../bdsim
make -j4
make install
```

### 4.6 Troubleshooting

Below is a list of possible encountered problems. If you experience problems beyond these, please contact us (see Support).

1) Visualisation does not work:

   "parameter value is not listed in the candidate List."

Check which graphics systems BDSIM has available. This is shown in the terminal when you run BDSIM

```
You have successfully registered the following graphics systems.
Current available graphics systems are:
ASCII Tree (ATree)
DAWNFILE (DAWNFILE)
G4HepRep (HepRepXML)
G4HepRepFile (HepRepFile)
OpenGLImmediateQt (OGLI, OGLIQt)
OpenGLImmediateX (OGLIX)
OpenGLImmediateXm (OGLIXm, OGLI_FALLBACK, OGLIQt_FALLBACK)
OpenGLStoredQt (OGL, OGLS, OGLSQt)
OpenGLStoredX (OGLSX)
OpenGLStoredXm (OGLSXm, OGL_Fallback, OGLSQt_FALLBACK)
RayTracer (RayTracer)
RayTracerX (RayTracerX)
VRMLlFILE (VRMLlFILE)
VRML2FILE (VRML2FILE)
gMocrenFile (gMocrenFile)
```

If your favourite is not there check that Geant4 is correctly compiled with that graphics system. You will have to reconfigure Geant4 and install any necessary libraries (such as Qt or XMotif), then recompile Geant4, then recompile bdsim.

2) Huge print out and failure when trying to load data in Python:

```python
In [1]: import pybdsim
d =

In [2]: d = pybdsim.Data.Load("run1.root")
```

**Error** in cling::AutoloadingVisitor::InsertIntoAutoloadingState:

(continues on next page)
Missing FileEntry for ../parser/beamBase.h
requested to autoload type GMAD::BeamBase
Error in cling::AutoloadingVisitor::InsertIntoAutoloadingState:
Missing FileEntry for ../parser/optionsBase.h
requested to autoload type GMAD::OptionsBase
HeaderDict dictionary payload:33:10: fatal error: 'BDSOutputROOTEventHeader.hh'
#include "BDSOutputROOTEventHeader.hh"
^~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Error in <TInterpreter::AutoParse>: Error parsing payload code for class
-- Header with content:

#line 1 "HeaderDict dictionary payload"

#ifndef G__VECTOR_HAS_CLASS_ITERATOR
#define G__VECTOR_HAS_CLASS_ITERATOR 1
#endif
#ifndef __ROOTBUILD__
#define __ROOTBUILD__ 1
#endif
#define _BACKWARD_BACKWARD_WARNING_H
/*
Beam Delivery Simulation (BDSIM) Copyright (C) Royal Holloway,

This file is part of BDSIM.

BDSIM is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published
by the Free Software Foundation version 3 of the License.

BDSIM is distributed in the hope that it will be useful, but
WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with BDSIM. If not, see <http://www.gnu.org/licenses/>.
*/
#ifndef ANALYSISHEADER_H
#define ANALYSISHEADER_H
#include "TROOT.h"
#include "BDSOutputROOTEventHeader.hh"
#include "RebdsimTypes.hh"

class TTree;
/**
 * @brief Options loader.
 * @author Laurie Nevay.
 */
class Header
{
public:
    Header();
}
Header(bool debugIn);
virtual ~Header();

/// Set the branch addresses to address the contents of the file.
void SetBranchAddress(TTree* t);

/// Member that ROOT can map file data to locally.
BDSOutputROOTEventHeader* header;

private:
bool debug;

ClassDef(Header,1);
}
#endif

Error in <TClass::LoadClassInfo>: no interpreter information for class Header_is available even though it has a TClass initialization routine.
Error in <TClass::LoadClassInfo>: no interpreter information for class Header_is available even though it has a TClass initialization routine.
---------------------------------------------------------------------------
RuntimeError
Traceback (most recent call last)
<ipython-input-2-ab00b7718588> in <module>()
----> 1 d = pybdsim.Data.Load("run1.root")
/Users/nevay/physics/reps/pybdsim/pybdsim/Data.pyc in Load(filepath)
 60 return _LoadAscii(filepath)
 61 elif extension == 'root':
--> 62 return _LoadRoot(filepath)
/Users/nevay/physics/reps/pybdsim/pybdsim/Data.pyc in _LoadRoot(filepath)
 149 LoadROOTLibraries()
 150 --> 151 fileType = _ROOTFileType(filepath) #throws warning if not a bdsim_file
 152 if fileType == "BDSIM":
 153 if not htree:
/Users/nevay/physics/reps/pybdsim/pybdsim/Data.pyc in _ROOTFileType(filepath)
 133 if not htree:
 134 raise Warning("ROOT file "'{}" is not a BDSIM one".
--> 135 h = _ROOT.Header()
 136 h.SetBranchAddress(htree)
 137 htree.GetEntry(0)

RuntimeError: Header::Header() =>
could not resolve ::()
G4OpenGLImmediateX::CreateViewer: error flagged by negative view id in G4OpenGLImmediateXViewer creation.

Check that your graphics card driver is installed correctly for your memory card and possibly reinstall them. For Ubuntu for example, run:

`fglrxinfo`

If fglrx is installed and working well you should see an output similar to:

```
> fglrxinfo
display: :0 screen: 0
OpenGL vendor string: Advanced Micro Devices, Inc.
OpenGL renderer string: ATI Radeon HD 4300/4500 Series
OpenGL version string: 3.3.11399 Compatibility Profile Context
```

For more info see [https://help.ubuntu.com/community/BinaryDriverHowto/AMD](https://help.ubuntu.com/community/BinaryDriverHowto/AMD)

3) Build does not work - GLIBCXX errors, where a message similar to this is shown

```
Linking CXX executable bdsim
/afs/cern.ch/sw/lcg/external/geant4/9.6.p02/x86_64-slc6-gcc46-opt
/lib64/libG4analysis.so: undefined reference to `std::__detail::_List_node_base::_M_unhook()@GLIBCXX_3.4.15'
```

This means that the compiler version for BDSIM is different from the one used to compile Geant4. Make sure it is the same compiler version. Remember to start from a clean build directory, otherwise CMake does NOT update the compiler version.

4) Build does not work - linker errors with xml and zlib like

```
/usr/lib64/libxml2.so: undefined reference to `gzdirect@ZLIB_1.2.2.3'
collect2: error: ld returned 1 exit status
```

This probably means that the xml library is not properly installed. The easiest option may be not to use this part of BDSIM by switching off the CMake variable USE_LCDD (in cmake).
RUNNING BDSIM

5.1 Basic Operation

The minimum required information to run BDSIM is an input GMAD file.

```bash
bdsim --file=lattice.gmad
```

The following section describes other ‘executable’ options that may specify various options, such as whether to run in ‘batch’ mode or interactively with a visualiser and where to write output.

5.2 Executable Options

BDSIM can be executed in a terminal with extra arguments to specify various inputs. The angular brackets here are to indicate a user-specified value and should not be used literally. The following command may be used to display all options:

```bash
bdsim --help
```
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-file=&lt;file&gt;</td>
<td>Specifies the input gmad file</td>
</tr>
<tr>
<td>-batch</td>
<td>Batch mode - no graphics</td>
</tr>
<tr>
<td>-circular</td>
<td>Assumes circular machine - turn control. See note below.</td>
</tr>
<tr>
<td>-colours</td>
<td>Print out all colours predefined in BDSIM and exit</td>
</tr>
<tr>
<td>-distrFile=&lt;file&gt;</td>
<td>Which file to use for the bunch distribution</td>
</tr>
<tr>
<td>-exportGeometryTo=&lt;file&gt;</td>
<td>Exports the geometry to a file extension, which determines the format where possible extensions are (“gdml”)</td>
</tr>
<tr>
<td>-geant4Macro=&lt;file&gt;</td>
<td>Optional Geant4 macro file to run after the visualisation has started. Only works in interactive visualisation.</td>
</tr>
<tr>
<td>-generatePrimariesOnly</td>
<td>Generates primary particle coordinates only then exits without simulating anything</td>
</tr>
<tr>
<td>-help</td>
<td>Lists all executable commands</td>
</tr>
<tr>
<td>-materials</td>
<td>Lists materials included in BDSIM by default</td>
</tr>
<tr>
<td>-ngenerate=N</td>
<td>The number of primary events to simulate overrides the ngenerate option in the input file.</td>
</tr>
<tr>
<td>-output=&lt;fmt&gt;</td>
<td>Outputs the format “rootevent” (default) or “none”</td>
</tr>
<tr>
<td>-outfile=&lt;file&gt;</td>
<td>Outputs file name. Will be appended with _N where N = 0, 1, 2, 3,...</td>
</tr>
<tr>
<td>-recreate=&lt;file&gt;</td>
<td>The rootevent output file to recreate events from.</td>
</tr>
<tr>
<td>-seed=&lt;N&gt;</td>
<td>Seed for the random number generator</td>
</tr>
<tr>
<td>-seedStateFileName=&lt;file&gt;</td>
<td>File containing CLHEP::Random seed state NB - this overrides other seed values</td>
</tr>
<tr>
<td>-startFromEvent=N</td>
<td>Event offset to start from when recreating events when using --recreate</td>
</tr>
<tr>
<td>-survey=&lt;file&gt;</td>
<td>Prints survey info to &lt;file&gt;</td>
</tr>
<tr>
<td>-printFractionEvents=N</td>
<td>How often to print out events as a fraction of the total number of events to simulate (default 0.1 i.e. 10%). -1 for all.</td>
</tr>
<tr>
<td>-printFractionTurns=N</td>
<td>How often to print out turns as a fraction of the total number of turns to simulate (default 0.2 i.e. 20%). -1 for all. Will only print out on an event that will print out as well.</td>
</tr>
<tr>
<td>--verbose</td>
<td>Displays general parameters before run</td>
</tr>
<tr>
<td>--verbose_event</td>
<td>Displays information for every event</td>
</tr>
<tr>
<td>--verbose_step</td>
<td>Displays tracking information after each step</td>
</tr>
<tr>
<td>--verbose_event_num=&lt;N&gt;</td>
<td>Displays tracking information for event ‘N’</td>
</tr>
<tr>
<td>--vis_debug</td>
<td>Displays all volumes in the visualiser</td>
</tr>
<tr>
<td>--vis_mac=&lt;file&gt;</td>
<td>File with the visualisation macro script. Default provided by BDSIM: openGL (OGLSQt)).</td>
</tr>
<tr>
<td>--writeSeedState</td>
<td>Writes an ASCII file seed state for each event</td>
</tr>
</tbody>
</table>

BDSIM can be run in one of two ways: interactively or in batch mode, which are described in the following sections.

When run interactively, a Geant4 visualiser is invoked that produces a window with an image of the BDSIM model as well as a terminal prompt to control it. No events are simulated without user input. BDSIM provides a basic visualisation “macro” for Geant4 using the Qt visualiser to start and add the geometry to the scene. This is found from the BDSIM installation directory or failing that the build directory. The user may provide their own custom visualisation macro with the executable command `--vis_mac=mymac.mac`.

Alternatively, BDSIM can be run in batch mode, where no visualiser is used and the specified number of primary events is simulated and feedback is printed to the terminal. Batch mode is typically much faster than the interactive mode, but the interactive mode is very useful for understanding the model and a typical event in the simulation, i.e. where a particle hits.

**Note:** For more information on the `--circular` option and using BDSIM with circular machines, see *Circular Machines.*
5.3 Interactively

Features:

• Default option
• Interactive visualisation of the accelerator model
• Ability to view and rotate the accelerator model
• Ability to run and view individual events
• Visualise events generated at run time
• Typically slower than batch mode
• No events run without user input

To execute BDSIM in interactive mode, the user must simply not use the --batch command. The user can also specify a macro file using the --vis_mac option above; otherwise, BDSIM will look for “vis.mac” in the current working directory. If not found, BDSIM will use its own default visualisation settings (typically the Qt visualiser, with extra menu buttons).

Example:

```
bdsim --file=sm.gmad --outfile=run2
```

This executes BDSIM for the simpleMachine example in bdsim/examples/simpleMachine with ROOT output (default) to a file named “run2”. The program is run interactively and the window shown below appears. From here, the user types into the visualiser terminal:

```
/run/beamOn 1
```

This runs one event and visualises it. Each time this command is used, a new output file with a numerical suffix will be created.

To exit the visualiser, in the visualiser terminal type:

```
exit
```

More details of how to use the visualiser and common commands can be found in Visualisation.

Note: The visualiser is part of Geant4, so if the desired visualiser isn’t available, you must recompile Geant4 with the correct visualiser (and subsequently BDSIM afterwards). Geant4 also uses the CMake configuration system. The visualiser shown is the OpenGL Qt visualiser, which we recommend for its ease of use and high level of interactivity.

Note: BDSIM simulates one particle at a time from the primary distribution and all of the associated secondaries. Each event is independent and different particles in the input bunch cannot interact with each other or their secondaries. This is an underlying feature of Geant4.

5.4 In Batch Mode

Features:

• No interactive visualiser
• Faster
• Number of events runs, then program quits
Fig. 1: BDSIM running interactively with OpenGL Qt visualiser from Geant4.
• No user input
• Typical use for a “job” on a farm

To execute BDSIM in batch mode, simply use the \texttt{--batch} execution option.

Examples:

\texttt{bdsim --file=atf2.gmad --outfile=run1 --batch --seed=123}

This executes BDSIM for the ATF2 example with ROOT output to a file name “run1” in batch mode with a seed value of 123. The simulation runs the number of events specified by the \texttt{ngenerate} options parameter in the input \texttt{gmad} file, which is 1 by default.

### 5.5 Recreate Mode

After performing a simulation in BDSIM, it is possible to reproduce one or more events exactly the same again - this is called “strong recreation”. To do this, the original input \texttt{gmad} files (and any associated external geometry, field maps, beam distribution files e.g. all the input) are required and should be the same as was originally used. Along with this, a BDSIM ROOT output file is required.

The output file is used to load the random number generator seed states at the start of each event such that the beam and physics processes will be the same. For example:

\texttt{bdsim --file=mymodel.gmad --outfile=run1 --batch --ngenerate=100}

Now let us recreate event 87 (0 counting):

\texttt{bdsim --file=mymodel.gmad --outfile=selectevent --batch --ngenerate=1 --recreate=run1.root --startFromEvent=87}

The relevant executable options are \texttt{recreate}, \texttt{startFromEvent}. These are also documented in \textit{General Run Options}.

Recreation can also be used by specifying options in the input \texttt{gmad} file. For example:

\texttt{! start with the original model}
\texttt{include mymodel.gmad;}
\texttt{option, recreate=1,
   recreateFileName="run1.root",
   startFromEvent=87,
   ngenerate=1;}

If the above GMAD syntax was in a file called “recreation1.gmad”, we would run it like:

\texttt{bdsim --file=recreation1.gmad --outfile=selectevent --batch}

This would be equivalent to the recreation example above. Note, the option \texttt{recreate} in GMAD is a Boolean (set to 1 or 0) but as an executable option it’s the path to the file (a string).

Notes:

• The event offset counting is 0 counting. So, the first event is index 0. This is consistent with the print out of event number in BDSIM.

• If the recreation goes beyond the stored number of events, the random number generator will proceed as normal. e.g. starting from event 80/100 and generating 30 events, will result in 10 new events.

• Executable options override whatever options were used (and therefore stored in the output) in the initial run of BDSIM.
• Changing physics options in your input as compared to the original model will result in different results. The primary particle coordinates will of course be the same. The random number generator is set at the beginning of each new event.

• If a user supplied bunch distribution is used, the reading of the bunch file will start from the correct event to fully recreate the exact same event again.
The following sections describe how to prepare a BDSIM model. These sections are provided in order of requirement.

- **Basic Language**
  - Lattice Description
  - GMAD Syntax
  - Mathematical Functions
  - Coordinates & Units

- **Circular Machines**

- **Beamline Elements**
  - Aperture Parameters
  - Magnet Geometry Parameters
  - Offsets & Tilts - Component Misalignment
  - Cavity Geometry Parameters

- **External Fields & Geometry**
  - Fields
  - Externally Provided Geometry Formats
  - Placements of Geometry

- **Sequence of Elements**
  - Lattice Sequence

- **Output at a Plane - Samplers**
  - Attaching a Sampler to a Beamline Element
  - Sampler Dimensions
  - Sampler Visualisation
  - Output at an Arbitrary Plane - User Placed Sampler

- **Physics Processes**
  - Modular Physics Lists
  - Geant4 Reference Physics Lists
  - Complete Physics Lists

- **Physics Biasing**

- **Options**

- **Beam Parameters**
• Advanced Topics
  – Tunnel Geometry
  – Materials and Atoms
  – Crystals
  – Regions
  – More details about Bends
  – Colours
  – Controlling Simulation Speed
  – Multiple Beam Lines

6.1 Lattice Description

A model of the accelerator is given to BDSIM via input text files in the GMAD Syntax. The overall program structure should follow:

1) Component definition (see Beamline Elements)
2) Sequence definition using defined components (see Lattice Sequence)
3) Which sequence to use (see use - Defining which Line to Use)
4) Where to record output (see Output at a Plane - Samplers)
5) Options, including which physics lists, number to simulate etc. (see Options)
6) A beam definition (see Beam Parameters)

These are described in the following sections. Aside from these standard parameters, more detail may be added to the model through:

• Magnet Geometry Parameters.
• Custom Fields.
• Adding Externally Provided Geometry.
• Offsets & Tilts - Component Misalignment.

6.2 GMAD Syntax

GMAD is a language specifically for BDSIM that is made to be human readable. The name comes from the design intention of MAD-X syntax and extensions for Geant4. While GMAD is very similar to MAD-X, not all MAD-X commands are supported.

• S.I. units are used except where explicitly specified
• Variables can be defined using name = value; syntax
• Arithmetic expressions can be defined
• Binary operators +, -, *, /, ^, are valid
• Unary operators +, -, are valid
• Boolean operators <, >, <=, >=, <> (not equal), ==, are valid
• Every expression must end with a semi-colon;
• No variable name can begin with a number
• !Comments start with an exclamation mark “!”
• A variable may inherit values (via copy) from another variable using `newvariable : existingvariable`;

### 6.2.1 Mathematical Functions

The following mathematical functions are provided:

- `sqrt`
- `cos`
- `sin`
- `tan`
- `exp`
- `log`
- `acos`
- `asin`
- `atan`
- `abs`

### 6.2.2 Other Commands

- `print;` Prints all elements
- `print, line;` Prints all elements that are in the beam line defined by `use`. See also `use - Defining which Line to Use`
- `print, option;` Prints the value of some options
- `print, variable;` Prints the value of a numerical variable, which could be your own defined variable
- `length = d1"l";` A way to access properties of elements, in this case, length of element d1.
- `stop;` or `return;` Exists parser
- `if () {} ;` ‘if’ construct
- `if () {} else {} ;` ‘if-else’ construct
- `include ../some/other/file.gmad;` Includes another file to be parsed. Note that the path provided must be relative, not absolute.

### 6.2.3 Examples

Examples:

```plaintext
x = 1;
y = 2.5-x;
z = sin(x) + log(y) - 8e5;
mat = "copper";
```

### 6.2.4 Common Pitfalls

The following is an example of a common mistake that’s not easy to spot:
With this syntax we expect to create a design beam of 3 TeV electrons but the central energy of 1 TeV for the bunch. 3 TeV is used to calculate the magnet strengths and it’s expected to fire a 1 TeV electron. However, the E0 parameter here just defines a variable called E0 that isn’t used. The indentation (white-space) is ignored. The error is the semi-colon at the end of the second line. This is the correct version:

```plaintext
beam, particle="e-",
    energy=3*TeV,
    E0=1*TeV;
```

6.3 Coordinates & Units

In Geant4, global Euclidean coordinates are used for tracking purposes. However, in describing a lattice with BDSIM, curvilinear coordinates are used, as is common with accelerators (X,Y,S).

GMAD uses SI units

<table>
<thead>
<tr>
<th>Name</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>[m] (metres)</td>
</tr>
<tr>
<td>time</td>
<td>[s] (seconds)</td>
</tr>
<tr>
<td>angle</td>
<td>[rad] (radians)</td>
</tr>
<tr>
<td>quadrupole coefficient</td>
<td>[m⁻²]</td>
</tr>
<tr>
<td>multipole coefficient</td>
<td>[m⁻ⁿ]</td>
</tr>
<tr>
<td>electric voltage</td>
<td>[V] (Volts)</td>
</tr>
<tr>
<td>electric field strength</td>
<td>[V/m]</td>
</tr>
<tr>
<td>particle energy</td>
<td>[GeV]</td>
</tr>
<tr>
<td>particle mass</td>
<td>[GeV/c²]</td>
</tr>
<tr>
<td>particle momentum</td>
<td>[GeV/c²]</td>
</tr>
<tr>
<td>beam current</td>
<td>[A] (Amperes)</td>
</tr>
<tr>
<td>particle charge</td>
<td>[e] (elementary charges)</td>
</tr>
<tr>
<td>emittance</td>
<td>[π m mrad]</td>
</tr>
<tr>
<td>density</td>
<td>[g/cm³]</td>
</tr>
<tr>
<td>temperature</td>
<td>[K] (Kelvin)</td>
</tr>
<tr>
<td>pressure</td>
<td>[atm] (atmosphere)</td>
</tr>
<tr>
<td>frequency</td>
<td>[Hz] (Hertz)</td>
</tr>
<tr>
<td>mass number</td>
<td>[g/mol]</td>
</tr>
</tbody>
</table>

Some useful predefined values / units are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>pi</td>
<td>3.14159265358979</td>
</tr>
<tr>
<td>degrees</td>
<td>π / 180</td>
</tr>
<tr>
<td>GeV</td>
<td>1</td>
</tr>
<tr>
<td>eV</td>
<td>(10^{-9})</td>
</tr>
<tr>
<td>keV</td>
<td>(10^{-6})</td>
</tr>
<tr>
<td>MeV</td>
<td>(10^{-3})</td>
</tr>
<tr>
<td>TeV</td>
<td>(10^{3})</td>
</tr>
<tr>
<td>V</td>
<td>1</td>
</tr>
<tr>
<td>kV</td>
<td>(10^{4})</td>
</tr>
<tr>
<td>MV</td>
<td>(10^{9})</td>
</tr>
<tr>
<td>Tesla</td>
<td>1</td>
</tr>
</tbody>
</table>

Continued on next page
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>1</td>
</tr>
<tr>
<td>rad</td>
<td>1</td>
</tr>
<tr>
<td>mrad</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>urad</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>clight</td>
<td>$2.99792458 \times 10^8$</td>
</tr>
<tr>
<td>km</td>
<td>$10^3$</td>
</tr>
<tr>
<td>m</td>
<td>1</td>
</tr>
<tr>
<td>cm</td>
<td>$10^{-2}$</td>
</tr>
<tr>
<td>mm</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>um</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>mum</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>nm</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>ang</td>
<td>$10^{-10}$</td>
</tr>
<tr>
<td>pm</td>
<td>$10^{-12}$</td>
</tr>
<tr>
<td>s</td>
<td>1</td>
</tr>
<tr>
<td>ms</td>
<td>$10^{-3}$</td>
</tr>
<tr>
<td>us</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>ns</td>
<td>$10^{-9}$</td>
</tr>
<tr>
<td>ps</td>
<td>$10^{-12}$</td>
</tr>
<tr>
<td>Hz</td>
<td>1</td>
</tr>
<tr>
<td>kHz</td>
<td>$10^3$</td>
</tr>
<tr>
<td>MHz</td>
<td>$10^6$</td>
</tr>
<tr>
<td>GHz</td>
<td>$10^9$</td>
</tr>
</tbody>
</table>

As an example, one can write either $100*eV$ or $0.1*keV$ to specify an energy value in GMAD. Both are equivalent.

6.4 Circular Machines

To simulate circular machines, BDSIM should be executed with the --circular executable option (see Executable Options). This installs special beam line elements called the teleporter and terminator at the end of the lattice that are described below.

Note: There must be a minimum $0.2 \mu m$ gap between the last element and the beginning of the sequence to accommodate these elements. This has a minimal impact on tracking.

Both the terminator and teleporter are invisible and very thin elements that are not normally shown in the visualiser. These can be visualised by executing BDSIM with the --vis_debug executable option.

The turn number is automatically stored in the energy loss output in the data when the circular option is used.

6.4.1 Terminator

In a Geant4 / BDSIM model, all particles are tracked down to zero energy or until they leave the world volume. In the case of a circular accelerator, the particles may circulate indefinitely as they lose no energy traversing the magnetic fields. To control this behaviour and limit the number of turns taken in the circular machine, the terminator is inserted. This is a very thin disk that has dynamic limits attached to it. It is normally transparent to all particles and composed of vacuum. After the desired number of turns of the primary particle has elapsed, it switches to being an infinite absorber. It achieves this by setting limits (G4UserLimits) with a maximum allowed energy of 0eV.

The user should set the option nturns (default 1) (see Common Options).
option, nturns=56;

6.4.2 Teleporter

Not all optical models close perfectly in Cartesian coordinates, i.e. the ends don’t perfectly align. Some small offsets may be tolerable, as most tracking codes use curvilinear coordinates. To account for this, the teleporter is a small disk volume inserted to make up the space and shift particles transversely as if the ends matched up perfectly. This is automatically calculated and constructed when using the `--circular` executable option.

Although the teleporter may not be required in a well-formed model that closes, the minimum gap of $0.2\mu m$ is required for the terminator.

6.4.3 One Turn Map

Geant4 mandates that there are no overlaps between solids, which in BDSIM means that a thin 1 nm gap is placed between each lattice element. Whilst these thin gaps have a negligible effect for a single pass or turn, over several turns it introduces a sizeable inaccuracy in the tracking (in the context of large circular models). To correct for this, BDSIM models can be supplemented with a one turn map which is applied at the end of each turn to right the primary back onto the correct trajectory. To ensure physical results the one turn map is only applied to primaries, if they did not interact on the previous turn, and if they are within 5% of the reference momentum. The one turn map is also not applied on the first turn where there the beam is offset in S, but applied on following turns, still accounting for the exceptions mentioned above.

The map must be of the format as written by MAD-X-PTC’s `PTC_NORMAL` command. A one turn map (in this case, 12th order) can be generated in MAD-X with the following

```plaintext
PTC_CREATE_UNIVERSE;
PTC_CREATE_LAYOUT, model=2, method=6, nst=10, exact=true, resplit, xbend;
PTC_NORMAL, maptable, icase=5, no=12;
write, table="map_table", file="my_oneturn_map_file";
PTC_END;
```

To use then use the one turn map with BDSIM

```plaintext
option, ptcOneTurnMapFileName="path/to/my_oneturn_map_file";
```

- This can only be used with circular machines.

6.5 Beamline Elements

BDSIM provides a variety of different elements each with their own function, geometry and potential fields. Any element in BDSIM is described with the following pattern

```plaintext
name: type, parameter=value, parameter="string";
```

**Note:** Note the colon `:` between name and type. The double (not single) inverted commas for a string parameter and that each functional line must end with a semi-colon. Spaces will be ignored.

The following elements may be defined

- *drift*
- *rbend*
- *sbend*
• quadrupole
• sextupole
• octupole
• decapole
• multipole
• thinmultipole
• vkicker
• hkicker
• kicker
• tkicker
• rf
• rcol
• jcol
• ecol
• degrader
• muspoiler
• shield
• dump
• solenoid
• laser
• gap
• crystalcol
• undulator
• transform3d
• element
• marker
• wirescanner

These are detailed in the following sections.

6.5.1 Simple example, extend and copy

Example:

d1: drift, l=5*m;

This defines a drift element with name d1 and a length of 5 metres. The definition can later be changed or extended with

d1: l=3*m, aper=0.1*m;

Note the omission of the type drift. This will change the length of d1 to 3 metres and set the aperture size to 10 centimetres.
An element can also be defined by copying an existing element

\[
d2: \ d1, \ l=2*m;
\]

Element \( d2 \) is a drift with the properties of \( d1 \) and a length of 2 metres. Note that if \( d1 \) is changed again, \( d2 \) will not change.

### 6.5.2 Magnet Strength Polarity

**Note:** BDSIM strictly follows the MAD-X definition of magnet strength parameter \( k \) - a positive \( k \) corresponds to horizontal focussing for a positively charged particle. This therefore indicates a positive \( k \) corresponds to horizontal defocussing for a negatively charged particle. However, MAD-X treats all particles as positively charged for tracking purposes.

**Warning:** BDSIM currently treats \( k \) absolutely, so to convert a MAD-X lattice for negatively particles, the MAD-X \( k \) values must be multiplied by -1. The pybdsim converter provides an option called `flipmagnets` for this purpose. This may be revised in future releases depending on changes to MAD-X.

### 6.5.3 Component Strength Scaling

In the case of acceleration or energy degradation, the central energy of the beam may change. However, BDSIM constructs all fields with respect to the rigidity calculated from the particle species and the `energy` parameter in the beam definition (not \( E0 \), but `energy`). To easily scale the strengths, every beam line element has the parameter `scaling` that enables its strength to be directly scaled.

In the case of a dipole, this scales the field but not the angle (the field may be calculated from the angle if none is specified). For example

\[
\text{beam, particle="e-", energy=10*GeV;}
\text{sbl1: sbend, l=2.5*m, angle=0.1;}
\text{d1: drift, l=1*m;}
\text{cav1: rf, l=1*m, gradient=50, frequency=0;}
\text{sb2: sbend, l=2.5*m, angle=0.1, scaling=1.005;}
\text{l1: line=(sbl1,d1,cav1,d1,sb2,d1);}
\]

In this example an rf cavity is used to accelerate the beam by 50 MeV (50 MeV / m for 1 m). The particle passes through one bend, the cavity and then another. As the second bend is scaled (by a factor of \((10 \text{ GeV} + 50 \text{ MeV}) / 10 \text{ GeV} = 1.005\) a particle starting at \((0,0)\) with perfect energy will appear at \((0,0)\) after this lattice.

In the case of a quadrupole or other magnet, the scaling is internally applied to the \( k/l \) or appropriate parameter that is used along with the design rigidity to calculate a field gradient.

An example is included in `examples/features/components/scaling.gmad`.

**Note:** The user should take care to use this linear scaling parameter wisely - particularly in sub-relativistic regimes. The fields should typically be scaled with momentum and not total energy of the particle.
### 6.5.4 drift

*drift* defines a straight beam pipe with no field.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>l</em></td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>vacuumMaterial</td>
<td>The vacuum material to use, can be user defined</td>
<td>vacuum</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:
- The *aperture parameters* may also be specified.

Examples:

```bash
1203b: drift, l=1*m;
1204c: drift, l=3*cm, beampipeRadius=10*cm;
```

### 6.5.5 rbend

*rbend* defines a rectangular bend magnet. Either the total bending angle, *angle*, or the magnetic field, *B*, (in Tesla) for the nominal beam energy can be specified. If both are defined the magnet is under or over-powered. The faces of the magnet are normal to the chord of the input and output points. Can be specified using:

1) *angle only* - *B* calculated from the angle and the beam design rigidity.
2) *B only* - the angle is calculated from the beam design rigidity.
3) *angle & B* - physically constructed using the angle, and field strength as *B*.

Pole face rotations can be applied to both the input and output faces of the magnet, based upon the reference system shown in the figure below. A pure dipole field is provided in the beam pipe and a more general dipole (as
described by *General Yoke Multipole*) is provided for the yoke. A quadrupolar component can be specified using the \( k_1 \) parameter that is given by:

\[
k_1 = \frac{1}{\rho} \frac{dB_y}{dx} \quad [m^{-2}]
\]

If \( k_1 \) is specified, the integrator from the `bdsimmatrix` integrator set is used. This results in no physical pole face angle being constructed for tracking purposes. The tracking still includes the pole face effects.

**Note:** See *Bends* for important notes about dipole tracking.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l )</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>angle</td>
<td>Angle [rad]</td>
<td>0</td>
<td>Yes, and or ( B )</td>
</tr>
<tr>
<td>( B )</td>
<td>Magnetic field [T]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( e1 )</td>
<td>Input pole face angle [rad]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( e2 )</td>
<td>Output pole face angle [rad]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>material</td>
<td>Magnet outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
<tr>
<td>yokeOnInside</td>
<td>Yoke on inside of bend</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( hStyle )</td>
<td>H style poled geometry</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( k1 )</td>
<td>Quadrupole coefficient for function magnet</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( fint )</td>
<td>Fringe field integral for the entrance face of the rbend</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( fintx )</td>
<td>Fringe field integral for the exit face of the rbend. -1 means default to the same as fint. 0 there will be no effect.</td>
<td>-1</td>
<td>No</td>
</tr>
<tr>
<td>( fintK2 )</td>
<td>Second fringe field integral for the entrance face of the rbend</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( fintxK2 )</td>
<td>Second fringe field integral for the exit face of the rbend</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( hgap )</td>
<td>The half gap of the poles for <em>fringe field purposes only</em></td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( h1 )</td>
<td>input poleface curvature [m⁻¹]</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>( h2 )</td>
<td>output poleface curvature [m⁻¹]</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

Notes:

- The *aperture parameters* may also be specified.
- The *magnet geometry parameters* may also be specified.

A few points about rbends:

1) For large angles (> 100 mrad), particles may hit the aperture, as the beam pipe is represented by a straight (chord) section and even nominal energy particles may hit the aperture, depending on the degree of tracking accuracy specified. In this case, consider splitting the `rbend` into multiple ones.

2) The pole face rotation angle is limited to \( \pm \pi/4 \) radians.

3) If a non-zero pole face rotation angle is specified, the element preceding / succeeding the rotated magnet face must either be a drift or an rbend with opposite rotation (e.g. an rbend with \( e2 = 0.1 \) can be followed by an rbend with \( e1 = -0.1 \)). The preceding / succeeding element must be longer than the projected length from the rotation, given by \( 2 \tan(eX) \).

4) Fringe field kicks are applied in a thin fringe field magnet (1 micron thick by default) at the beginning or at the end of the rbend. The length of the fringe field element can be set by the option `thinElementLength` (see `options`).

5) In the case of finite \( fint \) or \( fintx \) and \( hgap \), a fringe field is used even if \( e1 \) and \( e2 \) have no angle.

6) The \( fintK2 \) and \( fintxK2 \) parameters are for a second fringe field correction term that are included to enable optics comparisons with TRANSPORT. Whilst this term is not available in MAD-X, the default values of 0
mean this second fringe field correction will not be applied unless \( f_{\text{int}K2} \) or \( f_{\text{int}xK2} \) are explicitly specified as non-zero.

7) The effect of pole face rotations and fringe field kicks can be turned off for all dipoles by setting the option includeFringeFields=0 (see options).

8) The poleface curvature does not construct the curved geometry. The effect is instead applied in the thin fringefield magnet.

9) Rbends are limited in angle to less than \( \pi/2 \).

Examples:

```
MRB20: rbend, l=3*m, angle=0.003;
r1: rbend, l=5.43m, beampipeRadius=10*cm, B=2*Tesla;
RB04: rbend, l=1.8*m, angle=0.05, e1=0.1, e2=-0.1
```

### 6.5.6 sbend

\( \text{sbend} \) defines a sector bend magnet. Either the total bending angle, \( \text{angle} \), or the magnetic field, \( B \), (in Tesla) for the nominal beam energy can be specified. If both are defined the magnet is under or over-powered. The faces of the magnet are normal to the curvilinear coordinate system. \( \text{sbend} \) magnets are made of a series of straight segments. Can be specified using:

1) \( \text{angle} \) only - \( B \) calculated from the angle and the beam design rigidity.
2) $B$ only - the angle is calculated from the beam design rigidity.

3) angle & $B$ - physically constructed using the angle, and field strength as $B$.

Pole face rotations can be applied to both the input and output faces of the magnet, based upon the reference system shown in the figure below. A pure dipole field is provided in the beam pipe and a more general dipole (as described by General Yoke Multipole) is provided for the yoke. A quadrupolar component can be specified using the $k1$ parameter that is given by:

$$k_1 = \frac{1}{\rho} \frac{dB}{dx} \left[ m^{-2} \right]$$

If $k1$ is specified, the integrator from $bdsimmatrix$ integrator set is used. This results in no physical pole face angle being constructed for tracking purposes. The tracking still includes the pole face effects.

The $sbend$ geometry is constructed as many small straight sections with angled faces. This makes no effect on tracking, but allows a much higher variety of apertures and magnet geometry to be used given the Geant4 geometry. The number of segments is computed such that the maximum tangential error in the aperture is 1 mm.

**Note:** See *Bends* for important notes about dipole tracking.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l$</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>angle</td>
<td>Angle [rad]</td>
<td>0</td>
<td>Yes, and $B$ or $B$</td>
</tr>
<tr>
<td>$B$</td>
<td>Magnetic field [T]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>$e1$</td>
<td>Input poleface angle [rad]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>$e2$</td>
<td>Output poleface angle [rad]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>material</td>
<td>Magnet outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
<tr>
<td>yokeOnInside</td>
<td>Yoke on inside of bend</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>$k1$</td>
<td>Quadrupole coefficient for function magnet</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>fint</td>
<td>Fringe field integral for the entrance face of the rbend</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>fintx</td>
<td>Fringe field integral for the exit face of the rbend. -1 means default to the same as fint. 0 there will be no effect.</td>
<td>-1</td>
<td>No</td>
</tr>
<tr>
<td>fintK2</td>
<td>Second fringe field integral for the entrance face of the rbend</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>fintxK2</td>
<td>Second fringe field integral for the exit face of the rbend</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>hgap</td>
<td>The half gap of the poles for fringe field purposes only</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>$h1$</td>
<td>input poleface curvature [m$^{-1}$]</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>$h2$</td>
<td>output poleface curvature [m$^{-1}$]</td>
<td>0</td>
<td>no</td>
</tr>
</tbody>
</table>

Notes:

- The *aperture parameters* may also be specified.
- The *magnet geometry parameters* may also be specified.

A few points about sbends:

1) The pole face rotation angle is limited to $\pm \pi/4$ radians.

2) If a non-zero pole face rotation angle is specified, the element preceding / succeeding the rotated magnet face must either be a drift or an sbend with the opposite rotation (e.g. an sbend with $e2 = 0.1$ can be followed by an sbend with $e1 = -0.1$). The preceding / succeeding element must be longer than the projected length from the rotation, given by $2 \tan(eX)$.

3) Fringe field kicks are applied in a thin fringe field magnet (1 micron thick by default) at the beginning or at the end of the sbend. The length of the fringe field magnet can be set by the option *thinElementLength* (see *options*).
Fig. 2: Pole face notation for an sbend.

4) In the case of finite \( f_{int} \) or \( f_{intx} \) and \( h_{gap} \) a fringe field is used even if \( e1 \) and \( e2 \) have no angle.

5) The \( f_{intK2} \) and \( f_{intxK2} \) parameters are for a second fringe field correction term that are included to enable optics comparisons with TRANSPORT. Whilst this term is not available in MAD-X, the default values of 0 mean this second fringe field correction will not be applied unless \( f_{intK2} \) or \( f_{intxK2} \) are explicitly specified as non-zero.

6) The effect of pole face rotations and fringe field kicks can be turned off for all dipoles by setting the option \texttt{includeFringeFields=0} (see \texttt{options}).

7) The poleface curvature does not construct the curved geometry. The effect is instead applied in the thin fringefield magnet.

8) Sbends are limited in angle to less than 2 pi. If the sbends are not split with the option \texttt{dontSplitSBends}, an sbend will be limited in angle to a maximum of pi/2.

Examples:

\begin{verbatim}
sbend, l=14.5*m, angle=0.005, magnetGeometryType="lhccright";
sbend, l=304.2*cm, b=1.5*Tesla;
sbend, l=0.61*m, angle=0.016, e1=-0.05, e2=0.09
\end{verbatim}

6.5.7 quadrupole

\textit{quadrupole} defines a quadrupole magnet. The strength parameter \( k_1 \) is defined as

\[ k_1 = \frac{1}{B \rho} \frac{dB_y}{dx} \left[ \text{m}^{-2} \right] \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l )</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( k1 )</td>
<td>Quadrupole coefficient</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>\textit{material}</td>
<td>Magnet outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:
• The *aperture parameters* may also be specified.

• The *magnet geometry parameters* may also be specified.

• See *Magnet Strength Polarity* for polarity notes.

A pure quadrupolar field is provided in the beam pipe and a more general multipole (as described by *General Yoke Multipole*) is provided for the yoke.

Examples:

```
q1: quadrupole, l=0.3*m, k1=45.23;
qm15ff: quadrupole, l=20*cm, k1=95.2;
```

### 6.5.8 sextupole

`sextupole` defines a sextupole magnet. The strength parameter $k_2$ is defined as

$$k_2 = \frac{1}{B_\rho} \frac{dB^2_y}{dx^2} [m^{-3}]$$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l$</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>$k2$</td>
<td>Sextupole coefficient</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td><code>material</code></td>
<td>Magnet outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:

• The *aperture parameters* may also be specified.

• The *magnet geometry parameters* may also be specified.

• See *Magnet Strength Polarity* for polarity notes.
A pure sextupolar field is provided in the beam pipe and a more general multipole (as described by *General Yoke Multipole*) is provided for the yoke.

Examples:

```
sx1: sextupole, l=0.5*m, k2=4.678;
sx2: sextupole, l=20*cm, k2=45.32, magnetGeometry="normalconducting";
```

### 6.5.9 octupole

octupole defines an octupole magnet. The strength parameter \( k_3 \) is defined as

\[
k_3 = \frac{1}{B_\rho} \frac{dB_y}{dx^3} [m^{-4}]
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l )</td>
<td>Length ([m])</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( k3 )</td>
<td>Octupole coefficient</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( material )</td>
<td>Magnet outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:

- The *aperture parameters* may also be specified.
- The *magnet geometry parameters* may also be specified.
- *See Magnet Strength Polarity* for polarity notes.

A pure octupolar field is provided in the beam pipe and a more general multipole (as described by *General Yoke Multipole*) is provided for the yoke.

Examples:

```
oct4b: octupole, l=0.3*m, k3=32.9;
```

### 6.5.10 decapole

decapole defines a decapole magnet. The strength parameter \( k_4 \) is defined as

\[
k_4 = \frac{1}{B_\rho} \frac{dB_y}{dx^4} [m^{-5}]
\]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l )</td>
<td>Length ([m])</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( k4 )</td>
<td>Decapole coefficient</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( material )</td>
<td>Magnet outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
</tbody>
</table>
A pure decapolar field is provided in the beam pipe and a more general multipole (as described by General Yoke Multipole) is provided for the yoke.

- The aperture parameters may also be specified.
- The magnet geometry parameters may also be specified.
- See Magnet Strength Polarity for polarity notes.

Examples:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>knl</td>
<td>List of normal coefficients</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>ksl</td>
<td>List of skew coefficients</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>material</td>
<td>Magnet outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:

- The aperture parameters may also be specified.
- The magnet geometry parameters may also be specified.
- See Magnet Strength Polarity for polarity notes.
- No yoke field is provided.

Examples:

OCTUPOLE1 : multipole, l=0.5*m , knl={ 0,0,1 }, ksl={ 0,0,0 };
6.5.13 vkicker

`vkicker` can either be a thin or thick vertical dipole magnet. If specified with a finite length \( l \), it will be constructed as a thick dipole. However, if no length (or a length of exactly 0 is specified), a thin kicker will be built. In practice, the thin version is constructed as a 1um slice with only the aperture geometry and no surrounding geometry and is not visible with the default visualisation settings.

The strength is specified by the parameter `vkick`, which is the fractional momentum kick in the vertical direction. A positive value corresponds to an increase in \( p_y \). In the case of the thin kicker the position is not affected, whereas with the thick kicker, the position will change.

The strength may also be specified by the magnetic field \( B \). A positive field value corresponds to an increase in \( p_y \) for a positively charged particle.

**Warning:** `vkick` will supercede the strength even if \( B \) is specified. Therefore, the user should specify only `vkick` or \( B \).

In the case of a thick kicker, the resulting bending angle is calculated using:

\[
\theta = \sin^{-1}(p_x)
\]

The dipole field strength is then calculated with respect to the chord length:

\[
B = B\rho \frac{\theta}{\text{chord length}}
\]

- The *aperture parameters* may also be specified.
- For a `vkicker` with a finite length, the *magnet geometry parameters* may also be specified.

**Note:** Pole face rotations and fringe fields can be applied to `vkickers` by supplying the same parameters that would be applied to an `rbend` or `sbend`. If the `vkicker` is zero length, the \( B \) field value must be supplied in order to calculate the bending radius which required to apply the effects correctly.

- Fringe field kicks are applied in a thin fringe field magnet (1 micron thick by default) at the beginning or at the end of the `vkicker`. The length of the fringe field element can be set by the option `thinElementLength` (see *options*).
- For zero length `vkickers`, the pole face and fringe field kicks are applied in the same thin element as the `vkick`.
- In the case of finite `fint` or `fintx` and `hgap`, a fringe field is used even if `e1` and `e2` have no angle.
- The effect of pole face rotations and fringe field kicks can be turned off for all magnets by setting the option `includeFringeFields=0` (see *options*).
- No pole face geometry is constructed.

A pure dipole field is provided in the beam pipe and a more general multipole (as described by *General Yoke Multipole*) is provided for the yoke.

Examples:
6.5.14 hkicker

hkicker can either be a thin horizontal kicker or a thick horizontal dipole magnet. If specified with a finite length \( l \), it will be constructed as a dipole. However, if no length (or a length of exactly 0) is specified, a thin kicker will be built. This is typically a 1um slice with only the shape of the aperture and no surrounding geometry. It is also typically not visible with the default visualisation settings.

The strength is specified by the parameter \( hkick \), which is the fractional momentum kick in the vertical direction. A positive value corresponds to an increase in \( p_x \). In the case of the thin kicker the position is not affected, whereas with the thick kicker, the position will change.

The strength may also be specified by the magnetic field \( B \). A positive field value corresponds to an decrease in \( p_x \) (note right-handed coordinate frame) for a positively charged particle.

\textbf{Warning:} \( hkick \) will supercede the strength even if \( B \) is specified. Therefore, the user should specify only \( hkick \) or \( B \).

\textbf{Note:} A positive value of \( hkick \) causes an increase in horizontal momentum, so the particle will bend to the left looking along the beam line, i.e. in positive \( x \). This is the opposite of a bend where a positive angle causes a deflection in negative \( x \).

- The \textit{aperture parameters} may also be specified.
- For a hkicker with a finite length, the \textit{magnet geometry parameters} may also be specified.

\textbf{Note:} Pole face rotations and fringe fields can be applied to hkickers by supplying the same parameters that would be applied to an \textit{rbend} or \textit{sbend} . If the hkicker is zero length, the \( B \) field value must be supplied in order to calculate the bending radius which required to apply the effects correctly.

- Fringe field kicks are applied in a thin fringe field magnet (1 micron thick by default) at the beginning or at the end of the hkicker. The length of the fringe field element can be set by the option \textit{thinElementLength} (see \textit{options}).
- For zero length hkickers, the pole face and fringe field kicks are applied in the same thin element as the hkick.
- In the case of finite fint or fintx and hgap, a fringe field is used even if e1 and e2 have no angle.
- The effect of pole face rotations and fringe field kicks can be turned off for all magnets by setting the option \textit{includeFringeFields}=0 (see \textit{options}).
- No pole face geometry is constructed.

A pure dipole field is provided in the beam pipe and a more general multipole (as described by \textit{General Yoke Multipole}) is provided for the yoke.

Examples:

\begin{verbatim}
KXX15h: hkicker, hkick=0.01;
KXX15h: hkicker, hkick=-1.3e-5, l=0.2*m;
KXX21h: hkicker, B=0.03*T;
\end{verbatim}
6.5.15 kicker

`kicker` defines a combined horizontal and vertical kicker. Either both or one of the parameters `hkick` and `vkick` may be specified. Like the `hkicker` and `vkicker`, this may also be thin or thick. In the case of the thick kicker, the field is the linear sum of two independently calculated fields.

**Note:** Pole face rotation and fringe fields kicks are unavailable for plain kickers

Example:

```plaintext
kick1: kicker, l=0.45*m, hkick=1.23e-4, vkick=0.3e-4;
```

6.5.16 tkicker

BDSIM, like MAD-X, provides a `tkicker` element. This is an alias in BDSIM for a `kicker`, however MAD-X differentiates the two on the basis of fitting parameters. BDSIM does not make this distinction. See `kicker` for more details.

In the case of a `tkicker`, the field $B$ cannot be used and only `hkick` and `vkick` can be used.

**Note:** Pole face rotation and fringe fields kicks are unavailable for tkickers

6.5.17 rf

`rf` or `rfcavity` defines an RF cavity with a time varying electric or electromagnetic field. There are several geometry and field options as well as ways to specify the strength. The default field is a uniform (in space) electric-only field that is time varying according to a cosine (see Sinusoidal Electric Field). Optionally, the electromagnetic field for a pill-box cavity may be used (see Pill-Box Cavity). The `G4 Classical RK4` numerical integrator is used to calculate the motion of particles in both cases.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l$</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>$E$</td>
<td>Electric field strength</td>
<td>0</td>
<td>Yes (or <code>gradient</code>)</td>
</tr>
<tr>
<td><code>gradient</code></td>
<td>Field gradient [MV/m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td><code>frequency</code></td>
<td>Frequency of oscillation (Hz)</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td><code>phase</code></td>
<td>Phase offset (rad)</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td><code>tOffset</code></td>
<td>Offset in time (s)</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td><code>material</code></td>
<td>Outer material</td>
<td>&quot;&quot;</td>
<td>Yes</td>
</tr>
<tr>
<td><code>cavityModel</code></td>
<td>Name of cavity model object</td>
<td>&quot;&quot;</td>
<td>No</td>
</tr>
</tbody>
</table>

**Note:** The design energy of the machine is not affected, so the strength and fields of components after an RF cavity in a lattice are calculated with respect to the design energy, the particle and therefore, design rigidity. The user should scale the strength values appropriately if they wish to match the increased energy of the particle.
Warning: The elliptical cavity geometry may not render or appear in the Geant4 QT visualiser. The geometry exists and is valid, but this is due to deficiencies of the Geant4 visualisation system. The geometry exists and is fully functional.

- The field is such that a positive E-field results in acceleration of the primary particle.
- The phase is calculated automatically such that zero phase results in the peak E-field at the centre of the component for its position in the lattice.
- Either \( t_{\text{Offset}} \) or \( \text{phase} \) may be used to specify the phase of the oscillator.
- The material must be specified in the \( \text{rf gmad} \) element or in the attached cavity model by name. The cavity model will override the element material.

If \( t_{\text{Offset}} \) is specified, a phase offset is calculated from this time for the speed of light in a vacuum. Otherwise, the curvilinear S-coordinate of the centre of the rf element is used to find the phase offset.

Note: As the phase offset is calculated from the speed of light in a vacuum, this is only correct for already relativistic beams. Development is underway to improve this calculation for sub-relativistic beams.

If \( \text{phase} \) is specified, this is added to the calculated phase offset from either the lattice position or \( t_{\text{Offset}} \).

Simple examples:

```
rf1: rf, l=10*cm, E=10*MV, frequency=90*MHz, phase=0.02;
rf2: rf, l=10*cm, gradient=14*MV / m, frequency=450*MHz;
rf3: rf, l=10*cm, E=10*MV, frequency=90*MHz, tOffset=3.2*ns;
```

Rather than just a simple E-field, an electromagnetic field that is the solution to a cylindrical pill-box cavity may be used. A cavity object (described in more detail below) is used to specify the field type. All other cavity parameters may be safely ignored and only the field type will be used. The field is described in Pill-Box Cavity.

Pill-box field example:

```
rffield: field, type="rfcavity";
rf4: rf, l=10*cm, E=2*kV, frequency=1.2*GHz, fieldVacuum="rffield";
```

Elliptical SRF cavity geometry is also provided and may be specified by use of another ‘cavity’ object in the parser. This cavity object can then be attached to an rf object by name. Details can be found in Cavity Geometry Parameters.

### 6.5.18 rcol

An rcol defines a rectangular collimator. The aperture is rectangular and the external volume is square.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l )</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( xsize )</td>
<td>Horizontal half aperture [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( ysize )</td>
<td>Half height of jaws [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>material</td>
<td>Outer material</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>( horizontalWidth )</td>
<td>Outer full width [m]</td>
<td>0.5 m</td>
<td>No</td>
</tr>
<tr>
<td>( xsizeLeft )</td>
<td>Left jaw aperture [m]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( xsizeRight )</td>
<td>Right jaw aperture [m]</td>
<td>0</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:

- \( horizontalWidth \) should be big enough to encompass the \( xsize \) and \( ysize \).
- If no \( xsize \) or \( ysize \) are provided, they are assumed to be 0 and a solid block is made.
- The parameter *minimumKineticEnergy* (GeV by default) may be specified to artificially kill particles below this kinetic energy in the collimator. This is useful to match other simulations where collimators can be assumed to be infinite absorbers. If this behaviour is required, the user should specify an energy greater than the total beam energy.
- The collimator can be tapered by specifying an exit aperture size with \( xsizeOut \) and \( ysizeOut \), with the \( xsize \) and \( ysize \) parameters defining the entrance aperture.
- All collimators can be made infinite absorbers with the general option *collimatorsAreInfiniteAbsorbers* (see Tracking Options).

Examples:

```plaintext
! Standard
TCP15: rcol, l=1.22*m, material="graphite", xsize=104*um, ysize=5*cm;

! Tapered
TCP16: rcol, l=1.22*m, material="graphite", xsize=104*um, ysize=5*cm,
\(--\)xsizeOut=208*um, ysizeOut=10*cm;

! with kinetic energy limit
TCP6CD: rcol, l=0.6*m, material="C", xsize=200*um, ysize=5*cm,
\(--\)minimumKineticEnergy=10*MeV;
```

6.5.19 *ecol*

*ecol* defines an elliptical collimator. This is exactly the same as *rcol* except that the aperture is elliptical and the \( xsize \) and \( ysize \) define the horizontal and vertical half-axes respectively. When tapered, the ratio between the horizontal and vertical half-axes of the entrance aperture must be the same ratio for the exit aperture.

- All the same conditions for *rcol* apply for *ecol*.
6.5.20 jcol

jcol defines a jaw collimator with two square blocks on either side in the horizontal plane. If a vertical jcol is required, the tilt parameter should be used to rotate it by $\pi/2$. The horizontal position of each jaw can be set separately with the xsizeLeft and xsizeRight apertures which are the distances from the centre of element to the left and right jaws respectively.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>xsize</td>
<td>Horizontal half aperture [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>ysize</td>
<td>Half height of jaws [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>material</td>
<td>Outer material</td>
<td>None</td>
<td>Yes</td>
</tr>
<tr>
<td>xsizeLeft</td>
<td>Left jaw aperture [m]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>xsizeRight</td>
<td>Right jaw aperture [m]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>horizontalWidth</td>
<td>Outer full width [m]</td>
<td>0.5 m</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:

- The horizontalWidth must be greater than 2x xsize.
- To prevent the jaws overlapping with one another, a jaw cannot be constructed that crosses the X axis of the element (i.e. supplying a negative xsizeLeft or xsizeRight will not work). Should you require this, please offset the element using the element parameters offsetX and offsetY instead.
- To construct a collimator jaws with one jaw closed (i.e. an offset of 0), the horizontal half aperture must be set to 0, with the other jaws half aperture set as appropriate.
- If xsize, xsizeLeft and xsizeRight are not specified, the collimator will be constructed as a box with no aperture.
- Specifying a jaw aperture which is larger than half the horizontalWidth value will result in that jaw not being constructed. If both jaw apertures are greater than half the horizontalWidth, no jaws will be built and BDSIM will exit.
- The parameter minimumKineticEnergy (GeV by default) may be specified to artificially kill particles below this kinetic energy in the collimator. This is useful to match other simulations where collimators can be assumed to be infinite absorbers. If this behaviour is required, the user should specify an energy greater than the total beam energy.
- All collimators can be made infinite absorbers with the general option collimatorsAreInfiniteAbsorbers (see Tracking Options).

Examples:
6.5.21 degrader

degraded defines interleaved pyramidal pieces of material. Depending on the physics list used, this is capable of reducing the beam energy. This happens only through interaction and the use of a physics list. Note, the default physics list in BDSIM is no physics and only magnetic tracking, in which case this component will have no effect.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>numberWedges</td>
<td>Number of degrader wedges</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>wedgeLength</td>
<td>Degrader wedge length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>degraderHeight</td>
<td>Degrader height [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>materialThickness</td>
<td>Amount of material seen by the beam [m]</td>
<td>0</td>
<td>Yes/No*</td>
</tr>
<tr>
<td>degraderOffset</td>
<td>Horizontal offset of both wedge sets</td>
<td>0</td>
<td>Yes/No*</td>
</tr>
<tr>
<td>material</td>
<td>Degrader material</td>
<td>Carbon</td>
<td>Yes</td>
</tr>
<tr>
<td>horizontalWidth</td>
<td>Outer full width [m]</td>
<td>global</td>
<td>No</td>
</tr>
</tbody>
</table>

Note: Either `materialThickness` or `degraderOffset` can be specified to adjust the horizontal lateral wedge position, and consequently the total material thickness the beam can propagate through. If both are specified, `degraderOffset` will be ignored. When `numberWedges` is specified to be n, the degrader will consist of n-1 `full` wedges and two `half` wedges. When viewed from above, a `full` wedge appears as an isosceles triangle, and a `half` wedge appears as a right-angled triangle.

Examples:

DEG1: degrader, l=0.25*m, material="carbon", numberWedges=5, wedgeLength=100*mm, degraderHeight=100*mm, materialThickness=200*mm;

DEG2: degrader, l=0.25*m, material="carbon", numberWedges=5, wedgeLength=100*mm, degraderHeight=100*mm, degraderOffset=50*mm;
6.5.22 muspoiler

`muspoiler` defines a muon spoiler, which is a rotationally magnetised iron cylinder with a beam pipe in the middle. There is no magnetic field in the beam pipe.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>l</code></td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td><code>B</code></td>
<td>Magnetic field [T]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td><code>material</code></td>
<td>Outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
<tr>
<td><code>horizontalWidth</code></td>
<td>Outer full width [m]</td>
<td>global</td>
<td>No</td>
</tr>
</tbody>
</table>

6.5.23 shield

`shield` defines a square block of material with a square aperture. The user may choose the outer width and inner horizontal and vertical apertures of the block. A beam pipe is also placed inside the aperture. If the beam pipe dimensions (including thickness) are greater than the aperture, the beam pipe will not be created.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>l</code></td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td><code>material</code></td>
<td>Outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
<tr>
<td><code>horizontalWidth</code></td>
<td>Outer full width [m]</td>
<td>global</td>
<td>No</td>
</tr>
<tr>
<td><code>xsize</code></td>
<td>Horizontal inner half aperture [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td><code>ysize</code></td>
<td>Vertical inner half aperture [m]</td>
<td>0</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:
• The *aperture parameters* may also be specified.

### 6.5.24 dump

*dump* defines a square or circular block of material that is an infinite absorber. All particles impacting the dump will be absorbed irrespective of the particle and physics processes.

This is intended as a useful way to absorb a beam with no computational time. Normally, the world volume is filled with air. If the beam reaches the end of the beam line it will hit the air and likely create a shower of particles that will take some time to simulate. In this case, when this isn’t required, it is recommended to use a dump object to absorb the beam.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>l</em></td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>horizontalWidth</td>
<td>Outer full width [m]</td>
<td>global</td>
<td>No</td>
</tr>
<tr>
<td>apertureType</td>
<td>Which shape</td>
<td>rectangular</td>
<td>No</td>
</tr>
</tbody>
</table>

How this works: the material of the dump is actually vacuum, but G4UserLimits are used to kill particles. This requires the cuts and limits physics process that is included automatically. In the case of using a Geant4 reference physics list (see *Geant4 Reference Physics Lists*), the necessary process is added automatically to enforce this.

The dump may accept *apertureType* with the value of either *circular* or *rectangular* for the shape of the dump. By default it is rectangular.

Examples:

```plaintext
d1: dump, l=0.2*m, horizontalWidth=20*cm;
d2: dump, l=0.4*m, apertureType="circular";
d3: dump, l=0.3*m, apertureType="rectangular";
```

### 6.5.25 solenoid

*solenoid* defines a solenoid magnet. This utilises a thick lens transfer map with a hard edge field profile. Fringes for the edge effects are provided by default and are controllable with the option *includeFringeFields*.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>ks</td>
<td>Solenoid strength</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>B</td>
<td>Magnetic field</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>material</td>
<td>Outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
<tr>
<td>horizontalWidth</td>
<td>Outer full width [m]</td>
<td>global</td>
<td>No</td>
</tr>
</tbody>
</table>

- A positive field corresponds to a field in along the direction of positive S.
- The entrance / exit solenoid fringes are not constructed if the previous / next element is also a solenoid.
- See *Magnet Strength Polarity* for polarity notes.
- No yoke field is provided.

Examples:

```
atlassol: solenoid, l=20*m, ks=0.004;
```

### 6.5.26 wirescanner

![wirescanner diagram]

*wirescanner* defines a cylindrical object within a beam pipe to represent a wire scanner typically use in an accelerator.

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
<th>default</th>
<th>required</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>length of drift section around wire</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>wireDiameter</td>
<td>diameter of wire [m]</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>wireLength</td>
<td>length of wirescanner [m]</td>
<td>0</td>
<td>yes</td>
</tr>
<tr>
<td>angle</td>
<td>angle of the wire w.r.t. vertical</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>wireOffsetX</td>
<td>x offset of the wire from the center [m]</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>wireOffsetY</td>
<td>y offset of the wire from the center [m]</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>wireOffsetZ</td>
<td>z offset of the wire from the center [m]</td>
<td>0</td>
<td>no</td>
</tr>
<tr>
<td>wireMaterial</td>
<td>material of wire</td>
<td>carbon</td>
<td>no</td>
</tr>
</tbody>
</table>

Notes:

- The angle is the rotation from vertical in the clock-wise direction looping in the positive S direction (the usually direction of the beam).

The offsets are with respect to the centre of the beam pipe section the wire is placed inside. This should therefore be less than half the element length l. The usual beam pipe parameters can be specified and apply the to the beam.
pipe. For example, material is used for the beam pipe material whereas wireMaterial is used for the material of the wire.

The user should take care to define a wire long enough to intercept the beam but be sufficiently short to fit inside the beam pipe given the offsets in x, y and z. Checks are made on the end points of the wire.

Examples:

```
ws45Deg: wirescanner, l=4*cm, wireDiameter=0.1*mm, wireLength=5*cm,
        wireOffsetX=1*cm, angle=pi/4, wireMaterial="C",
        aper1=5*cm;
wsVertical: wirescanner, l=4*cm, wireDiameter=0.1*mm, wireLength=5*cm,
        wireOffsetX=1*cm, wireOffsetZ=1.6*cm, wireMaterial="C";
```

### 6.5.27 laser

laser defines a drift section with a laser beam inside. The laser acts as a static target of photons.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Length of drift section [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>x, y, z</td>
<td>Components of laser direction vector (normalised)</td>
<td>(1,0,0)</td>
<td>yes</td>
</tr>
<tr>
<td>waveLength</td>
<td>Laser wavelength [m]</td>
<td>532*nm</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Examples:

```
laserwire: laser, l=1*um, x=1, y=0, z=0, wavelength=532*nm;
```

### 6.5.28 gap

gap defines a gap where no physical geometry is placed. It will be a region of the world, composed of the same material as the world volume.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>l</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>angle</td>
<td>Angle [rad]</td>
<td>0</td>
<td>No</td>
</tr>
</tbody>
</table>

Examples:

```
GAP1: gap, l=0.25*m, angle=0.01*rad;
```

### 6.5.29 crystalcol

crystalcol defines a crystal collimator that uses crystals to channel particles. It is composed of a beam pipe with one or two crystals inside it. The crystals can be the same (but placed at different angles) or different. The crystals are placed + xsize away from the centre.

The crystal is defined in a separate object in the parser and referred to by the name of that object. At least one of crystalBoth, crystalLeft and crystalRight must be specified.

**Warning:** This requires the user to use the “completechannelling” or “channelling” physics list for channelling to take place.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l )</td>
<td>Length ([\text{m}])</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( xsize )</td>
<td>Half-gap distance of each crystal from centre ([\text{m}])</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( \text{material} )</td>
<td>Material</td>
<td>&quot;&quot;</td>
<td>Yes</td>
</tr>
<tr>
<td>( \text{crystalBoth} )</td>
<td>Name of crystal object for both crystals</td>
<td>&quot;&quot;</td>
<td>No</td>
</tr>
<tr>
<td>( \text{crystalLeft} )</td>
<td>Name of crystal object for left crystal</td>
<td>&quot;&quot;</td>
<td>No</td>
</tr>
<tr>
<td>( \text{crystalRight} )</td>
<td>Name of crystal object for right crystal</td>
<td>&quot;&quot;</td>
<td>No</td>
</tr>
<tr>
<td>( \text{crystalAngleYAxisLeft} )</td>
<td>Rotation angle of left crystal ([\text{rad}])</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( \text{crystalAngleYAxisRight} )</td>
<td>Rotation angle of right crystal ([\text{rad}])</td>
<td>0</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:

- Crystal channelling potential files are required for this - see *Crystals* for more details.
- If only \( \text{crystalLeft} \) or \( \text{crystalRight} \) is specified, only one crystal will be placed.
- If both \( \text{crystalLeft} \) and \( \text{crystalRight} \) are specified, both will be constructed uniquely and placed.
- If \( \text{crystalBoth} \) is specified, \( \text{crystalLeft} \) and \( \text{crystalRight} \) will be ignored and the \( \text{crystalBoth} \) definition will be used for both crystals. The angles will be unique.

**Note:** Crystal channelling is only available in Geant4.10.4 onwards. If BDSIM is compiled with a Geant4 version below this, the geometry will be constructed correctly but the channelling physics process will not be used and the crystal will not channel particles.

- See *Crystals* for the definition of a crystal object.

Examples:

```python
lovelycrystal: crystal, material = "G4_Si",
    data = "data/Si220pl",
    shape = "box",
    lengthY = 5*cm,
    lengthX = 0.5*mm,
    lengthZ = 4*mm,
    sizeA = 5.43*ang,
    sizeB = 5.43*ang,
    sizeC = 5.43*ang,
    alpha = 1,
```

(continues on next page)
6.5.30 undulator

**undulator** defines an undulator magnet which has a sinusoidally varying field along the element with field components:

\[
B_x = 0 \\
B_y = B \cdot \cos \left( \frac{2\pi}{\lambda} \right) \cosh \left( \frac{2\pi}{\lambda} y \right) \\
B_z = -B \cdot \sin \left( \frac{2\pi}{\lambda} \right) \sinh \left( \frac{2\pi}{\lambda} y \right)
\]

where \( \lambda \) is the undulator period.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>( l )</td>
<td>Length [m]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( B )</td>
<td>Magnetic field [T]</td>
<td>0</td>
<td>Yes</td>
</tr>
<tr>
<td>( \text{undulatorPeriod} )</td>
<td>Undulator magnetic period [m]</td>
<td>1</td>
<td>Yes</td>
</tr>
<tr>
<td>( \text{undulatorGap} )</td>
<td>Undulator gap [m]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( \text{undulatorMagnetHeight} )</td>
<td>Undulator magnet height [m]</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>( \text{material} )</td>
<td>Magnet outer material</td>
<td>Iron</td>
<td>No</td>
</tr>
</tbody>
</table>

Notes:

- The undulator period must be an integer factor of the undulator length. If not, BDSIM will exit.
- The undulator gap is the total distance between the upper and lower sets of magnets. If not supplied, it is set to twice the beam pipe diameter.
- The undulator magnet height is the vertical height of the sets of magnets. If not supplied, it is set to the 0.5*horizontalWidth - undulator gap.
- The **aperture parameters** may also be specified.
- See **Magnet Strength Polarity** for polarity notes.
- To generate radiation from particles propagating through the undulator field, synchrotron radiation physics must be included in the model’s physicsList. See **Physics Processes** for further details.

Examples:
6.5.31 transform3d

`transform3d` defines an arbitrary three-dimensional transformation of the curvilinear coordinate system at that point in the beam line sequence. This is often used to rotate components by a large angle.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x offset</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>y</td>
<td>y offset</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>z</td>
<td>z offset</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>phi</td>
<td>phi Euler angle</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>theta</td>
<td>theta Euler angle</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>psi</td>
<td>psi Euler angle</td>
<td>0</td>
<td>No</td>
</tr>
</tbody>
</table>

**Note:** this permanently changes the coordinate frame, so care must be taken to undo any rotation if intended for only one component.

Examples:

```plaintext
rcolrot: transform3d, psi=pi/2;
```

6.5.32 element

`element` defines an arbitrary element that’s defined by externally provided geometry. It includes the possibility of overlaying a field as well. Several geometry formats are supported. The user must supply the length (accurately) as well as a diameter, such that the geometry will be contained in a box that has horizontal and vertical sizes of diameter.

The geometry is simply placed in the beam line. There is no placement offset other than the offset and tilt of the element in the beam line. Therefore, the user must prepare geometry with the placement as required.

An alternative strategy is to use the `gap` beam line element and make a placement at the appropriate point in global coordinates.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometryFile</td>
<td>Filename of geometry</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>l</td>
<td>Length. Arc length in case of a finite angle.</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>horizontalWidth</td>
<td>Diameter of component [m]</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>fieldAll</td>
<td>Name of field object to use</td>
<td>NA</td>
<td>No</td>
</tr>
<tr>
<td>angle</td>
<td>Angle the component bends the beam line.</td>
<td>0</td>
<td>No</td>
</tr>
</tbody>
</table>

`geometryFile` should be of the format `format:filename`, where `format` is the geometry format being used (gdml \ gmad \ mokka) and `filename` is the path to the geometry file. See [Externally Provided Geometry](#) for more details.

`fieldAll` should refer to the name of a field object the user has defined in the input gmad file. The syntax for this is described in [Fields](#).

**Note:** The length must be larger than the geometry so that it is contained within it and no overlapping geometry will be produced. However, care must be taken, as the length will be the length of the component inserted in the input sequence.
beamline. If this is much larger than the size required for the geometry, the beam may be mismatched into the rest of the accelerator. A common practice is to add a picometre to the length of the geometry.

**Simple example:**

```plaintext
detector: element, geometryFile="gdml:atlasreduced.gdml", horizontalWidth=10*m, → l=44*m;
```

**Example with field:**

```plaintext
somefield: field, type="ebmap2d",
   eScaling = 3.1e3,
   bScaling = 0.5,
   integrator = "g4classicalrk4",
   magneticFile = "poisson2d:/Path/To/File.TXT",
   magneticInterpolator = "nearest2D",
   electricFile = "poisson2d:/Another/File.TXT",
   electricInterpolator = "linear2D";

detec: element, geometryFile="mokka:qq.sql", fieldAll="somefield", l=5*m, → horizontalWidth=0.76*m;
```

**Note:** For GDML geometry, we preprocess the input file prepending all names with the name of the element. This is to compensate for the fact that the Geant4 GDML loader does not handle unique file names. However, in the case of very large files with many vertices, the preprocessing can dominate. In this case, the option `preprocessGDML` should be turned off. The loading will only work with one file in this case.

### 6.5.33 marker

**marker** defines a point in the lattice. This element has no physical length and is only used as a reference. For example, a *sampler* (see [*Output at a Plane - Samplers*](#)) is used to record particle passage at the front of a component, but how would you record particles exiting a particular component? The intended method is to use a **marker** and place it in the sequence after that element, then attach a sampler to the marker.

**Examples:**

```plaintext
ml: marker;
```

### 6.6 Aperture Parameters

For elements that contain a beam pipe, several aperture models can be used. These aperture parameters can be set as the default for every element using the `option` command (see [*options*](#)) and can be overridden for each element by specifying them with the element definition. The aperture is controlled through the following parameters:

- `apertureType`
- `beampipeRadius` or `aper1`
- `aper2`
- `aper3`
- `aper4`
- `vacuumMaterial`
- `beampipeThickness`
- `beampipeMaterial`
For each aperture model, a different number of parameters are required. Here, we follow the MAD-X convention and have four parameters. The user must specify them as required for that model. BDSIM will check to see if the combination of parameters is valid. `beampipeRadius` and `aper1` are degenerate.

Up to four parameters can be used to specify the aperture shape (`aper1, aper2, aper3, aper4`). These are used differently for each aperture model and match the MAD-X aperture definitions. The required parameters and their meaning are given in the following table.

<table>
<thead>
<tr>
<th>Aperture Model</th>
<th># of parameters</th>
<th>aper1</th>
<th>aper2</th>
<th>aper3</th>
<th>aper4</th>
</tr>
</thead>
<tbody>
<tr>
<td>circular</td>
<td>1</td>
<td>radius</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>rectangular</td>
<td>2</td>
<td>x half-width</td>
<td>y half-width</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>elliptical</td>
<td>2</td>
<td>x semi-axis</td>
<td>y semi-axis</td>
<td>radius of circle</td>
<td>NA</td>
</tr>
<tr>
<td>lhc</td>
<td>3</td>
<td>x half-width of rectangle</td>
<td>y half-width of rectangle</td>
<td>radius of circle</td>
<td>NA</td>
</tr>
<tr>
<td>lhc-detailed</td>
<td>3</td>
<td>x half-width of rectangle</td>
<td>y half-width of rectangle</td>
<td>radius of circle</td>
<td>NA</td>
</tr>
<tr>
<td>rectellipse</td>
<td>4</td>
<td>x half-width of rectangle</td>
<td>y half-width of rectangle</td>
<td>x semi-axis of ellipse</td>
<td>y semi-axis of ellipse</td>
</tr>
<tr>
<td>racetrack</td>
<td>3</td>
<td>horizontal offset of circle</td>
<td>vertical offset of circle</td>
<td>radius of circular part</td>
<td>NA</td>
</tr>
<tr>
<td>octagonal</td>
<td>4</td>
<td>x half-width</td>
<td>y half-width</td>
<td>x point of start of edge</td>
<td>y point of start of edge</td>
</tr>
<tr>
<td>clicpcl</td>
<td>4</td>
<td>x half-width</td>
<td>top ellipse y half-height</td>
<td>bottom ellipse y half-height</td>
<td>y separation between ellipses</td>
</tr>
</tbody>
</table>

These parameters can be set with the `option` command, as the default parameters and also on a per element basis that overrides the defaults for that specific element.

In the case of `clicpcl` (CLIC Post Collision Line), the beam pipe is asymmetric. The centre is the same as the geometric centre of the bottom ellipse. Therefore, `aper4`, the y separation between ellipses is added on to the 0 position. The parameterisation is taken from Phys. Rev. ST Accel. Beams 12, 021001 (2009).

### 6.7 Magnet Geometry Parameters

As well as the beam pipe, magnet beam line elements also have further outer geometry beyond the beam pipe. This geometry typically represents the magnetic poles and yoke of the magnet but there are several geometry types to choose from. The possible different styles are described below and syntax examples can be found in `examples/features/geometry/4_magnets/`.

- Externally provided geometry can also be wrapped around the beam pipe (detailed below).

The magnet geometry is controlled by the following parameters.

**Note:** These can all be specified using the `option` command as well as on a per element basis, but in this case they act as a default that will be used if none are specified by the element.

**Note:** The option `ignoreLocalMagnetGeometry` exists and if it is true (1), any per-element magnet geometry definitions will be ignored and the ones specified in Options will be used.
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>magnetGeometryType</td>
<td>The style of magnet geometry to use. One of: <code>cylindrical</code>, <code>polescircular</code>, <code>polessquare</code>, <code>polesfacet</code>, <code>polesfacetcrop</code>, <code>lhclleft</code>, <code>lhclright</code>, <code>none</code> and <code>format:path</code>.</td>
<td><code>polessquare</code></td>
<td>No</td>
</tr>
<tr>
<td>horizontalWidth</td>
<td><strong>Full</strong> horizontal width of the magnet (m)</td>
<td>0.6 m</td>
<td>No</td>
</tr>
<tr>
<td>outerMaterial</td>
<td>Material of the magnet</td>
<td>“iron”</td>
<td>No</td>
</tr>
<tr>
<td>yokeOnInside</td>
<td>Whether the yoke of a dipole appears on the inside of the bend and if false, it’s on the outside. Applicable only to dipoles.</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>hStyle</td>
<td>Whether a dipole (only a dipole) will be an H style one or a C style one (c style by default. True (‘1’) or False (‘0’).)</td>
<td>0</td>
<td>No</td>
</tr>
<tr>
<td>vhRatio</td>
<td>The vertical to horizontal ratio of a magnet. The width will always be the <code>horizontalWidth</code> and the height will scale according to this ratio. In the case of a vertical kicker it will be the height that is <code>horizontalWidth</code> (as the geometry is simply rotated). Ranges from 0.1 to 10. This currently only applies to dipoles with poled geometry.</td>
<td>0.8</td>
<td>No</td>
</tr>
</tbody>
</table>
Examples:

```plaintext
option, magnetGeometryType = "polesfacetcrop",
    horizontalWidth = 0.5*m;
```

```plaintext
m1: quadrupole, l=0.3*m,
    k1=0.03,
    magnetGeometryType="gdml:geometryfiles/quad.gdml",
    horizontalWidth = 0.5*m;
```

**Warning:** The choice of magnet outer geometry will significantly affect the beam loss pattern in the simulation, as particles and radiation may propagate much further along the beam line when a magnet geometry with poles is used.

**Note:** Should a custom selection of various magnet styles be required for your simulation, please contact us (see Feature Request) and this can be added - it is a relatively simple process.

### 6.7.1 No Magnet Outer Geometry - “none”

No geometry for the magnet outer part is built at all and nothing is placed in the model. This results in only a beam pipe with the correct fields being provided.

![Beam Pipe](image)

### 6.7.2 Cylindrical - “cylindrical”

The beam pipe is surrounded by a cylinder of material (the default is iron) whose outer diameter is controlled by the `horizontalWidth` parameter. In the case of beam pipes that are not circular in cross-section, the cylinder fits directly against the outside of the beam pipe.

This geometry is useful when a specific geometry is not known. The surrounding magnet volume acts to produce secondary radiation as well as act as material for energy deposition, therefore this geometry is best suited for the most general studies.

### 6.7.3 Poles Circular - “polescircular”

This magnet geometry has simple iron poles according to the order of the magnet and the yoke is represented by an annulus. Currently no coils are implemented. If a non-symmetric beam pipe geometry is used, the larger of the horizontal and vertical dimensions of the beam pipe will be used to create the circular aperture at the pole tips.
6.7.4 Poles Square (Default) - “polessquare”

This magnet geometry has again, individual poles according to the order of the magnet but the yoke is an upright square section to which the poles are attached. This geometry behaves in the same way as polescircular with regard to the beam pipe size.

horizontalWidth is the full width of the magnet horizontally as shown in the figure below, not the diagonal width.

6.7.5 Poles Faceted - “polesfacet”

This magnet geometry is much like polessquare; however, the yoke is such that the pole always joins at a flat piece of yoke and not in a corner. This geometry behaves in the same way as polescircular with regards to the beam pipe size.

horizontalWidth is the full width as shown in the figure.
6.7. Magnet Geometry Parameters
6.7.6 Poles Faceted with Crop - “polesfacetcrop”

This magnet geometry is quite similar to polesfacet, but the yoke in between each pole is cropped to form another facet. This results in the magnet geometry having double the number of poles as sides.

horizontalWidth is the full width horizontally as shown in the figure.

6.7.7 LHC Left & Right - “lhcleft” | “lhcright”

lhcleft and lhcright provide more detailed magnet geometry appropriate for the LHC. Here, the left and right suffixes refer to the shift of the magnet body with respect to the reference beam line. Therefore, lhcleft has the magnet body shifted to the left in the direction of beam travel and the ‘active’ beam pipe is the right one. Vice versa for the lhcright geometry.

For this geometry, only the sbend and quadrupole have been implemented. All other magnet geometry defaults to the cylindrical set.

This geometry is parameterised to a degree regarding the beam pipe chosen. Of course, parameters similar to the LHC make most sense, as does use of the lhcdetailed aperture type. Examples are shown with various beam pipes and both sbend and quadrupole geometries.
6.7. Magnet Geometry Parameters
6.8 Offsets & Tilts - Component Misalignment

To simulate a real accelerator it may be necessary to introduce measured placement offsets or misalignments and rotations. Every component can be displaced transversely and rotated along the axis of the beam propagation.

**Note:** Components that have a finite angle (rbend and sbend) will only respond to tilt and not vertical or horizontal offsets. This is because these would change the length of the bend about its central axis. This is not currently handled but may be implemented in future releases.

**Note:** A tilt on a component with a finite angle causes the axis the angle is induced in (typically the y-axis) to be rotated without rotating the reference frame of the beam, i.e. a dipole with a \( \frac{\pi}{2} \) tilt will become a vertical bend without flipping x and y in the sampler or subsequent components. This matches the behaviour of MAD8 and MAD-X.

**Note:** A right-handed coordinate system is used and the beamline is built along the \( z \) direction.

The misalignments can be controlled through the following parameters.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>offsetX</td>
<td>Horizontal displacement of the component [m]</td>
</tr>
<tr>
<td>offsetY</td>
<td>Vertical displacement of the component [m]</td>
</tr>
<tr>
<td>tilt</td>
<td>Clockwise rotation of the component, facing in the direction of the beamline ( z ) [rad]. In the case of an rbend or sbend, this rotates the axis about which the beam bends.</td>
</tr>
</tbody>
</table>

Examples:

```plaintext
d1: drift, l=1*m, offsetX=1*cm;
d2: drift, l=0.5*m, offsetY = 0.3*cm, tilt=0.003;
```

6.9 Cavity Geometry Parameters

A more detailed rf cavity geometry may be described by constructing a ‘cavity’ object in gmad and attaching it by name to an element. The following parameters may be added to a cavity object:
### Parameter | Required | Description
--- | --- | ---
**name** | Yes | Name of the object
**type** | Yes | (elliptical | rectangular | pillbox)
**material** | Yes | The material for the cavity
**irisRadius** | No | The radius of the narrowest part
**equatorRadius** | No | The radius of the widest part
**halfCellLength** | No | Half-length along a cell
**equatorHorizontalAxis** | Elliptical only | Horizontal semi-axis of the ellipse at the cavity equator
**equatorVerticalAxis** | Elliptical only | Vertical semi-axis of the ellipse at the cavity equator
**irisHorizontalAxis** | Elliptical only | Horizontal semi-axis of the ellipse at the iris
**irisVerticalAxis** | Elliptical only | Vertical semi-axis of the ellipse at the iris
**tangentLineAngle** | Elliptical only | Angle to the vertical line connecting two ellipses
**thickness** | No | Thickness of material
**numberOfPoints** | No | Number of points to generate around 2 \( \pi \).
**numberOfCells** | No | Number of cells to construct

Example:

```plaintext
shinyCavity: cavity, type="elliptical",
irisRadius = 35*mm,
equatorRadius = 103.3*mm,
halfCellLength = 57.7*mm,
equatorHorizontalAxis = 40*mm,
equatorVerticalAxis = 42*mm,
irisHorizontalAxis = 12*mm,
irisVerticalAxis = 19*mm,
tangentLineAngle = 13.3*pi/180,
thickness = 1*mm,
numberOfPoints = 24,
numberOfCells = 1;
```

6.9. Cavity Geometry Parameters 75
The parameterisation used to define elliptical cavities in BDSIM. The symbols used in the figure map to the cavity options according to the table below.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>BDSIM Cavity Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{R}$</td>
<td>equatorRadius</td>
</tr>
<tr>
<td>$r$</td>
<td>irisRadius</td>
</tr>
<tr>
<td>$A$</td>
<td>equatorHorizontalAxis</td>
</tr>
<tr>
<td>$B$</td>
<td>equatorVerticalAxis</td>
</tr>
<tr>
<td>$a$</td>
<td>irisHorizontalAxis</td>
</tr>
<tr>
<td>$b$</td>
<td>irisVerticalAxis</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>tangentLineAngle</td>
</tr>
<tr>
<td>$L$</td>
<td>halfCellLength</td>
</tr>
</tbody>
</table>

### 6.10 Fields

BDSIM provides the facility to overlay pure magnetic, pure electric or combined electromagnetic fields on an element, as defined by an externally provided field map. This can be done for 1) only the vacuum volume; 2) only the volume outside the vacuum (i.e. the yoke); 3) or one full map for the whole element. BDSIM allows any Geant4 integrator to be used to calculate the motion of the particle, which can be chosen given knowledge of the smoothness of the field or the application. BDSIM also provides a selection of 1-4D interpolators that are used to provide the field value in between the data points in the supplied field map.

To overlay a field, one must define a field ‘object’ in the parser and then ‘attach’ it to an element.

- Fields are in a local Cartesian coordinate system with respect to the origin of the element they are attached to.
- The field may be attached to everything “fieldAll”; the vacuum volume “fieldVacuum”, or the yoke “fieldOuter”.
- Magnetic and electric field maps are specified in separate files and may have different interpolators.
- Fields may have up to four dimensions.

The dimensions are (by default) in order $x, y, z, t$. For example, specifying a 3D field will be $x, y, z$ and a 2D field $x, y$.

For BDSIM format fields (see Formats, Field Map File Formats and BDSIM Field Format Different Dimensions), the user can however specify different dimension with the other dimensions being assumed constant. For example, a field that varies in $x, z$ is possible (assumed constant in $y$). For BDSIM format fields, this is detected automatically by the column labelling and the keys in the header of the file that specify the ranges in each dimension. The dimensions must however be in ascending or descending order.

**Note:** Currently only **regular** (evenly spaced) grids are supported with field maps. It would require significant development to extend this to irregular grids. It’s strongly recommended the user re-sample any existing field map into a regular grid. A regular grid is also much faster for tracking purposes.

Here is example syntax to define a field object named ‘somefield’ in the parser and overlay it onto a drift pipe where it covers the full volume of the drift (not outside it though):

```plaintext
somefield: field, type="ebmap2d",
eScaling = 3.0,
bScaling = 0.4,
integrator = "g4classicalrk4",
magneticFile = "poisson2d:/Path/To/File.TXT",
magneticInterpolator = "nearest2D",
electricFile = "poisson2d:/Another/File.TXT",
electricInterpolator = "linear2D";
```

(continues on next page)
Each beam line element will allow “fieldAll”, “fieldVacuum” and “fieldOuter” to be specified.

When defining a field, the following parameters can be specified.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>See type table below.</td>
</tr>
<tr>
<td>eScaling</td>
<td>A numerical scaling factor that all electric field vectors’ amplitudes will be multiplied by</td>
</tr>
<tr>
<td>bScaling</td>
<td>A numerical scaling factor that all magnetic field vectors’ amplitudes will be multiplied by</td>
</tr>
<tr>
<td>integrator</td>
<td>The integrator used to calculate the motion of the particle in the field. See below for full list of supported integrators.</td>
</tr>
<tr>
<td>globalTransform</td>
<td>Boolean. Whether a transform from local curvilinear coordinates to global coordinates should be provided (default true).</td>
</tr>
<tr>
<td>magneticFile</td>
<td>“format:filePath” - see formats below.</td>
</tr>
<tr>
<td>magneticInterpolator</td>
<td>Which interpolator to use - see below for a full list.</td>
</tr>
<tr>
<td>electricFile</td>
<td>“format:filePath” - see formats below.</td>
</tr>
<tr>
<td>electricInterpolator</td>
<td>Which interpolator to use - see below for a full list.</td>
</tr>
<tr>
<td>x</td>
<td>x-offset from element it’s attached to</td>
</tr>
<tr>
<td>y</td>
<td>y-offset from element it’s attached to</td>
</tr>
<tr>
<td>z</td>
<td>z-offset from element it’s attached to</td>
</tr>
<tr>
<td>t</td>
<td>t-offset from Global t in seconds</td>
</tr>
<tr>
<td>phi</td>
<td>Euler phi rotation from the element the field is attached to</td>
</tr>
<tr>
<td>theta</td>
<td>Euler theta rotation from the element the field is attached to</td>
</tr>
<tr>
<td>psi</td>
<td>Euler psi rotation from the element the field is attached to</td>
</tr>
<tr>
<td>axisAngle</td>
<td>(Boolean) Use axis angle rotation variables. Default 0 (Euler).</td>
</tr>
<tr>
<td>axisX</td>
<td>x-component of axis defining axis / angle rotation</td>
</tr>
<tr>
<td>axisY</td>
<td>y-component of axis defining axis / angle rotation</td>
</tr>
<tr>
<td>axisZ</td>
<td>z-component of axis defining axis / angle rotation</td>
</tr>
<tr>
<td>angle</td>
<td>angle (rad) of defining axis / angle rotation</td>
</tr>
<tr>
<td>autoScale</td>
<td>This automatically calculates the field gradient at the origin and the field magnitude will be automatically scaled according to the normalised k strength (such as k1 for a quadrupole) for the magnet it’s attached to. Only applicable for when attached to magnets.</td>
</tr>
<tr>
<td>maximumStepLength</td>
<td>The maximum permitted step length through the field. (m)</td>
</tr>
</tbody>
</table>

**Note:** Either axis angle (with unit axis 3-vector) or Euler angles can be used to provide the rotation between the element the field maps are attached to and the coordinates of the field map. Use axisAngle=1 to use the axis angle rotation scheme.

**Note:** A right-handed coordinate system is used in Geant4, so positive x is out of a ring.

### 6.10.1 Field Types

- These are not case sensitive.
### Type String

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmap1d</td>
<td>1D magnetic only field map</td>
</tr>
<tr>
<td>bmap2d</td>
<td>2D magnetic only field map</td>
</tr>
<tr>
<td>bmap3d</td>
<td>3D magnetic only field map</td>
</tr>
<tr>
<td>bmap4d</td>
<td>4D magnetic only field map</td>
</tr>
<tr>
<td>emap1d</td>
<td>1D electric only field map</td>
</tr>
<tr>
<td>emap2d</td>
<td>2D electric only field map</td>
</tr>
<tr>
<td>emap3d</td>
<td>3D electric only field map</td>
</tr>
<tr>
<td>emap4d</td>
<td>4D electric only field map</td>
</tr>
<tr>
<td>ebmap1d</td>
<td>1D electric-magnetic field map</td>
</tr>
<tr>
<td>ebmap2d</td>
<td>2D electric-magnetic field map</td>
</tr>
<tr>
<td>ebmap3d</td>
<td>3D electric-magnetic field map</td>
</tr>
<tr>
<td>ebmap4d</td>
<td>4D electric-magnetic field map</td>
</tr>
</tbody>
</table>

### 6.10.2 Formats

**Note:** BDSIM field maps by default have units cm, s.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdsim1d</td>
<td>1D BDSIM format file (Units cm, s)</td>
</tr>
<tr>
<td>bdsim2d</td>
<td>2D BDSIM format file (Units cm, s)</td>
</tr>
<tr>
<td>bdsim3d</td>
<td>3D BDSIM format file (Units cm, s)</td>
</tr>
<tr>
<td>bdsim4d</td>
<td>4D BDSIM format file (Units cm, s)</td>
</tr>
<tr>
<td>poisson2d</td>
<td>2D Poisson Superfish SF7 file</td>
</tr>
<tr>
<td>poisson2dquad</td>
<td>2D Poisson Superfish SF7 file for 1/8th of quadrupole</td>
</tr>
<tr>
<td>poisson2ddipole</td>
<td>2D Poisson Superfish SF7 file for positive quadrant that’s reflected to produce a full windowed dipole field</td>
</tr>
</tbody>
</table>

Field maps in the following formats are accepted:

- BDSIM’s own format (both uncompressed .dat and gzip compressed files. \gzip must be in the file name for this to load correctly.)
- Superfish Poisson 2D SF7

These are described in detail below. More field formats can be added relatively easily - see Feature Request. A detailed description of the formats is given in Field Map File Formats. A preparation guide for BDSIM format files is provided here BDSIM Field Map File Preparation.

### 6.10.3 Integrators

The following integrators are provided. The majority are interfaces to Geant4 integrators. \texttt{g4classicalrk4} is typically the recommended default and is very robust. \texttt{g4cakskarprkf45} is similar but slightly less CPU-intensive. For version Geant4.10.4 onwards, \texttt{g4dormandprince745} is the default recommended by Geant4 (although not the BDSIM default currently). Note: any integrator capable of operating on EM fields will work on solely B- or E-fields.

We recommend looking at the source .hh files in the Geant4 source code for an explanation of each, as this is where they are documented. The source files can be found in \texttt{<geant4-source-dir>/source/geometry/magneticfield/include}. 
6.10.4 Interpolators

The field may be queried at any point inside the volume, so an interpolator is required to provide a value of the field in between specified points in the field map. There are many algorithms that can be used to interpolate the field map data. A mathematical description of the ones provided in BDSIM as well as example plots is shown in \textit{Field Map Interpolators}.

- This string is case-insensitive.

<table>
<thead>
<tr>
<th>String</th>
<th>B/EM</th>
<th>Time Varying</th>
<th>Required Geant4 Version (&gt;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>g4cashkarpkf45</td>
<td>EM</td>
<td>Y</td>
<td>10.0</td>
</tr>
<tr>
<td>g4classicalrk4</td>
<td>EM</td>
<td>Y</td>
<td>10.0</td>
</tr>
<tr>
<td>g4constrk4</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4expliciteuler</td>
<td>EM</td>
<td>Y</td>
<td>10.0</td>
</tr>
<tr>
<td>g4impliciteuler</td>
<td>EM</td>
<td>Y</td>
<td>10.0</td>
</tr>
<tr>
<td>g4simpleheum</td>
<td>EM</td>
<td>Y</td>
<td>10.0</td>
</tr>
<tr>
<td>g4simplerunge</td>
<td>EM</td>
<td>Y</td>
<td>10.0</td>
</tr>
<tr>
<td>g4exacthelixstepper</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4helixexpliciteuler</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4helixheum</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4heliximpliciteuler</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4helixmixedstepper</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4helixsimplerunge</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4nystromrk4</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4rk3stepper</td>
<td>B</td>
<td>N</td>
<td>10.0</td>
</tr>
<tr>
<td>g4bogackishampine23</td>
<td>EM</td>
<td>Y</td>
<td>10.3</td>
</tr>
<tr>
<td>g4bogackishampine45</td>
<td>EM</td>
<td>Y</td>
<td>10.3</td>
</tr>
<tr>
<td>g4dolomcprik34</td>
<td>EM</td>
<td>Y</td>
<td>10.3</td>
</tr>
<tr>
<td>g4dormandprince745</td>
<td>EM</td>
<td>Y</td>
<td>10.3</td>
</tr>
<tr>
<td>g4dormandprincerk56</td>
<td>EM</td>
<td>Y</td>
<td>10.3</td>
</tr>
<tr>
<td>g4dormandprincerk78</td>
<td>EM</td>
<td>Y</td>
<td>10.3</td>
</tr>
<tr>
<td>g4tsitourasrk45</td>
<td>EM</td>
<td>Y</td>
<td>10.3</td>
</tr>
<tr>
<td>g4rk547feq1</td>
<td>EM</td>
<td>Y</td>
<td>10.4</td>
</tr>
<tr>
<td>g4rk547feq2</td>
<td>EM</td>
<td>Y</td>
<td>10.4</td>
</tr>
<tr>
<td>g4rk547feq3</td>
<td>EM</td>
<td>Y</td>
<td>10.4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nearest1d</td>
<td>Nearest neighbour in 1D</td>
</tr>
<tr>
<td>nearest2d</td>
<td>Nearest neighbour in 2D</td>
</tr>
<tr>
<td>nearest3d</td>
<td>Nearest neighbour in 3D</td>
</tr>
<tr>
<td>nearest4d</td>
<td>Nearest neighbour in 4D</td>
</tr>
<tr>
<td>linear1d</td>
<td>Linear interpolation in 1D</td>
</tr>
<tr>
<td>linear2d</td>
<td>Linear interpolation in 2D</td>
</tr>
<tr>
<td>linear3d</td>
<td>Linear interpolation in 3D</td>
</tr>
<tr>
<td>linear4d</td>
<td>Linear interpolation in 4D</td>
</tr>
<tr>
<td>cubic1d</td>
<td>Cubic interpolation in 1D</td>
</tr>
<tr>
<td>cubic2d</td>
<td>Cubic interpolation in 2D</td>
</tr>
<tr>
<td>cubic3d</td>
<td>Cubic interpolation in 3D</td>
</tr>
<tr>
<td>cubic4d</td>
<td>Cubic interpolation in 4D</td>
</tr>
</tbody>
</table>
6.11 Externally Provided Geometry

BDSIM provides the ability to use externally provided geometry in the Geant4 model constructed by BDSIM. A variety of formats are supported (see Geometry Formats). External geometry can be used in three ways:

1) A placement of a piece of geometry unrelated to the beam line.
2) Wrapped around the beam pipe in a BDSIM magnet element.
3) As a general element in the beam line where the geometry constitutes the whole object.
4) As the world volume in which the BDSIM beamline is placed.

These are discussed in order in Placements, External Magnet Geometry and Element.

6.11.1 Geometry Formats

The following geometry formats are supported. More may be added in collaboration with the BDSIM developers - please see Feature Request. The syntax and preparation of these geometry formats are described in more detail in External Geometry Formats.

<table>
<thead>
<tr>
<th>Format String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gdml</td>
<td>Geometry Description Markup Language - Geant4’s official geometry persistency format - recommended</td>
</tr>
<tr>
<td>ggmad</td>
<td>Simple text interface provided by BDSIM to some simple Geant4 geometry classes</td>
</tr>
<tr>
<td>mokka</td>
<td>An SQL style description of geometry</td>
</tr>
</tbody>
</table>

Note: BDSIM must be compiled with the GDML build option in CMake turned on for gdml loading to work.

Note: For GDML geometry, we preprocess the input file prepending all names with the name of the element. This is to compensate for the fact that the Geant4 GDML loader does not handle unique file names. However, in the case of very large files with many vertices, the preprocessing can dominate. In this case, the option preprocessGDML should be turned off. The loading will only work with one file in this case.

Warning: If a geometry file path is defined relative to the location of the GMAD file and that GMAD file is included in a parent file in a different location, the file will not be correctly located (i.e. main.gmad includes ../somedit/anotherfile.gmad, which defines geometry in “../a/relative/path/geometryfile.gdml”. The file will not be found). If all GMAD files are located in the same directory, this will not be a problem. It is better / cleaner overall to use multiple GMAD input files and include them.
6.11.2 External World Geometry

External geometry can be supplied as the world volume with the option `worldGeometryFile` (see Geometry Options). The BDSIM beamline will be placed inside this world volume provided in the file.

Unlike the standard BDSIM world volume whose size is set dynamically, the external world volume will have fixed dimensions, therefore the user should supply a world volume of sufficient size so as to fully encompass the BDSIM beamline. Should the extents of the BDSIM beamline be larger than the world extents, the beamline will not be constructed and BDSIM will exit.

`worldGeometryFile` should be of the format `format:filename`, where `format` is the geometry format being used (`gdml` | `gmad` | `mokka`) and `filename` is the path to the geometry file. See Externally Provided Geometry for more details.

- See also Geometric Importance Sampling for usage of this.

6.12 Placements

Geometry provided in an external file may be placed at any location in the world with any rotation. This is intended to place geometry alongside the beam line and not inside or as part of it. The user is responsible for ensuring that the geometry does not overlap with any other geometry including the beam line. Only in special cases, such as for a magnet yoke, can externally provided geometry be placed “inside” BDSIM geometry.

For geometry to be placed in the beam line, use the `element`.

**Warning:** If the geometry overlaps, tracking faults may occur from Geant4 as well as incorrect results and there may not always be warnings provided. For this reason, BDSIM will always use the Geant4 overlap checker when placing external geometry into the world volume. This only ensures the container doesn’t overlap with BDSIM geometry, not that the internal geometry is valid.

**Warning:** You cannot place external geometry ‘inside’ an accelerator component with a placement. Although it may appear OK in the visualiser, the hierarchy of the geometry will be wrong and the tracking will not work as expected. Avoid this.

There are 3 possible ways to place a piece of geometry.

1) In global Cartesian coordinates.
   - `x`, `y`, `z` and any rotation are with respect to the world frame of reference.

2) In curvilinear coordinates.
   - `s`, `x`, `y` are used along with a rotation. The transform for the distance `s` along the beamline is first applied. `x`, `y` and the rotation are with respect to that frame.

3) In curvilinear coordinates with respect to a beam line element by name.
   - The name of an element is used to look up its `s` coordinate. `s`, `x`, `y` and the rotation are with respect to the centre of that element. Therefore, `s` in this case is local curvilinear `s`.

The scenario is automatically selected based on which parameters are set. If `s` is finite, then it is either scenario 2 or 3. If `referenceElement` is specified, scenario 3 is assumed.

**Warning:** For both scenarios 2) and 3), a placement can only be made inside the S length of the accelerator - it is not possible to place something beyond the accelerator currently. In this case, the user should resort to a global placement.
The following parameters may be specified with a placement in BDSIM:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>geometryFile</td>
<td>format:file - which geometry format and file to use</td>
</tr>
<tr>
<td>x</td>
<td>Offset in global x</td>
</tr>
<tr>
<td>y</td>
<td>Offset in global y</td>
</tr>
<tr>
<td>z</td>
<td>Offset in global z</td>
</tr>
<tr>
<td>s</td>
<td>Curvilinear s coordinate (global</td>
</tr>
<tr>
<td>phi</td>
<td>Euler angle phi for rotation</td>
</tr>
<tr>
<td>theta</td>
<td>Euler angle theta for rotation</td>
</tr>
<tr>
<td>psi</td>
<td>Euler angle psi for rotation</td>
</tr>
<tr>
<td>axisX</td>
<td>Axis angle rotation x-component of unit vector</td>
</tr>
<tr>
<td>axisY</td>
<td>Axis angle rotation y-component of unit vector</td>
</tr>
<tr>
<td>axisZ</td>
<td>Axis angle rotation z-component of unit vector</td>
</tr>
<tr>
<td>angle</td>
<td>Axis angle, angle to rotate about unit vector</td>
</tr>
<tr>
<td>axisAngle</td>
<td>Boolean whether to use axis angle rotation scheme (default false)</td>
</tr>
<tr>
<td>sensitive</td>
<td>Whether the geometry records energy deposition (default true)</td>
</tr>
<tr>
<td>referenceElement</td>
<td>Name of element to place geometry with respect to (string)</td>
</tr>
<tr>
<td>referenceElementNumber</td>
<td>Occurrence of referenceElement to place with respect to if it is used more than once in the sequence. Zero counting.</td>
</tr>
</tbody>
</table>

`referenceElementNumber` is the occurrence of that element in the sequence. For example, if a sequence was:

```plaintext
l1: line=(d1,sb1,d2,qd1,d2,dfl,d2,sb1,d1);
```

and we wanted to place with respect to the first element, we would use:

```plaintext
p1: placement, referenceElement="d1",
    referenceElementNumber=0;
```

If 0, the `referenceElementNumber` argument is optional. If we want to place with respect to the third usage of “d2”, we would use:

```plaintext
p1: placement, referenceElement="d2",
    referenceElementNumber=3;
```

**Note:** Dipoles are split in BDSIM into many small straight sections. These must have a unique name to appear correctly in the Geant4 visualisation system. The splitting is done dynamically based on the angle of the bend and if it has pole face rotations on one or both sides. The names are mangled and so the original name will not be found. The user should run the visualiser first and identify the name of the segment of the dipole they wish to place with respect to. Alternatively, in the case of low angle bends, the element before or after can be used with a finite `s` offset.

- Examples can be found in `bdsim/examples/features/geometry/13_placements`.
- The file path provided in `geometryFile` should either be relative to where bdsim is executed from or an absolute path.
- The main beam line begins at (0,0,0) by default but may be offset. See `Offset for Main Beam Line` for more details.

Two styles of rotation can be used: either a set of three Euler angles, or the axis angle rotation scheme, where a `unit` vector is provided in `x,y,z` and an angle to rotate about that. These variables are used to construct a `G4RotationMatrix` directly, which is also the same as a `CLHEP::HepRotation`.

**Note:** Geant4 uses a right-handed coordinate system and `m` and `rad` are the default units for offsets and angles in BDSIM.
The following is an example syntax used to place a piece of geometry:

```plaintext
leadblock: placement, x = 10*m,
y = 3*cm,
z = 12*m,
geometryFile="gdml:mygeometry/detector.gdml";
```

**Warning:** Care must be taken not to define the same placement name twice. If `leadblock` were declared again here, the first definition would be updated with parameters from the second, leading to possibly unexpected geometry.

### 6.12.1 External Magnet Geometry

A geometry file may be placed around a beam pipe inside a BDSIM magnet instance. The beam pipe will be constructed as normal and will use the appropriate BDSIM tracking routines, but the yoke geometry will be loaded from the file provided. The external geometry must have a cut out in its container volume for the beam pipe to fit, i.e. both the beam pipe and the yoke exist at the same level in the geometry hierarchy (both are placed in one container for the magnet). The beam pipe is not placed ‘inside’ the yoke.

This will work for *solenoid, sbend, rbend, quadrupole, sextupole, octupole, decapole, multipole, muonspoiler, vkicker, hkicker* element types in BDSIM.

Example:

```plaintext
q1: quadrupole, l=20*cm, k1=0.0235, magnetGeometryType="gdml:mygeometry/atf2quad.gdml";
```

### 6.12.2 Element

A general piece of geometry may be placed in the beam line along with any externally provided field map using the `element` beam line element. See `element`.

### 6.13 Lattice Sequence

Once all the necessary components have been defined, they must be placed in a sequence to make a lattice. Elements can be repeated. A sequence of elements is defined by a `line`. Lines of lines can be made to describe the accelerator sequence programmatically, i.e.

```plaintext
d1: drift, l=3*m;
q1: quadrupole, l=0.1*m, k1=0.684;
q2: quadrupole, l=0.1*m, k1=-0.684;
fodo: line = (q1,d1,q2,d1);
transportline: line(fodo, fodo, fodo, fodo);
```

### 6.13.1 Line

`line` defines a sequence of elements.

```plaintext
name: line=(element1, element2, element3, ...);
```

Here, `element` can be any element or line. Lines can also be reversed using

```plaintext
line_name : line=(line_2)
```
or within another line by:

```plaintext
line=(line_1,-line_2)
```

Reversing a line also reverses all nested lines within.

## 6.13.2 use - Defining which Line to Use

Once all elements and at least one line is defined, the main sequence of the beam line can be defined. This must be defined using the following syntax:

```plaintext
use, period=<line_name>
```

Examples:

```plaintext
d1: drift, l=3.2*m;
q1: quadrupole, l=20*cm, k1=4.5;
q2: quadrupole, l=20*cm, k1=-4.5;
fodo: line=(d1,q1,d1,q2,d1);
use, period=fodo;
```

The beam line is placed in the world volume (the outermost coordinate system) starting at position (0,0,0) with direction (0,0,1) - i.e. pointing in positive \( z \). The user may specify an initial offset and rotation for the beam line with respect to the world volume using the options described in Offset for Main Beam Line.

Multiple beam lines may also be visualised - but only visualised (not suitable for simulations currently). Details are provided in Multiple Beam Lines.

## 6.14 Output at a Plane - Samplers

BDSIM provides a sampler as a means to observe the particle distribution at a point in the lattice. A sampler is ‘attached’ to an already defined element and records all the particles passing through a plane at the exit face of that element.

A sampler will record any particles passing through that plane in any direction. It is defined in reality by a box 5 x 5 m that is 1 nm thick. The user may consider it an infinitely thin plane.

### 6.14.1 Attaching a Sampler to a Beamline Element

They are defined using the following syntax:

```plaintext
sample, range=<element_name>;
```

where `element_name` is the name of the element you wish to sample. Depending on the output format chosen, the element name may be recorded in the output (‘rootevent’ output only).

**Note:** Samplers can only be defined after the main sequence has been defined using the `use` command (see use - Defining which Line to Use). Failure to do so will result in an error and BDSIM will exit.

**Note:** Samplers record all particles impinging on them (i.e. both forwards and backwards). Even secondary particles that may originate from further along the lattice are recorded. They have no material so they do not absorb or affect particles, only witness them.
To place a sampler before an item, attach it to the previous item. If however, you wish to record the coordinates with another name rather than the name of the element before, you can define a marker; place it in the sequence; and then define a sampler that uses that marker:

\[
\begin{align*}
  d1 & : \text{drift, } l=2.4\text{ m;} \\
  d2 & : \text{drift, } l=1\text{ m;} \\
  \text{interestingplane} & : \text{marker;} \\
  l1 & : \text{line=(d1,d1,interestingplane,d2,d1);} \\
  \text{use, period=l1;} \\
  \text{sample, range } = \text{interestingplane};
\end{align*}
\]

When an element is defined multiple times in the line (such as “d1” in the above example), samplers will be attached to all instances. If you wish to sample only one specific instance, the following syntax can be used:

\[
\text{sample, range=<element_name>[index];}
\]

To attach samplers after all elements:

\[
\text{sample, all;}
\]

And to attach samplers after all elements of a specific type:

\[
\text{sample, <type>};
\]

\[
\text{e.g.}
\]

\[
\text{sample, quadrupole;}
\]

**Note:** If a sampler is placed at the very beginning of the lattice, it may appear that only approximately half of the primary particles seem to pass through it. This is the correct behaviour, as unlike an optics program such as MAD-X, the sampler represents a thin plane in 3D space in BDSIM. If the beam distribution has some finite extent in \(z\) or \(t\), particles may start beyond this first sampler and never pass through it.

**Warning:** The record of the primary particle coordinates in the output (“Primary”) may resemble a sampler but it is just a record of the initial coordinates. It is not a sampler and cannot record other secondary particles.

### 6.14.2 Sampler Dimensions

The sampler is represented by a box solid that is 1 nm thick along \(z\) and 5m wide transversely in \(x\) and \(y\). If a smaller or larger capture area for the samplers is required, the option \textit{samplerDiameter} may be specified in the input \textit{gmad}.

\[
\text{option, samplerDiameter=3 m;}
\]

This affects all samplers.

**Note:** For a very low energy lattice with large angle bends, the default samplerDiameter may cause geometrical overlap warnings from Geant4. This situation is difficult to avoid automatically, but easy to remedy by setting the samplerDiameter to a lower value. We recommend reducing \textit{samplerDiameter} for low energy or strongly curving accelerators.
6.14.3 Sampler Visualisation

The samplers are normally invisible and are built in a parallel world geometry in Geant4. To visualise them, the following command should be used in the visualiser:

```
/vis/drawVolume worlds
```

The samplers will appear in semi-transparent green, as well as the curvilinear geometry used for coordinate transforms (cylinders).

6.14.4 Output at an Arbitrary Plane - User Placed Sampler

The user may place a sampler anywhere in the model with any orientation. This is called a samplerplacement. The sampler may have either a circular or rectangular (including square) shape and be placed with any orientation. A samplerplacement will record all particles travelling in any direction through it. A branch in the Event output will be create with the name of the samplerplacement. The user may define an arbitrary number of samplerplacement's. A 'samplerplacement' is defined with the following syntax:

```
s1: samplerplacement, referenceElement="d1", referenceElementNumber=1, x=20*cm, y=-1*cm, z=30*cm, axisAngle=1, axisY=1, angle=pi/4, aper1=10*cm;
```

This defines a circular (by default) sampler with radius 10 cm positioned with respect to the 2nd instance of the d1 element (zero counting) in the main beam line with a rotation about the unit Y axis of $\pi/4$.

Placement

A samplerplacement may be placed in 3 ways.

1) In global Cartesian coordinates.
2) In curvilinear coordinates.
3) In curvilinear coordinates with respect to a beam line element by name.

The strategy is automatically determined based on the parameters set. The full list of parameters is described below, but the required ones for each scenario are described in Placements.

**Warning:** This sampler can nominally overlap with any geometry. However, the user should avoid co-planar overlaps with other geometry. e.g. do not place one just at the end of an element or perfectly aligned with the face of an object. This will cause bad tracking and overlaps. This is a limitation of Geant4. The user placed samplers are slightly thicker than normal ones to help avoid this problem.

Shape

The sampler will be 1 nm thick in reality but may be treated by the user as an infinitely thin plane. It is composed of vacuum and should not interfere with the ongoing physics of the simulation. The user may select the shape of the sampler from either circular or rectangular (including square). The parameter apertureType should be specified as either "circular" or "rectangular". The aperture parameters typically used in BDSIM should also be used - these are aper1 and aper2. The meaning of these parameters is described in Aperture Parameters.
Parameters

The samplerplacement object accepts a number of parameters similar to other parts of BDSIM. These include a subset of the aperture parameters (see Aperture Parameters) and the position information from the placements. The full list of accepted parameters is given below.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>Offset in global x</td>
</tr>
<tr>
<td>y</td>
<td>Offset in global y</td>
</tr>
<tr>
<td>z</td>
<td>Offset in global z</td>
</tr>
<tr>
<td>s</td>
<td>Curvilinear s coordinate (global</td>
</tr>
<tr>
<td>phi</td>
<td>Euler angle phi for rotation</td>
</tr>
<tr>
<td>theta</td>
<td>Euler angle theta for rotation</td>
</tr>
<tr>
<td>psi</td>
<td>Euler angle psi for rotation</td>
</tr>
<tr>
<td>axisX</td>
<td>Axis angle rotation x-component of unit vector</td>
</tr>
<tr>
<td>axisY</td>
<td>Axis angle rotation y-component of unit vector</td>
</tr>
<tr>
<td>axisZ</td>
<td>Axis angle rotation z-component of unit vector</td>
</tr>
<tr>
<td>angle</td>
<td>Axis angle, angle to rotate about unit vector</td>
</tr>
<tr>
<td>axisAngle</td>
<td>Boolean whether to use axis angle rotation scheme (default false)</td>
</tr>
<tr>
<td>sensitive</td>
<td>Whether the geometry records energy deposition (default true)</td>
</tr>
<tr>
<td>referenceElement</td>
<td>Name of element to place geometry with respect to (string)</td>
</tr>
<tr>
<td>referenceElementNumber</td>
<td>Occurence of referenceElement to place with respect to if it is used more than once in the sequence. Zero counting.</td>
</tr>
<tr>
<td>apertureType</td>
<td>The shape of the sampler desired as described using the aperture syntax of BDSIM. Currently, only circular and rectangular are supported.</td>
</tr>
<tr>
<td>shape</td>
<td>An intuitive alias to apertureType.</td>
</tr>
<tr>
<td>aper1</td>
<td>Aperture parameter #1.</td>
</tr>
<tr>
<td>aper2</td>
<td>Aperture parameter #2.</td>
</tr>
<tr>
<td>aper3</td>
<td>Aperture parameter #3.</td>
</tr>
<tr>
<td>aper4</td>
<td>Aperture parameter #4.</td>
</tr>
</tbody>
</table>

Examples

The following are examples of samplerplacement:

```
s1: samplerplacement, referenceElement="d1", referenceElementNumber=1, x=20*cm, y=-1*cm, z=30*cm, axisAngle=1, axisY=1, angle=pi/4, aper1=10*cm;
```

This places a circular sampler called “s1” with respect to the 2nd instance of the beam line element “d1”. The x,y,z are offsets from the centre of this element along the direction of travel of the beam. The sampler is rotated about the unit Y axis (again with respect to the centre of the beam line element rotation) by an angle of π/4. The sampler will be circular (by default) with a radius of 10 cm.

```
s2: samplerplacement, x=0.2*m, y=-1*cm, z=30.123*m, axisAngle=1, axisY=1, angle=-pi/6, aper1=10*cm, aper2=5*cm, shape="rectangular";
```

This will place a sampler called “s2” in global Cartesian coordinates approximately 30 m forward from the centre of model. The placement rotation is done in the global coordinate system. The sampler shape is rectangular and is 20 cm wide and 10 cm tall.
User Sampler Visualisation

Samplers are by default invisible. To visualise the sampler placement, all samplers should be visualised as described in Sampler Visualisation. The scene tree can then be explored in the visualiser to hide other hidden volumes (such as the ‘curvilinear’ coordinate transform worlds) and other samplers. It is recommended to tick and un-tick the desired element to see it appear and disappear repeatedly.

6.15 Physics Processes

BDSIM can exploit all the physics processes that come with Geant4. It is advantageous to define only the processes required so that the simulation covers the desired outcome but is also efficient. Geant4 says, “There is no one model that covers all physics at all energy ranges.”

By default, only tracking in magnetic fields is provided (e.g. no physics) and other processes must be specified to be used.

Rather than specify each individual particle physics process on a per-particle basis, a series of “physics lists” are provided that are a predetermined set of physics processes suitable for a certain application. BDSIM follows the Geant4 ethos in this regard and the majority of those in BDSIM are simple shortcuts to the Geant4 ones.

There are 3 ways to specify physics lists in BDSIM:

1) BDSIM’s modular physics lists as described in Modular Physics Lists:

```python
option, physicsList = "em qgsp_bert"
```

These are modular and can be added independently. BDSIM provides a ‘physics list’ for a few discrete processes that aren’t covered inside Geant4 reference physics lists such as crystal channelling and cherenkov radiation. It is possible to create a physics list similar to a Geant4 reference physics list using BDSIM’s modular approach as internally Geant4 does the same thing.

2) Geant4’s reference physics lists as described in Geant4 Reference Physics Lists:

```python
option, physicsList = "g4FTFP_BERT"
```

These are more complete “reference physics lists” that use several modular physics lists from Geant4 like BD-SIM but in a predefined way that Geant4 quote for references results. These have rather confusingly similar names. ftfp_bert causes BDSIM to use G4HadronPhysicsFTFP_BERT whereas g4FTFP_BERT uses FTFP_BERT in Geant4. We refer the pattern 1) as ‘modular physics lists’ and pattern 2) as Geant4 reference physics lists.

3) A complete physics list. This is a custom solution for a particular application that is hard coded in BDSIM. These all start with ‘complete’. See Complete Physics Lists.

```python
option, physicsList = "completechannelling"
```

For general high energy hadron physics we recommend:

```python
option, physicsList = "em ftfp_bert decay muon hadronic_elastic em_extra"
```

Some physics lists are only available in later versions of Geant4. These are filtered at compile time for BDSIM and it will not recognise a physics list that requires a later version of Geant4 than BDSIM was compiled with respect to.

A summary of the available physics lists in BDSIM is provided below (others can be easily added by contacting the developers - see Feature Request).

See the Geant4 documentation for a more complete explanation of the physics lists.

- Physics List Guide
- User Case Guide
6.15.1 Modular Physics Lists

A modular physics list can be made by specifying several physics lists separated by spaces. These are independent.

- The strings for the modular physics list are case-insensitive.

Examples:

```plaintext
option, physicsList="em ftfp_bert";
option, physicsList="em_low decay ion hadron_elastic qgsp_bert em_extra;"
```

**Warning:** Not all physics lists can be used with all other physics lists. BDSIM will print a warning and exit if this is the case. Generally, lists suffixed with “hp” should not be used along with the unsuffixed ones (e.g. “qgsp_bert” and “qgsp_bert_hp” should not be used together). Similarly, the standard electromagnetic variants should not be used with the regular “em”.

<table>
<thead>
<tr>
<th>String to use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charge_exchange</td>
<td>Transportation of primary particles only - no scattering in material</td>
</tr>
<tr>
<td>channelling</td>
<td>G4ChargeExchangePhysics</td>
</tr>
<tr>
<td>cherenkov</td>
<td>Provides Cherenkov radiation for all charged particles. Issued by the BDSIM physics builder BDSPhysicsCherenkov that provides the process G4CherenkovProcess.</td>
</tr>
<tr>
<td>decay</td>
<td>Provides radioactive decay processes using G4DecayPhysics. Crucial for pion decay for example.</td>
</tr>
<tr>
<td>decay_radioactive</td>
<td>Radioactive decay of long-lived nuclei. Uses G4RadioactiveDecayPhysics.</td>
</tr>
<tr>
<td>dna</td>
<td>G4EmDNAPhysics list. Only applies to G4_WATER material.</td>
</tr>
<tr>
<td>dna_1</td>
<td>Variant 1 of G4EmDNAPhysics list. Uses G4EmDNAPhysics_option1.</td>
</tr>
<tr>
<td>dna_X</td>
<td>Variant X of G4EmDNAPhysics list, where X is one of 1,2,3,4,5,6,7.</td>
</tr>
<tr>
<td>em_extra</td>
<td>G4EmPhysicsExtra. Responds to the option useLENGaammaNuclear that requires the G4LENDDATA environmental variable to be set for the optional LEND data set (see ** below). Additional options described below also allow different parts of this model to be turned on or off.</td>
</tr>
<tr>
<td>em_gs</td>
<td>G4EmStandardPhysicsGS. Available from Geant4.10.2 onwards.</td>
</tr>
<tr>
<td>em_livermore</td>
<td>G4EmLivermorePhysics</td>
</tr>
<tr>
<td>em_livermore_polarised</td>
<td>G4EmLivermorePolarizedPhysics</td>
</tr>
<tr>
<td>em_low_ep</td>
<td>G4EmLowEPPhysics</td>
</tr>
<tr>
<td>em_penelope</td>
<td>The same as em, but using low-energy electromagnetic models. Uses G4EmPenelopePhysics</td>
</tr>
<tr>
<td>em_ss</td>
<td>G4EmStandardPhysicsSS</td>
</tr>
<tr>
<td>em_wvi</td>
<td>G4EmStandardPhysicsWVI</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>String to use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>em_1</td>
<td>G4EmStandardPhysics_option1</td>
</tr>
<tr>
<td>em_2</td>
<td>G4EmStandardPhysics_option2</td>
</tr>
<tr>
<td>em_3</td>
<td>G4EmStandardPhysics_option3</td>
</tr>
<tr>
<td>em_4</td>
<td>G4EmStandardPhysics_option4</td>
</tr>
<tr>
<td>fftp_bert</td>
<td>Fritiof Precompound Model with Bertini Cascade Model. The FTF model is based on the FRITIOF description of string excitation and fragmentation. This is provided by G4HadronPhysicsFTFP_BERT. All FTF physics lists require G4HadronElasticPhysics to work correctly.</td>
</tr>
<tr>
<td>fftp_bert_hp</td>
<td>Similar to FTFP_BERT, but with the high precision neutron package. This is provided by G4HadronPhysicsFTFP_BERT_HP.</td>
</tr>
<tr>
<td>hadronic_elastic</td>
<td>Elastic hadronic processes. This is provided by G4HadronElasticPhysics.</td>
</tr>
<tr>
<td>hadronic_elastic_d</td>
<td>G4HadronDElasticPhysics</td>
</tr>
<tr>
<td>hadronic_elastic_h</td>
<td>G4HadronHElasticPhysics</td>
</tr>
<tr>
<td>hadronic_elastic_hp</td>
<td>G4HadronElasticPhysicsHP</td>
</tr>
<tr>
<td>hadronic_elastic_lend **</td>
<td>G4HadronElasticPhysicsLEND</td>
</tr>
<tr>
<td>hadronic_elastic_xs</td>
<td>G4HadronElasticPhysicsXS</td>
</tr>
<tr>
<td>ion</td>
<td>G4IonElasticPhysics</td>
</tr>
<tr>
<td>ion_binary (*)</td>
<td>G4IonBinaryCascadePhysics</td>
</tr>
<tr>
<td>ion_elastic</td>
<td>G4IonElasticPhysics</td>
</tr>
<tr>
<td>ion_elastic_qmd</td>
<td>G4IonQMDPhysics</td>
</tr>
<tr>
<td>ion_inclxx (*)</td>
<td>G4IonINCLXXPhysics</td>
</tr>
<tr>
<td>ion_php (*)</td>
<td>G4IonPhysicsPHP. Available from Geant4.10.3 onwards.</td>
</tr>
<tr>
<td>lw</td>
<td>Laserwire photon producing process as if the laserwire had scattered photons from the beam. Not actively developed, but will register process.</td>
</tr>
<tr>
<td>muon</td>
<td>Provides muon production and scattering processes. Gamma to muons, annihilation to muon pair, ‘ee’ to hadrons, pion decay to muons, multiple scattering for muons, muon Bremsstrahlung, pair production and Cherenkov light are all provided. Given by BDSIM physics builder (a la Geant4) BDSPhysicsMuon.</td>
</tr>
<tr>
<td>neutron_tracking_cut</td>
<td>G4NeutronTrackingCut allows neutrons to be killed via their tracking time (i.e. time of flight) and minimum kinetic energy. These options are set via the option command, neutronTimeLimit (s) and neutronKineticEnergyLimit (GeV).</td>
</tr>
<tr>
<td>optical</td>
<td>Optical physics processes including absorption, Rayleigh scattering, Mie scattering, optical boundary processes, scintillation and Cherenkov. This uses G4OpticalPhysics class.</td>
</tr>
<tr>
<td>qgsp_bert</td>
<td>Quark-Gluon String Precompound Model with Bertini Cascade model. This is based on the G4HadronPhysicsQGSP_BERT class and includes hadronic elastic and inelastic processes. Suitable for high energy (&gt;10 GeV).</td>
</tr>
<tr>
<td>qgsp_bert_hp</td>
<td>Similar to QGSP_BERT, but with the addition of data-driven high precision neutron models to transport neutrons below 20 MeV down to thermal energies. This is provided by G4HadronPhysicsQGSP_BERT_HP.</td>
</tr>
<tr>
<td>qgsp_bic</td>
<td>Like QGSP, but using Geant4 Binary cascade for primary protons and neutrons with energies below ~10GeV, thus replacing the use of the LEP model for protons and neutrons. In comparison to the LEP model, Binary cascade better describes production of secondary particles produced from interactions of protons and neutrons with nuclei. This is provided by G4HadronPhysicsQGSP_BIC.</td>
</tr>
</tbody>
</table>
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>String to use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>qgsp_bic_hp</td>
<td>Similar to QGSP_BIC, but with the high precision neutron package. This is provided by G4HadronPhysicsQGSP_BIC_HP.</td>
</tr>
<tr>
<td>shielding</td>
<td>G4HadronPhysicsShielding. Inelastic hadron physics suitable for shielding applications.</td>
</tr>
<tr>
<td>shielding_lend(**)</td>
<td>G4HadronPhysicsShieldingLEND. Similar to shielding, but requires LEND data set for low-energy neutrons. Available from Geant4.10.4 onwards.</td>
</tr>
<tr>
<td>stopping</td>
<td>G4StoppingPhysics. Hadronic physics for stopping particles.</td>
</tr>
<tr>
<td>synch_rad</td>
<td>Provides synchrotron radiation for all charged particles. Provided by BDSIM physics builder BDSPhysicsSynchRad that provides the process G4SynchrotronRadiation.</td>
</tr>
</tbody>
</table>

The following are also accepted as aliases to current physics lists. These are typically previously used names.

<table>
<thead>
<tr>
<th>Physics List</th>
<th>Alias To</th>
</tr>
</thead>
<tbody>
<tr>
<td>cerenkov</td>
<td>cerenkov</td>
</tr>
<tr>
<td>em_low</td>
<td>em_penelope</td>
</tr>
<tr>
<td>hadronic</td>
<td>ftfp_bert</td>
</tr>
<tr>
<td>hadronic_hp</td>
<td>ftfp_bert_hp</td>
</tr>
<tr>
<td>ionbinary</td>
<td>ion_binary</td>
</tr>
<tr>
<td>ioninclxx</td>
<td>ion_inclxx</td>
</tr>
<tr>
<td>ionphp</td>
<td>ion_php</td>
</tr>
<tr>
<td>spindecay</td>
<td>decay_spin</td>
</tr>
<tr>
<td>synchrad</td>
<td>synch_rad</td>
</tr>
</tbody>
</table>

Warning: (*) These physics lists require the optional high-precision data from Geant4. The user should download this data from the Geant4 website and install it (for example: extract to <install-dir>/share/Geant4-10.3.3/data/ beside the other data) and export the environmental variable G4PARTICLEHPDATA to point to this directory.

Warning: (**) These physics lists require the optional LEND data set that can be downloaded from the Geant4 website. It should be extracted and the environmental variable G4LENDDATA set to the directory containing it.

6.15.2 em_extra Physics Notes

The em_extra model is an interface to G4EmExtraPhysics that collects a variety of extra electromagnetic models together. Not all of these are activated by default. BDSIM provides options to turn these components on and off. See Physics Processes for more details on the specific options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Minimum Geant4 Version</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>useLENDGammaNuclear</td>
<td>10.4</td>
<td>Off</td>
</tr>
<tr>
<td>useElectroNuclear</td>
<td>10.4</td>
<td>On</td>
</tr>
<tr>
<td>useMuonNuclear</td>
<td>10.2</td>
<td>On</td>
</tr>
<tr>
<td>useGammaToMuMu</td>
<td>10.3</td>
<td>Off</td>
</tr>
<tr>
<td>usePositronToMuMu</td>
<td>10.3</td>
<td>Off</td>
</tr>
<tr>
<td>usePositronToHadrons</td>
<td>10.3</td>
<td>Off</td>
</tr>
</tbody>
</table>

Example:
option, physicsList="em em_extra",
    useMuonNuclear=1,
    useGammaToMuMu=1;

The options will always be accepted by BDSIM if an earlier version of Geant4 is used than supported, however, these will have no effect.

*G4EmExtraPhysics* provides a simple interface to increase the cross-section of some processes. This interface is not used in BDSIM, as it does not propagate the associated weights correctly. Biasing should be done through the generic biasing interface with the name of the process (described in the following section), as this will propagate the weights correctly.

### 6.15.3 Geant4 Reference Physics Lists

BDSIM allows use of the Geant4 reference physics lists directly and more details can be found in the Geant4 documentation:

- Physics List Guide
- User Case Guide

**Notes:**

- Only one Geant4 reference physics list can be used and it cannot be used in combination with any modular physics list.
- The range cuts specified with BDSIM options apply by default and the option `g4PhysicsUseBDSIMRangeCuts` should be set to 0 (‘off’) to avoid this if required. The defaults are 1 mm, the same as Geant4.
- If the option `minimumKineticEnergy` is set to a value greater than 0 (the default), a physics process will be attached to the Geant4 reference physics list to enforce this cut. This must be 0 and `g4PhysicsUseBDSIMCutsAndLimits` option off to not use the physics process to enforce cuts and limits and therefore achieve the exact reference physics list. This is the default option.

**Warning:** Turning off all limits may result in tracking warnings. The events should still proceed as normal, but Geant4 by default requests step lengths of 10 km or more, which often break the validity of the accelerator tracking routines. This is unavoidable, hence why we use the limits by default. BDSIM, by default applies step length limits of 110% the length of each volume.

The following reference physics lists are included as of Geant4.10.4.p02. These must be prefix with “g4” to work in BDSIM.

- FTFP_BERT
- FTFP_BERT_TRV
- FTFP_BERT_ATL
- FTFP_BERT_HP
- FTFQGSP_BERT
- FTFP_INCLXX
- FTFP_INCLXX_HP
- FTF_BIC
- LBE
- QBBC
- QGSP_BERT
• QGSP_BERT_HP
• QGSP_BIC
• QGSP_BIC_HP
• QGSP_BIC_AllHP
• QGSP_FTFP_BERT
• QGSP_INCLXX
• QGSP_INCLXX_HP
• QGS_BIC
• Shielding
• ShieldingLEND
• ShieldingM
• NuBeam

The **optional** following suffixes may be added to specify the electromagnetic physics variant:

• _EMV
• _EMX
• _EMY
• _EMZ
• _LIV
• _PEN
• __GS
• __LE
• __WVI
• __SS

**Examples:**

```
option, physicsList="g4QBBC";

option, physicsList="g4QBBC_EMV";

option, physicsList="g4FTFP_BERT_PEN",
   g4PhysicsUSEBSDIMCutsAndLimits=0;
```

This last example turns off the minimum kinetic energy and also the maximum step length limit which is by default 110% the length of the element. If bad tracking behaviour is experienced (stuck particles etc.) this should be considered.

```
option, physicsList="g4FTFP_BERT";
```

This following example will enforce a minimum kinetic energy but also limit the maximum step length (consequently) to 110% the length of the component and provide more robust tracking.

```
option, physicsList="g4FTFP_BERT",
   minimumKineticEnergy=20*GeV;
```

**Note:** “g4” is not case sensitive but the remainder of the string is. The remainder is passed to the Geant4 physics list that constructs the appropriate physics list and this is case sensitive.
6.15.4 Complete Physics Lists

These are complete physics lists provided for specialist applications. Currently, only one is provided for crystal channelling physics. These all begin with “complete”.

These cannot be used in combination with any other physics processes.

<table>
<thead>
<tr>
<th>Physics List</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>completechannelling</td>
<td>Modified em option 4 plus channelling as per the Geant4 example for crystal channelling. The exact same physics as used in their example.</td>
</tr>
</tbody>
</table>

**Note:** The range cuts specified with BDSIM options to not apply and cannot be used with a ‘complete’ physics list.

6.16 Physics Biasing

BDSIM currently provides two ways to artificially interfere with the physics processes to make the desired outcome happen more often. In both cases, the goal is to simulate the correct physical outcome, but more efficiently in the parameters of interest, i.e. variance reduction.

The two cases are **Cross-Section Biasing** and **Geometric Importance Sampling**, each described below.

6.16.1 Cross-Section Biasing

The cross-section for a physics process for a specific particle can be artificially altered by a numerical scaling factor using cross-section biasing. This is done on a per-particle and per-physics-process basis. The biasing is defined with the keyword `xsecbias`, to define a bias ‘object’. This can then be attached to various bits of the geometry or all of it. This is provided with the Geant4 generic biasing feature.

Geant4 automatically includes the reciprocal of the factor as a weighting, which is recorded in the BDSIM output as “weight” in each relevant piece of data. Any data used should be multiplied by the weight to achieve the correct physical result.

**Note:** This only works with Geant4 version 10.1 or higher. It does not work Geant4.10.3.X series.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Biasing process name</td>
</tr>
<tr>
<td>particle</td>
<td>Particle that will be biased</td>
</tr>
<tr>
<td>proc</td>
<td>Process(es) to be biased</td>
</tr>
<tr>
<td>xsecfact</td>
<td>Biasing factor(s) for the process(es)</td>
</tr>
<tr>
<td>flag</td>
<td>Flag which particles are biased for the process(es) (1=all, 2=primaries, 3=secondaries)</td>
</tr>
</tbody>
</table>

- Particle names should be exactly as they are in Geant4 (case-sensitive). The best way to find these out is to run a single event with the desired physics list. The physics list print out will name particles used.
- The process name should be exactly as they are in Geant4 (case-sensitive). Similarly, the best way to find these names is to run a single event with the desired physics list.
- A special particle name ‘all’ will bias all defined particles. (case-sensitive).
- In the case of an ion beam, the particle name should be “GenericIon”. The biasing will apply to all ions, so the flag should be used to select primary or secondary or all particles. This is because Geant4 uses the concept of a generic ion as there are so many possible ions.
• Examples can be found in bdsim/examples/features/processes/5_biasing.

Example:

```
biasDef1: xsecBias, particle="e-", proc="all", xsecfact=10, flag=3;
biasDef2: xsecBias, particle="e+", proc="eBrem eIoni msc", xsecfact={10,1,5}, flag=→{1,1,2};
```

The process can also be attached to a specific element using the keywords `biasVacuum` or `biasMaterial` for the biasing to be attached the vacuum volume or everything outside the vacuum respectively:

```
q1: quadrupole, l=1*m, material="Iron", biasVacuum="biasDef1 biasDef2"; // uses the →process biasDef1 and biasDef2
q2: quadrupole, l=0.5*m, biasMaterial="biasDef2";
```

### 6.16.2 Geometric Importance Sampling

To enable importance sampling, the user must provide both a mass world and a separate importance sampling world as external geometry files. The mass world file should contain the appropriate volumes as if you were conducting a standard simulation without importance sampling. The importance world file should contain the volumes that will be the importance cells only. A third text file must also be provided which contains a map of the physical volumes that form the importance cells and their corresponding importance volumes.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>worldGeometryFile</td>
<td>Geometry file containing the mass world</td>
</tr>
<tr>
<td>importanceWorldGeometryFile</td>
<td>Geometry file containing the importance sampling world</td>
</tr>
<tr>
<td>importanceVolumeMap</td>
<td>ASCII file containing a map of the importance world physical volumes and their corresponding importance values</td>
</tr>
</tbody>
</table>

Example:

```
option, worldGeometryFile="gdml:shielding-world.gdml",
    importanceWorldGeometryFile="gdml:importance-cell-world.gdml",
    importanceVolumeMap="importanceValues.dat";
```

An example of the world volume geometry (top), the importance sampling world geometry (middle), and an importance volume map (bottom) are shown below with an example beamline.

In the output a new branch in the event tree calls “ElossWorldContents” is automatically added when using importance sampling. This is the global energy deposition hits from any volumes that were in the externally supplied world - such as shielding blocks. This distinguishes the energy deposition in the world volume itself (i.e. the air).

• Both the mass world and importance sampling world must be the same size.
• Both the mass world and importance sampling world must be large enough to encompass the machine beamline. If not, BDSIM will exit.
• It is down to the user to ensure the importance cells are correctly positioned.
• If a importance cell volume exists in the importance world geometry and is not listed in the ASCII map file with a importance value, BDSIM will exit.
• The importance sampling world volume has an importance value of 1.

### 6.17 Options

Various simulation details can be controlled through the `option` command. Options are defined using the following syntax:
option, <option_name>=<value>;

Values accepted can be a number (integer, floating point or scientific notation), a string with the value enclosed in "double inverted commas", or a Boolean. For Boolean options (described as on or off, or true or false) a number 1 or 0 is used.

Multiple options can be defined at once using the following syntax:

option, <option1> = <value>,
       <option2> = <value>;

**Note:** No options are required to be specified to run a BDSIM model. Defaults will be used in all cases. However, we do recommend you select an appropriate physics list and beam pipe radius, as these will have a large impact on the outcome of the simulation.

Below is a full list of all options in BDSIM. Please also see *Executable Options* for options that are used on the command line when executing BDSIM. The executable options override whatever options are specified in the input gmad files.

Common options are duplicated below for convenience as these are the most useful ones. All options are described in the following sub-sections:

- General Run Options
- Geometry Options
- Tracking Options
- Physics Processes
- Visualisation
- Output Options
- One Turn Map
- Offset for Main Beam Line
- Scoring Map
- Developer Options
6.17.1 Common Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>beampipeRadius</td>
<td>Default beam pipe inner radius [m]</td>
</tr>
<tr>
<td>beampipeThickness</td>
<td>Default beam pipe thickness [m]</td>
</tr>
<tr>
<td>beampipeMaterial</td>
<td>Default beam pipe material</td>
</tr>
<tr>
<td>elossHistoBinWidth</td>
<td>The width of the default energy deposition and particle loss histogram</td>
</tr>
<tr>
<td></td>
<td>bins made as BDSIM runs [m] Default 1.0 m</td>
</tr>
<tr>
<td>eventNumberOffset</td>
<td>Event that the recreation should start from</td>
</tr>
<tr>
<td>hStyle</td>
<td>Whether default dipole style is H-style vs. C-style (default false)</td>
</tr>
<tr>
<td>ngenerate</td>
<td>Number of primary particles to simulate</td>
</tr>
<tr>
<td>nturns</td>
<td>The number of revolutions that the particles are allowed to complete in</td>
</tr>
<tr>
<td></td>
<td>a circular accelerator. Requires –circular executable option to work.</td>
</tr>
<tr>
<td>horizontalWidth</td>
<td>Default accelerator component full width [m] Default is 0.5 m.</td>
</tr>
<tr>
<td>physicsList</td>
<td>The physics list to use</td>
</tr>
<tr>
<td>printFractionEvents</td>
<td>How often to print out the event number as a fraction of the total number</td>
</tr>
<tr>
<td></td>
<td>of events to simulation (default is 0.1 i.e. 10%). Varies from 0 to 1.</td>
</tr>
<tr>
<td></td>
<td>-1 for all.</td>
</tr>
<tr>
<td>printFractionTurns</td>
<td>How often to print out the turn number as a fraction of the total number</td>
</tr>
<tr>
<td></td>
<td>of turns to simulation (default is 0.2 i.e. 20%). Varies from 0 to 1. -1</td>
</tr>
<tr>
<td></td>
<td>for all. Will only print out in an event that also prints out.</td>
</tr>
<tr>
<td>prodCutPhotons</td>
<td>Standard overall production cuts for photons (default 1e-3) [m]</td>
</tr>
<tr>
<td>prodCutElectrons</td>
<td>Standard overall production cuts for electrons (default 1e-3) [m]</td>
</tr>
<tr>
<td>prodCutPositrons</td>
<td>Standard overall production cuts for positrons (default 1e-3) [m]</td>
</tr>
<tr>
<td>prodCutProtons</td>
<td>Standard overall production cuts for protons (default 1e-3) [m]</td>
</tr>
<tr>
<td>removeTemporaryFiles</td>
<td>Whether to delete temporary files (typically gdml) when BDSIM exits.</td>
</tr>
<tr>
<td></td>
<td>Default true.</td>
</tr>
<tr>
<td>seed</td>
<td>The integer seed value for the random number generator</td>
</tr>
<tr>
<td>stopSecondaries</td>
<td>Whether to stop secondaries or not (default = false)</td>
</tr>
<tr>
<td>worldMaterial</td>
<td>The default material surrounding the model. This is by default air.</td>
</tr>
</tbody>
</table>

6.17.2 General Run Options

For a description of recreating events, see [Recreate Mode](#).

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ngenerate</td>
<td>Number of primary particles to simulate</td>
</tr>
<tr>
<td>nturns</td>
<td>The number of revolutions particles are allowed to complete in a circular</td>
</tr>
<tr>
<td></td>
<td>accelerator - requires –circular executable option to work.</td>
</tr>
<tr>
<td>printFractionEvents</td>
<td>How often to print out the event number as a fraction of the total number</td>
</tr>
<tr>
<td></td>
<td>of events to simulation (default is 0.1 i.e. 10%). Varies from 0 to 1.</td>
</tr>
<tr>
<td></td>
<td>-1 for all.</td>
</tr>
<tr>
<td>printFractionTurns</td>
<td>How often to print out the turn number as a fraction of the total number</td>
</tr>
<tr>
<td></td>
<td>of turns to simulation (default is 0.2 i.e. 20%). Varies from 0 to 1. -1</td>
</tr>
<tr>
<td></td>
<td>for all. Will only print out in an event that also prints out.</td>
</tr>
<tr>
<td>recreate</td>
<td>Whether to use recreation mode or not (default 0). If used as an executable</td>
</tr>
<tr>
<td></td>
<td>option, this should be a string with a path to the recreateFileName.</td>
</tr>
<tr>
<td>recreateFileName</td>
<td>Path to BDSIM output file to use for recreation.</td>
</tr>
<tr>
<td>removeTemporaryFiles</td>
<td>Whether to delete temporary files (typically gdml) when BDSIM exits.</td>
</tr>
<tr>
<td></td>
<td>Default true.</td>
</tr>
<tr>
<td>seed</td>
<td>The integer seed value for the random number generator</td>
</tr>
<tr>
<td>startFromEvent</td>
<td>Number of event to start from when recreating. 0 counting.</td>
</tr>
<tr>
<td>writeSeedState</td>
<td>Writes the seed state of the last event start in ASCII</td>
</tr>
</tbody>
</table>
6.17.3 Geometry Options

These affect the construction of the 3D model in BDSIM. Tunnel parameters are also described in *Tunnel Geometry*.

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>aper1</td>
<td>Default aper1 parameter</td>
</tr>
<tr>
<td>aper2</td>
<td>Default aper2 parameter</td>
</tr>
<tr>
<td>aper3</td>
<td>Default aper3 parameter</td>
</tr>
<tr>
<td>aper4</td>
<td>Default aper4 parameter</td>
</tr>
<tr>
<td>beampipeRadius</td>
<td>Default beam pipe inner radius - alias for aper1 [m]</td>
</tr>
<tr>
<td>beampipeThickness</td>
<td>Default beam pipe thickness [m]</td>
</tr>
<tr>
<td>beampipeMaterial</td>
<td>Default beam pipe material</td>
</tr>
<tr>
<td>buildTunnel</td>
<td>Whether to build a tunnel (default = 0)</td>
</tr>
<tr>
<td>buildTunnelStraight</td>
<td>Whether to build a tunnel, ignoring the beamline and just in a straight line (default = 0).</td>
</tr>
<tr>
<td>buildTunnelFloor</td>
<td>Whether to add a floor to the tunnel</td>
</tr>
<tr>
<td>checkOverlaps</td>
<td>Whether to run Geant4’s geometry overlap checker during geometry construction (slower)</td>
</tr>
<tr>
<td>coilWidthFraction</td>
<td>0.05 - 0.98 - fraction of available horizontal space between pole and yoke that coil will occupy</td>
</tr>
<tr>
<td>coilHeightFraction</td>
<td>0.05 - 0.98 - fraction of available vertical space between pole tip and yoke that coil will occupy</td>
</tr>
<tr>
<td>dontSplitSBends</td>
<td>If true, do not split sbends into multiple segments (default = false).</td>
</tr>
<tr>
<td>emptyMaterial</td>
<td>This is ‘vacuum’ material where no material is required. Note: this is not ‘vacuum’ in the beam pipe, but the lowest density material (G4_GALACTIC) Geant4 can muster, as all materials must have a finite density. This is used for the gap between tight-fitting container volumes and objects.</td>
</tr>
<tr>
<td>horizontalWidth</td>
<td>The default full width of a magnet</td>
</tr>
<tr>
<td>hStyle</td>
<td>Whether default dipole style is H-style vs. C-style (default false)</td>
</tr>
<tr>
<td>ignoreLocalAperture</td>
<td>If this is true (1), any per-element aperture definitions will be ignored and the ones specified in Options will be used.</td>
</tr>
<tr>
<td>ignoreLocalMagnetGeometry</td>
<td>If this is true (1), any per-element magnet geometry definitions will be ignored and the ones specified in Options will be used.</td>
</tr>
<tr>
<td>includeIronMagFields</td>
<td>Whether to include magnetic fields in the magnet poles</td>
</tr>
<tr>
<td>magnetGeometryType</td>
<td>The default magnet geometry style to use</td>
</tr>
<tr>
<td>outerMaterial</td>
<td>The default material to use for the yoke of magnet geometry</td>
</tr>
<tr>
<td>preprocessGDML</td>
<td>Whether to prepend the element name at the front of every tag in a temporary copy of the GDML file. loaded. This is to compensate for the Geant4 GDML loader that cannot load multiple files correctly. On by default.</td>
</tr>
<tr>
<td>removeTemporaryFiles</td>
<td>Whether to delete temporary files (typically gdml) when BDSIM exits. Default true.</td>
</tr>
<tr>
<td>samplerDiameter</td>
<td>Diameter of samplers (default 5 m) [m]. This is also the diameter of the curvilinear world volumes used in curvilinear transforms. In the case of lower energy machines with strong bending angles (10s of degrees), this should be reduced to prevent overlaps between curvilinear volumes along the beam line.</td>
</tr>
<tr>
<td>sensitiveBeamPipe</td>
<td>Whether the beam pipe records energy loss. This includes cavities.</td>
</tr>
<tr>
<td>sensitiveOuter</td>
<td>Whether the outer part of each component (other than the beam pipe records energy loss</td>
</tr>
<tr>
<td>sensitiveVacuum</td>
<td>Whether energy deposition in the residual vacuum gas is recorded.</td>
</tr>
<tr>
<td>soilMaterial</td>
<td>Material for soil outside tunnel wall</td>
</tr>
<tr>
<td>thinElementLength</td>
<td>The length of all thummultipoles and dipole fringefields in a lattice (default 1e-6) [m]</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>tunnelIsInfiniteAbsorber</td>
<td>Whether all particles entering the tunnel material should be killed or not (default = false)</td>
</tr>
<tr>
<td>tunnelType</td>
<td>Which style of tunnel to use - one of: circular, elliptical, square, rectangular (more to come in v0.9)</td>
</tr>
<tr>
<td>tunnelAper1</td>
<td>Tunnel aperture parameter #1 - typically horizontal [m]</td>
</tr>
<tr>
<td>tunnelAper2</td>
<td>Tunnel aperture parameter #2 - typically vertical [m]</td>
</tr>
<tr>
<td>tunnelFloorOffset</td>
<td>The offset of the tunnel floor from the centre of the tunnel (not the beam line)</td>
</tr>
<tr>
<td>tunnelMaterial</td>
<td>Material for tunnel wall</td>
</tr>
<tr>
<td>tunnelOffsetX</td>
<td>Horizontal offset of the tunnel with respect to the beam line reference trajectory</td>
</tr>
<tr>
<td>tunnelOffsetY</td>
<td>Vertical offset of the tunnel with respect to the beam line reference trajectory</td>
</tr>
<tr>
<td>tunnelSoilThickness</td>
<td>Soil thickness outside tunnel wall [m]</td>
</tr>
<tr>
<td>tunnelThickness</td>
<td>Thickness of tunnel wall [m]</td>
</tr>
<tr>
<td>vacuumMaterial</td>
<td>The material to use for the beam pipe vacuum (default = “Vacuum”)</td>
</tr>
<tr>
<td>vacuumPressure</td>
<td>The pressure of the vacuum gas [bar]</td>
</tr>
<tr>
<td>vhRatio</td>
<td>Default vertical to horizontal ratio for dipoles</td>
</tr>
<tr>
<td>worldVolumeMargin</td>
<td>The margin added in all directions to the world volume [m]. Default 5m, minimum 2m.</td>
</tr>
<tr>
<td>worldMaterial</td>
<td>The default material surrounding the model. This is by default air.</td>
</tr>
<tr>
<td>worldGeometryFile</td>
<td>The filename of the world geometry file. See External World Geometry for more details. Default = “”.</td>
</tr>
<tr>
<td>yokeFields</td>
<td>Whether to include a general multipolar field for the yoke of each magnet (using a fourth order Runge-Kutta integrator). Default true.</td>
</tr>
</tbody>
</table>

### 6.17.4 Tracking Options

These control over the tracking routines used, as well as roughly the speed of the simulation with various options. Tracking integrator sets are described in detail in *Integrator Sets and Integrator Algorithms*. 
<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Function</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>collimatorsAreInfiniteAbsorbers</td>
<td>When turned on, all particles that enter the material of a collimator (rcol, ecol and jcol) are killed and the energy recorded as deposited there.</td>
</tr>
<tr>
<td>includeFringeFields</td>
<td>Places thin fringe field elements on the end of bending magnets with finite pole face angles. The length of the total element is conserved. (default = false).</td>
</tr>
<tr>
<td>integratorSet</td>
<td>Set of tracking routines to use (“bdsimmatrix”, “bdsimtwo”, “bdsimmatrrixfringescaling”, “geant4”, or “geant4dp”)</td>
</tr>
<tr>
<td>killNeutrinos</td>
<td>Whether to always stop tracking neutrinos for increased efficiency (default = true)</td>
</tr>
<tr>
<td>maximumStepLength</td>
<td>Maximum step length [m] (default = 20 m)</td>
</tr>
<tr>
<td>maximumTrackingTime</td>
<td>The maximum time of flight allowed for any particle before it is killed [s]</td>
</tr>
<tr>
<td>maximumTrackLength</td>
<td>The maximum length in metres of any track passing through any geometry in the model (not including the world volume)</td>
</tr>
<tr>
<td>minimumKineticEnergy</td>
<td>A particle below this energy will be killed and the energy deposition recorded at that location [GeV]</td>
</tr>
<tr>
<td>minimumKineticEnergyTunnel</td>
<td>A particle below this energy in any BDSIM-generated tunnel sections will be killed and the energy deposition recorded at that location [GeV]</td>
</tr>
<tr>
<td>minimumRadiusOfCurvature</td>
<td>Minimum tolerable radius of curvature of a particle, below which, the energy will be decreased by 2% on each use of the integrators to prevent infinite loops - should be just greater than width of beam pipe [m].</td>
</tr>
<tr>
<td>minimumRange</td>
<td>A particle that would not travel this range (a distance) in the current material will be cut [m]</td>
</tr>
<tr>
<td>ptcOneTurnMapFileName</td>
<td>File name for a one turn map prepared in PTC that is used in the teleporter to improve the accuracy of circular tracking. See One Turn Map.</td>
</tr>
<tr>
<td>stopSecondaries</td>
<td>Whether to stop secondaries or not (default = false)</td>
</tr>
<tr>
<td>tunnelIsInfiniteAbsorber</td>
<td>Whether all particles entering the tunnel material should be killed or not (default = false)</td>
</tr>
</tbody>
</table>
### 6.17.5 Physics Processes

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultBiasVacuum</td>
<td>Name of bias object to be attached to vacuum volumes by default</td>
</tr>
<tr>
<td>defaultBiasMaterial</td>
<td>Name of bias object to be attached to general material of components</td>
</tr>
<tr>
<td>defaultRangeCut</td>
<td>The default predicted range at which a particle is cut. Overwrites other</td>
</tr>
<tr>
<td>g4PhysicsUseBDSIMCutsAndLimits</td>
<td>If on, the maximum step length will be limited to 110% of the component</td>
</tr>
<tr>
<td></td>
<td>- length - this makes the tracking more robust and is the default</td>
</tr>
<tr>
<td></td>
<td>- with a regular BDSIM physics list. The minimum kinetic option is also</td>
</tr>
<tr>
<td></td>
<td>- obeyed. Default off.</td>
</tr>
<tr>
<td>g4PhysicsUseBDSIMRangeCuts</td>
<td>If on, this will apply the BDSIM range cut lengths to the Geant4 physics</td>
</tr>
<tr>
<td></td>
<td>- list used. This is off by default.</td>
</tr>
<tr>
<td>minimumKineticEnergy</td>
<td>A particle below this energy will be killed and the energy deposition</td>
</tr>
<tr>
<td></td>
<td>- recorded at that location [GeV]</td>
</tr>
<tr>
<td>minimumKineticEnergyTunnel</td>
<td>A particle below this energy in any BDSIM-generated tunnel sections</td>
</tr>
<tr>
<td></td>
<td>- will be killed and the energy deposition recorded at that location</td>
</tr>
<tr>
<td></td>
<td>- [GeV]</td>
</tr>
<tr>
<td>minimumRange</td>
<td>A particle that would not travel this range (a distance) in the current</td>
</tr>
<tr>
<td></td>
<td>- material will be cut [m]</td>
</tr>
<tr>
<td>neutronTimeLimit</td>
<td>Maximum allowed tracking time for a neutron when using the neuron Tracking</td>
</tr>
<tr>
<td></td>
<td>- cut physics list [s]</td>
</tr>
<tr>
<td>neutronKineticEnergyLimit</td>
<td>Minimum allowed energy for neutrons when using the neuron Tracking cut</td>
</tr>
<tr>
<td></td>
<td>- physics list [GeV]</td>
</tr>
<tr>
<td>physicsEnergyLimitLow</td>
<td>Optional lower energy level for all physics models. This is usually 990</td>
</tr>
<tr>
<td></td>
<td>- eV by default in Geant4. The user may change this if required. Warning,</td>
</tr>
<tr>
<td></td>
<td>- this must be used only if the user understands how this will affect</td>
</tr>
<tr>
<td></td>
<td>- the running of Geant4. [GeV]</td>
</tr>
<tr>
<td>physicsEnergyLimitHigh</td>
<td>Optional upper energy level for all physics models. This is usually 100</td>
</tr>
<tr>
<td></td>
<td>- TeV by default in Geant4. The user may change this if required. Warn-</td>
</tr>
<tr>
<td></td>
<td>- ing, this must be used only if the user understands how this will</td>
</tr>
<tr>
<td></td>
<td>- affect the running of Geant4. [GeV]</td>
</tr>
<tr>
<td>physicsList</td>
<td>Which physics lists to use - default tracking only</td>
</tr>
<tr>
<td>physicsVerbose</td>
<td>Prints out all processes linked to primary particle and all physics</td>
</tr>
<tr>
<td></td>
<td>- processes registered in general</td>
</tr>
<tr>
<td>prodCutPhotons</td>
<td>Standard overall production cuts for photons (default 1e-3) [m]</td>
</tr>
<tr>
<td>prodCutElectrons</td>
<td>Standard overall production cuts for electrons (default 1e-3) [m]</td>
</tr>
<tr>
<td>prodCutPositrons</td>
<td>Standard overall production cuts for positrons (default 1e-3) [m]</td>
</tr>
<tr>
<td>prodCutProtons</td>
<td>Standard overall production cuts for protons (default 1e-3) [m]</td>
</tr>
<tr>
<td>stopSecondaries</td>
<td>Whether to stop secondaries or not (default = false)</td>
</tr>
<tr>
<td>synchronRadOn</td>
<td>Whether to use synchrotron radiation processes</td>
</tr>
<tr>
<td>tunnelIsInfiniteAbsorber</td>
<td>Whether all particles entering the tunnel material should be killed or</td>
</tr>
<tr>
<td></td>
<td>- not (default = false)</td>
</tr>
<tr>
<td>turnOnCerenkov</td>
<td>Whether to produce Cherenkov radiation</td>
</tr>
<tr>
<td>useElectroNuclear</td>
<td>Uses electro-nuclear processes when em_extra physics list is used. De-</td>
</tr>
<tr>
<td></td>
<td>- fault On. Requires Geant4.10.4 or greater.</td>
</tr>
<tr>
<td>useGammaToMuMu</td>
<td>Uses gamma to muon pair production process when using em_extra physics</td>
</tr>
<tr>
<td></td>
<td>- list is used. Default Off. Requires Geant4.10.3 onwards.</td>
</tr>
<tr>
<td>useLENDGammaNuclear</td>
<td>Uses the low-energy neutron data set, as provided by the environmen-</td>
</tr>
<tr>
<td></td>
<td>- tal variable G4LENDDATA when using the em_extra physics list. Boolean,</td>
</tr>
<tr>
<td></td>
<td>- Available in Geant4.10.4 onwards.</td>
</tr>
<tr>
<td>useMuonNuclear</td>
<td>Uses muon-nuclear interaction processes when using em_extra physics</td>
</tr>
<tr>
<td>usePositronToMuMu</td>
<td>Uses muon pair production from positron annihilation when using em_extra</td>
</tr>
<tr>
<td>usePositronToHadrons</td>
<td>Uses hadron production from positron-electron annihilation process when</td>
</tr>
<tr>
<td></td>
<td>- using em_extra physics list. Default Off. Requires Geant4.10.3 on-</td>
</tr>
<tr>
<td></td>
<td>- wards.</td>
</tr>
</tbody>
</table>
6.17.6 Visualisation

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>nSegmentsPerCircle</td>
<td>the number of facets per $2 \pi$ in the visualiser. Note, this does not affect the accuracy of the geometry - only the visualisation (default = 50).</td>
</tr>
</tbody>
</table>

6.17.7 Output Options

The particle physics simulation in BDSIM can produce an impressive quantity of output information. The data describing a full record of every particle and their interaction would prove too difficult to manage or analyse sensibly. BDSIM records the most useful information, but provides options to record less or even more data. This is controlled with the following options.

**Note:** These options may increase the output file size by a large amount. Use only the ones you need.

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>elossHistoBinWidth</td>
<td>The width of the histogram bins [m]</td>
</tr>
<tr>
<td>nperfile</td>
<td>Number of events to record per output file</td>
</tr>
<tr>
<td>sensitiveOuter</td>
<td>Whether the outer part of each component (other than the beam pipe) records energy loss. storeEloss is required to be on for this to work. The user may turn off energy loss from the beam pipe and retain losses from the magnet outer in combination with the next option sensitiveBeamPipe. Both are stored together in Eloss branch of the Event Tree in the output. Default on.</td>
</tr>
<tr>
<td>sensitiveBeamPipe</td>
<td>Whether the beam pipe records energy loss. This includes cavities. This can be used in combination with the above option sensitiveOuter, to control which energy loss is recorded. Energy loss from this option is recorded in the Eloss branch of the Event Tree in the output. Default on.</td>
</tr>
<tr>
<td>sensitiveVacuum</td>
<td>Whether energy deposition in the residual vacuum gas is recorded. Energy loss from this option is recorded in the Eloss branch of the Event Tree in the output. Default on.</td>
</tr>
<tr>
<td>storeCollimatorHitsIons</td>
<td>If storeCollimatorInfo is on and collimator hits are generated, isIon, ionA and ionZ variables are filled. Collimator hits will now also be generated for all ions whether primary or secondary. Default off.</td>
</tr>
<tr>
<td>storeCollimatorHitsAll</td>
<td>If storeCollimatorInfo is on and collimator hits are generated, hits will be generated for all particles interacting with the collimators whether primary or secondary and whether ion or not. Default off.</td>
</tr>
<tr>
<td>storeCollimatorInfo</td>
<td>With this option on, summary information in the Model Tree about only collimators is filled. Collimator structures are created in the Event Tree of the output for each collimator and prefixed with “COLL_” and contain hits from (only) primary particles. Collimator summary histograms are also created and stored. Default off.</td>
</tr>
<tr>
<td>storeCollimatorLinks</td>
<td>If storeCollimatorInfo is on and collimator hits are generated, extra information is stored for each collimator hit.</td>
</tr>
<tr>
<td>storeEloss</td>
<td>Whether to store the energy deposition hits. Default on. By turning off, sensitiveBeamPipe, sensitiveOuter and sensitiveVacuum have no effect. Saves run time memory and output file size. See next option storeEloss for combination.</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Option</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>storeElossHistograms</td>
<td>Whether to store energy deposition histograms $E_{loss}$ and $E_{lossPE}$. This will automatically be on if $storeEloss$ is on. With $storeEloss$ off, this option can be turned on to retain the energy deposition histograms. If both this and $storeEloss$ are off, no energy deposition hits will be generated saving memory.</td>
</tr>
<tr>
<td>storeElossVacuum</td>
<td>Whether to store energy deposition from the vacuum volumes as hits in the $ElossVacuum$ branch and the corresponding summary histograms. Default off.</td>
</tr>
<tr>
<td>storeElossVacuumHistograms</td>
<td>Whether to generate summary histograms of energy deposition in the vacuum volumes. If $storeElossVacuum$ is on, this will be on. The user may turn off $storeElossVacuum$ but turn this on to store the energy deposition histograms.</td>
</tr>
<tr>
<td>storeElossTunnel</td>
<td>Whether to store energy deposition hits from the tunnel geometry in the $ElossTunnel$ branch of the Event Tree. Default off.</td>
</tr>
<tr>
<td>storeElossTunnelHistograms</td>
<td>Whether to generate summary histograms of energy deposition in the tunnel volumes. If $storeElossTunnel$ is on, this will be on. The user may turn off $storeElossTunnel$ but turn this on to store the energy deposition histograms.</td>
</tr>
<tr>
<td>storeElossWorld</td>
<td>Whether to record energy deposition in the world volume and, in the case of using Geant4.10.3 or newer, the energy leaving the world volume as well. Default off.</td>
</tr>
<tr>
<td>storeElossWorldContents</td>
<td>Whether to record energy deposition in the daughter volumes within the world volume when supplied as external world geometry. Default off.</td>
</tr>
<tr>
<td>storeElossGlobal</td>
<td>Global coordinates will be stored for each energy deposition hit and for each trajectory point. Default off.</td>
</tr>
<tr>
<td>storeElossLinks</td>
<td>For each energy deposition hit, the particle ID, track ID, parent ID and beam line index will be stored - this is intended to help ‘link’ the energy deposition back to other information. Default off.</td>
</tr>
<tr>
<td>storeElossLocal</td>
<td>Local coordinates will be stored for each energy deposition hit and for each trajectory point. Default off.</td>
</tr>
<tr>
<td>storeElossModelID</td>
<td>Store the beam line index of the object the energy deposition hit was in. If $storeElossLinks$ is on, this will be on irrespective of this option.</td>
</tr>
<tr>
<td>storeElossTime</td>
<td>The time since the start of the event will be stored for each point of energy deposition and trajectory. Default off.</td>
</tr>
<tr>
<td>storeElossTurn</td>
<td>Store the turn number of each energy deposition hit. Default off, but automatically on when using a circular machine with the (also executable) option $circular$</td>
</tr>
<tr>
<td>storeElossStepLength</td>
<td>Stores the step length for each energy deposition hit or not. Default off.</td>
</tr>
<tr>
<td>storeElossPreStepKineticEnergy</td>
<td>Stores the kinetic energy of the particle causing energy deposition as taken from the beginning of the step before it made it. Default off.</td>
</tr>
<tr>
<td>storeGeant4Data</td>
<td>Whether to store basic particle information for all particles used in the simulation under Geant4Data in the output. This can be relatively large when ions are used as there are many thousands of ion definitions. Default on.</td>
</tr>
<tr>
<td>storeModel</td>
<td>Whether to store the model information in the output. Default on.</td>
</tr>
<tr>
<td>storeSamplerAll</td>
<td>Convenience option to turn on all optional sampler output. Equivalent to turning on $storeSamplerCharge$, $storeSamplerKineticEnergy$, $storeSamplerMass$, $storeSamplerRigidity$, $storeSamplerIon$. Overrides these options even if they are explicitly set to off (0).</td>
</tr>
<tr>
<td>storeSamplerCharge</td>
<td>Stores corresponding charge of particle for every entry in sampler</td>
</tr>
<tr>
<td>storeSamplerKineticEnergy</td>
<td>Stores corresponding kinetic energy of particle for every entry in sampler.</td>
</tr>
<tr>
<td>storeSamplerMass</td>
<td>Stores corresponding mass (in GeV) of particle for every entry in the sampler.</td>
</tr>
<tr>
<td>Option</td>
<td>Function</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>storeSamplerPolarCoords</td>
<td>Calculate and store the polar coordinates (r, phi) and (rp, phip) for the sampler data.</td>
</tr>
<tr>
<td>storeSamplerRigidity</td>
<td>Stores the rigidity (in Tm) of particle for every entry in sampler</td>
</tr>
<tr>
<td>storeSamplerIon</td>
<td>Stores A, Z and Boolean whether the entry is an ion or not</td>
</tr>
<tr>
<td>storeTrajectory</td>
<td>Whether to store trajectories. If turned on, all trajectories are stored. This must be turned on to store any trajectories at all.</td>
</tr>
<tr>
<td>storeTrajectories</td>
<td>An alias to storeTrajectory</td>
</tr>
<tr>
<td>storeTrajectoryDepth</td>
<td>The depth of the particle tree to store the trajectories to 0 is the primary, 1 is the first generation of secondaries, etc.</td>
</tr>
<tr>
<td>storeTrajectoryELossSRange</td>
<td>Ranges in curvilinear S coordinate that if a particular track causes energy deposition in this range, its trajectory will be stored. The value should be a string inside which are pairs of numbers separated by a colon and ranges separated by whitespace such as “0.3:1.23 45.6:47.6”.</td>
</tr>
</tbody>
</table>
| storeTrajectoryParticle| The Geant4 name of particle(s) to only store trajectories for. This is case sensitive. Multiple particle names can be used with a space between them. e.g. “proton pi-“.
| storeTrajectoryParticleID| The PDG ID of the particle(s) to only store trajectories for. Multiple particle IDs can be supplied with a space between them. e.g. “11 12 22 13”. |
| storeTrajectoryEnergyThreshold| The threshold energy for storing trajectories. Trajectories for any particles with energy less than this amount (in GeV) will not be stored. |
| storeTrajectorySamplerID| If a trajectory reaches the name of these samplers, store that trajectory. This value supplied should be a whitespace separated string such as “cd1 qf32x”. |
| trajConnect            | Stores all the trajectories that connect a trajectory to be stored all the way to the primary particle. For example, if the filters from other trajectory options are to store only muons with an energy greater than 10 GeV, the few trajectories stored would appear unrelated. This option forces the storage of only the trajectories of any particles (irrespective of filters) that lead to the muon in question. |
| trajNoTransportation   | Suppresses trajectory points generated by transportation. When a particle hits a volume boundary, two trajectories would be created for before and afterwards, even if it didn’t interact or change. This option removes these points. |
| trajCutGTZ             | Only stores trajectories whose global z-coordinate is greater than this value in metres [m]. |
| trajCutLTR             | Only stores trajectories whose global radius is from the start position (sqrt(x^2, y^2)). |

### 6.17.8 Offset for Main Beam Line

The following options may be used to offset the main beam line with respect to the world volume, which is the outermost coordinate system.
Two styles of rotation can be used: either a set of three Euler angles, or the axis angle rotation scheme where a unit vector is provided in \( x, y, z \) and an angle to rotate about it. These variables are used to construct a \( \text{G4RotationMatrix} \) directly, which is also the same as a \( \text{CLHEP::HepRotation} \).

**Note:** Geant4 uses a right-handed coordinate system and \( m \) and \( rad \) are the default units for offsets and angles in BDSIM.

Example:

```plaintext
option, beamlineX = 3*m,
        beamlineY = 20*cm,
        beamlineZ = -30*m,
        beamlineAxisAngle = 1,
        beamlineAxisY = 1,
        beamlineAngle = 0.2;
```

This offsets the beam line by (3,0.2,-30) m and rotates about the unit vector (0,1,0) (i.e. in the horizontal plane - \( x,z \)) by 0.2 rad.

### 6.17.9 Scoring Map

BDSIM provides the capability to create one 3D histogram of energy deposition hits irrespective of the geometry. The hits are only created where the geometry exists and are sensitive. The histogram is independent of the geometry.
6.17.10 Developer Options

These are documented here, but use with caution, as they lead to undesirable behaviour and should only be used with understanding.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chordStepMinimum</td>
<td>Minimum step size</td>
</tr>
<tr>
<td>deltaChord</td>
<td>Chord finder precision</td>
</tr>
<tr>
<td>deltaIntersection</td>
<td>Boundary intersection precision</td>
</tr>
<tr>
<td>deltaOneStep</td>
<td>Set position error acceptable in an integration step</td>
</tr>
<tr>
<td>lengthSafety</td>
<td>Element overlap safety (caution!)</td>
</tr>
<tr>
<td>maximumEpsilonStep</td>
<td>Maximum relative error acceptable in stepping</td>
</tr>
<tr>
<td>minimumEpsilonStep</td>
<td>Minimum relative error acceptable in stepping</td>
</tr>
<tr>
<td>sampleElementsWithPoleface</td>
<td>Default false. Samplers are not to be attached to elements with pole-face rotations, as the sampler will overlap with the mass world geometry, resulting in incorrect tracking. This only occurs in integrator sets which construct the poleface geometry, namely bdsimtwo, geant4, and geant4dp. This option overrides this, allowing samplers to be attached. This option will not affect the default integrator set, bdsimmatrix.</td>
</tr>
<tr>
<td>teleporterFullTransform</td>
<td>Default true. Whether to use the newer teleporter offset method that uses a G4Transform3D to apply both an offset and a rotation. The newer method works in any 3D orientation whereas the old one only works with the beam line starting along unit Z (i.e. no beam line offset or rotation.</td>
</tr>
<tr>
<td>beam, offsetSampleMean=1</td>
<td>Default false. If true, this will remove the sample mean from the bunch distribution to match the central values. This is useful for optical function calculation. BDSIM is not currently able to reproduce results when this option is used and coordinates will be different for each run, or even when using –recreate. Only suitable for large (&gt;100) numbers of particles. Note: this isn’t an option, but part of the beam command. This cannot be used with the visualiser.</td>
</tr>
</tbody>
</table>

6.18 Beam Parameters

BDSIM starts each event by simulating one particle from a beam distribution. A distribution is chosen by the user in the input GMAD and the particle coordinates are randomly generated according to this distribution. To specify the input particle distribution, the `beam` command is used. This also specifies the particle species and reference total energy, which is the design total energy of the machine. This is used along with the particle species to calculate the momentum of the reference particle and therefore the magnetic rigidity that normalised magnetic field strengths are calculated with respect to. For example, the field of dipole magnets is calculated using this if only the angle parameter has been specified.

Apart from the design particle and energy, a beam of particles of a different species and total energy may be specified. By default, if only one particle is specified this is assumed to be both the design particle and the particle distribution will be used as well as default parameters. The minimum beam definitions are:

Note: The design energy is required to be specified, but the central energy, of say a bunch with a Gaussian distribution, can be also be specified with $E_0$.

Note: energy here is the total energy of the particle. This must be greater than the rest mass of the particle.

The user must specify at least energy and the particle type. In this case the reference distribution will be used as well as default parameters. The minimum beam definitions are:
Other parameters, such as the beam distribution type, distrType, are optional and can be specified as described in the following sections. The beam is defined using the following syntax:

```
beam, particle="proton",
   energy=4.0*TeV,
   distrType="reference";
```

Energy is the total energy in GeV. The beam particle may be one of the following:

- $e^-$ or $e^+$
- proton or antiproton
- gamma
- neutron
- $\mu^-$ or $\mu^+$
- $\pi^-$ or $\pi^+$
- photon or gamma

In fact, the user may specify any particle that is available through the physics lists used. The particle must be given by the Geant4 name. The ones above are always defined and so can always safely be used irrespective of the physics lists used. If the particle definition is not found, BDSIM will print a warning and exit.

### 6.18.1 Ion Beams

The user may also specify any ion with the following syntax:

```
beam, particle="ion A Z";
```

or:

```
beam, particle="ion A Z Q";
```

where $A$, $Z$, and $Q$ should be replaced by the atomic number, the number of protons in the nucleus and the charge. The charge is optional and by default is $Z$ (i.e. a fully ionised ion). In this case, it is recommended to use the ion physics list.

Available input distributions and their associated parameters are described in the following section.

### 6.18.2 Different Beam and Design Particles

The model may use one particle for design and one for the beam distribution. The “design” particle is used to calculate the rigidity that is used along with normalised field strengths (such as $k_1$ for quadrupoles) to calculate an absolute field or field gradient. However, it is often useful to simulate a beam of other particles. To specify a different central energy, the parameter $E_0$ should be used. If a different particle is required the parameter beamParticleName should be used.

Examples:

```
beam, particle="e-",
   energy=100*GeV,
   beamParticleName="e+";
```

This specifies that the magnet field strengths are calculated with respect to a 100 GeV electron and the beam tracked is a 100 GeV positron beam (along with any other relevant distribution parameters).
beam, particle="e-",
   energy=100*GeV,
   beamParticleName="e+",
   E0=20*GeV;

This specified that the magnet field strengths are calculated with respect to a 100 GeV electron and the beam tracked is a 20 GeV positron beam.

- If no beamParticleName variable is specified, it's assumed to be the same as particle.
- If no E0 variable is specified, it's assumed to be the same as energy.

6.18.3 Generate Only the Input Distribution

BDSIM can generate only the input distribution and store it to file without creating a model or running any physics simulation. This is very fast and can be used to verify the input distribution with a large number of particles (for example, 10k to 100k in under one minute).

BDSIM should be executed with the option --generatePrimariesOnly as described in Executable Options.

6.18.4 Beam in Output

All of the beam parameters are stored in the output, as described in Beam Tree. The particle coordinates used in the simulation are stored directly in the Primary branch of the Event Tree, as described in Event Tree.

Note: These are the exact coordinates supplied to Geant4 at the beginning of the event. Conceptually, these are 'local' coordinates with respect to the start of the beam line. However, if a finite S0 is specified, the bunch distribution is transformed to that location in the World, therefore the coordinates are the global ones used.

Warning: For large S0 in a large model, the particles may be displaced by a large distance as compared to the size of the beam, e.g. 1km offset for 1um beam. In this case, the limited precision of the float used to store the coordinates in the output may not show the beam distribution as expected. Internally, double precision numbers are used so that the beam distribution is accurate. A float typically has seven significant figures and a double 15.

6.18.5 Beam Distributions

The following beam distributions are available in BDSIM

- reference
- gaussmatrix
- gauss
- gausstwiss
- circle
- square
- ring
- eshell
- halo
- composite
• userfile
• ptc

Note: For gauss, gaussmatrix and gausstwiss, the beam option beam, offsetSampleMean=1 documented in Developer Options can be used to pre-generate all particle coordinates and subtract the sample mean from these, effectively removing any small systematic offset in the bunch at the beginning of the line. This is used only for optical comparisons currently.

6.18.6 reference

This is a single particle with the same position and angle defined by the following parameters. The coordinates are the same for every particle fired using the reference distribution. It is therefore not likely to be useful to generate a large number of repeated events with this distribution unless the user wishes to explore the different outcome from the physics processes, which will be different each time should the particle interact. This distribution may be referred to as a 'pencil' distribution by other codes.

These parameters also act as central parameters for all other distributions. For example, a Gaussian distribution may be defined with the gauss parameters, but with $X0$ set to offset the centroid of the Gaussian with respect to the reference trajectory. Note: energy is total energy of the particle - including the rest mass.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X0$</td>
<td>Horizontal position [m]</td>
<td>0</td>
</tr>
<tr>
<td>$Y0$</td>
<td>Vertical position [m]</td>
<td>0</td>
</tr>
<tr>
<td>$Z0$</td>
<td>Longitudinal position [m]</td>
<td>0</td>
</tr>
<tr>
<td>$T0$</td>
<td>Longitudinal position [s]</td>
<td>0</td>
</tr>
<tr>
<td>$Xp0$</td>
<td>Horizontal canonical momentum</td>
<td>0</td>
</tr>
<tr>
<td>$Yp0$</td>
<td>Vertical canonical momentum</td>
<td>0</td>
</tr>
<tr>
<td>$E0$</td>
<td>Central total energy of bunch distribution (GeV)</td>
<td>'energy'</td>
</tr>
</tbody>
</table>

Examples:

```
beam, particle = "e-",
   energy = 10*GeV,
   distrType = "reference";
```

Generates a beam with all coordinates=0 at the nominal energy.

```
beam, particle = "e-",
   energy = 10*GeV,
   distrType = "reference",
   X0 = 100*um,
   Y0 = 3.5*um;
```

Generates a particle with an offset of 100 μm horizontally and 3.5 μm vertically.

6.18.7 gaussmatrix

Uses the $N$ dimensional Gaussian generator from CLHEP, CLHEP::RandMultiGauss. The generator is initialised by a $6 \times 1$ means vector and $6 \times 6$ sigma matrix.

• All parameters from reference distribution are used as centroids.
**Option** | **Description**
--- | ---
`sigmaNM` | Sigma matrix element (N,M)

- Only the upper-right half of the matrix and diagonal should be populated, as the elements are symmetric across the diagonal.
- The coordinates are in order 1:x (m), 2:xp, 3:y (m), 4:yp, 5:t (s), 6:E (GeV).

The user should take care to ensure they specify a positive definite matrix. BDSIM will emit an error and stop running if this is not the case.

**Examples:**

```plaintext
beam, particle = "e-",
energy = 10*GeV,
distrType = "gaussmatrix",
sigma11 = 100*um,
sigma22 = 3*um,
sigma33 = 50*um,
sigma44 = 1.4*um,
sigma55 = 1e-12
sigma66 = 1e-4,
sigma12 = 1e-2,
sigma34 = 1.4e-3;
```

**Note:** One should take care in defining, say, `sigma16`, as this is the covariance of the x position and energy. However, this may be proportional to momentum and not total energy. Note, for such a correlation between x and E, other off-diagonal terms in the covariance matrix should be finite also.

### 6.18.8 gauss

Uses the `gaussmatrix` beam generator but with simplified input parameters, as opposed to a complete beam sigma matrix. This beam distribution has a diagonal σ-matrix and does not allow for correlations between phase space coordinates, so

\[
\begin{align*}
\sigma_{11} &= \sigma_z^2 \\
\sigma_{22} &= \sigma_x^2 \\
\sigma_{33} &= \sigma_y^2 \\
\sigma_{44} &= \sigma_y^2 \\
\sigma_{55} &= \sigma_T^2 \\
\sigma_{66} &= \sigma_E^2.
\end{align*}
\]

- The coordinates are in order 1:x (m), 2:xp, 3:y (m), 4:yp, 5:t (s), 6:E (GeV).
- All parameters from *reference* distribution are used as centroids.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sigmaX</code></td>
<td>Horizontal Gaussian sigma [m]</td>
</tr>
<tr>
<td><code>sigmaY</code></td>
<td>Vertical Gaussian sigma [m]</td>
</tr>
<tr>
<td><code>sigmaXp</code></td>
<td>Sigma of the horizontal canonical momentum</td>
</tr>
<tr>
<td><code>sigmaYp</code></td>
<td>Sigma of the vertical canonical momentum</td>
</tr>
<tr>
<td><code>sigmaE</code></td>
<td>Relative energy spread $\sigma_E/E$</td>
</tr>
<tr>
<td><code>sigmaT</code></td>
<td>Sigma of the temporal distribution [s]</td>
</tr>
</tbody>
</table>
6.18.9 gausstwiss

The beam parameters are defined by the usual Twiss parameters $\alpha$, $\beta$ and $\gamma$, plus dispersion $\eta$, from which the beam $\sigma$-matrix is calculated, using the following equations:

$$
\begin{align*}
\sigma_{11} &= \epsilon_x \beta_x + \eta_x^2 \sigma_E^2 \\
\sigma_{12} &= -\epsilon_x \alpha_x + \eta_x \eta_{xp} \sigma_E^2 \\
\sigma_{21} &= -\epsilon_x \alpha_x + \eta_x \eta_{xp} \sigma_E^2 \\
\sigma_{22} &= \epsilon_x \gamma_x + \eta_{xp} \sigma_E^2 \\
\sigma_{33} &= \epsilon_y \beta_y + \eta_y^2 \sigma_E^2 \\
\sigma_{34} &= -\epsilon_y \alpha_y + \eta_y \eta_{yp} \sigma_E^2 \\
\sigma_{43} &= -\epsilon_y \alpha_y + \eta_y \eta_{yp} \sigma_E^2 \\
\sigma_{44} &= \epsilon_y \gamma_y + \eta_{yp} \sigma_E^2 \\
\sigma_{13} &= \eta_x \eta_y \sigma_E^2 \\
\sigma_{31} &= \eta_x \eta_y \sigma_E^2 \\
\sigma_{23} &= \eta_{xp} \eta_y \sigma_E^2 \\
\sigma_{32} &= \eta_{xp} \eta_y \sigma_E^2 \\
\sigma_{14} &= \eta_x \eta_{yp} \sigma_E^2 \\
\sigma_{41} &= \eta_x \eta_{yp} \sigma_E^2 \\
\sigma_{24} &= \eta_{xp} \eta_{yp} \sigma_E^2 \\
\sigma_{16} &= \eta_x \sigma_E^2 \\
\sigma_{61} &= \eta_x \sigma_E^2 \\
\sigma_{26} &= \eta_{xp} \sigma_E^2 \\
\sigma_{62} &= \eta_{xp} \sigma_E^2 \\
\sigma_{36} &= \eta_y \sigma_E^2 \\
\sigma_{63} &= \eta_y \sigma_E^2 \\
\sigma_{46} &= \eta_{yp} \sigma_E^2 \\
\sigma_{64} &= \eta_y \sigma_E^2 \\
\sigma_{55} &= \sigma_T^2 \\
\sigma_{66} &= \sigma_T^2
\end{align*}
$$

• All parameters from reference distribution are used as centroids.
• Longitudinal parameters $\sigma_E$ and $\sigma_T$ used as defined in gauss.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emitx</td>
<td>Horizontal beam core emittance [m]</td>
</tr>
<tr>
<td>emity</td>
<td>Vertical beam core emittance [m]</td>
</tr>
<tr>
<td>betx</td>
<td>Horizontal beta function [m]</td>
</tr>
<tr>
<td>bety</td>
<td>Vertical beta function [m]</td>
</tr>
<tr>
<td>alfx</td>
<td>Horizontal alpha function</td>
</tr>
<tr>
<td>alfy</td>
<td>Vertical alpha function</td>
</tr>
<tr>
<td>dispx</td>
<td>Horizontal dispersion function [m]</td>
</tr>
<tr>
<td>dispy</td>
<td>Vertical dispersion function [m]</td>
</tr>
<tr>
<td>dispxp</td>
<td>Horizontal angular dispersion function</td>
</tr>
<tr>
<td>dispyp</td>
<td>Vertical angular dispersion function</td>
</tr>
</tbody>
</table>
6.18.10 circle

Beam of randomly distributed particles with a uniform distribution within a circle in each dimension of phase space - \(x, xp, y, yp, T, E\) with each uncorrelated. Each parameter defines the maximum absolute extent in that dimension, i.e. the possible values range from \(-envelopeX\) to \(envelopeX\) for example. Total energy is also uniformly distributed between \(\pm envelopeE\).

- All parameters from reference distribution are used as centroids.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>envelopeR</td>
<td>Maximum position</td>
</tr>
<tr>
<td>envelopeRp</td>
<td>Maximum canonical momentum</td>
</tr>
<tr>
<td>envelopeT</td>
<td>Maximum time offset [s]</td>
</tr>
<tr>
<td>envelopeE</td>
<td>Maximum energy offset [GeV]</td>
</tr>
</tbody>
</table>

6.18.11 square

This distribution has similar properties to the circle distribution, with the exception that the particles are randomly uniformly distributed within a square. Total energy is also uniformly distributed between \(\pm envelopeE\).

- All parameters from reference distribution are used as centroids.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>envelopeX</td>
<td>Maximum position in X [m]</td>
</tr>
<tr>
<td>envelopeXp</td>
<td>Maximum canonical momentum in X</td>
</tr>
<tr>
<td>envelopeY</td>
<td>Maximum position in Y [m]</td>
</tr>
<tr>
<td>envelopeYp</td>
<td>Maximum canonical momentum in Y</td>
</tr>
<tr>
<td>envelopeT</td>
<td>Maximum time offset [s]</td>
</tr>
<tr>
<td>envelopeE</td>
<td>Maximum energy offset [GeV]</td>
</tr>
</tbody>
</table>

6.18.12 ring

The ring distribution randomly and uniformly fills a ring in \(x\) and \(y\) between two radii. For all other parameters, the reference coordinates are used, i.e. \(xp, yp\) etc.

- All parameters from reference distribution are used as centroids.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rmin</td>
<td>Minimum radius in (x) and (y) [m]</td>
</tr>
<tr>
<td>Rmax</td>
<td>Maximum radius in (x) and (y) [m]</td>
</tr>
</tbody>
</table>

- No variation in \(z, xp, yp, t, s\) and total energy. Only central values.

6.18.13 eshell

Defines an elliptical annulus in phase space in each dimension that’s uncorrelated.

- All parameters from reference distribution are used as centroids.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>shellX</code></td>
<td>Ellipse semi-axis in phase space in horizontal position [m]</td>
</tr>
<tr>
<td><code>shellXp</code></td>
<td>Ellipse semi-axis in phase space in horizontal canonical momentum</td>
</tr>
<tr>
<td><code>shellY</code></td>
<td>Ellipse semi-axis in phase space in vertical position [m]</td>
</tr>
<tr>
<td><code>shellYp</code></td>
<td>Ellipse semi-axis in phase space in vertical momentum</td>
</tr>
<tr>
<td><code>shellXWidth</code></td>
<td>Spread of ellipse in phase space in horizontal position [m]</td>
</tr>
<tr>
<td><code>shellXpWidth</code></td>
<td>Spread of ellipse in phase space in horizontal canonical momentum</td>
</tr>
<tr>
<td><code>shellYWidth</code></td>
<td>Spread of ellipse in phase space in vertical position [m]</td>
</tr>
<tr>
<td><code>shellYpWidth</code></td>
<td>Spread of ellipse in phase space in vertical momentum</td>
</tr>
<tr>
<td><code>sigmaE</code></td>
<td>Extent of energy spread in fractional total energy. Uniformly distributed between $\pm \sigmaE$.</td>
</tr>
</tbody>
</table>

- No variation in $t$, $z$, and $s$. Only central values.

### 6.18.14 halo

The halo distribution is effectively a flat phase space with the central beam core removed at $\epsilon_{core}$. The beam core is defined using the standard Twiss parameters described previously. The implicit general form of a rotated ellipse is

$$\gamma x'^2 + 2\alpha x x' + \beta x'^2 = \epsilon$$

where the parameters have their usual meanings. A phase space point can be rejected or weighted depending on the single particle emittance, which is calculated as

$$\epsilon_{SP} = \gamma x'^2 + 2\alpha x x' + \beta x'^2$$

if the single particle emittance is less than beam emittance, such that $\epsilon_{SP} < \epsilon_{core}$ the particle is rejected. `haloPSWeightFunction` is a string that selects the function $f_{haloWeight}(\epsilon_{SP})$ which is 1 at the ellipse defined by $\epsilon_{core}$. The weighting functions are either `flat`, one over emittance `oneoverr` or exponential `exp`.

- All parameters from `reference` distribution are used as centroids.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>emitx</code></td>
<td>Horizontal beam core emittance [m] $\epsilon_{core,x}$</td>
</tr>
<tr>
<td><code>emity</code></td>
<td>Vertical beam core emittance [m] $\epsilon_{core,y}$</td>
</tr>
<tr>
<td><code>betx</code></td>
<td>Horizontal beta function [m] $\epsilon_{core,x}$</td>
</tr>
<tr>
<td><code>bety</code></td>
<td>Vertical beta function [m] $\epsilon_{core,y}$</td>
</tr>
<tr>
<td><code>alfx</code></td>
<td>Horizontal alpha function</td>
</tr>
<tr>
<td><code>alfy</code></td>
<td>Vertical alpha function</td>
</tr>
<tr>
<td><code>haloNSigmaXInner</code></td>
<td>Inner radius of halo in x (multiples of sigma)</td>
</tr>
<tr>
<td><code>haloNSigmaXOuter</code></td>
<td>Outer radius of halo in x (multiples of sigma)</td>
</tr>
<tr>
<td><code>haloNSigmaYInner</code></td>
<td>Inner radius of halo in y (multiples of sigma)</td>
</tr>
<tr>
<td><code>haloNSigmaYOuter</code></td>
<td>Outer radius of halo in y (multiples of sigma)</td>
</tr>
<tr>
<td><code>haloPSWeightFunction</code></td>
<td>Phase space weight function [string]</td>
</tr>
<tr>
<td><code>haloPSWeightParameter</code></td>
<td>Phase space weight function parameters [ ]</td>
</tr>
<tr>
<td><code>haloXCutInner</code></td>
<td>X position cut in halo (multiples of sigma)</td>
</tr>
<tr>
<td><code>haloYCutInner</code></td>
<td>Y position cut in halo (multiples of sigma)</td>
</tr>
</tbody>
</table>

- No variation in $t$, total energy, $z$ and $s$. Only central values.
Example:

```plaintext
beam, particle = "e-",
energy = 1.0*GeV,
distrType = "halo",
betx = 0.6,
bety = 1.2,
alfx = -0.023,
alfy = 1.3054,
emix = 5e-9,
emity = 4e-9,
haloNSigmaXInner = 0.1,
haloNSigmaXOuter = 2,
haloNSigmaYInner = 0.1,
haloNSigmaYOuter = 2,
haloPSWeightParameter = 1,
haloPSWeightFunction = "oneoverr";
```

6.18.15 composite

The horizontal, vertical and longitudinal phase spaces can be defined independently. The xDistrType, yDistrType and zDistrType can be selected from all the other beam distribution types. All of the appropriate parameters need to be defined for each individual distribution.

- All parameters from reference distribution are used as centroids.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xDistrType</td>
<td>Horizontal distribution type</td>
</tr>
<tr>
<td>yDistrType</td>
<td>Vertical distribution type</td>
</tr>
<tr>
<td>zDistrType</td>
<td>Longitudinal distribution type</td>
</tr>
</tbody>
</table>

**Note:** It is currently not possible to use two differently specified versions of the same distribution within the composite distribution, i.e. gaussTwiss (parameter set 1) for x and gaussTwiss (parameter set 2) for y. They will have the same settings.

Examples:

```plaintext
beam, particle="proton",
energy=3500*GeV,
distrType="composite",
xDistrType="eshell",
yDistrType="gausstwiss",
zDistrType="gausstwiss",
betx = 0.5*m,
bety = 0.5*m,
alfx = 0.00001234,
alfy = -0.0005425,
emix = 1e-9,
emity = 1e-9,
sigmaE = 0.00008836,
sigmaT = 0.00000000001,
shellX = 150*um,
shellY = 103*um,
shellXp = 1.456e-6,
shellYp = 2.4e-5,
shellXWidth = 10*um,
shellYWidth = 15*um,
```

(continues on next page)
6.18.16 userFile

The userFile distribution allows the user to supply an ASCII text file with particle coordinates that are tab-delimited. The column names and the units are specified in an input string.

The file may also be compressed using gzip. Any file with the extension .gz will be automatically decompressed during the run without any temporary files. This is recommended, as compressed ASCII is significantly smaller in size.

If the number of particles to be generated with ngenerate is greater than the number of particles defined in the file, the bunch generation will reload the file and read the particle coordinates from the beginning. A warning will be printed out in this case.

This distribution reads lines at the start of each event to be memory efficient. However, this prevents reading a whole file by the number of lines in the file unlike the ptc distribution that loads all lines and can use the beam option matchDistrFileLength.

**Note:** For gzip support, BDSIM must be compiled with GZIP. This is normally sourced from Geant4 and is on by default.

- tar + gz will not work. The file must be a single file compressed through gzip only.
- Lines starting with # will be ignored.
- Empty lines will also be ignored.
- A warning will be printed if the line is shorter than the number of variables specified in distrFileFormat and the event aborted - the simulation safely proceeds to the next event.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>distrFile</td>
<td>File path to ASCII data file</td>
</tr>
<tr>
<td>distrFileFormat</td>
<td>A string that details the column names and units</td>
</tr>
<tr>
<td>nlinesIgnore</td>
<td>Number of lines to ignore when reading user bunch input files</td>
</tr>
<tr>
<td>matchDistrFileLength</td>
<td>Option for certain distributions to simulate the same number of events as are in the file. Currently only for the ptc distribution.</td>
</tr>
</tbody>
</table>

Acceptable tokens for the columns are:

<table>
<thead>
<tr>
<th>Token</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>“E”</td>
<td>Total energy</td>
</tr>
<tr>
<td>“Ek”</td>
<td>Kinetic energy</td>
</tr>
<tr>
<td>“P”</td>
<td>Momentum</td>
</tr>
<tr>
<td>“t”</td>
<td>Time</td>
</tr>
<tr>
<td>“x”</td>
<td>Horizontal position</td>
</tr>
<tr>
<td>“y”</td>
<td>Vertical position</td>
</tr>
<tr>
<td>“z”</td>
<td>Longitudinal position</td>
</tr>
<tr>
<td>“xp”</td>
<td>Horizontal angle</td>
</tr>
<tr>
<td>“yp”</td>
<td>Vertical angle</td>
</tr>
<tr>
<td>“zp”</td>
<td>Longitudinal</td>
</tr>
<tr>
<td>“pt”</td>
<td>PDG particle ID</td>
</tr>
<tr>
<td>“w”</td>
<td>Weight</td>
</tr>
<tr>
<td>“-”</td>
<td>Skip this column</td>
</tr>
</tbody>
</table>
Length Units “m”, “cm”, “mm”, “mum”, “um”, “nm”  
Angle Units “rad”, “mrad”, “murad”, “urad”  
Time Units “s”, “ms”, “mus”, “us”, “ns”, “mm/c”, “nm/c”  

Examples:

```
beam, particle = "e-",
   energy = 1*GeV,
   distrType = "userfile",
   distrFile = "Userbeamdata.dat",
   distrFileFormat = "X[mum]:x[mrad]:y[mum]:yp[mrad]:z[cm]:E[MeV]";
```

The corresponding `userbeamdata.dat` file looks like:

```
0 1 2 1 0 1000
0 1 0 1 0 1002
0 1 0 0 0 1003
0 0 2 0 0 1010
0 0 0 2 0 1100
0 0 0 4 0 1010
0 0 0 3 0 1010
0 0 0 4 0 1020
0 0 0 2 0 1000
```

### 6.18.17 ptc

Output from MAD-X PTC used as input for BDSIM.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>distrFile</td>
<td>PTC output file</td>
</tr>
</tbody>
</table>

- Reference offsets specified in the gmad file such as $X0$ are added to each coordinate.

### 6.19 Tunnel Geometry

BDSIM can build a tunnel around the beam line. Currently, there are two main ways to control this.

1. The tunnel follows the beam line, bending automatically (recommended)
2. The tunnel is just built in a straight line - this may be useful for linear colliders but may also cause geometry overlaps (the user is responsible for checking this!)

**Warning:** With option 2, the user is entirely responsible to ensure no overlaps occur (through good design). Also note that the samplers may overlap the tunnel depending on the tunnel geometry (samplers are square with half-width of $samplerRadius$). In practice, however, we haven’t observed many ill effects because of this. Problems would take the form of ‘stuck particles’ and Geant4 would terminate that event.

Examples of tunnel geometry can be found with the BDSIM source code in `/examples/features/geometry/tunnel` and are described in `Tunnel Geometry`.

The automatic tunnel building is controlled through the following options used with the `option` command.
Tunnel Parameters | Description
---|---
buildTunnel | Whether to build a tunnel (default = 0)
builtTunnelStraight | Whether to build a tunnel, ignoring the beamline and just in a straight line (default = 0)
builtTunnelFloor | Whether to add a floor to the tunnel
tunnelIsInfiniteAbsorber | Whether all particles entering the tunnel material should be killed or not (default = false)
tunnelType | Which style of tunnel to use - one of: circular, elliptical, square, rectangular (more to come in v0.9)
tunnelAper1 | Tunnel aperture parameter #1 - typically horizontal (m)
tunnelAper2 | Tunnel aperture parameter #2 - typically vertical (m)
tunnelThickness | Thickness of tunnel wall (m)
tunnelSoilThickness | Soil thickness outside tunnel wall (m)
tunnelMaterial | Material for tunnel wall
soilMaterial | Material for soil outside tunnel wall
tunnelOffsetX | Horizontal offset of the tunnel with respect to the beam line reference trajectory
tunnelOffsetY | Vertical offset of the tunnel with respect to the beam line reference trajectory
tunnelFloorOffset | The offset of the tunnel floor from the centre of the tunnel (not the beam line)

These parameters are shown schematically in the figure below (gaps not to scale, elliptical shown as an example).

The soil around the tunnel is typically symmetric, with the `tunnelSoilThickness` being added to the larger of the horizontal and vertical tunnel dimensions.
Construction of the tunnel geometry may fail in particular cases of different beam lines. Beam lines with very strong bends ( > 0.5 rad) over a few metres may cause overlapping geometry. In future, it will be possible to override the automatic algorithm between certain elements in the beamline, but for now such situations must be avoided.

**Note:** Surrounding the beam line with a tunnel completely means that every particle simulated will have to eventually hit something and not escape. This means that every single particle will likely create a shower of particles down to 0 energy. This can increase simulation time. To avoid this, or at least control this behaviour, it is recommended to use the options minimumKineticEnergyTunnel or tunnelIsInfiniteAbsorber.

### 6.20 Materials and Atoms

All chemical elements are available in BDSIM as well as the Geant4 NIST database of materials for use. Custom materials and can also be added via the parser. All materials available in BDSIM can be found by executing BDSIM with the `--materials` option.

```bash
bdsim --materials
```

Aside from these, several materials useful for accelerator applications are already defined that are listed in *Predefined Materials*.

Generally, each beam line element accepts an argument “material” that is the material used for that element. It is used differently depending on the element. For example, in the case of a magnet, it is used for the yoke and for a collimator for the collimator block.

#### 6.20.1 Single Element

In the case of an element, the chemical symbol can be specified:

```plaintext
rc1: rcol, l=0.6*m, xsize=1.2*cm, ysize=0.6*cm, material="W";
```

These are automatically prefixed with `G4_` and retrieved from the NIST database of materials.

The user can also define their own material and then refer to it by name when defining a beam line element.

#### 6.20.2 Custom Single Element Material

If the material required is composed of a single element, but say of a different density or state than the default NIST one provided, it can be defined using the `matdef` command with the following syntax:

```plaintext
materialname : matdef, Z=<int>, A=<double>, density=<double>, T=<double>, P=←<double>, state=<char*>;
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>Atomic number</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>Mass number [g/mol]</td>
<td></td>
</tr>
<tr>
<td>density</td>
<td>Density [g/cm³]</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>Temperature [K]</td>
<td>300</td>
</tr>
<tr>
<td>P</td>
<td>Pressure [atm]</td>
<td>1</td>
</tr>
<tr>
<td>state</td>
<td>“solid”, “liquid” or “gas”</td>
<td>“solid”</td>
</tr>
</tbody>
</table>

Example:
A compound material can be specified in two manners:

### 6.20.3 Compound Material by Atoms

If the number of atoms of each component in a material unit is known, the following syntax can be used:

```plaintext
<material> : matdef, density=<double>, T=<double>, P=<double>,
          state=<char*>, components=<[list<char*>]>,
        componentsWeights=<{list<int>}>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>density</td>
<td>Density in [g/cm³]</td>
</tr>
<tr>
<td>components</td>
<td>List of symbols for material components</td>
</tr>
<tr>
<td>componentsWeights</td>
<td>Number of atoms for each component in material unit</td>
</tr>
</tbody>
</table>

Example:

```
NbTi : matdef, density=5.6, T=4.0, components=["Nb","Ti"], componentsWeights={1,1};
```

### 6.20.4 Compound Material by Mass Fraction

On the other hand, if the mass fraction of each component is known, the following syntax can be used:

```plaintext
<material> : matdef, density=<double>, T=<double>, P=<double>,
          state=<char*>, components=<[list<char*>]>,
        componentsFractions=<{list<double>}>
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>components</td>
<td>List of symbols for material components</td>
</tr>
<tr>
<td>componentsFractions</td>
<td>Mass fraction of each component in material unit</td>
</tr>
</tbody>
</table>

Example:

```
SmCo : matdef, density=8.4, T=300.0, components=["Sm","Co"], componentFractions = [0.338,0.662];
```

The second syntax can also be used to define materials which are composed by other materials (and not by atoms).

**Note:** Square brackets are required for the list of element symbols, curly brackets for the list of weights or fractions.

New elements can be defined with the `atom` keyword:

```plaintext
elementname : atom, Z=<int>, A=<double>, symbol=<char>;
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z</td>
<td>Atomic number</td>
</tr>
<tr>
<td>A</td>
<td>Mass number [g/mol]</td>
</tr>
<tr>
<td>symbol</td>
<td>Atom symbol</td>
</tr>
</tbody>
</table>

Example:
6.20.5 Predefined Materials

The following elements are available by full name that refer to the Geant4 NIST elements:

- aluminium
- beryllium
- carbon
- chromium
- copper
- iron
- lead
- magnesium
- nickel
- nitrogen
- silicon
- titanium
- tungstem
- uranium
- vanadium
- zinc

The following materials are also defined in BDSIM. The user should consult `bdsim/src/BDSMaterials.cc` for the full definition of each including elements, mass fractions, temperature and state.

- air (G4_AIR)
- airbdsim
- aralditef
- awakeplasma
- berylliumcopper
- bn5000
- bp_carbonmonoxide
- calciumcarbonate
- carbonfiber
- carbonmonoxide
- carbonsteel
- cellulose (G4_CELLULOSE_CELLOPHANE)
- clay
- clayousMarl
• concrete
• cu_4k
• dy061
• epoxyresin3
• fusedsilica
• gos_lanex
• gos_ri1
• graphite
• graphitefoam
• hy906
• invar
• kapton
• lanex
• lanex2
• laservac (same as vacuum but with different name)
• leadtungstate
• lhconcrete
• lhc_rock
• lhe_1.9k
• limousmarl
• liquidhelium
• marl
• medex
• mild_steel
• niobium_2k
• nbti.1
• nbti_4k
• nbti_87k
• nb_2k (niobium_2k)
• nb_87k
• n-bk7
• perspex
• pet
• pet_lanex
• pet_opaque
• polyurethane
• quartz
• smco
• soil
• solidhydrogen
• solidnitrogen
• solidoxygen
• stainlesssteel
• stainless_steel_304L
• stainless_steel_304L_87K
• stainless_steel_304LN
• stainless_steel_304LN_87K
• ti_87k
• tungsten Heavy_Alloy
• ups923a
• vacuum
• water (G4_WATER)
• weightiron
• yag

6.20.6 Vacuum and Air

The default “vacuum” material used in all beam pipes is composed of H, C and O with the following fractions:

<table>
<thead>
<tr>
<th>Element</th>
<th>Mass Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>0.482</td>
</tr>
<tr>
<td>C</td>
<td>0.221</td>
</tr>
<tr>
<td>O</td>
<td>0.297</td>
</tr>
</tbody>
</table>

The default pressure is 1e-12 bar, the temperature is 300K and the density is 1.16336e-9 g/cm³.

“air” is the G4_AIR material. As of Geant4.10.04.p02 (see geant4/source/materials/src/G4NistMaterialBuilder.cc), it is composed of C, N, O, Ar with the following fractions:

<table>
<thead>
<tr>
<th>Element</th>
<th>Mass Fraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>0.000124</td>
</tr>
<tr>
<td>N</td>
<td>0.755267</td>
</tr>
<tr>
<td>O</td>
<td>0.231781</td>
</tr>
<tr>
<td>Ar</td>
<td>0.012827</td>
</tr>
</tbody>
</table>

It is a gas with density of 1.20479 mg/cm³.

6.21 Crystals

To use various crystal components in BDSIM such as crystalcol, a crystal definition must first be made. This contains all of the required information to construct the crystal. The following parameters are required:
<table>
<thead>
<tr>
<th><strong>Parameter</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>material</td>
<td>Material that the crystal will be composed of</td>
</tr>
<tr>
<td>data</td>
<td>Path to data files, including first part of file name</td>
</tr>
<tr>
<td>shape</td>
<td>Geometry used - one of (box, cylinder, torus)</td>
</tr>
<tr>
<td>lengthX</td>
<td>X-dimension full length [m]</td>
</tr>
<tr>
<td>lengthY</td>
<td>Y-dimension full length [m]</td>
</tr>
<tr>
<td>lengthZ</td>
<td>Z-dimension full length [m]</td>
</tr>
<tr>
<td>sizeA</td>
<td>Unit cell a dimension [m]*</td>
</tr>
<tr>
<td>sizeB</td>
<td>Unit cell b dimension [m]*</td>
</tr>
<tr>
<td>sizeC</td>
<td>Unit cell c dimension [m]*</td>
</tr>
<tr>
<td>alpha</td>
<td>Interaxial angle $\alpha$ in units of $\pi/2$</td>
</tr>
<tr>
<td>beta</td>
<td>Interaxial angle $\beta$ in units of $\pi/2$</td>
</tr>
<tr>
<td>gamma</td>
<td>Interaxial angle $\gamma$ in units of $\pi/2$</td>
</tr>
<tr>
<td>spaceGroup</td>
<td>Space grouping of lattice (integer)</td>
</tr>
<tr>
<td>bendingAngleYAxis</td>
<td>Angle that the crystal is bent about Y-axis [rad].</td>
</tr>
<tr>
<td>bendingAngleZAxis</td>
<td>Angle that the crystal is bent about Z-axis [rad].</td>
</tr>
</tbody>
</table>

- (*) Note, the units of metres may seem ridiculous, but the parser is consistently in S.I. (or as much as possible). We recommend using units in the parser such as Angstroms. See Coordinates & Units.

**Note:** Depending on the shape chosen, the geometry may or may not represent the bending angle. The `bendingAngleYAxis` is always supplied to the channelling physics process irrespective of the geometry. This is important to note that the crystal may be a box, but the ‘crystal’ inside (in terms of the physics process) is not related to the geometry and is bent. The physical geometry is merely a volume where the crystal parameters apply.

**Note:** If there is no vertical bending angle, the torus geometry will reduce to the cylinder geometry, as this is faster for tracking. Similarly, if the cylinder is used and there is no horizontal bending angle, a box will be used, as it’s not possible to construct a cylinder with an infinite bending radius.

It is entirely possible to add more shapes to the code. Please contact the developers **Feature Request**.

Examples:

```python
lovelycrystal: crystal, material = "G4_Si",
data = "data/Si220pl",
shape = "box",
lengthY = 5*cm,
lengthX = 0.5*mm,
lengthZ = 4*mm,
sizeA = 5.43*ang,
sizeB = 5.43*ang,
sizeC = 5.43*ang,
alpha = 1,
beta = 1,
gamma = 1,
spaceGroup = 227,
bendingAngleYAxis = 0.1*rad,
bendingAngleZAxis = 0;

uglycrystal: crystal, material = "G4_Si",
data = "data/Si220pl",
shape = "box",
lengthY = 5*cm,
lengthX = 0.5*mm,
lengthZ = 4*mm,
sizeA = 5.43*ang,
sizeB = 5.43*ang,
```

(continues on next page)
sizeC = 5.43*ang,
alpha = 1,
beta = 1,
gamma = 1,
spaceGroup = 227,
bendingAngleYAxis = -0.1*rad,
bendingAngleZAxis = 0;

More examples can be found in *Crystals*.

### 6.22 Regions

In Geant4, it is possible to drive different *regions* - each with their own production cuts and user limits. In BDSIM, there is one default region to which the options prodCutXXXX apply (see *Options*) and then the user may define additional regions and attach them to the objects desired. For example:

```plaintext
precisionRegion: cutsregion, prodCutProtons=1*m,
prodCutElectrons=10*m,
prodCutPositrons=10*m,
prodCutPhotons = 1*mm;
d1: drift, l=10*m, region="precisionRegion";
```

### 6.23 Bends

#### 6.23.1 Fringe Field Integral Behaviour

Fringe fields can be specified for dipole magnets through the parameters *hgap, fint* and *fintx*. *fint* is the fringe field integral, as described in *BDSIM Dipole Fringe* for the entrance face of the dipole. *fintx* is for the same, but for the exit face. Even when there is no pole face rotation, there is still a small fringe field effect.

If *fint* is specified but *fintx* is not, *fintx* will default to the same value as *fint*. If, however, *fintx* is set to 0, it will in fact be 0 and will not take the value of *fint*. This is the same default behaviour as MAD-X. MAD-X will write out a value of *fintx* as -1 in this case in any output. BDSIM will write out the value used, even if it’s equal to 0.

#### 6.23.2 Pole Face Rotations

The *bdsimtwo* integrator set (see *Integrator Sets*) provides tracking through a uniform magnetic field in a dipole. The field exists wherever the magnet exists; in the case of pole face rotations on the end of a dipole, the magnet is constructed with the appropriate angled face. The field therefore also has a hard edge with exactly no field immediately outside the magnet volume.

The tracking routine for dipoles in the *bdsimtwo* integrator set (see *BDSIM Dipole Rodrigues 2*) tracks the particle using the analytical helical solution in a pure magnetic field in Cartesian coordinates. This however does not agree with the tracking provided by MAD-X. We therefore provide an equivalent to MAD-X in *bdsimmatrix* integrator set (the default). The vertical focussing provided by the fringe field is the same in both cases.

The difference between the two is negligible for small pole face angles - for example, the LHC lattice shows no difference between the integrator sets (~14mrad bending angle and very low pole face angles). However, with higher angle bends and stronger pole face angles (maximum is up to 45 degrees), the difference is non-negligible.

The integrator for dipoles in *bdsimtwo* is computationally faster and should be used for lattices like the LHC, where speed matters and the pole faces are not a strong feature.
**Note:** To provide equivalent tracking to MAD-X with the `bdsimmatrix` integrator set, the magnet geometry is constructed with flat ends (i.e. always an sbend). Rbends are constructed as sbends with additional poleface rotation angles equal to half the bend angle. Instead of constructing the poleface geometry, the effect of a poleface rotation is applied in a thin fringefield magnet (1 micron thick by default) at the beginning (for non-zero e1) or at the end (for non-zero e2) of the dipole. In future, this will be decoupled to allow both the physical angled faces in the model as well as accurate tracking, using the MAD-X style matrix integrators.

### 6.23.3 Large Angle Bends

For a model that includes large angle bends (for example > 0.1 rad), the user should consider reducing the sampler diameter (see Sampler Dimensions and Geometry Options). This is because the default 5m width of a sampler may cause overlaps between samplers, or each sampler may record particles from multiple positions in the beam line.

One other point is that the parallel geometry used for curvilinear transforms (the “curvilinear world”) may overlap with other curvilinear elements earlier in the beam line. The size of the curvilinear world cylinders is based on the samplerDiameter and reducing the samplerDiameter will reduce its size. There is some automatic provision for this in BDSIM where the sampler diameter is automatically reduced when large angle bends are present in the lattice, but this is based on a heuristic, rather than direct overlap checks.

In short, we recommend running with option, checkOverlaps=1; once to verify there are no problems for a machine with large angle bends. If there are any overlaps, reduce the sampler diameter to the typical full width of a magnet.

### 6.24 Colours

Most items allow you to define a custom colour for them to aid in visualisation. This includes all magnets and collimators, the shield and degrader. The colour can be defined with red, green and blue components, as well as a level of transparency, alpha. RGB values can range from 0 to 255. Once defined, a colour may not be redefined. The syntax to define a colour is

```plaintext
NAME: newcolour, red=#, green=#, blue=#, alpha=#
```

Examples:

```plaintext
purple: newcolour, red=128, green=0, blue=128;
coll: rcol, l=0.2*m, xsize=5*cm, ysize=4*cm, colour="purple", material="copper";
```

and:

```plaintext
purple: newcolour, red=128, green=0, blue=128;
orange: newcolour, red=255, green=140, blue=0;
nicegreen: newcolour, red=0, green=128, blue=0;
d1: drift, l=1*m;
basebend: sbend, l=2*m, angle=0.9;
sb1: basebend, colour="purple";
sb3: basebend, colour="nicegreen";
sb4: basebend, colour="yellow";
sb5: basebend, colour="orange";
sb6: basebend, colour="red";
beaml ine: line=(d1,sb1,d1,basebend,d1,sb3,d1,sb4,d1,sb5,d1,sb6,d1);
use, beaml ine;
sample, all;
```

(continues on next page)
beam, particle="proton",
energy= 50*GeV;

This example is from bdsim/examples/features/visualisation/coloured_s bend.gmad and produces the model shown below.

- Colours can only be specified on an element-by-element basis.
- Colour names are case-sensitive.
- New colour names must not clash with predefined BDSIM colour names.

All available colours in BDSIM can be found by running BDSIM with the --colours command:

bdsim --colours

For convenience the predefined colours in BDSIM are:

<table>
<thead>
<tr>
<th>Name</th>
<th>R</th>
<th>G</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>LHCcoil</td>
<td>229</td>
<td>191</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>LHCcollar</td>
<td>229</td>
<td>229</td>
<td>229</td>
<td>1</td>
</tr>
<tr>
<td>LHCcopperskin</td>
<td>184</td>
<td>133</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>LHCyoke</td>
<td>0</td>
<td>127</td>
<td>255</td>
<td>1</td>
</tr>
<tr>
<td>LHCyoked</td>
<td>209</td>
<td>25</td>
<td>25</td>
<td>1</td>
</tr>
<tr>
<td>awakescreen</td>
<td>175</td>
<td>196</td>
<td>222</td>
<td>1</td>
</tr>
<tr>
<td>awakespectrometer</td>
<td>0</td>
<td>102</td>
<td>204</td>
<td>1</td>
</tr>
<tr>
<td>beampipe</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>1</td>
</tr>
<tr>
<td>black</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>blue</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>1</td>
</tr>
<tr>
<td>brown</td>
<td>114</td>
<td>63</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>coil</td>
<td>184</td>
<td>115</td>
<td>51</td>
<td>1</td>
</tr>
<tr>
<td>collimator</td>
<td>76</td>
<td>102</td>
<td>51</td>
<td>1</td>
</tr>
<tr>
<td>crystal</td>
<td>175</td>
<td>196</td>
<td>222</td>
<td>1</td>
</tr>
<tr>
<td>cyan</td>
<td>0</td>
<td>255</td>
<td>255</td>
<td>1</td>
</tr>
<tr>
<td>decapole</td>
<td>76</td>
<td>51</td>
<td>178</td>
<td>1</td>
</tr>
<tr>
<td>default</td>
<td>229</td>
<td>229</td>
<td>229</td>
<td>1</td>
</tr>
<tr>
<td>degrader</td>
<td>159</td>
<td>159</td>
<td>159</td>
<td>1</td>
</tr>
<tr>
<td>dipolefringe</td>
<td>229</td>
<td>229</td>
<td>229</td>
<td>1</td>
</tr>
<tr>
<td>drift</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>1</td>
</tr>
<tr>
<td>ecol</td>
<td>76</td>
<td>102</td>
<td>51</td>
<td>1</td>
</tr>
<tr>
<td>element</td>
<td>229</td>
<td>229</td>
<td>229</td>
<td>1</td>
</tr>
</tbody>
</table>

Continued on next page
6.25 Controlling Simulation Speed

The particle showers created in high energy particle interactions with matter can lead to a very large number of particles being produced in an event. These in turn each take time to track through the model and the computational time per event increases. When simulating a very high-energy scale, the user may not be interested in very low-energy particles, however these may dominate the simulation time.

To improve efficiency, there are several options the user can adjust. These however may reduce the accuracy of the results obtained and must be used cautiously and only where required.
6.25.1 Range Cuts

The production range cuts are the recommended method from Geant4, who strongly advocate these over energy-based tracking cuts. These produce the most accurate results while reducing simulation time. Approximately, these are the length a secondary must travel before interacting. If the secondary would not travel further than this (depending on the secondary species, physics lists, material and energy), the secondary will not be produced. These can be set globally or for a region (see Regions) that is attached to individual volumes through the “region” parameter for that accelerator element. In fact, a range cut always exists in Geant4 (to prevent infrared divergence) and is by default 0.7 mm.

\[
\text{rangecut}=3\text{ cm};
\]

```plaintext
option, prodCutPhotons = rangecut,
prodCutElectrons = rangecut,
prodCutPositrons = rangecut,
defaultRangeCut = rangecut;
```

**Warning:** The range cut should not be longer than the typical dimension of the objects (i.e. a range cut of 1 km is likely to produce very rough energy deposition around boundaries).

6.25.2 Minimum Kinetic Energy

The user may specify a minimum kinetic energy, below which any particle will be killed. This may break conservation of energy if used aggressively. The default is 0 eV, as all particles are tracked to zero energy (allowing for the above range cuts).

```plaintext
option, minimumKineticEnergy=10\text{ MeV};
```

**Warning:** This will affect the location of energy deposition - i.e. the curve of energy deposition of a particle showering in a material will be different.

6.25.3 Minimum Range

The user may specify a minimum range for a particle to travel. Any particles with step sizes proposed below this will be killed. Again, this can break energy conservation if used aggressively.

```plaintext
option, minimumRange=2\text{ cm};
```

**Warning:** This will affect the location of energy deposition - i.e. the curve of energy deposition of a particle showering in a material will be different.

6.26 Multiple Beam Lines

BDSIM has the ability to use multiple beam lines. This feature is still in development and is currently only for visualisation purposes. Secondary beam lines are placed either with respect to the world coordinate system or with respect to a particular element in the main beam line. A few caveats:

- Only for visualisation purposes
- Beam lines cannot be placed with respect to an element in another secondary beam line.
- Secondary beam lines are not suitable for tracking.
• Secondary beam lines are not sensitive to energy deposition, nor do they produce output.

• The user is entirely responsible for overlapping geometry. The visualiser will render the geometry, but of course it will not be suitable for simulations, as overlaps lead to volume navigation problems and incorrect tracking.

The user may use any sequence defined in the parser before the `use` command. The secondary beam line is produced by declaring a placement. The placement definition (see `Placements`) is augmented with the following parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence</td>
<td>Name of the sequence (with <code>line</code>) to use for the secondary beam line</td>
</tr>
<tr>
<td>referenceElement</td>
<td>The element in the sequence with respect to which the beam line will be placed</td>
</tr>
<tr>
<td>referenceElementNumber</td>
<td>The ( i )th instance of the element in the sequence (zero counting) ( \text{ (i.e. 2} \rightarrow \text{the 3rd instance of \text{referenceElement} in the sequence) } )</td>
</tr>
</tbody>
</table>

### 6.26.1 Examples

This example is shown in `bdsim/examples/features/geometry/10_multiple_beamlines`. It defines a simple beam line and two other sequences that are placed alongside it. Further explanation is given below the example.

```bash
d1: drift, l=1*m;
d2: drift, l=3*m;
d3: drift, l=5*m;
sbl1: sbend, l=1*m, angle=0.5;
sb2: sbend, l=1*m, angle=-0.5;
q1: quadrupole, l=0.2*m, k1=4.166666;
q2: quadrupole, l=0.2*m, k1=-4.166666;
fodo: line=(d1,q1,d1,q2);
mainLine: line=(d2,sb1,d2,sb2,d2,fodo,fodo);
auxLine1: line=(d3,sb1,d1,sb2,d1,fodo,d1);
auxLine2: line=(d1,sb1,d1,sb2,d1,fodo,d1);
auxLine3: line=(fodo);

use, mainLine;

beam, particle="e-",
  energy=3*GeV;

auxLine1Place: placement, sequence = "auxLine1",
   referenceElement = "d2",
   referenceElementNumber = 2,
   x = -5*cm,
   z = -1*m,
   axisAngle = 1,
   axisY = 1,
   angle = -0.2;

auxLine2Place: placement, sequence = "auxLine2",
   referenceElement = "d2",
   referenceElementNumber = 2,
   x = -10*cm,
   z = -1*m,
   axisAngle = 1,
   axisY = 1,
   angle = -0.5;
```

(continues on next page)
Firstly, a series of simple elements are defined (drifts, quadrupoles and bends). A simple sequence called *fodo* is defined and also the main beam line, called *mainLine*. After this, extra sequences are defined that we will use for secondary beam lines. The *use* command selects which beam line the simulation will be based on.

```
use, mainLine;
```

After this, the beam is defined (required for any simulation for rigidity calculations), then the placement of secondary beam lines.

The first placement *auxLine1Place* is a placement that will place the sequence named *auxLine1* with respect to the third instance of the element *d2* in the primary sequence (*mainLine*).

```
auxLine1Place: placement, sequence="auxLine1",
  referenceElement="d2",
  referenceElementNumber=2,
```

The placement is generally with respect to the centre of the element described in the primary beam line and along the direction it’s pointing. Without any displacement, the geometry would therefore overlap. Here, an offset and rotation are specified for this particular placement. An offset in *x* of -5 cm and -1 m in *z* is specified. The coordinate system is right-handed with positive-*z* pointing along the direction of travel in the beam line. A negative-*x* displacement is therefore to the right, looking along the direction of travel and 1 m in *z* is towards the beginning of the element from the centre. Rotations are described in *Placements*. Here, an axis angle rotation is used. The beam line is rotated about the unit Y-axis (local to that element) by -0.2 rad.

The second placement uses a different sequence, but in a similar fashion.

The third placement doesn’t specify a *referenceElement*, so the placement is with respect to the beginning of the beam line.

The model is shown below.

![Model Diagram]

The drift segments do not of course connect but are merely placed close to each other. In future, continuous vacuum points will be provided.
Currently, three formats are supported by BDSIM. GDML is recommended as this is thoroughly supported by Geant4 and the geometry extent can be automatically determined. More can be added in collaboration with the BDSIM development team - please see Feature Request.

**Warning:** The ggmad and Mokka formats are not currently developed or maintained in BDSIM.

### 7.1 GDML

GDML (Geometry Description Markup Language) is an XML schema for detector description. To use Geant4 and BDSIM needs to be built with GDML usage on (default true). For more information please refer to the GDML website and manual.

This format is widely supported and other geometry software may be able to export geometry in GDML format. The Geant4 GDML parser will not reload a volume if one by the same name is already loaded. Instead, it will use that volume. In the case of multiple GDML files being used in BDSIM, this would result in incorrect geometry. BDSIM includes a preprocessor using the xercesc library that will make a temporary copy of any GDML files loaded and prepend all names with the name of the element or placement being used. The user will not normally notice this and the temporary files are deleted after use.

The BDSIM GDML preprocessor has some limitations. We cannot support variables in values. In this case, the user should load a GDML file with Geant4 and re-export it. This will ‘flatten’ / resolve any variables, e.g.

```xml
<variable name="offsetX" value="3"/>
<position x="offsetX+3" y="0" z="-3"/>
```

would not work, as the variable “offsetX” is referred to in the value “x” in the position tag.

**Warning:** The Geant4 GDML parser typically requires internet access for the schema. To overcome this deficiency we provide a copy of the latest GDML schema in BDSIM, which it uses. If the user specifies a path on the file system in the GDML tag (presumably to their own modified schema) this will be used.

**Note:** For GDML geometry, we preprocess the input file prepending all names with the name of the element. This is to compensate for the fact that the Geant4 GDML loader does not handle unique file names. However, in the case of very large files with many, many vertices, the preprocessing can dominate. In this case, the option `preprocessGDML` should be turned off. The loading will only work with one file in this case.
7.2 GDML Preparation

A Python utility has been created to aid preparation, visualisation and overlap checking of GDML geometry. Please see Python Utilities for pyg4ometry.

7.3 ggmad

ggmad is a simple format used as wrapper to (some) Geant4 geometry classes. It can be used for specifying more or fewer simple geometries, such as collimators. Example:

```
Cons {
  x0=0,
y0=0,
z0=375,
  rmin1=100,
rmax1=500,
rmin2=5,
rmax2=500,
  z=125,
material=Graphite,
  phi0=0,
dphi=360,
  temperature=1
}
```

A file can contain several objects which will be placed sequentially into the volume.

**Note:** The user has to make sure that there is no overlap between them.

Available shapes are:

### 7.3.1 Box

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>x-origin</td>
</tr>
<tr>
<td>y0</td>
<td>y-origin</td>
</tr>
<tr>
<td>z0</td>
<td>z-origin</td>
</tr>
<tr>
<td>x</td>
<td>x-size</td>
</tr>
<tr>
<td>y</td>
<td>y-size</td>
</tr>
<tr>
<td>z</td>
<td>z-size</td>
</tr>
<tr>
<td>phi</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>theta</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>psi</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>material</td>
<td>Material name</td>
</tr>
</tbody>
</table>
### 7.3.2 Cons

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>x-origin</td>
</tr>
<tr>
<td>y0</td>
<td>y-origin</td>
</tr>
<tr>
<td>z0</td>
<td>z-origin</td>
</tr>
<tr>
<td>rmin1</td>
<td>Inner radius at start</td>
</tr>
<tr>
<td>rmax1</td>
<td>Outer radius at start</td>
</tr>
<tr>
<td>rmin2</td>
<td>Inner radius at end</td>
</tr>
<tr>
<td>rmax2</td>
<td>Outer radius at end</td>
</tr>
<tr>
<td>z</td>
<td>z-size</td>
</tr>
<tr>
<td>phi</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>theta</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>psi</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>phi0</td>
<td>Angle for start of sector</td>
</tr>
<tr>
<td>dphi</td>
<td>Angle swept by sector</td>
</tr>
<tr>
<td>material</td>
<td>Material name</td>
</tr>
</tbody>
</table>

### 7.3.3 Tubs

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>x-origin</td>
</tr>
<tr>
<td>y0</td>
<td>y-origin</td>
</tr>
<tr>
<td>z0</td>
<td>z-origin</td>
</tr>
<tr>
<td>rmin</td>
<td>Inner radius</td>
</tr>
<tr>
<td>rmax</td>
<td>Outer radius</td>
</tr>
<tr>
<td>z</td>
<td>z-size</td>
</tr>
<tr>
<td>phi</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>theta</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>psi</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>material</td>
<td>Material name</td>
</tr>
</tbody>
</table>

### 7.3.4 Trd

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x0</td>
<td>x-origin</td>
</tr>
<tr>
<td>y0</td>
<td>y-origin</td>
</tr>
<tr>
<td>z0</td>
<td>z-origin</td>
</tr>
<tr>
<td>x1</td>
<td>Half-length at wider side</td>
</tr>
<tr>
<td>x2</td>
<td>Half-length at narrower side</td>
</tr>
<tr>
<td>y1</td>
<td>Half-length at wider side</td>
</tr>
<tr>
<td>y2</td>
<td>Half-length at narrower side</td>
</tr>
<tr>
<td>z</td>
<td>z-size</td>
</tr>
<tr>
<td>phi</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>theta</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>psi</td>
<td>Euler angle for rotation</td>
</tr>
<tr>
<td>material</td>
<td>Material name</td>
</tr>
</tbody>
</table>
7.4 Mokka

This format is currently in the form of a dumped MySQL database format. Note that throughout any of the Mokka files, a # may be used to represent a commented line. There are three key stages, which are detailed in the following sections, that are required for setting up the Mokka geometry:

- Describing the geometry
- Creating a geometry list
- Defining a Mokka Element

7.4.1 Describing the geometry

An object must be described by creating a MySQL file containing commands that would typically be used for uploading/creating a database and a corresponding new table into a MySQL database. BDSIM supports only a few such commands - specifically the CREATE TABLE and INSERT INTO commands. When writing a table to describe a solid, there are some parameters that are common to all solid types (such as NAME and MATERIAL) and some that are more specific (such as those relating to radii for cone objects). A full list of the standard and specific table parameters, as well as some basic examples, are given below with each solid type. All files containing geometry descriptions must have the following database creation commands at the top of the file:

```
DROP DATABASE IF EXISTS DATABASE_NAME;
CREATE DATABASE DATABASE_NAME;
USE DATABASE_NAME;
```

A table must be created to allow for the insertion of the geometry descriptions. A table is created using the following MySQL compliant commands:

```
CREATE TABLE TABLE-NAME_GEOMETRY-TYPE (
 TABLE-PARAMETER VARIABLE-TYPE,
 TABLE-PARAMETER VARIABLE-TYPE,
 TABLE-PARAMETER VARIABLE-TYPE
);
```

Once a table has been created, values must be entered into it in order to define the solids and position them. The insertion command must appear after the table creation and must be followed by the MySQL compliant table insertion command:

```
INSERT INTO TABLE-NAME_GEOMETRY-TYPE VALUES(value1, value2, "char-value", ...);
```

The values must be inserted in the same order, as their corresponding parameter types are described in the table creation. Note that ALL length types must be specified in mm and that ALL angles must be in radians.

An example of two simple boxes with no visual attribute set is shown below. The first box is a simple vacuum cube, whilst the second is an iron box with length x = 10mm, length y = 150mm, length z = 50mm, positioned at x=1m, y=0, z=0.5m and with zero rotation:

```
CREATE TABLE mytable_BOX (
 NAME VARCHAR(32),
 MATERIAL VARCHAR(32),
 LENGTHX DOUBLE(10,3),
 LENGTHY DOUBLE(10,3),
 LENGTHZ DOUBLE(10,3),
 POSX DOUBLE(10,3),
 POSY DOUBLE(10,3),
 POSZ DOUBLE(10,3),
 ROTPSI DOUBLE(10,3),
 ROTTHETA DOUBLE(10,3),
 ROTPHI DOUBLE(10,3)
);
```

(continues on next page)
Further examples of the Mokka geometry implementation can be found in the examples/features/geometry/Mokka/General directory. See the common table parameters and solid type sections below for more information on the table parameters available for use.

**Common Table Parameters**

The following is a list of table parameters that are common to all solid types, either as an optional or mandatory parameter:

- **NAME**
  - Variable type: VARCHAR(32)
  - Optional parameter
  - If supplied, then the Geant4 LogicalVolume associated with the solid will be labelled with this name. The default is set to be the table’s name plus an automatically assigned volume number.

- **MATERIAL**
  - Variable type: VARCHAR(32)
  - Optional parameter
  - If supplied, then the volume will be created with this material type - note that the material must be given as a character string inside double quotation marks("`). The default material is set as Vacuum.

- **PARENTNAME**
  - Variable type: VARCHAR(32)
  - Optional parameter
  - If supplied, then the volume will be placed as a daughter volume to the object with ID equal to PARENTNAME. The default parent is set to be the Component Volume. Note that if PARENTID is set to the Component Volume, then POSZ will be defined with respect to the start of the object. Else POSZ will be defined with respect to the centre of the parent object.

- **INHERITSTYLE**
  - Variable type: VARCHAR(32)
  - Optional parameter to be used with PARENTNAME.
  - If set to “SUBTRACT” then instead of placing the volume within the parent volume as an inherited object, it will be subtracted from the parent volume in a Boolean solid operation. The default for this value is set to “” - which sets to the usual mother/daughter volume inheritance.

- **ALIGNIN**
  - Variable type: INTEGER(11)
  - Optional parameter
  - If set to 1 then the placement of components will be rotated and translated such that the incoming beamline will pass through the z-axis of this object. The default is set to zero.

- **ALIGNOUT**
  - Variable type: INTEGER(11)
  - Optional parameter
  - If set to 1 then the placement of the next beamline component will be rotated and translated such that the outgoing beamline will pass through the z-axis of this object. The default is set to zero.

- **SETSENSITIVE**
  - Variable type: INTEGER(11)
Optional parameter
If set to 1 then the object will be set up to register energy depositions made within it and also record the z-position at which this deposition occurs. This information will be saved in the ELoss Histogram if using ROOT output. The default is set to zero.

• MAGTYPE
  Variable type: VARCHAR(32)
  Optional parameter
  If supplied, then the object will be set up to produce the appropriate magnetic field using the supplied K1 or K2 table parameter values. Three magnet types are available - “QUAD”, “SEXT” and “OCT”. The default is set to no magnet type. Note that if MAGTYPE is set to a value whilst K1/K2/K3 are not set, then no magnetic field will be implemented.

• K1
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set to a value other than zero, in conjunction with MAGTYPE set to “QUAD”, then a quadrupole field with this K1 value will be set up within the object. Default is set to zero.

• K2
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set to a value other than zero, in conjunction with MAGTYPE set to “SEXT”, then a sextupole field with this K2 value will be set up within the object. Default is set to zero.

• K3
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set to a value other than zero, in conjunction with MAGTYPE set to “OCT”, then a sextupole field with this K3 value will be set up within the object. Default is set to zero.

• POSX, POSY, POSZ
  Variable type: DOUBLE(10,3)
  Required parameters
  They form the position in mm used to place the object in the component volume. POSX and POSY are defined with respect to the centre of the component volume and with respect to the component volume’s rotation. POSZ is defined with respect to the start of the component volume. Note that if the object is being placed inside another volume using PARENTNAME then the position will refer to the centre of the parent object.

• ROTPSI, ROTTHETA, ROTPHI
  Variable type: DOUBLE(10,3)
  Optional parameters
  These are the Euler angles in radians used to rotate the object before it is placed. The default is set to zero for each angle.

• RED, BLUE, GREEN
  Variable type: DOUBLE(10,3)
  Optional parameters
  They are the RGB colour components assigned to the object and should be a value between 0 and 1. The default is set to zero for each colour.

• VISATT
  Variable type: VARCHAR(32)
  Optional parameter
  This is the visual state setting for the object. Setting this to “W” results in a wireframe display of the object. “S” produces a shaded solid and “I” leaves the object invisible. The default is set to be solid.

• FIELDX, FIELDY, FIELDZ

Chapter 7. External Geometry Formats
Variable type: DOUBLE(10,3)

Optional parameters
They can be used to apply a uniform field to any volume, with default units of Tesla. Note that if there is a solenoid field present throughout the entire element, then this uniform field will act in addition to the solenoid field.

‘Box’ Solid Types

Append _BOX to the table name in order to make use of the G4Box solid type. The following table parameters are specific to the box solid:

- LENGTHX, LENGTHY, LENGTHZ
  Variable type: DOUBLE(10,3)
  Required parameters
  These values will be used to specify the box’s dimensions.

‘Trapezoid’ Solid Types

Append _TRAP to the table name in order to make use of the G4Trd solid type - which is defined as a trapezoid with the X and Y dimensions varying along z-functions. The following table parameters are specific to the trapezoid solid:

- LENGTHXPLUS
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the x-extent of the box’s dimensions at the surface positioned at +dz.

- LENGTHXPMINUS
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the x-extent of the box’s dimensions at the surface positioned at -dz.

- LENGTHYPLUS
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the y-extent of the box’s dimensions at the surface positioned at +dz.

- LENGTHYPMINUS
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the y-extent of the box’s dimensions at the surface positioned at -dz.

- LENGTHZ
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the z-extent of the box’s dimensions.

‘Cone’ Solid Types

Append _CONE to the table name in order to make use of the G4Cons solid type. The following table parameters are specific to the cone solid:

- LENGTH
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the z-extent of the cone’s dimensions.
• **RINNERSTART**
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set, this value will be used to specify the inner radius of the start of the cone. The default value is zero.

• **RINNEREND**
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set, this value will be used to specify the inner radius of the end of the cone. The default value is zero.

• **ROUTERSTART**
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the outer radius of the start of the cone.

• **ROUTEREND**
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the outer radius of the end of the cone.

• **STARTPHI**
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set, this value will be used to specify the starting angle of the cone. The default value is zero.

• **DELTAPHI**
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set, this value will be used to specify the delta angle of the cone. The default value is 2*pi.

### ‘Torus’ Solid Types

Append _TORUS to the table name in order to make use of the G4Torus solid type. The following table parameters are specific to the torus solid:

• **RINNER**
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set, this value will be used to specify the inner radius of the torus tube. The default value is zero.

• **ROUTER**
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the outer radius of the torus tube.

• **RSWEPT**
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the swept radius of the torus. It is defined as being the distance from the centre of the torus ring to the centre of the torus tube. For this reason, this value should not be set to less than ROUTER.

• **STARTPHI**
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set, this value will be used to specify the starting angle of the torus. The default value is zero.

• **DELTAPHI**
Optional parameter

If set, this value will be used to specify the delta swept angle of the torus. The default value is \(2\pi\).

**'Polycone' Solid Types**

Append _POLYCONET to the table name in order to make use of the G4Polycone solid type. The following table parameters are specific to the polycone solid:

- **NZPLANES**
  Variable type: INTEGER(11)
  Required parameter
  This value will be used to specify the number of z-planes to be used in the polycone. This value must be set to greater than 1.

- **PLANEPOS1, PLANEPOS2, ..., PLANEPOSN**
  Variable type: DOUBLE(10,3)
  Required parameters
  These values will be used to specify the z-position of the corresponding z-plane of the polycone. There should be as many PLANEPOS parameters set as the number of z-planes. For example, three z-planes will require that PLANEPOS1, PLANEPOS2, and PLANEPOS3 are all set up.

- **RINNER1, RINNER2, ..., RINNERN**
  Variable type: DOUBLE(10,3)
  Required parameters
  These values will be used to specify the inner radius of the corresponding z-plane of the polycone. There should be as many RINNER parameters set as the number of z-planes. For example, three z-planes will require that RINNER1, RINNER2, and RINNER3 are all set up.

- **ROUTER1, ROUTER2, ..., ROUTERN**
  Variable type: DOUBLE(10,3)
  Required parameters
  These values will be used to specify the outer radius of the corresponding z-plane of the polycone. There should be as many ROUTER parameters set as the number of z-planes. For example, three z-planes will require that ROUTER1, ROUTER2, and ROUTER3 are all set up.

- **STARTPHI**
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set, this value will be used to specify the starting angle of the polycone. The default value is zero.

- **DELTAPHI**
  Variable type: DOUBLE(10,3)
  Optional parameter
  If set, this value will be used to specify the delta angle of the polycone. The default value is \(2\pi\).

**'Elliptical Cone' Solid Types**

Append _ELLIPticalCONE to the table name in order to make use of the G4Ellipticalcone solid type. The following table parameters are specific to the elliptical cone solid:

- **XSEMIAXIS**
  Variable type: DOUBLE(10,3)
  Required parameter
  This value will be used to specify the Semiaxis in X.

- **YSEMIAXIS**
Variable type: DOUBLE(10,3)
Required parameter
This value will be used to specify the Semiaxis in Y.

• LENGTHZ
Variable type: DOUBLE(10,3)
Required parameter
This value will be used to specify the height of the elliptical cone.

• ZCUT
Variable type: DOUBLE(10,3)
Required parameter
This value will be used to specify the upper cut plane level.

Note that the above parameters are used to define an elliptical cone with the following parametric equations (in the usual Geant4 way):

\[
X = X_{\text{SEMIAXIS}} \times \left( \frac{\text{LENGTHZ} - u}{u} \right) \times \cos(v)
\]

\[
Y = Y_{\text{SEMIAXIS}} \times \left( \frac{\text{LENGTHZ} - u}{u} \right) \times \sin(v)
\]

\[
Z = u
\]

where \(v\) is between 0 and 2 \(\pi\) and \(u\) between 0 and \(h\), respectively.

### 7.4.2 Creating a geometry list

A geometry list is a simple file consisting of a list of file names that contain geometry descriptions. This is the file that should be passed to the GMAD file when defining the Mokka element. An example of a geometry list containing 'boxes.sql' and 'cones.sql' would be:

```plaintext
# '#' symbols can be used for commenting out an entire line
/directory/boxes.sql
/directory/cones.sql
```

### 7.4.3 Defining a Mokka element

The Mokka element can be defined by the following command:

```plaintext
collimator : element, geometry=mokka:coll_geomlist.sql
```
A BDSIM model can be prepared either manually in a hand-written fashion, or using a provided suite of python tools to automatically convert the description of an accelerator lattice from other formats, such as MAD-X, MAD8 or Transport, to that of BDSIM - gmad.

The automatic conversion is typically achieved by preparing a ‘rendered’ or ‘flat’ output description of each element in the accelerator from whatever optics program you use to design the accelerator, then converting this using our Python utility pybdsim - see Python Utilities.

Additionally, the python tools can be used to programmatically create an accelerator lattice of your own design, which is described in Python Builder.

8.1 Manual Preparation

An input gmad (text) file can easily be prepared manually in your favourite text editor by defining (in order):

1. Individual elements
2. The order they appear in a line
3. Which period to use - the above line
4. Options such as the physics list and tracking cuts
5. Input beam distribution

Please see Model Description - Input Syntax for a description of the input syntax.

8.2 MAD-X Conversion

A MAD-X lattice can be easily converted to a BDSIM gmad input file using the supplied python utilities. This is achieved by:

1. Preparing a TFS file with MAD-X containing all Twiss table information
2. Converting the TFS file to gmad using pybdsim.

The Twiss file can be prepared by appending the following MAD-X syntax to the end of your MAD-X script:

```madx
select,flag-twiss, clear;
twiss,sequence=SEQUENCENAME, file=twiss.tfs;
```

where SEQUENCENAME is the name of the sequence in MAD-X. By not specifying the output columns, a very large file is produced containing all possible columns. This is required to successfully convert the lattice. If the tfs file contains insufficient information, pybdsim will not be able to convert the model. Use pybdsim in Python to convert the model:

```python
>>> import pybdsim
>>> pybdsim.Convert.MadxTfs2Gmad('inputfile.tfs', 'latticev1')
```
This is described in more detail in the dedicated pybdsim documentation http://www.pp.rhul.ac.uk/bdsim/pybdsim/convert.html.

**Note:** The python utilities require “.tfs” suffix as the file type to work properly.

To convert the tfs file, pybdsim should be used. pybdsim along with other utilities can be found in the utils directory in the BDSIM source directory.

**Note:** If these folders are empty, please update the submodules as described in *From the GIT Repository*.

### 8.3 MAD8 Conversion

This can be prepared in a similar fashion to a MAD-X model. The user must have our pymad8 and pybdsim packages (see *Python Utilities*).

This is described in more detail in the dedicated pybdsim documentation http://www.pp.rhul.ac.uk/bdsim/pybdsim/convert.html.

### 8.4 Python Builder

Using the classes in the pybdsim package that are used in the conversion process, the user may create a BDSIM model programmatically, i.e. the user can write a Python script to create the lattice they want.

This is described in more detail in the dedicated pybdsim documentation http://www.pp.rhul.ac.uk/bdsim/pybdsim/builder.html

The pymadx package also has a similar functionality and is documented in its dedicated documentation http://www.pp.rhul.ac.uk/bdsim/pymadx/moduledocs.html#module-pymadx.Builder.

### 8.5 Optical Validation

Once a model has been prepared or converted, it is often useful to compare the optical functions of the accelerator to a known reference. This ensures that the model has been prepared correctly. If, for example, a drift between magnets was too short or magnetic strengths were wrong, the optical functions would disagree and this would clearly indicate where the problem lies. This is an important validation step before physics studies begin, where such errors may not be so clear.

Simulating the passage of several thousand particles validates optical functions with a distribution according to the core Gaussian distribution. Samplers are attached to all elements to record the particle distribution after each element. A provided optical analysis is used to calculate optical functions and moments of the particle coordinates that can then be compared to a reference.

#### 8.5.1 Recommended Options for Generating Optics

Options in gmad:

```python
option, physicsList="em",
    stopSecondaries=1,
    aper1=5*cm;

sample, all;
```
• If no physics is present, particles will not interact with material and will not stop if lost.
• ‘Stop secondaries’ will kill any secondary particles - faster and doesn’t pollute data.
• A large aperture size should be used throughout to avoid clipping the beam.
• Samplers should be attached to all elements to record the beam distribution after each element.

**Note:** Attaching samplers to all elements vastly increases the output data file size and the user should generally only attach samplers to points of specific interest.

A Gaussian beam according to the Twiss parameters at the start of the beam line should be used. An emittance should be chosen that is used in the source of optics (i.e. in the MAD-X model and therefore appears in the header of the TFS Twiss output), but an emittance that ensures the beam size is small enough throughout the machine to avoid particles clipping.

A target number of particles is 1000 for linear optics with no energy spread and approximately 10000 for a beam with energy spread. The optical analysis calculates the statistical uncertainty associated with estimating each parameter, given there is a finite number of particles. With approximately 1000 particles, it is very unlikely to have any particles beyond $3\sigma$ for a Gaussian distribution, therefore as long as most apertures are above this, no particles should be lost.

### 8.5.2 Generating Optics Data

BDSIM should be executed in batch mode running between 1000 and 10000 particles (for example).

```
    bdsim --file=mymodel.gmad --outfile=op1 --batch --ngenerate=2000
```

### 8.5.3 Analysing Optics Data

The output file can then be analysed with `rebdsimOptics`. `rebdsimOptics` is a simple interface to `rebdsim` to calculate optical functions only. It is possible to run `rebdsim` with an analysis configuration text file specifying `CalculateOpticalFunctions` (see Analysis Configuration File). Rebdsim takes arguments as the input file(s) and the desired output file name for the optical functions. The optical functions are written to a separate ROOT format file only containing the optical functions.

```
    rebdsimOptics op1.root op1_optics.root
```

To calculate the optical functions, the emittance is calculated from the distribution at each sampler. In this case, only linear optics should be used, as non-linear optical elements, such as sextupoles and higher order magnets, couple the emittance in horizontal and vertical directions and the calculation is invalid for Twiss parameters. The default option in `rebdsimOptics` is to calculate the emittance only once for the first sampler and assume this for all subsequent samplers. In the case of acceleration, or where the geometrical emittance is expected to change, the emittance can be calculated at each sampler freshly. The executable option `--emittanceOnFly` should be used.

```
    rebdsimOptics op1.root op1_optics.root --emittanceOnFly
```

**Note:** When using rebdsimOptics to analyse multiple files for optics, the input file name should be surrounded with inverted commas - i.e. `rebdsimOptics "*.root" optics.root`.

### 8.5.4 Loading Optical Data

After calculating the optical functions, the data file can be loaded in ROOT manually or using the provided `pybdsim` utility in Python using ROOT.
This provides arrays of all the optical functions in a dictionary.

### 8.5.5 Comparison of Optics

For each of the formats BDSIM supports for converting models, there is a comparison plotting script in *pybdsim* to allow easy comparison. For MAD-X conversion, for example, the Twiss output in a TFS file can be used.

```python
>>> import pybdsim
>>> pybdsim.Compare.MadxVsBDSIM('madxtwiss.tfs', 'op1_optics.root')
```

This will create a series of plots with both the optical functions from MAD-X and those calculated by *rebdsimOptics* on the same plot with a colour machine diagram on top. A few example plots are shown below.

All plots are also written by default to a single pdf file whose name is by default based on the the *rebdsimOptics* output file name given to the comparison. In this case, it would be *op1_optics.pdf*.

![Example comparison of beam centroids for ATF2.](image)

Fig. 1: Example comparison of beam centroids for ATF2.

### 8.5.6 Comparison Notes

It is worth noting that we are drawing a sample of particle coordinates from a probability distribution and then calculating moments of the sample distribution to estimate the original parameters of the probability distribution. With a higher number of samples, the estimate improves but is still an estimate. The optical comparison is useful to validate the model preparation and extremely small differences in optical functions should not be interpreted as poor tracking.

### 8.5.7 Advanced Comparison

For very precise comparison, there are two further possible options that can be used.
Fig. 2: Example comparison of beam size for ATF2.

Fig. 3: Example comparison of $\sigma_{xp,yp}$ for ATF2.
Fig. 4: Example comparison of Twiss $\beta$ functions.

Fig. 5: Example comparison of Twiss $\alpha$ functions.
1) BDSIM can be compiled with double precision output as opposed to the default floating-point precision. This is a CMake option (see *Configuring the BDSIM Build with CMake*). The option `ROOT_DOUBLE_OUTPUT` should be turned on and BDSIM recompiled and installed.

2) `beam, offsetSampleMean=1` should be used in the input gmad. This precalculates all coordinates of the particles and subtracts the small sample mean from each coordinate, removing any small systematic offset at the beginning that typically propagates throughout the beam line. This will not work in the visualiser and breaks the strong-reproducibility in BDSIM - use with caution.
By default, BDSIM writes an output file with a summary of the model created, the particle coordinates generated, the options used, and basic energy deposition histograms. This can be turned off by executing BDSIM with:

```
bdsim --output=none
```

The output file name can be specified with:

```
bdsim --outfile="mydesiredname"
```

with no extension.

### 9.1 File Writing Policy

- The default file name is “output”.
- If no output file name is given and there is already a file called “output”, a suffix with an integer will be added, i.e. “output-1”.
- BDSIM will overwrite an output file if `--outfile` is supplied with the same name again.
- The behaviour is the same in both visualiser mode and batch mode.
- A new output file is created for each `/run/beamOn` command in the visualiser.

### 9.2 Output Information

The following extra information can be optionally recorded from a BDSIM simulation:

- Particle coordinates at a plane after each element - ‘sampler’ information.
- Particle coordinates at a plane that is placed in the world by the user - ‘samplerplacement’ (see [Output at an Arbitrary Plane - User Placed Sampler](#)).
- Energy deposition ‘hits’ from any component, the air or the beam pipe vacuum.
- Trajectories of all or certain particles (optional - see [Output Options](#)).
- Detailed information from hits in a collimator - see [Output Options](#).
- A single 3D histogram of any hits in the simulation (optional - see [Scoring Map](#)).

Samplers are ‘attached’ to a beam line element by using the sample command:

```
sample, range=<element_name>;
```

See [Output at a Plane - Samplers](#) for more details.

All components are sensitive to energy deposition by default. Any energy deposition will therefore be recorded in the output as well as stored in pre-made energy deposition histograms per event.
Trajectories are vectors of trajectory points that record the information about a particle at each step in the simulation. This records information, such as: all coordinates, particle type, state and the physics process that determined that step.

9.3 Particle Identification

BDSIM uses the standard Particle Data Group identification numbers for each particle type, similarly to Geant4. These are typically referred to as “partID”. A table of the particles and explanation of the numbering scheme can be found online:


Notes:

- These are integers.
- A negative value represents the opposite charge from the definition of the particle, but which doesn’t necessarily mean it’s negatively charged.

A table of common particles is listed below:

<table>
<thead>
<tr>
<th>Name</th>
<th>PDG ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>proton</td>
<td>2212</td>
</tr>
<tr>
<td>electron</td>
<td>11</td>
</tr>
<tr>
<td>positron</td>
<td>-11</td>
</tr>
<tr>
<td>gamma or photon</td>
<td>22</td>
</tr>
<tr>
<td>neutron</td>
<td>2112</td>
</tr>
<tr>
<td>pion positive</td>
<td>211</td>
</tr>
<tr>
<td>pion negative</td>
<td>-211</td>
</tr>
<tr>
<td>pion zero</td>
<td>111</td>
</tr>
<tr>
<td>muon negative</td>
<td>13</td>
</tr>
<tr>
<td>muon positive</td>
<td>-13</td>
</tr>
</tbody>
</table>

9.4 Output Data Selection & Reduction

Not all the variables in the output are filled by default, but are kept (empty) to maintain a consistent structure (as much as possible). The default level of output is judged to be the most commonly useful for the purpose of BDSIM but there are many extra options to control the detail of the output as well as the ability to turn bits off. This granularity is very useful when you have made small studies with the options you desire and now want to scale up the simulation to large statistics and the size of the data may become difficult to deal with. At this point, the user can turn off any data they may not need to save space.

If some output is not required, BDSIM will not generate the ‘hit’ information with sensitive detector classes automatically to improve computational speed and reduce memory usage during the simulation. This is handled automatically in BDSIM.

It is thoroughly recommend to consult all the options at Output Options. However, consider the following points to reduce output data size:

- If energy loss hits are not required (e.g. maybe only the pre-made histograms will suffice), turn these off with the option storeEloss.
- Eloss normally dominates the size of the output file as it has the largest number of hits with typically $10^4$ energy deposition hits per primary.
- By default some basic information is store in “Geant4Data” for all particles used in the simulation. For a big study, it is worth turning this off as it’s replicated in every file.
9.5 Output Files

This section only describes the structure. Loading and analysis instructions can be found in *Output Analysis*. The output format ‘rootevent’ is written to a ROOT file. This format is preferred as it lends itself nicely to particle physics information; is stored as compressed binary internally; and can store and load complex custom structures.

- Units, unless specified, are SI (i.e. m, rad).
- Energy is in GeV and is the total energy of a particle.
- Time is measured in nanoseconds.
- Small letters denote local (to that object) coordinates, whereas capital letters represent global coordinates.

Not all information described may be written by default. Options described in *Output Options* allow control over what is stored. The default options give a detailed picture with an acceptable file size. The true amount of information produced in the simulation of every particle and the steps taken is tremendous and cannot be usefully stored.

<table>
<thead>
<tr>
<th>Format</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>--output=none</td>
<td>No output is written</td>
</tr>
<tr>
<td>ROOT Event (Default)</td>
<td>--output=rootevent</td>
<td>A ROOT file with details of the model built, options used, seed states, and event-by-event information (default and recommended).</td>
</tr>
</tbody>
</table>

As a general guideline, the following naming conventions are used:

<table>
<thead>
<tr>
<th>Short Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phits</td>
<td>Primary hits</td>
</tr>
<tr>
<td>Ploss</td>
<td>Primary losses</td>
</tr>
<tr>
<td>Eloss</td>
<td>Energy loss</td>
</tr>
<tr>
<td>PE</td>
<td>Per element</td>
</tr>
<tr>
<td>Coll</td>
<td>Collimator</td>
</tr>
</tbody>
</table>

**Note:** A “hit” is the point of first contact, whereas a “loss” is the last point that particle existed - in the case of a primary it is where it stopped being a primary.

**Note:** Energy loss is the energy deposited by particles along their step.

9.6 Basic Data Inspection

To view the data as shown here, we recommend using a ROOT tree browser - *TBrowser*. Start ROOT (optionally with the file path specified to put it at the top of the list).

- The -l option stops the logo splash screen coming up and is slightly quicker.

While in the ROOT interpreter, enter the following command to ‘construct’ a TBrowser object.

Double-click the file and then the ‘Trees’ (small folders with green leaf on them) to explore the hierarchy of the file. Eventually, individual variables can be double-clicked on to give a preview histogram on-the-fly that is a
histogram of all entries in the Tree (i.e. all events in the Event Tree). If the variable is a vector, each item in the vector is entered (‘filled’) into the histogram.

**Note:** If a file is open in ROOT in a TBrowser but has been overwritten externally, it will not show the correct contents - close the TBrowser and ROOT and reopen it.

### 9.7 Structure Of Output

BDSIM uses a series of classes to accumulate information about a Geant4 Run and Event. Instances of these classes are ‘filled’ with information during the simulation and copied to the output.

In the case of the ROOT event output format, these classes are stored directly in the file so that the same classes can be used by the output analysis tool (rebdsim) to read and process the data. A BDSIM ROOT event file has the following structure:
The file consists of four ROOT ‘trees’ each with ‘branches’ that represent instances of the BDSIM classes. The trees are:

<table>
<thead>
<tr>
<th>Tree Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>Details about the file type and software versions</td>
</tr>
<tr>
<td>Geant4Data</td>
<td>Information about all particles and ions used in the simulation</td>
</tr>
<tr>
<td>Beam</td>
<td>A record of all options associated with the beam definition</td>
</tr>
<tr>
<td>Options</td>
<td>A record of all options used by BDSIM</td>
</tr>
<tr>
<td>Model</td>
<td>A record of the lengths and placement transforms of every element built by BDSIM in the accelerator beam line suitable for recreating global coordinates or visualising trajectories</td>
</tr>
<tr>
<td>Run</td>
<td>Information collected per Run</td>
</tr>
<tr>
<td>Event</td>
<td>Information collected per Event</td>
</tr>
</tbody>
</table>

9.7.1 Header Tree

The header tree contains a single branch called “Header,” (note the “.”). This branch represents a single instance of BDSOutputROOTEventHeader. This stores the various software libraries BDSIM is compiled against, as well as the BDSIM version. It also stores the time the file was created and the file type, i.e. whether the file is from BDSIM, rebdsim or rebdsimCombine.

**BDSOutputROOTEventHeader**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdsimVersion</td>
<td>std::string</td>
<td>Version of BDSIM used</td>
</tr>
<tr>
<td>geant4Version</td>
<td>std::string</td>
<td>Version of Geant4 used</td>
</tr>
<tr>
<td>rootVersion</td>
<td>std::string</td>
<td>Version of ROOT used</td>
</tr>
<tr>
<td>clhepVersion</td>
<td>std::string</td>
<td>Version of CLHEP used</td>
</tr>
<tr>
<td>timeStamp</td>
<td>std::string</td>
<td>Time and date file was created</td>
</tr>
<tr>
<td>fileType</td>
<td>std::string</td>
<td>String describing what stage of simulation the file came from</td>
</tr>
<tr>
<td>dataVersion</td>
<td>int</td>
<td>BDSIM data format version</td>
</tr>
<tr>
<td>doublePrecisionOutput</td>
<td>bool</td>
<td>Whether BDSIM was compiled with double precision for output</td>
</tr>
</tbody>
</table>
9.7.2 Geant4Data Tree

The Geant4Data tree contains a single branch called “Geant4Data.” (note the “.”). This branch represents a single instance of BDSOutputROOTGeant4Data. This stores two maps (like dictionaries) of the particle and ion information for each particle / ion used in the simulation (only, i.e. not all that Geant4 supports). The map goes from an integer, the Particle Data Group ID, to the particle or ion info that are stored in simple C++ structures called BDSOutputROOTGeant4Data::ParticleInfo and BDSOutputROOTGeant4Data::IonInfo respectively. These contain the name, charge, mass, and in the case of ions, additionally A and Z. The both have a function called rigidity that can calculate the rigidity of the particle for a given total energy - this is used during the execution of BDSIM when rigidities are requested to be stored.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>particles</td>
<td>std::map&lt;int, BDSOutputROOTGeant4Data::ParticleInfo&gt;</td>
<td>Map of PDG ID to particle info.</td>
</tr>
<tr>
<td>ions</td>
<td>std::map&lt;int, BDSOutputROOTGeant4Data::IonInfo&gt;</td>
<td>Map of PDG ID to ion info.</td>
</tr>
</tbody>
</table>

**ParticleInfo Struct**

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>std::string</td>
<td>Name of particle</td>
</tr>
<tr>
<td>charge</td>
<td>int</td>
<td>Particle charge in units of e</td>
</tr>
<tr>
<td>mass</td>
<td>double</td>
<td>Particle Data Group mass in GeV</td>
</tr>
</tbody>
</table>
9.7.3 Beam Tree

The beam tree contains a single branch called “Beam.” (note the “.”). This branch represents an instance of `parser/BeamBase.hh`. The tree typically contains one entry, as only one definition of the beam was used per execution of BDSIM.

9.7.4 Options Tree

The options tree contains a single branch called “Options.” (note the “.”). This branch represents an instance of `parser/OptionsBase.hh`. The tree typically contains one entry, as only one set of options were used per execution of BDSIM.
9.7.5 Model Tree

This tree contains a single branch called “Model”. This branch represents an instance of `include/BDSOutputROOTEventModel.hh`. There is also typically one entry, as there is one model. Note that some variables here appear as ‘leaf’ icons and some as ‘branch’ icons. This is because some of the variables are vectors.

BDSOutputROOTEventModel

One entry in the model tree represents one beam line.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>samplerNamesUnique</td>
<td>std::vector<a href="">std::string</a></td>
<td>The unique names of each of the samplers. These are identical to the names of the sampler branches found in the Event tree.</td>
</tr>
<tr>
<td>componentName</td>
<td>std::vector<a href="">std::string</a></td>
<td>The beamline component names</td>
</tr>
<tr>
<td>placementName</td>
<td>std::vector<a href="">std::string</a></td>
<td>Unique name for each placement</td>
</tr>
<tr>
<td>componentType</td>
<td>std::vector<a href="">std::string</a></td>
<td>Beamline component type; “drift”, “sbend”, etc.</td>
</tr>
<tr>
<td>length</td>
<td>std::vector&lt;float&gt;</td>
<td>Component length (metres)</td>
</tr>
<tr>
<td>staPos</td>
<td>std::vector&lt;TVector3&gt;</td>
<td>Global coordinates of start of beamline element (metres)</td>
</tr>
<tr>
<td>midPos</td>
<td>std::vector&lt;TVector3&gt;</td>
<td>Global coordinates of middle of beamline element (metres)</td>
</tr>
<tr>
<td>endPos</td>
<td>std::vector&lt;TVector3&gt;</td>
<td>Global coordinates of end of beamline element (metres)</td>
</tr>
<tr>
<td>staRot</td>
<td>std::vector&lt;TRotation&gt;</td>
<td>Global rotation for the start of this beamline element</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>midRot</td>
<td>std::vector&lt;TRotation&gt;</td>
<td>Global rotation for the middle of this beamline element</td>
</tr>
<tr>
<td>endRot</td>
<td>std::vector&lt;TRotation&gt;</td>
<td>Global rotation for the end of this beamline element</td>
</tr>
<tr>
<td>staRefPos</td>
<td>std::vector&lt;TVector3&gt;</td>
<td>Global coordinates for the start of the beamline elements along the reference trajectory and without any tilt or rotation from the component</td>
</tr>
<tr>
<td>midRefPos</td>
<td>std::vector&lt;TVector3&gt;</td>
<td>Global coordinates for the middle of the beamline elements along the reference trajectory and without any tilt or rotation from the component</td>
</tr>
<tr>
<td>endRefPos</td>
<td>std::vector&lt;TVector3&gt;</td>
<td>Global coordinates for the start of the beamline elements along the reference trajectory and without any tilt or rotation from the component</td>
</tr>
<tr>
<td>staRefRot</td>
<td>std::vector&lt;TRotation&gt;</td>
<td>Global rotation matrix for start of the beamline elements along the reference trajectory and without any tilt or rotation from the component</td>
</tr>
<tr>
<td>midRefRot</td>
<td>std::vector&lt;TRotation&gt;</td>
<td>Global rotation matrix for middle of the beamline elements along the reference trajectory and without any tilt or rotation from the component</td>
</tr>
<tr>
<td>endRefRot</td>
<td>std::vector&lt;TRotation&gt;</td>
<td>Global rotation matrix for middle of the beamline elements along the reference trajectory and without any tilt or rotation from the component</td>
</tr>
<tr>
<td>staS</td>
<td>std::vector&lt;float&gt;</td>
<td>S-position of start of start of element (metres)</td>
</tr>
<tr>
<td>midS</td>
<td>std::vector&lt;float&gt;</td>
<td>S-position of start of middle of element (metres)</td>
</tr>
<tr>
<td>endS</td>
<td>std::vector&lt;float&gt;</td>
<td>S-position of start of end of element (metres)</td>
</tr>
<tr>
<td>beamPipeType</td>
<td>std::vector<a href="">std::string</a></td>
<td>Aperture type; “circular”, “lhc”, etc.</td>
</tr>
<tr>
<td>beamPipeAper1</td>
<td>std::vector&lt;double&gt;</td>
<td>Aperture aper1 (metres)</td>
</tr>
<tr>
<td>beamPipeAper2</td>
<td>std::vector&lt;double&gt;</td>
<td>Aperture aper2 (metres)</td>
</tr>
<tr>
<td>beamPipeAper3</td>
<td>std::vector&lt;double&gt;</td>
<td>Aperture aper3 (metres)</td>
</tr>
<tr>
<td>beamPipeAper4</td>
<td>std::vector&lt;double&gt;</td>
<td>Aperture aper4 (metres)</td>
</tr>
<tr>
<td>material</td>
<td>std::vector<a href="">std::string</a></td>
<td>Main material associated with an element. For a drift, this is the beam pipe material; for a magnet, the yoke material; a collimator, the main material.</td>
</tr>
<tr>
<td>k1 - k12</td>
<td>std::vector&lt;float&gt;</td>
<td>Normalised magnet strength associated with element (1st - 12th order)</td>
</tr>
<tr>
<td>k12 - k122</td>
<td>std::vector&lt;float&gt;</td>
<td>Normalised skew magnet strength associated with element (1st - 12th order)</td>
</tr>
<tr>
<td>ks</td>
<td>std::vector&lt;float&gt;</td>
<td>Normalised solenoid strength</td>
</tr>
<tr>
<td>hkick</td>
<td>std::vector&lt;float&gt;</td>
<td>Fractional momentum kick in horizontal direction</td>
</tr>
<tr>
<td>vkick</td>
<td>std::vector&lt;float&gt;</td>
<td>Fractional momentum kick in vertical direction</td>
</tr>
<tr>
<td>bField</td>
<td>std::vector&lt;float&gt;</td>
<td>Magnetic field magnitude (T)</td>
</tr>
<tr>
<td>eField</td>
<td>std::vector&lt;float&gt;</td>
<td>Electric field magnitude (MV)</td>
</tr>
<tr>
<td>e1</td>
<td>std::vector&lt;float&gt;</td>
<td>Input pole face angle (note sbend / rbend convention) (rad)</td>
</tr>
</tbody>
</table>
Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2</td>
<td>std::vector&lt;float&gt;</td>
<td>Output pole face angle (rad)</td>
</tr>
<tr>
<td>hgap</td>
<td>std::vector&lt;float&gt;</td>
<td>Half-gap of pole tips for dipoles (m)</td>
</tr>
<tr>
<td>fint</td>
<td>std::vector&lt;float&gt;</td>
<td>Fringe-field integral</td>
</tr>
<tr>
<td>fintx</td>
<td>std::vector&lt;float&gt;</td>
<td>Fringe-field integral for exit pole face</td>
</tr>
<tr>
<td>fintk2</td>
<td>std::vector&lt;float&gt;</td>
<td>2nd fringe-field integral</td>
</tr>
<tr>
<td>fintxk2</td>
<td>std::vector&lt;float&gt;</td>
<td>2nd fringe-field integral for exit pole face</td>
</tr>
</tbody>
</table>

Optional collimator information also store in the model.

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>storeCollimator-Info</td>
<td>bool</td>
<td>Whether the optional collimator information was stored.</td>
</tr>
<tr>
<td>collimatorIndices</td>
<td>std::vector&lt;int&gt;</td>
<td>Index of each collimator in this beam line. Optional.</td>
</tr>
<tr>
<td>collimatorIndices-ByName</td>
<td>std::map&lt;std::string, int&gt;</td>
<td>Map of collimator names to beam line indices. Includes both the accelerator component name and the placement name which is unique.</td>
</tr>
<tr>
<td>collimator-BranchName-Unique</td>
<td>std::vector<a href="">std::string</a></td>
<td>Name of branches in Event tree created specifically for collimator hits.</td>
</tr>
</tbody>
</table>

BDSOutputROOTEventCollimatorInfo

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentName</td>
<td>std::string</td>
<td>Collimator name</td>
</tr>
<tr>
<td>componentType</td>
<td>std::string</td>
<td>Type of collimator</td>
</tr>
<tr>
<td>length</td>
<td>double</td>
<td>Length (m)</td>
</tr>
<tr>
<td>tilt</td>
<td>double</td>
<td>Tilt (rad)</td>
</tr>
<tr>
<td>offsetX</td>
<td>offsetX</td>
<td>Horizontal offset (m)</td>
</tr>
<tr>
<td>offsetY</td>
<td>offsetY</td>
<td>Vertical offset (m)</td>
</tr>
<tr>
<td>material</td>
<td>std::string</td>
<td>Collimator material</td>
</tr>
<tr>
<td>xSizeIn</td>
<td>double</td>
<td>Horizontal half aperture at entrance (m)</td>
</tr>
<tr>
<td>ySizeIn</td>
<td>double</td>
<td>Vertical half aperture at entrance (m)</td>
</tr>
<tr>
<td>xSizeOut</td>
<td>double</td>
<td>Horizontal half aperture at exit (m)</td>
</tr>
<tr>
<td>ySizeOut</td>
<td>double</td>
<td>Vertical half aperture at exit(m)</td>
</tr>
</tbody>
</table>

9.7.6 Run Tree

This tree contains two branches called “Histos.” and “Summary.” which represent instances of include/BDSOutputROOTEventHistograms.hh and include/BDSOutputROOTEventInfo, respectively. Histos contains two vectors of 1D and 2D histograms that are produced per run.

9.7.7 Event Tree

This tree contains information on a per-event basis. Everything shown in the above tree has a different value per-event run in BDSIM.
<table>
<thead>
<tr>
<th>Branch Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary (+)</td>
<td>BDSOutputROOTEventInfo</td>
<td>Per-event summary information</td>
</tr>
<tr>
<td>Primary</td>
<td>BDSOutputROOTEventSampler&lt;float&gt;</td>
<td>A record of the coordinates at the start of the simulation (before tracking)</td>
</tr>
<tr>
<td>PrimaryGlobal</td>
<td>BDSOutputROOTEventCoords</td>
<td>Global Cartesian coordinates of the primary particle. These are the same as those in “Primary” unless S0 is specified in the beam distribution.</td>
</tr>
<tr>
<td>Eloss</td>
<td>BDSOutputROOTEventLoss</td>
<td>Coordinates of energy deposition in the accelerator material.</td>
</tr>
<tr>
<td>ElossVacuum (*)</td>
<td>BDSOutputROOTEventLoss</td>
<td>Coordinates of energy deposition in the accelerator vacuum only</td>
</tr>
<tr>
<td>ElossTunnel (*)</td>
<td>BDSOutputROOTEventLoss</td>
<td>Coordinates of energy deposition in the tunnel material</td>
</tr>
<tr>
<td>ElossWorld (*)</td>
<td>BDSOutputROOTEventLoss</td>
<td>Coordinates of energy deposition in the world volume - by default the air.</td>
</tr>
<tr>
<td>ElossWorldContent(++)</td>
<td>BDSOutputROOTEventLossWorld</td>
<td>Global coordinates of energy deposition in any volume supplied inside an externally supplied world volume.</td>
</tr>
<tr>
<td>ElossWorldExit (*)</td>
<td>BDSOutputROOTEventLossWorld</td>
<td>Global coordinates of the point any track exits the world volume and therefore the simulation.</td>
</tr>
<tr>
<td>PrimaryFirstHit</td>
<td>BDSOutputROOTEventLoss</td>
<td>Energy deposit ‘hit’ representing the first step on the primary trajectory that wasn’t due to tracking, i.e. the first interaction where a physics process was induced</td>
</tr>
<tr>
<td>PrimaryLastHit</td>
<td>BDSOutputROOTEventLoss</td>
<td>The end point of the primary trajectory. If S is -1 (m) it means the particle finished away from the beam line where there was no curvilinear coordinate system present.</td>
</tr>
<tr>
<td>Trajectory</td>
<td>BDSOutputROOTEventTrajectory</td>
<td>A record of all the steps the primary particle took and the associated physics processes</td>
</tr>
<tr>
<td>Histos</td>
<td>BDSOutputROOTEventHistograms</td>
<td>Per-event histograms in vectors</td>
</tr>
<tr>
<td>xxxxx</td>
<td>BDSOutputROOTEventSampler&lt;float&gt;</td>
<td>A dynamically generated branch created per sampler (here named ‘xxxxx’) that contains a record of all particles that passed through the sampler during the event. Note: this includes both primary and secondary particles.</td>
</tr>
<tr>
<td>COLL_xxxx (**)</td>
<td>BDSOutputROOTEventCollimator</td>
<td>A dynamically generated branch created per collimator when the storeCollimatorInfo is used. Stores collimator hit information by default only for primary particle hits.</td>
</tr>
</tbody>
</table>

- (+) This was called “Info” in BDSIM before V1.3.
- (++) ElossWorldContents is only included if the option storeElossWorldContents is turned on or importance sampling is used.
- (*) ElossVacuum, ElossTunnel, ElossWorld and ElossWorldExit are empty by default and controlled by the option storeElossWorld.
- (**) COLL_xxxx is only added per collimator when the option storeCollimatorInfo is used.

The types and names of the contents of each class can be found in the header files in bdsim/include/BDSOutputROOTEvent*.hh. The contents of the classes are described below.
**Warning:** For large $S0$ in a large model, a large distance as compared to the size of the beam may displace the primary coordinates, e.g. 1km offset for 1um beam. For this reason the PrimaryGlobal structure always uses double precision numbers, unlike the Primary structure and the other samplers that use floating point precision numbers (unless the ROOTDOUBLE CMake option is used at compilation time for double precision in the samplers).

---

**BDSOutputROOTEventInfo**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>startTime</td>
<td>time_t</td>
<td>Time stamp at start of event</td>
</tr>
<tr>
<td>stopTime</td>
<td>time_t</td>
<td>Time stamp at end of event</td>
</tr>
<tr>
<td>duration</td>
<td>float</td>
<td>Duration of event in seconds</td>
</tr>
<tr>
<td>seedStateAtStart</td>
<td>std::string</td>
<td>State of random number generator at the start of the event as provided by CLHEP</td>
</tr>
<tr>
<td>index</td>
<td>int</td>
<td>Index of the event (0 counting)</td>
</tr>
<tr>
<td>aborted</td>
<td>bool</td>
<td>Whether event was aborted or not</td>
</tr>
<tr>
<td>primaryHitMachine</td>
<td>bool</td>
<td>Whether the primary particle hit the machine. This is judged by whether there are any energy deposition hits or not. If no physics processes are registered this won’t work correctly.</td>
</tr>
<tr>
<td>primaryAbsorbedInCollimator</td>
<td>bool</td>
<td>Whether the primary particle stopped in a collimator or not.</td>
</tr>
<tr>
<td>memoryUsageMb</td>
<td>double</td>
<td>Memory usage of the whole program at the the current event including the geometry.</td>
</tr>
<tr>
<td>energyDeposited</td>
<td>double</td>
<td>(GeV) Integrated energy in Eloss including the statistical weights.</td>
</tr>
<tr>
<td>energyDepositedVacuum</td>
<td>double</td>
<td>(GeV) Integrated energy in ElossVacuum the statistical weights.</td>
</tr>
<tr>
<td>energyDepositedWorld</td>
<td>double</td>
<td>(GeV) Integrated energy in the ElossWorld structure including the statistical weight.</td>
</tr>
<tr>
<td>energyDepositedTunnel</td>
<td>double</td>
<td>(GeV) Integrated energy in the ElossTunnel including the statistical weight.</td>
</tr>
<tr>
<td>energyWorldExit</td>
<td>double</td>
<td>(GeV) Integrated energy of all particles including their rest mass leaving the world volume and therefore the simulation.</td>
</tr>
<tr>
<td>energyKilled</td>
<td>double</td>
<td>(GeV) Integrated energy including their rest mass of any particles that were artificially killed in the stacking action.</td>
</tr>
<tr>
<td>energyTotal</td>
<td>double</td>
<td>The sum of the above energies for the current event.</td>
</tr>
<tr>
<td>nCollimatorsInteracted</td>
<td>int</td>
<td>The number of collimators the primary particle interacted with.</td>
</tr>
</tbody>
</table>

**Note:** `energyDepositedVacuum` will only be non-zero if the option `storeElossVacuum` is on which is off by default.

**Note:** `energyDepositedWorld` will only be non-zero if the option `storeElossWorld` is on which is off by default.

**Note:** `energyWorldExit` will only be non-zero if Geant4.10.3 or later is used as well as the option...
storeElossWorld is on that is off by default.

**Note:** nCollimatorsInteracted will only be non-zero if the option storeCollimatorInfo is turned on which is off by default.

**Warning:** One would expect the parameter energyTotal which is the sum of the energies to be equal to the incoming beam energy. This in reality depends on the physics list used as well as the production range cuts. Furthermore, ions from the accelerator material may be liberated leading to an inflated total energy as their rest mass is also counted. This is non-trivial to correct and this value is provided only as a guide. The physics library and BDSIM-provided tracking both conserve energy but it is highly non-trivial to ensure all changes are recorded.

**BDSOutputROOTEventLoss**

Energy deposition hits are the most numerous, so not all information is recorded by default. Extra information can be recorded but this typically dominates the output file size.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>int</td>
<td>The number of energy deposition hits for this event</td>
</tr>
<tr>
<td>energy</td>
<td>std::vector&lt;float&gt;</td>
<td>Vector of energy of each piece of energy deposition</td>
</tr>
<tr>
<td>S</td>
<td>std::vector&lt;float&gt;</td>
<td>Corresponding curvilinear S-position (m) of energy deposition</td>
</tr>
<tr>
<td>weight</td>
<td>std::vector&lt;float&gt;</td>
<td>Corresponding weight</td>
</tr>
<tr>
<td>partID</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Particle ID of particle that caused energy deposition</td>
</tr>
<tr>
<td>trackID</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Track ID of particle that caused energy deposition</td>
</tr>
<tr>
<td>parentID</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Track ID of the parent particle</td>
</tr>
<tr>
<td>modelID</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Index in model tree for where deposition occurred</td>
</tr>
<tr>
<td>turn</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Turn in circular machine on which hit occurred</td>
</tr>
<tr>
<td>x</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Local X of energy deposition (m)</td>
</tr>
<tr>
<td>y</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Local Y of energy deposition (m)</td>
</tr>
<tr>
<td>z</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Local Z of energy deposition (m)</td>
</tr>
<tr>
<td>X</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Global X of energy deposition (m)</td>
</tr>
<tr>
<td>Y</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Global Y of energy deposition (m)</td>
</tr>
<tr>
<td>Z</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Global Z of energy deposition (m)</td>
</tr>
<tr>
<td>T</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Global time-of-flight since beginning of event (ns)</td>
</tr>
<tr>
<td>stepLength</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Length of step that the energy deposition was produced in (m)</td>
</tr>
<tr>
<td>preStepKineticEnergy</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) The kinetic energy of the particle (any species) at the starting point of the step that the energy deposition was produced in</td>
</tr>
<tr>
<td>storeLinks</td>
<td>bool</td>
<td>Whether extra information was stored (partID, trackID, parentID, modelID, turn)</td>
</tr>
<tr>
<td>storeLocal</td>
<td>bool</td>
<td>Whether x, y, z were stored</td>
</tr>
<tr>
<td>storeGlobal</td>
<td>bool</td>
<td>Whether X, Y, Z were stored</td>
</tr>
<tr>
<td>storeTime</td>
<td>bool</td>
<td>Whether T was stored</td>
</tr>
<tr>
<td>storeStepLength</td>
<td>bool</td>
<td>Whether stepLength was stored</td>
</tr>
<tr>
<td>storePreStepKineticEnergy</td>
<td>bool</td>
<td>Whether preStepKineticEnergy was stored</td>
</tr>
</tbody>
</table>

**BDSOutputROOTEventLossWorld**

For the point where particles exit the world, there is no concept of a curvilinear coordinate system so there are only global coordinates recorded.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>int</td>
<td>The number of exits for this event</td>
</tr>
<tr>
<td>totalEnergy</td>
<td>std::vector&lt;float&gt;</td>
<td>Vector of total energy of each particle exiting</td>
</tr>
<tr>
<td>postStepKineticEnergy</td>
<td>std::vector&lt;float&gt;</td>
<td>The kinetic energy of the particle (any species) at the end point as the particle exited.</td>
</tr>
<tr>
<td>X</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Global X of exit point (m)</td>
</tr>
<tr>
<td>Y</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Global Y of exit point (m)</td>
</tr>
<tr>
<td>Z</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Global Z of exit point (m)</td>
</tr>
<tr>
<td>T</td>
<td>std::vector&lt;float&gt;</td>
<td>(optional) Global time-of-flight since beginning of event (ns)</td>
</tr>
<tr>
<td>partID</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Particle ID of particle</td>
</tr>
<tr>
<td>trackID</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Track ID of particle</td>
</tr>
<tr>
<td>parentID</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Track ID of the parent particle</td>
</tr>
<tr>
<td>weight</td>
<td>std::vector&lt;float&gt;</td>
<td>Corresponding weight</td>
</tr>
<tr>
<td>turn</td>
<td>std::vector&lt;int&gt;</td>
<td>(optional) Turn in circular machine on loss</td>
</tr>
</tbody>
</table>

**BDSOutputROOTEventTrajectory**

By default, only the primary particle trajectory is stored - see *Output Options* for which options to set to control the level of detail stored in the trajectories.

Currently, some degenerate information is stored for completeness. This may be removed in future versions (e.g. the pre-step point of the part of the trajectory is the same as the post-step point of the previous part of the trajectory).

Each entry in the vectors in BDSOutputROOTEventTrajectory represents one step along the particle trajectory with a ‘pre-step’ and ‘post-step’ point - information associated with the start and end of that step.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>int</td>
<td>The number of trajectories stored for this event</td>
</tr>
<tr>
<td>partID</td>
<td>std::vector&lt;int&gt;</td>
<td>The PDG ID for the particle in each trajectory step</td>
</tr>
<tr>
<td>trackID</td>
<td>std::vector&lt;unsigned int&gt;</td>
<td>The track ID for the particle in each trajectory step</td>
</tr>
<tr>
<td>parentID</td>
<td>std::vector&lt;float&gt;</td>
<td>The track ID of the parent particle for each trajectory step</td>
</tr>
<tr>
<td>parentIndex</td>
<td>std::vector&lt;int&gt;</td>
<td>The index in the vectors of this class that correspond to parent particle (the one that lead to the creation of the particle in the current entry)</td>
</tr>
<tr>
<td>parentStepIndex</td>
<td>std::vector&lt;int&gt;</td>
<td>TBC</td>
</tr>
<tr>
<td>preProcessTypes</td>
<td>std::vector&lt;std::vector&lt;int&gt;&gt;</td>
<td>Geant4 enum of pre-step physics process - general category</td>
</tr>
<tr>
<td>preProcessSubTypes</td>
<td>std::vector&lt;std::vector&lt;int&gt;&gt;</td>
<td>Geant4 enum of pre-step physics process - specific process ID within category</td>
</tr>
<tr>
<td>postProcessTypes</td>
<td>std::vector&lt;std::vector&lt;int&gt;&gt;</td>
<td>Geant4 enum of post-step physics process - general category</td>
</tr>
<tr>
<td>postProcesssSubTypes</td>
<td>std::vector&lt;std::vector&lt;int&gt;&gt;</td>
<td>Geant4 enum of post-step physics process - specific process ID within category</td>
</tr>
<tr>
<td>preWeights</td>
<td>std::vector&lt;std::vector&lt;double&gt;&gt;</td>
<td>Weighting associated with pre-step point</td>
</tr>
<tr>
<td>postWeights</td>
<td>std::vector&lt;std::vector&lt;double&gt;&gt;</td>
<td>Weighting associated with post-step point</td>
</tr>
<tr>
<td>energies</td>
<td>std::vector&lt;std::vector&lt;double&gt;&gt;</td>
<td>Total energy of particle in current trajectory step</td>
</tr>
<tr>
<td>trajectories</td>
<td>std::vector&lt;std::vector&lt;TVector3&gt;&gt;</td>
<td>The ‘position’ of the trajectory according to Geant4 - from G4Track-&gt;GetPosition()</td>
</tr>
<tr>
<td>momenta</td>
<td>std::vector&lt;std::vector&lt;TVector3&gt;&gt;</td>
<td>Momentum of the track (GeV)</td>
</tr>
<tr>
<td>modelIndicies</td>
<td>std::vector&lt;std::vector&lt;int&gt;&gt;</td>
<td>Index in beam line of which element the trajectory is in (-1 if not inside an accelerator component)</td>
</tr>
</tbody>
</table>

In addition, some maps are stored to link the entries together conceptually.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>trackID_trackIndex</td>
<td>std::map&lt;int, int&gt;</td>
<td>A map of all trackIDs to the index in this class</td>
</tr>
<tr>
<td>trackIndex_trackProcess</td>
<td>std::map&lt;int, std::pair&lt;int,int&gt;&gt;</td>
<td>A map from the index in this class to track process</td>
</tr>
<tr>
<td>trackIndex_modelIndex</td>
<td>std::map&lt;int, int&gt;</td>
<td>A map from the index in this class to the model index</td>
</tr>
<tr>
<td>modelIndex_trackIndex</td>
<td>std::map&lt;int, std::vector&lt;int&gt;&gt;</td>
<td>A map from the model index to the index in this class</td>
</tr>
</tbody>
</table>

Functions are provided that allow exploration of the data through the connections stored.
<table>
<thead>
<tr>
<th>Function</th>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>findParentProcess(int trackIndex)</td>
<td>std::pair&lt;int, int&gt;</td>
<td>Find the parent track index and process index from the ultimate parent of this particle up the trajectory table.</td>
</tr>
<tr>
<td>trackInteractions(int trackID)</td>
<td>std::vector&lt;BDSOutputROOTEventTrajectoryPoint&gt;</td>
<td>Return vector of points where this particle interacted all the way to the primary. Transportation steps are suppressed.</td>
</tr>
<tr>
<td>primaryProcessPoint(int trackID)</td>
<td>BDSOutputROOTEventTrajectoryPoint</td>
<td>For a given track ID, return the point where the primary particle first interacted.</td>
</tr>
<tr>
<td>processHistory(int trackID)</td>
<td>std::vector&lt;BDSOutputROOTEventTrajectoryPoint&gt;</td>
<td>Full history up the trajectory table to the primary for a given track ID.</td>
</tr>
</tbody>
</table>

**BDSOutputROOTEventSampler**

Note: the sampler structure, like everything else in the event tree, is stored per event. However, for a given event, there may be multiple hits on a sampler, i.e. many secondary particles may have passed through a sampler. For this purpose, most variables are vectors of numbers, where the vector represents all the hits in that event.

As the sampler is considered infinitely thin and always in the same place, there is no point in storing the z-location or the S-location for every particle hit. Therefore, these variables are only stored once as a single number per event.

The class is templated to allow use of both double and float precision numbers. By default, T = float, i.e. float precision number is stored. BDSIM can be compiled with an option for double precision output (useful typically only for development or precision testing) but this doubles the output file size.
<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>int</td>
<td>The number in this event in this sampler</td>
</tr>
<tr>
<td>energy</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the total energy (GeV) of each hit in this sampler</td>
</tr>
<tr>
<td>x</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the x-coordinate of each hit (m)</td>
</tr>
<tr>
<td>y</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the y-coordinate of each hit (m)</td>
</tr>
<tr>
<td>z</td>
<td>T</td>
<td>Single entry of z for this sampler (m)</td>
</tr>
<tr>
<td>xp</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the fractional x transverse momentum</td>
</tr>
<tr>
<td>yp</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the fractional y transverse momentum</td>
</tr>
<tr>
<td>zp</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the fractional forward momentum</td>
</tr>
<tr>
<td>T</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the time-of-flight of the particle (ns)</td>
</tr>
<tr>
<td>weight</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the associated weights of the hits</td>
</tr>
<tr>
<td>partID</td>
<td>std::vector&lt;int&gt;</td>
<td>Vector of the PDG ID for the particle of each hit</td>
</tr>
<tr>
<td>parentID</td>
<td>std::vector&lt;int&gt;</td>
<td>Vector of the trackID of the progenitor of the particle that hit</td>
</tr>
<tr>
<td>trackID</td>
<td>std::vector&lt;int&gt;</td>
<td>Vector of the trackID of the particle that hit</td>
</tr>
<tr>
<td>modelID</td>
<td>int</td>
<td>The index to the BDSIM model of which element the sampler belonged to</td>
</tr>
<tr>
<td>turnNumber</td>
<td>std::vector&lt;int&gt;</td>
<td>Vector of the turn number of the particle that hit</td>
</tr>
<tr>
<td>S</td>
<td>T</td>
<td>S-position of the sampler (m)</td>
</tr>
<tr>
<td>r (*)</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the radius calculated from x and y (m)</td>
</tr>
<tr>
<td>rp (*)</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the radius calculated from xp and yp</td>
</tr>
<tr>
<td>charge (*)</td>
<td>std::vector&lt;int&gt;</td>
<td>Vector of the PDG charge of the particle for each hit</td>
</tr>
<tr>
<td>mass (*)</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the PDG mass of the particle for each hit (GeV)</td>
</tr>
<tr>
<td>rigidity (*)</td>
<td>std::vector&lt;T&gt;</td>
<td>Vector of the rigidity of the particle for each hit (Tm)</td>
</tr>
<tr>
<td>isIon (*)</td>
<td>std::vector&lt;bool&gt;</td>
<td>Vector of whether the particle is an ion or not</td>
</tr>
<tr>
<td>ionA (*)</td>
<td>std::vector&lt;int&gt;</td>
<td>Vector of the atomic mass number. 0 for non-nuclei.</td>
</tr>
<tr>
<td>ionZ (*)</td>
<td>std::vector&lt;int&gt;</td>
<td>Vector of the atomic number. 0 for non-nuclei.</td>
</tr>
</tbody>
</table>

**Note:** (*) These are not stored by default (i.e. the vectors exist but are empty). If these parameters are desired, please use the appropriate options to turn their storage on. See `Output Options` for more details.

**Warning:** A common issue is that apparently half of the particles missing in the first sampler in the beam line. If a sampler is placed at the beginning of the beam line and a bunch distribution with a finite z-width is used, approximately half of the particles will start in front of the sampler, never pass through it and never be registered. For this reason, one should refrain from putting a sampler at the beginning of a beam line to avoid confusion. The primary output records all primary coordinates before they enter the tracking in the geometry, so it always contains all primary particles.
### BDSOutputROOTEventCoords

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>double</td>
<td>Global Cartesian x coordinate (m)</td>
</tr>
<tr>
<td>y</td>
<td>double</td>
<td>Global Cartesian y coordinate (m)</td>
</tr>
<tr>
<td>z</td>
<td>double</td>
<td>Global Cartesian z coordinate (m)</td>
</tr>
<tr>
<td>xp</td>
<td>double</td>
<td>Global Cartesian unit momentum in x</td>
</tr>
<tr>
<td>yp</td>
<td>double</td>
<td>Global Cartesian unit momentum in y</td>
</tr>
<tr>
<td>zp</td>
<td>double</td>
<td>Global Cartesian unit momentum in z</td>
</tr>
<tr>
<td>T</td>
<td>double</td>
<td>Time (ns)</td>
</tr>
</tbody>
</table>

### BDSOutputROOTEventHistograms

This class contains the following data:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>histograms1D</td>
<td>std::vector&lt;TH1D*&gt;</td>
<td>Vector of 1D histograms stored in the simulation</td>
</tr>
<tr>
<td>histograms2D</td>
<td>std::vector&lt;TH2D*&gt;</td>
<td>Vector of 2D histograms stored in the simulation</td>
</tr>
<tr>
<td>histograms3D</td>
<td>std::vector&lt;TH3D*&gt;</td>
<td>Vector of 3D histograms stored in the simulation</td>
</tr>
</tbody>
</table>

These are histograms stored for each event. Whilst a few important histograms are stored by default, the number may vary depending on the options chosen and the histogram vectors are filled dynamically based on these. For this reason, the name of the histogram is given an not the index. BDSIM produces six histograms by default during the simulation. These are:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phits</td>
<td>Primary hit. S location of first physics process on the primary track.</td>
</tr>
<tr>
<td>Ploss</td>
<td>Primary loss. S location of the end of the primary track.</td>
</tr>
<tr>
<td>Eloss (*)</td>
<td>Energy deposition. Based on the data from ‘Eloss’ branch.</td>
</tr>
<tr>
<td>PhitsPE</td>
<td>Same as Phits, but binned per element in S. Note the values are not normalised to the bin width.</td>
</tr>
<tr>
<td>PlossPE</td>
<td>Same as Ploss, but binned per element in S. Note the values are not normalised to the bin width.</td>
</tr>
<tr>
<td>ElossPE (*)</td>
<td>Same as Eloss, but binned per element in S. Note the values are not normalised to the bin width. Based on the data from the Eloss branch.</td>
</tr>
<tr>
<td>ElossTunnel (**)</td>
<td>Energy deposition in the tunnel. Based on data from the ElossTunnel branch.</td>
</tr>
<tr>
<td>ElossTunnelPE (**)</td>
<td>Energy deposition in the tunnel with per element binning. Based on data from the ElossTunnel branch.</td>
</tr>
<tr>
<td>CollPhitsPE (***)</td>
<td>Primary hits where each bin is 1 collimator in the order they appear in the beam line. These are bins copied out of PhitsPE for only the collimators.</td>
</tr>
<tr>
<td>CollPlossPE (***)</td>
<td>Primary loss where each bin is 1 collimator in the order they appear in the beam line. These are bins copied out of PlossPE for only the collimators.</td>
</tr>
<tr>
<td>CollElossPE (***)</td>
<td>Energy deposition where each bin is 1 collimator in the order they appear in the beam line. These are bins copied out of ElossPE for only the collimators.</td>
</tr>
<tr>
<td>CollInteracted (***)</td>
<td>Each bin represents one collimator in the beam line in the order they appear and is filled with 1.0 if the primary particle interacted with that collimator in that event. Note, the primary may interact with multiple collimators each event.</td>
</tr>
</tbody>
</table>

- (*) The “Eloss” and “ElossPE” histograms are only created if storeELoss or storeElossHistograms are turned on (default is on).
- (**) The tunnel histograms are only created if storeELossTunnel or storeElossTunnelHistograms options are on (default is storeElossTunnelHistograms on

9.7. Structure Of Output 173
only when tunnel is built).

- (***) The histograms starting with “Coll” are only created if `storeCollimatorInfo` is turned on.

**Note:**  The per-element histograms are integrated across the length of each element so they will have different (uneven) bin widths.

The energy loss histograms are evenly binned according to the option `elossHistoBinWidth` (in metres).

**BDSOutputROOTEventCollimator**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>primaryInteracted</td>
<td>bool</td>
<td>Whether the primary interacted with this collimator this event</td>
</tr>
<tr>
<td>primaryStopped</td>
<td>bool</td>
<td>Whether the primary stopped in this collimator this event</td>
</tr>
<tr>
<td>n</td>
<td>int</td>
<td>Number of hits recorded and therefore the length of each vector here</td>
</tr>
<tr>
<td>energy</td>
<td>std::vector&lt;float&gt;</td>
<td>Total energy of the particle for the hit (GeV)</td>
</tr>
<tr>
<td>energyDeposited</td>
<td>std::vector&lt;float&gt;</td>
<td>Energy deposited in the step for the hit (GeV)</td>
</tr>
<tr>
<td>xIn</td>
<td>std::vector&lt;float&gt;</td>
<td>Pre step point x coordinate for the hit (m)</td>
</tr>
<tr>
<td>yIn</td>
<td>std::vector&lt;float&gt;</td>
<td>Pre step point y coordinate for the hit (m)</td>
</tr>
<tr>
<td>zIn</td>
<td>std::vector&lt;float&gt;</td>
<td>Pre step point z coordinate for the hit (m)</td>
</tr>
<tr>
<td>xpIn</td>
<td>std::vector&lt;float&gt;</td>
<td>Pre step point x fraction of unit momentum</td>
</tr>
<tr>
<td>ypIn</td>
<td>std::vector&lt;float&gt;</td>
<td>Pre step point y fraction of unit momentum</td>
</tr>
<tr>
<td>zpIn</td>
<td>std::vector&lt;float&gt;</td>
<td>Pre step point z fraction of unit momentum</td>
</tr>
<tr>
<td>T</td>
<td>std::vector&lt;float&gt;</td>
<td>Global time at hit (ns)</td>
</tr>
<tr>
<td>weight</td>
<td>std::vector&lt;float&gt;</td>
<td>Statistical weight associated with particle at hit</td>
</tr>
<tr>
<td>partID</td>
<td>std::vector&lt;int&gt;</td>
<td>PDG ID for particle type</td>
</tr>
<tr>
<td>parentID</td>
<td>std::vector&lt;int&gt;</td>
<td>TrackID of the progenitor of the particle that hit</td>
</tr>
<tr>
<td>turn</td>
<td>std::vector&lt;int&gt;</td>
<td>Turn number of the hit (1 counting)</td>
</tr>
<tr>
<td>firstPrimaryHitThisTurn</td>
<td>std::vector&lt;bool&gt;</td>
<td>Whether this is the first primary particle hit in this collimator this turn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Used to match first hit with other simulations when there may be more than</td>
</tr>
<tr>
<td></td>
<td></td>
<td>one primary hit in the same collimator as the particle passes through once.</td>
</tr>
<tr>
<td>impactParameterX</td>
<td>std::vector&lt;float&gt;</td>
<td>Depth into the collimator from its aperture in the frame of the collimator</td>
</tr>
<tr>
<td>impactParameterY</td>
<td>std::vector&lt;float&gt;</td>
<td>Depth into the collimator from its aperture in the frame of the collimator</td>
</tr>
<tr>
<td>isIon</td>
<td>std::vector&lt;bool&gt;</td>
<td>Whether the hit was made by an ion</td>
</tr>
<tr>
<td>ionA</td>
<td>std::vector&lt;int&gt;</td>
<td>Ion atomic mass number. 0 for non-nuclei</td>
</tr>
<tr>
<td>ionZ</td>
<td>std::vector&lt;int&gt;</td>
<td>Ion atomic number. 0 for non-nuclei</td>
</tr>
<tr>
<td>turnSet</td>
<td>std::set&lt;int&gt;</td>
<td>A set (no duplicate values) for which turns this collimator was hit (including non-primary particles depending on the options)</td>
</tr>
<tr>
<td>charge</td>
<td>std::vector&lt;int&gt;</td>
<td>PDG charge of the particle for each hit</td>
</tr>
<tr>
<td>kineticEnergy</td>
<td>std::vector&lt;float&gt;</td>
<td>Pre step point kinetic energy of the particle for each hit</td>
</tr>
<tr>
<td>mass</td>
<td>std::vector&lt;float&gt;</td>
<td>PDG mass of the particle for each hit (GeV)</td>
</tr>
<tr>
<td>rigidity</td>
<td>std::vector&lt;float&gt;</td>
<td>Rigidity of the particle for each hit (Tm)</td>
</tr>
</tbody>
</table>
CHAPTER
TEN

OUTPUT ANALYSIS

This section describes how to load and view data from the recommended output `rootevent` format. See Basic Data Inspection for how to view the data and make the most basic on-the-fly histograms.

10.1 Setup

1) BDSIM must be installed after compilation for the analysis tools to function properly.

2) Environmental variables should be set by sourcing `<bdsim-install-dir>/bin/bdsim.sh`.

3) A ROOT logon macro may optionally be written for convenience in loading libraries.

If the setup is correct, you should be able to execute ‘rebdsim’ in the terminal. See Building and Environmental Variables for more details.

If the analysis will be regularly used interactively, it is worth automating the library loading in root by finding and editing the `rootlogon.C` in your `<root-install-dir>/macros/` directory. Example text would be:

```cpp
cout << "Loading rebdsim libraries" << endl;
gSystem->Load("librebsim");
gSystem->Load("libbdsimRootEvent");
```

Note: The file extension is omitted on purpose.

The absolute path is not necessary, as the above environmental variables are used by ROOT to find the library.

10.2 Quick Recipes

10.2.1 Inspect Histograms

- Run rebdsimHistoMerge on BDSIM output file (quick).
- Browse output of rebdsimHistoMerge in TBrowser in ROOT.
- See rebdsimHistoMerge - Simple Histogram Merging for details.
10.2.2 Plot Energy Deposition & Losses

- Run rebdsimHistoMerge on BDSIM output file (quick).
- Plot in Python using `pybdsim` using dedicated plotting function.

```
rebdsmHistoMerge output.root results.root
ipython
>>> import pybdsim
>>> pybdsim.Plot.LossAndEnergyDeposition("results.root")
```

10.3 rebdsim - General Analysis Tool

BDSIM is accompanied by an analysis tool called `rebdsim` ("root event BDSIM") that provides the ability to use simple text input files to specify histograms and process data. It also provides the ability to calculate optical functions from the sampler data.

`rebdsim` is based on a set of analysis classes that are compiled into a library. These may be used through `rebdsim`, but also through the ROOT interpreter and in a user’s ROOT macro or compiled code. They may also be used through Python if the user has ROOT available through Python.

`rebdsim` is executed with one argument which is the path to an analysis configuration text file. This is a simple text file that describes which histograms to make from the data. Optionally, a second argument of a data file to operate on will override the one specified in the analysis configuration file. This allows the same analysis configuration to be used to analyse many different data files. A third optional argument (must have second argument specified) is the output file name that the resultant analysis will be written to.

Examples:

```
rebdsim analysisConfig.txt
rebdsim analysisConfig.txt output.root
rebdsim analysisConfig.txt output.root results.root
```

10.3.1 Preparing an Analysis Configuration File

The analysis configuration file is a simple text file. This can be prepared by copying and editing an example. The text file acts as a thin interface to an analysis in ROOT that would commonly use the `TTree->Draw()` method.

We strongly recommend browsing the data in a TBrowser beforehand and double-clicking the variables. This gives you an idea of the range of the data. See Basic Data Inspection for more details.

There are three types of histograms that `rebdsim` can produce:

1. "Simple" histograms - these are sum over all entries in that tree.
2. "Per-Entry" histograms - here an individual histogram is made for each entry in the tree and these are averaged across all entries. In the case of the Event tree, each entry is a single event. A per-entry histogram is therefore a per-event histogram.
3. "Merged" histograms - these are the mean taken across all entries of a histogram already in the output file. For example, there is an energy deposition histogram stored with each event. This would be merged into a per-event average.
10.3.2 Per-Entry and Simple Histograms

For the energy deposition histogram for example, the energy deposition hits are binned as a function of the curvilinear $S$ position along the accelerator. In fact, the $S$ position is binned with the weight of the energy. In each event, a single primary particle can lead to the creation of thousands of secondaries that can each create many energy deposition hits. In the case of a simple histogram, all energy deposition hits across all events are binned. This gives us a total for the simulation performed and the bin error (uncertainty associated with a given histogram bin) is proportional to $1/\sqrt{N}$, where $N$ is the number of entries in that bin. This, however, doesn’t correctly represent the variation seen from event to event. Using the per-event histograms, a single simple 1D histogram of energy deposition is created and these are averaged. The resultant histogram has the mean per-event (note the normalisation here versus the simple histograms) and the error on the bin is the standard error on the beam, i.e.

$$\text{bin error} = \frac{\sigma}{\sqrt{n_{\text{events}}}}$$

where $\sigma$ is the standard deviation of the values in that bin for all events.

Note: Per-entry histograms will only be calculated where there exists two or more entries in the tree. In the case of the Event tree, this corresponds to more than two events.

10.3.3 Analysis Configuration File

The input text file has roughly two sections: options and histogram definitions.

Examples can be found in:
- `<bdsim>/examples/features/io/1_rootevent/analysisConfig.txt`
- `<bdsim>/examples/features/analysis/simpleHistograms/analysisConfig.txt`
- `<bdsim>/examples/features/analysis/perEntryHistograms/analysisConfig.txt`

<table>
<thead>
<tr>
<th>Debug</th>
<th>True</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputFilePath</td>
<td>./output.root</td>
</tr>
<tr>
<td>OutputFileName</td>
<td>./ana_1.root</td>
</tr>
<tr>
<td>CalculateOpticalFunctions</td>
<td>True</td>
</tr>
<tr>
<td>CalculateOpticalFunctionsFileName</td>
<td>./ana_1.dat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># Object</th>
<th>Tree Name</th>
<th>Histogram Name</th>
<th># of Bins</th>
<th>Binning</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>Histogram1D Event.</td>
<td>Primaryx</td>
<td>100</td>
<td>{-0.1:0.1}</td>
<td>Primary.x</td>
</tr>
<tr>
<td></td>
<td>Histogram1D Event.</td>
<td>Primaryy</td>
<td>100</td>
<td>{-0.1:0.1}</td>
<td>Primary.y</td>
</tr>
<tr>
<td></td>
<td>Histogram1D Options.</td>
<td>seedState</td>
<td>200</td>
<td>0:200</td>
<td>Options.</td>
</tr>
<tr>
<td></td>
<td>Histogram1D Model.</td>
<td>componentLength</td>
<td>100</td>
<td>0.0:100</td>
<td>Model.length</td>
</tr>
<tr>
<td></td>
<td>Histogram1D Run.</td>
<td>runDuration</td>
<td>1000</td>
<td>0:1000</td>
<td>Summary.</td>
</tr>
<tr>
<td></td>
<td>Histogram2D Event.</td>
<td>XvsY</td>
<td>100,100</td>
<td>{-0.1:0.1,-0.1:0.1}</td>
<td>Primary.</td>
</tr>
<tr>
<td></td>
<td>Histogram3D Event.</td>
<td>PhaseSpace3D</td>
<td>50,50,50</td>
<td>{-5e-6:5e-6,-5e-6:5e-6,-5e-6:5e-6}</td>
<td>Primary.x:Primary.y:Primary.z</td>
</tr>
<tr>
<td></td>
<td>Histogram1DLog Event.</td>
<td>PrimaryXAbs</td>
<td>20</td>
<td>{-9:-3}</td>
<td>abs(Primary.x)</td>
</tr>
<tr>
<td></td>
<td>Histogram2DLinLog Event.</td>
<td>PhaseSpaceAbs</td>
<td>20,20</td>
<td>{-1e-6:1e-5,-9:-3}</td>
<td>Primary.</td>
</tr>
<tr>
<td></td>
<td>Histogram2DLog Event.</td>
<td>PhaseSpaceAbs2</td>
<td>20,20</td>
<td>{-9:-3,-1e-6:1e-5}</td>
<td>abs(Primary.x):Primary.y</td>
</tr>
</tbody>
</table>

- *HistogramND* defines an N-dimension per-entry histogram where $N$ is 1, 2 or 3.

10.3. rebdsim - General Analysis Tool

177
• SimpleHistogramND defines an N-dimension simple histogram where \( N \) is 1, 2 or 3.

• Arguments in the histogram rows must not contain any white space!

• Columns in the histogram rows must be separated by any amount of white space (at least one space).

• A line beginning with \# is ignored as a comment line.

• Empty lines are also ignored.

• For bins and binning, the dimensions are separated by , .

• For bins and binning, the range from low to high is specified by low:high.

• For a 2D or 3D histogram, x vs. y variables are specified by samplername_y:samplername_x. See warning below for order of variables.

• Variables must contain the full ‘address’ of a variable inside a Tree.

• Variables can also contain a value manipulation, e.g. 1000*(Primary.energy-0.938) (to get the kinetic energy of proton primaries in MeV).

• The selection is a weight. In the case of the Boolean expression, it is a weight of 1 or 0.

• Selection can be a Boolean operation (e.g. Primary.x>0) or simply 1 for all events.

• Multiple Boolean operations can be used e.g. Primary.x>0&&samplername.ParentID!=0.

• If a Boolean and a weight is desired, multiply both with the Boolean in brackets, e.g. Eloss.energy*(Eloss.S>145.3).

• True or False, as well as 1 or 0, may be used for Boolean options at the top.

• ROOT special variables can be used as well, such as Entry$ and Entries$. See the documentation link immediately below.

Note: Per-entry histograms will only be calculated where there exists two or more entries in the tree. In the case of the Event tree, this corresponds to more than two events. Whilst the per-entry histograms will work for any tree in the output, they are primarily useful for per-event analysis on the Event tree.

A full explanation on the combination of selection parameters is given in the ROOT TTree class: https://root.cern.ch/doc/master/classTTree.html. See the “Draw” method and “selection”.

10.3.4 Logarithmic Binning

Logarithmic binning may be used by specifying ‘Log’ after ‘HistogramND’ for each dimension. The dimensions specified in order are \( x, y, z \). If a linearly spaced dimension is required, the user should write ‘Lin’. If nothing is specified it is assumed to be linear.

Examples:

```
Histogram1D // linearly spaced
Histogram1DLog // logarithmically spaced
Histogram2D // X and Y are linearly spaced
Histogram2DLog // X is logarithmically spaced and Y linearly
Histogram2DLinLog // X is linearly spaced and Y logarithmically
```

The bin’s lower edges and upper edges should be an exponent of 10. For example, to generate a 1D histogram with thirty logarithmically spaced bins from 1e-3 to 1e3, the following syntax would be used:

```
```
**Warning:** The variable for plotting is really a simple interface to CERN ROOT’s TTree Draw method. This has some inconsistency. If 1D, there is just \( x \). If 2D, it’s \( y : x \). If 3D, it’s \( x : y : z \). This only applies to the variable and not to the bin specification.

### 10.3.5 Analysis Configuration Options

The following (case-insensitive) options may be specified in the top part.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackwardsCompatible</td>
<td>ROOT event output files from BDSIM prior to v0.994 do not have the header structure that is used to ensure the files are the right format and prevent a segfault from ROOT. If this option is true, the header will not be checked, allowing old files to be analysed.</td>
</tr>
<tr>
<td>CalculateOptics</td>
<td>Whether to calculate optical functions or not</td>
</tr>
<tr>
<td>Debug</td>
<td>Whether to print out debug information</td>
</tr>
<tr>
<td>EmittanceOnTheFly</td>
<td>Whether to calculate the emittance freshly at each sampler or simply use the emittance calculated from the first sampler (i.e. the primaries). The default is false and therefore calculates the emittance at each sampler.</td>
</tr>
<tr>
<td>EventStart</td>
<td>Event index to start from - zero counting. Default is 0.</td>
</tr>
<tr>
<td>EventEnd</td>
<td>Event index to finish analysis at - zero counting. Default is -1 that represents how ever many events there are in the file (or files if multiple are being analysed at once).</td>
</tr>
<tr>
<td>InputFilePath</td>
<td>The root event file to analyse (or regex for multiple).</td>
</tr>
<tr>
<td>MergeHistograms</td>
<td>Whether to merge the event level default histograms provided by BDSIM. Turning this off will significantly improve the speed of analysis if only separate user-defined histograms are desired.</td>
</tr>
<tr>
<td>OutputFileName</td>
<td>The name of the result file written to</td>
</tr>
<tr>
<td>OpticsFileName</td>
<td>The name of a separate text file copy of the optical functions output</td>
</tr>
<tr>
<td>PrintModuloFraction</td>
<td>The fraction of events to print out (default 0.01). If you require print out for every event, set this to 0.</td>
</tr>
<tr>
<td>ProcessSamplers</td>
<td>Whether to load the sampler data or not</td>
</tr>
</tbody>
</table>

### 10.3.6 Variables In Data

See [Basic Data Inspection](#) for how to view the data and make the most basic on-the-fly histograms.

### 10.4 rebdsimCombine - Output Combination

`rebdsimCombine` is a tool that can combine `rebdsim` output files correctly (i.e. the mean of the mean histograms) to provide the overall mean and error on the mean, as if all events had been analysed in one execution of `rebdsim`.

The combination of the histograms from the `rebdsim` output files is very quick in comparison to the analysis. `rebdsimCombine` is used as follows:

```
rebdsimCombine <result.root> <file1.root> <file2.root> ....
```

where `<result.root>` is the desired name of the merge output file and `<file.root>` etc. are input files to be merged. This workflow is shown schematically in the figure below.
10.5 rebdsimHistoMerge - Simple Histogram Merging

BDSIM, by default, records a few histograms per event that typically include the primary particle impact and loss location as well as the energy deposition. The histograms are stored in vectors inside the Event tree of the output. These cannot be viewed directly in the ROOT TBrowser as they are in a vector. Even then, each histogram is for one event only. To view the average of all the histograms, a dedicated tool is provided that provides a subset of the rebdsim functionality. rebdsim would automatically combine these histograms while performing analysis.

A tool rebdsimHistoMerge is provided to take the average of only the already existing histograms without the need to prepare an analysis configuration file. It is run as follows:

```
rebdsimHistoMerge output.root results.root
```

This creates a ROOT file called “results.root” that contains the average histograms across all events. This can only operate on BDSIM output files, not rebdsim output files.

10.6 rebdsimOptics - Optical Functions

rebdsimOptics is a tool to load sampler data from a BDSIM output file and calculate optical functions as well as beam sizes. It is run as follows:

```
rebdsimOptics output.root optics.root
```

This creates a ROOT file called “optics.root” that contains the optical functions of the sampler data.

See Optical Validation for more details.

10.7 Speed & Efficiency

Whilst the ROOT file IO is very efficient, the sheer volume of data to process can easily result in slow running analysis. To combat this, only the minimal variables should be loaded that need to be. rebdsim automatically activates only the ‘ROOT branches’ it needs for the analysis. A few possible ways to improve performance are:

- Reduce number of 2D or 3D histograms if possible. Analysis is linear in time with number of bins.
- Remove unnecessary histograms from your analysis configuration file.
- Avoid unnecessary filters in the selection.
- Turn off optical function calculations if they’re not needed or don’t make sense, i.e. if you’re analysing the spray from a collimator in a sampler, it makes no sense to calculate the optical functions of that distribution.
- Turn off the MergeHistograms option. If you’re only making your own histograms, this should considerably speed up the analysis for a large number of events.

Simple histograms to not require loading each entry in the tree and an analysis with only simple histograms will be quicker. Per-entry histograms of course, require loading each entry.

rebdsim “turns off” the loading of all data and only loads what is necessary for the given analysis.

10.7.1 Scaling Up - Parallelising Analysis

For high-statistics studies, it’s common to run multiple instances of BDSIM with different seeds (different seeds ensures different results) on a high throughout the computer cluster. There are two possible strategies to efficiently scale the statistics and analysis; both produce numerically identical output but make different use of computing resources. The more data stored per event in the output files, the longer it takes to load it from disk and the longer the analysis. Similarly, the more events simulated, the longer the analysis will take. Of course either strategy can be used.
10.7.2 Low-Data Volume

If the overall output data volume is relatively low, we recommend analysing all of the output files at once with rebdsim. In the Analysis Configuration File file, the InputFilePath should be specified as "*.root" to match all the root files in the current directory.

**Note:** For "*.root" all files should be from the same simulation and only BDSIM output files (i.e. not rebdsim output files).

rebdsim will ‘chain’ the files together to behave as one big file with all of the events. This is shown schematically in the figure below.

![Schematic of strategy for a low volume of data produced from a computationally intense simulation.](image)

**Fig. 1:** Schematic of strategy for a low volume of data produced from a computationally intense simulation. Multiple instances of BDSIM are executed, each producing their own output file. These are analysed all at once with rebdsim.

This strategy works best for a relatively low number of events and data volume (example numbers might be < 10000 events and < 10 GB of data).

10.7.3 High-Data Volume

In this case, it is better to analyse each output file with rebdsim separately and then combine the results. In the case of per-event histograms, rebdsim provides the mean per event, along with the error on the mean for the bin error. A separate tool, rebdsimCombine, can be used to combine these rebdsim output files into one single file. This is numerically equivalent to analysing all the data in one execution of rebdsim but significantly faster. See rebdsimCombine - Output Combination for more details.
Fig. 2: Schematic of strategy for a high-data volume analysis. Multiple instances of BDSIM are executed in a script that then executes rebdsim with a suitable analysis configuration. Only the output files from rebdsim are then combined into a final output identical to what would have been produced from analysing all data at once, but in vastly reduced time.
10.8 User Analysis

Whilst rebdsim will cover the majority of analyses, the user may desire a more detailed or customised analysis. Methods to accomplish this are detailed here for interactive or compiled C++ with ROOT, or through Python.

The classes used to store and load data in BDSIM are packaged into a library. This library can be used interactively in Python and ROOT to load the data manually.

10.8.1 Analysis in Python

This is the preferred method. Analysis in Python can be done using ROOT in Python directly or through our library pybdsim (see Python Utilities).

**Note:** ROOT must have been installed or compiled with Python support.

You can test whether ROOT works with your Python installation by starting Python and trying to import ROOT - there should be no errors.

```
>>> import ROOT
```

The library containing the analysis classes may be then loaded:

```
>>> import ROOT
>>> ROOT.gSystem.Load("librebdsim")
>>> ROOT.gSystem.Load("libbdsimRootEvent")
```

The classes in bdsim/analysis will now be available inside ROOT in Python.

This can also be conveniently achieved with pybdsim:

```
>>> import pybdsim
>>> pybdsim.Data.LoadROOTLibraries()
```

This raises a Python exception if the libraries aren’t found correctly. This is done automatically when any BDSIM output file is loaded using the ROOT libraries.

**IPython**

We recommend using IPython instead of pure Python to allow interactive exploration of the tools. After typing at the IPython prompt for example pybdsim., press the tab key and all of the available functions and objects inside pybdsim (in this case) will be shown.

For any object, function or class, type a question mark after it to see the docstring associated with it.

```
>>> import pybdsim
>>> d = pybdsim.Data.Load("combined-ana.root")
>>> d.
```

**General Data Loading**

Any output file from the BDSIM set of tools can be loaded with:
This will work for files from BDSIM, rebdsim, rebdsimCombine, rebdsimHistoMerge and rebdsimOptics. This function may return a different type of object depending on the file that was loaded. The two types are DataLoader, which is the same as the rebdsim C++ class but in Python, and RebdsimFile (defined in pybdsim/pybdsim/ Data.py), which is a Python class to hold the output from a rebdsim output file and conveniently convert ROOT histograms to numpy arrays. The type can easily be inspected:

```python
>>> type(d)
pybdsim.Data.RebdsimFile
```

### Looping Over Events

The following is an example of how to loop over events in a BDSIM output file using pybdsim.

```python
>>> import pybdsim
>>> import numpy

>>> d = pybdsim.Data.Load("myoutputfile.root")

>>> eventTree = d.GetEventTree()

>>> for event in eventTree:
...   print numpy.array(event.Primary.x)
```

In this example, the variable `event` will have the same structure as the Event tree in the BDSIM output. See `Basic Data Inspection` for more details on how to browse the data.

**Note:** The branch “Summary” in the Event and Run trees used to be called “Info” in BDSIM < V1.3. This conflicted with TObject::Info() so this looping in Python would work for any data in this branch, hence the change.

### Sampler Data

The following shows the convenience methods to access sampler data from a BDSIM output file using pybdsim:

```python
>>> import pybdsim
>>> import numpy

>>> d = pybdsim.Data.Load("myoutputfile.root")

>>> primaries = pybdsim.Data.SamplerData(d)

>>> primaries.data.keys()
['weight', 'trackID', 'energy', 'turnNumber', 'parentID', 'xp', 'zp', 'rigidity', 'ionZ', 'charge', 'ionA', 'modelID', 'S', 'T', 'yp', 'partID', 'n', 'mass', 'y',
```
The `SamplerData` function has an optional second argument that takes the index (zero counting) of the sampler or the name as it appears in the file. This includes the primaries ("Primary").

**Note:** This loads all data into memory at once and is generally not as efficient as looping over event by event. This is provided for convenience, but may not scale well to very large data sets.

**Warning:** This concatenates all events into one array, so the event by event nature of the data is lost. This may be acceptable in some cases, but it is worth considering making a 2D histogram directly using `rebdsim` rather than say loading the sampler data here and making a 2D plot. Certainly, if the statistical uncertainties are to be calculated, this is a far preferable route.

### REBDSIM Histograms

Output from `rebdsim` can be loaded using `pybdsim`. The histograms made by `rebdsim` are loaded as the ROOT objects they are, but are also converted to numpy arrays using classes provided by `pybdsim` for convenience. The Python converted ones are held in dictionaries suffixed with ‘py’. The histograms are loaded into dictionaries where the key is a string with the full path and name of the histogram in the `rebdsim` output file. The value is the histogram from the file.

```python
>>> import pybdsim
>>> d = pybdsim.Data.Load("rebdsimoutputfile.root")
>>> d.histograms
{'Event/MergedHistograms/ElossHisto': <ROOT.TH1D object ("ElossHisto") at 0x7fbe365e9520>,
 'Event/MergedHistograms/ElossPEHisto': <ROOT.TH1D object ("ElossPEHisto") at 0x7fbe365ea750>,
 'Event/MergedHistograms/ElossTunnelHisto': <ROOT.TH1D object ("ElossTunnelHisto") at 0x7fbe365eab40>,
 'Event/MergedHistograms/ElossTunnelPEHisto': <ROOT.TH1D object ("ElossTunnelPEHisto") at 0x7fbe365ea9e0>,
 'Event/MergedHistograms/PhitsHisto': <ROOT.TH1D object ("PhitsHisto") at 0x7fbe365ea9c0>,
 'Event/MergedHistograms/PhitsPEHisto': <ROOT.TH1D object ("PhitsPEHisto") at 0x7fbe365e9bc0>,
 'Event/MergedHistograms/PlossHisto': <ROOT.TH1D object ("PlossHisto") at 0x7fbe365ea8fc0>,
 'Event/MergedHistograms/PlossPEHisto': <ROOT.TH1D object ("PlossPEHisto") at 0x7fbe365e8a0>,
 'Event/PerEntryHistograms/EnergyLossManual': <ROOT.TH1D object ("EnergyLossManual") at 0x7fbe365a3a50>,
 'Event/PerEntryHistograms/EnergySpectrum': <ROOT.TH1D object ("EnergySpectrum") at 0x7fbe365a2e20>,
 'Event/PerEntryHistograms/EventDuration': <ROOT.TH1D object ("EventDuration") at 0x7fbe325907b0>,
 'Event/PerEntryHistograms/TunnelDeposition': <ROOT.TH3D object ("TunnelDeposition") at 0x7fbe35e2c800>,
 'Event/PerEntryHistograms/TunnelLossManual': <ROOT.TH1D object ("TunnelLossManual") at 0x7fbe365a40b0>,
 'Event/SimpleHistograms/Primaryx': <ROOT.TH1D object ("Primaryx") at 0x7fbe325cf9d0>,
```

10.8.2 Analysis in C++ or ROOT

The following commands can be used as either compiled C++ or as interactive C++ using ROOT. Here, we show their usage using ROOT interactively.

When using ROOT’s interpreter, you can use the functionality of the BDSIM classes dynamically. First, you must load the shared library (if not done so in your ROOT logon macro) to provide the classes:

```cpp
root> gSystem->Load("librebdsim");
root> gSystem->Load("libbdsimRootEvent");
```

Loading this library exposes all classes that are found in `<bdsim>/analysis`. If you are familiar with ROOT, you may use the ROOT file as you would any other given the classes provided by the library:

```cpp
root> TFile* f = new TFile("output.root", "READ");
root> TTree* eventTree = (TTree*)f->Get("Event");
root> BDSOutputROOTEventLoss* elosslocal = new BDSOutputROOTEventLoss();
root> eventTree->SetBranchAddress("Eloss.", &elosslocal);
root> eventTree->GetEntry(0);
root> cout << elosslocal->n << endl;
```

The header (".hh") files in `<bdsim>/analysis` provide the contents and abilities of each class.

### General Data Loading

This would of course be fairly tedious to load all the structures in the output. Therefore, a data loader class is provided that constructs local instances of all the objects and sets the branch address on them (links them to the open file). For example:

```cpp
root> gSystem->Load("librebdsim");
root> gSystem->Load("libbdsimRootEvent");
root> DataLoader* dl = new DataLoader("output.root");
root> Event* evt = dl->GetEvent();
root> TTree* evtTree = dl->GetEventTree();
```
Here, a file is loaded and by default all data is loaded in the file.

**Looping Over Events**

We get access to event by event information through a local event object and the linked event tree (here, a chain of all files) provided by the DataLoader class. We can then load a particular entry in the tree, which for the Event tree is an individual event:

```
root> evtTree->GetEntry(10);
```

The event object now contains the data loaded from the file.

```
root> evt->Eloss.n
(int_t) 430
```

For our example, the file has 430 entries of energy loss for event #10. The analysis loading classes are designed to have the same structure as the output file. Look at `bdsim/analysis/Event.hh` to see what objects the class has.

One may manually loop over the events in a macro:

```c
void DoLoop()
{
    gSystem->Load("librebdsim");
    DataLoader* dl = new DataLoader("output.root");
    Event* evt = dl->GetEvent();
    TTree* evtTree = dl->GetEventTree();
    int nentries = (int)evtTree->GetEntries();
    for (int i = 0; i < nentries; ++i)
    {
        evtTree->GetEntry(i)
        std::cout << evt->Eloss.n >> std::endl;
    }
}
```

```
root> .L myMacro.C
root> DoLoop()
```

This would loop over all entries and print the number of energy deposition hits per event.

**Sampler Data**

Samplers are dynamically added to the output based on the names the user decides in their input accelerator model. The names of the samplers can be accessed from the DataLoader class:

```
std::vector<std::string> samplerNames = dl->GetSamplerNames();
```

**REBDSIM Histograms**

To load histograms, the user should open the ROOT file and access the histograms directly:

```
root> TFile* f = new TFile("output.root");
root> TH1D* eloss = (TH1D*)f->Get("Event/MergedHistograms/ElossHist0");
```

It is recommended to use a TBrowser to get the exact names of objects in the file.

**Output Classes**

The following classes are used for data loading and can be found in `bdsim/analysis:`
10.9 Numerical Methods

Algorithms used to accurately calculate quantities are described here. These are documented explicitly as a simple implementation of the mathematical formulae would result in an inaccurate answer in some cases.

10.9.1 Numerically Stable Calculation of Mean & Variance

To calculate the mean in the per-entry histograms as well as the associated error (the standard error on the mean), the following formulae are used:

\[
\bar{x} = \frac{1}{n} \sum_{i=0}^{n} x_i \\
\sigma_x = \frac{1}{\sqrt{n}} \frac{\sigma}{\sqrt{n}} = \frac{1}{\sqrt{n}} \sqrt{\frac{1}{n} \sum_{i=0}^{n} (x_i - \bar{x})^2}
\]

These equations are however problematic to implement computationally. The formula above for the variance requires two passes through the data to first calculate the mean, then the variance using that mean. The above equation can be rearranged to provide the same calculation with a single pass through the data, however, such algorithms are typically numerically unstable, i.e. they rely on a small difference between two very large numbers. With the finite precision of a number represented in a C++ double type (~15 significant digits), the instability may lead to un-physical results (negative variances) and generally incorrect results.

The algorithm used in rebdsim to calculate the means and variances is an online, single-pass numerically stable one. This means that the variance is calculated as each data point is accumulated, it requires only one pass of the data, and does not suffer numerical instability. To calculate the mean, the following recurrence relation is used:

\[
\bar{x}_{i=0} = 0 \\
\bar{x}_{i+1} = \bar{x}_i + \frac{(x_i - \bar{x}_i)}{i}
\]

for \(i \in [1 \ldots n_{\text{event}}]\)

The variance is calculated with the following recurrence relation that requires the above online mean calculation:

\[
Var(x)_{i=0} = 0 \\
Var(x)_{i+1} = Var(x)_i + (x_i - \bar{x}_i)(x_i - \bar{x}_{i+1})
\]

for \(i \in [1 \ldots n_{\text{event}}]\)

After processing all entries, the variance is used to calculate the standard error on the mean with:

\[
\sigma_x = \frac{1}{\sqrt{n}} \sqrt{\frac{1}{\sqrt{n} - 1} Var(x)}
\]
10.9.2 Merging Histograms

`rebdsimCombine` merges histograms that already have the mean and the error on the mean in each bin. These are combined with a separate algorithm that is also numerically stable.

The mean is calculated as:

\[
\bar{x}_{i=0} = 0 \\
\delta = x_{i+1} - \bar{x}_i \\
\bar{x}_{i+1} = \bar{x}_i + n_{i+1} \frac{\delta}{n_i + n_{i+1}}
\]

\[
Var(x)_{i=0} = 0 \\
Var(x)_{i+1} = Var(x)_i + Var(x)_{i+1} + (n_i n_{i+1} \delta^2) \\
\quad \quad \text{for } i = 1 \ldots n_{rebdsim \text{ files}}
\]
Several python packages are provided to aid preparation and conversion of models to and from BDSIM’s gmad format and are described in the following sections.

- pybdsim - conversion and building of input as well as ASCII data analysis
- pymadx - file loader for MAD-X TFS file format
- pymad8 - file loader for MAD8 file format
- pytransport - file loader for Transport file format

11.1 Installing Python Utilities

Installation is described in the setup section in *Python Utilities*.

11.2 Documentation

In the following sections is individual documentation for each python package. These are also available when using ipython by typing ‘?’ after any module, class or function.


11.3 Geometry Preparation

For preparation of custom geometry, we have developed a Python geometry package called *pyg4ometry*. This allows scripting of geometry creation in Python, visualisation and overlap checking outside Geant4. The emitted GDML files can be loaded in BDSIM.

Please visit [https://bitbucket.org/jairhul/pyg4ometry](https://bitbucket.org/jairhul/pyg4ometry)
When BDSIM is executed without the --batch executable option, the Geant4 visualisation system is used. This is the default behaviour as it is typical to view and understand the typical outcome of a model before running larger simulations in batch mode. By default, BDSIM uses the OpenGL Qt visualiser as it’s very easy to work with and has a rich feature set.

To use this visualisation, Geant4 must have been compiled with the Qt visualiser option turned on, which is non-default. Qt is however, the most widely used visualiser and recommended by Geant4. See Adding Other Visualisers for more.

Below are the most commonly used commands and the full set of features is described in Visualisation Features.

Using the visualiser means that BDSIM will use more memory as it is required to store the trajectories of all particles to display them. By default, only 100 events will be accumulated for display. Events outside this will be discarded from the visualisation.

12.1 Colour Coding

The default colour coding of trajectories is the same as Geant4. This colour coding is green: neutral; blue: positively charged; red: negatively charged.

12.2 Common Useful Commands

Execute BDSIM with your input gmad file name:

bdsim --file=sm.gmad

The following is a list of our most commonly used commands that can be used in the session box (terminal prompt) inside the visualiser:

- /run/beamOn 3 - Runs three primary events
- exit - Exits the visualiser and BDSIM
- /vis/viewer/set/viewpointThetaPhi 0 90 - Sets the view point angle
- /vis/scene/add/axes 0 0 0 - Adds a set of unit vector axes at position (0,0,0)
- /vis/drawVolume worlds - view all invisible geometry including samplers
- /vis/viewer/set/lightsVector 1 1 1 - change the orientation of the lighting to roughly opposite
- /vis/viewer/set/viewpointThetaPhi 180 0 - look along the beam line
- /bds/beaml ine/goto d1 - reposition the view point at beam line element named d1
- /vis/viewer/set/projection p 75 - set the viewing style to perspective with angle 75 degrees
- /vis/viewer/set/projection o - set the projection to orthographic (no perspective)
• `/vis/viewer/addCutawayPlane 0 0 0 m 1 0 0` - add a cut away plane along the beam line making everything on one side invisible.
• `/vis/viewer/clearCutawayPlanes` - get rid of cut away planes

### 12.3 Visualisation Features

The default OpenGL Qt visualiser is shown below.

The visualiser is shown again below with some interesting parts highlighted. These are:

- **Green dashed box middle** Main visualiser window - view of the model
- **Purple dashed box top left** Control buttons that are described in more detail in *Control Buttons*
- **Blue box on the left** Scene tree - expand this to see a full list of all volumes in the simulation.
- **Orange box top left** Help browser where you can search for all commands in the visualiser
- **Red box bottom** Session - enter commands here.

#### 12.3.1 Control Buttons

#### 12.3.2 Drawing Styles

The model may be viewed as a wireframe model, wireframe and solid and in all cases with or without perspective. Some examples of this are shown below for the same model. These are all controlled easily from the buttons at the top. There are also commands that will work to control these as documented in Geant4.
12.3. Visualisation Features
Fig. 1: As a wireframe model.
Fig. 2: With both solid and wireframe visualisation (subtle lines on each piece of geometry).
Fig. 3: With perspective.
12.4 Default and Custom Visualisers

Strictly speaking, a visualisation macro must be supplied to Geant4 to tell it what to display. For convenience, BDSIM provides a set of macros that display the geometry and add a few useful buttons and menus to the user interface. To use these, the user need only not specify a specific visualisation macro.

```
bdsim --file=mylattice.gmad
```

- Note also no --batch command

If you wish to use a different visualiser, you may specify this by using your own visualisation macro with BDSIM. This can be done using the following command:

```
bdsim --file=mylattice.gmad --vis_mac=othervis.mac
```

where othervis.mac is your visualisation macro. It is recommended to copy and edit the default BDSIM visualisation macro (vis.mac) that can be found in the bdsim source directory as follows:

```
bdsim/vis/*.mac
```

When running, BDSIM looks for the macros in the installation directory and then the build directory if it exists. The user can edit this files directly as a default for BDSIM on their system. (e.g. `<bdsim-install-dir>/vis/*.mac`).

The user can also specify an optional macro to run after the visualisation has started. This way, you can use the default BDSIM visualisation but run your own macro at the beginning. This may be useful for particular view points or visualisation settings.

```
bdsim --file=mylattice.gmad --geant4Macro=viewpoint.mac
```

Note: This macro is run after the geometry is ‘closed’ in Geant4 terminology and the physics list is fixed.

12.5 Adding Other Visualisers

BDSIM makes use of the visualisers Geant4 was compiled with on your system. When BDSIM is started interactively (i.e. without the --batch command), Geant4 will print a list of all available visualisers that are available. Below is an example excerpt from the terminal output that shows the list of available visualisers on the developer’s system.

By default, BDSIM uses the OpenGL Qt visualiser - we highly recommend this, as it is the most modern one with the best feature set. It is also recommended by Geant4.

To add another available visualiser, you must change the build options of Geant4 (in ccmake), recompile and install it; then you must recompile BDSIM against the new Geant4. In the case where you simply update the Geant4 options in the same installation, this process is relatively quick and recompiling BDSIM only re-links the libraries together (the last quick step of compilation).

For Geant4 to enable other visualisers, it will require certain other 3rd party libraries to be present. On Mac, these can be found through a package manager such as MacPorts and on linux, through whatever package manager is available (e.g. yum). These must be installed before reconfiguring Geant4.

Visualization Manager instantiating with verbosity "warnings (3)"
Visualization Manager initialising...
Registering graphics systems...
You have successfully registered the following graphics systems.
Current available graphics systems are:
  ASCII free (AFree)
  DAWNFILE (DAWNFILE)
  G4HepRep (HepRepXML)
  G4HepRepFile (HepRepFile)
  RayTracer (RayTracer)
  VMML1FILE (VMML1FILE)
  VMML2FILE (VMML2FILE)
  gMocrenFile (gMocrenFile)
  OpenGLImmediatedt (GGLI0t, OGLI)
  OpenGLstoredt (GGL50t, OGL, OGLS)
  OpenGLimmediatex (GGLIXm, GGL0t_FALLBACK)
  OpenGLstoredx (GGL5Xm, GGL50t_FALLBACK)
  RayTracerX (RayTracerX)
Registering model factories...
You have successfully registered the following model factories.
Registered model factories:
  generic
drawByAttribute
drawByCharge
drawByOriginVolume
INTERFACING

As BDSIM is completely open source, it is possible to extend BDSIM or interface it with other software. This ability is driven by the needs of users, so we strongly encourage users or would-be developers to get in touch with the developers (see Support). We welcome extensions and contributions to BDSIM.

Below are details of various methods to extend BDSIM.

13.1 Compiling Against BDSIM

BDSIM includes CMake files in the installation so that another CMake application can use the BDSIM classes and libraries. CMake is a system to automatically create Make files for different operating systems and different computer systems with libraries in various locations. CMake is used as the build system for BDSIM, but also for Geant4, ROOT and CLHEP and is a common C++ software paradigm. Below is the most minimal CMakeLists.txt to create a C++ application using BDSIM:

```cmake
# cmake_minimum_required (VERSION 3.2)
project (externalcpp)
# point cmake to my own custom installation directory that's not a system dir
# this is where BDSIMConfig.cmake exists
set(CMAKE_PREFIX_PATH /Users/nevay/physics/reps/bdsim-develop-install/lib/cmake/bdsim)
# find the package and set up variables
find_package(BDSIM REQUIRED)
# define program includes
include_directories(${BDSIM_INCLUDE_DIR}/bdsim)
# make a program and link to gmad (parser) and bdsim libraries
add_executable(customprogram customprogram.cc)
target_link_libraries (customprogram gmad bdsim)
```

Along with this is a single C++ file called customprogram.cc. Below are the contents of this:

```cpp
#include "BDSIMClass.hh"
#include <iostream>

int main(int argc, char** argv) {
    BDSIM* bds = new BDSIM(argc, argv);
    if (!bds->Initialised())
        {std::cout << "Initialisation failed" << std::endl; return 1;}

    std::cout << "Custom stuff here" << std::endl;
```

(continues on next page)
bds->BeamOn();
delete bds;
return 0;
}

This example is provided in bdsim/examples/features/interfaces/externalcpp. The user should edit the CMakeLists.txt so that CMAKE_PREFIX_PATH points to their BDSIM installation directory if not in a system directory to allow CMake to find that installation.

13.2 Custom Beam Line Component

**Warning:** If there are any geometrical overlaps (broken hierarchy, touching solids, overlapping separate solids at the same level of hierarchy), the Geant4 tracking may be wrong and the results cannot be trusted. This may lead to slow running models, inaccurate results, excessive navigation warnings, or worst of all: no warnings but inaccurate results. When developing custom geometry, the developer must ensure no geometrical overlaps are present before the model is used for a physics study.

Whilst BDSIM provides the most common accelerator beam line components, we cannot foresee custom components that various accelerators may have. To insert a custom component, we would recommend using a geometry package such as `pyg4ometry` to prepare GDML geometry and using a generic beam line element `element`. See Geometry Preparation and the generic beam line element object, `element`. A field map can also be overlaid on this.

However, if the user is familiar with Geant4 C++, Geant4 geometry or requires a more detailed interaction with the simulation, it is possible to add a custom C++ beam line component to BDSIM. The user must define:

- A class that constructs the beam line component and that inherits BDSAcceleratorComponent.
- A factory class that constructs an instance of the component. This should inherit BDSComponentConstructor and translates information provided by BDSIM from the parsing of the input text files.
- A C++ main program that uses BDSIM and will be executed like BDSIM.
- A CMake file to compile the application sources.

A complete example is provided in bdsim/examples/features/interfaces/usercomponent that is described here. The contents of the directory are shown below.

This example builds a custom vertical dipole spectrometer. This makes use of the magnet geometry and beam pipe factories to build a magnet with custom proportions and an offset beam pipe with a screen inside it.
13.2. Custom Beam Line Component
### 13.2.1 Input GMAD

The key part in the input GMAD is to define a `usercomponent` beam line element. This takes and argument `userTypeName` to define the type of the element if more than one user component is registered. This beam line element can now be used normally in any line in BDSIM. To convey parameters to the new user-defined element, any parameter available for any other element may be used. These are defined in `parser/element.hh`.

Additional parameters may be supplied via the element member string “userParameters”. This should be a string with space delimited parameter value sets where each parameter and value are separated by a colon. For example:

```
userParameters="variable1:0.4 variable2:bananas"
```

The utility function `BDS::GetUserParametersMap` from `#include "BDSUtilities.hh"` will split this up into a map of strings to strings such as:

<table>
<thead>
<tr>
<th>Map Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable1</td>
<td>&quot;0.4&quot;</td>
</tr>
<tr>
<td>variable2</td>
<td>&quot;bananas&quot;</td>
</tr>
</tbody>
</table>

The variables are left as strings and it is up to the developer to know which variables to convert to numbers if required. The map can also be searched if some variables are optional. The usercomponent example shows this for passing the colour into the new element.

### 13.2.2 Component Class

The component class in this example is called “UDipole”. The user component can have any constructor it likes, but it must inherit BDSAcceleratorComponent and provide a name, length, angle it bends the beam line by and a string saying the name of the component. This information is used to construct the beam line and is passed to the output.

**Note:** The “length” is the arc length if there is a finite angle. If there is a finite angle, this is assumed to bend the beam line continuously throughout the component. e.g. no ‘S’ shaped component. The coordinate system is right-handed and a positive angle causes deflection to negative x coordinate while propagating in the direction z.

BDSIM makes extensive use of the concept of factories. These are bits of code that take a recipe class - a small class or struct with some simple parameters - and builds an object. The factory retains no ownership of the object and forgets about it. We use this pattern to create beam pipes and magnets for example and these can be placed inside each other or alongside each other safely.

Each object inherits BDSGeometryComponent and this deals with the memory management (ownership) of objects and the extent of the object.
13.2.3 Component Constructor

This is a class the user must implement that inherits BDSComponentConstructor. The user must implement a method called Construct that has pointers to the beam line element from the parser (GMAD::Element) as well as the ones before and after it. It should also make sure to change into Geant4 units from the parser’s SI units.

13.2.4 Class Documentation

BDSIM uses Doxygen for class documentation. This is a series of comments in C++ with extra comment characters that are built into a documentation system. Please look through the Doxygen website for BDSIM http://www.pp.rhul.ac.uk/bdsim/doxygen or the headers of the source code in bdsim/include/*.hh.
BDSIM includes a series of examples to illustrate its usage. These also form the basis of the test suite to ensure stable development and record any changes. Each example is self-contained and instructions are provided both in this documentation and in the .rst files beside each example.

14.1 Basic Examples

14.1.1 Air Water Target

- Based on bdsim/examples/airwatertarget/air.gmad

This is a trivial example to show how to create a section of air between beam line elements or at the end of a beam line.

The strategy is to create a drift without a surrounding beam pipe by using the `apertureType="circularvacuum"` element parameter and to change the `vacuumMaterial` parameter to the desired material such as air.

A second strategy is shown using a rectangular collimator. By not specifying the `xsize` and `ysize` of the collimator opening, a collimator without an opening is created, which is a block of material.

With the aperture type set to `circularvacuum` the drift is invisible as the vacuum volumes are always invisible. To view these, the executable option `--vis_debug` is used to show all volumes. With this option, invisible volumes are displayed in transparent green.

How to run:

```
bdsim --file=air.gmad --vis_debug
```
A simple example of a BDSIM model that sprays particles everywhere. It consists of four quadrupole magnets that are on but not designed to contain the beam. The beam is lost quickly creating a spray of secondary particles in both the machine components as well as the surrounding tunnel.

How to run:

```
bdsim --file=bl.gmad
```

Running 10 events with the following command in the visualiser session prompt, the following visualisation is produced.

```
/run/beamOn 10
```

The default colour coding is described in *Colour Coding*.

### 14.1.3 Collimation

- Based on `bdsim/examples/collimation/collimation.gmad`

This is an example to show how energy deposition and particle spectra from a small collimation system. There is a more detailed version of this example described in *Collimation* with detailed collimation-specific information.

The model consists of two collimators, followed by a triplet set of quadrupoles, a shielding wall and a third collimator. The collimators are made of successively denser material (carbon, copper and tungsten).

The first step is to prepare some data to analyse. 1000 events runs in around 20s on the developer’s computer. The files can be found in `bdsim/examples/collimation`. It can be run with the following command:
BDSIM Documentation, Release 1.3.2

This will create an output file called `data1.root`. We can run BDSIM with the visualisation to get an idea of what happens in the model. This time, we run without any data output:

```
bdsim --file=collimation.gmad --output=none
```

The following window (with Geant4 setup with the Qt visualiser) should appear.

If we type the following command in the terminal prompt at the bottom of this window, we can visualise 10 events:

```
/run/beamOn 10
```

This looks like:

The particles are colour coded by charge by default (positive: blue, negative: red, and neutral: green).

The following view was created by adding a ‘cut away plane’ that makes part of geometry on one side of a plane invisible. Also, the project was set from orthographic to perspective based using the button on the toolbar (see Control Buttons). The command for the cut away plane is:

```
/vis/viewer/addCutawayPlane 0 0 0 m 1 0 0
```

We can take a look at the data with ROOT with the following command:

```
root -l data1.root
```

The “-l” flag means no logo (slightly quicker), and specifying a file along with the command means this file will come at the top of any browser windows in ROOT.

We start a TBrowser to inspect the data. The intention here is to inspect the data and decide which histograms we might want to prepare from it.
ROOT takes commands in C++, so here we construct an ‘instance’ of the TBrowser class called “tb” (can be any name). The TBrowser brings up a window that allows graphical exploration of the data. This looks like:

The most interesting information is the in the Event tree. Double-click on this to expand it and look at the variables. A full explanation of the output here is described in Event Tree. This browser is most useful to get the exact names to prepare the analysis configuration text file that’s used for analysis.

To produce histograms, we prepare an input text file that describes which histograms we want to prepare. This file is described in detail in Preparing an Analysis Configuration File. Typically we start by copying an example from bdsim/examples/features/analysis/perEntryHistograms/analysisConfig.txt.

Below is an example analysis configuration called analysisConfigSimple.txt that is included in the same example directory.

The data can be analysed with the following command:

```
rebdsim analysisConfigSimple.txt data1.root data1-histos.root
```

This will produce an output file called data1-histos.root that contains the requested histograms as well as a merged copy of any pre-made histograms in the data file (such as energy deposition).

If we start another ROOT session, or click the refresh button (top left, near “Draw Option”, looks like a recycle symbol), the file view will refresh and we can browse the new output file and view the histograms in ROOT. We can also load the histograms in Python using the pybdsim utility package and make some nicer plots:

```
ipython
>>> import pybdsim
>>> d = pybdsim.Data.Load("data1-histos.root")
>>> d.histograms
```

This is described in the manual for pybdsim (see Python Utilities) and the relevant section is http://www.pp.rhul.ac.uk/bdsim/pybdsim/data.html. This is what should be seen:

The following commands can be used to make a few simple plots in Python:

```
14.1. Basic Examples 211
```
We leave it to the user to create their own plots, but the basic data exploration is provided and the user should consult the pybdsim source code (see pybdsim/pybdsim/Plot.py) for how we have made these plots using matplotlib.

The above commands create the following plots.

### 14.1.4 Simple Machine

- **Based on** bdsim/examples/simpleMachine/sm.gmad

A simple example of a BDSIM model. It has a few drift beam pipes with a quadrupole, collimator and sector bend dipole magnet.

**How to run:**

```
bdsim --file=sm.gmad
```

### 14.1.5 Model-Model

- **Based on several models in** /bdsim/examples/model-model
- **Specific example in** /bdsim/examples/model-model/bdsim/halo/h-main-h.gmad

---

14.1. Basic Examples 213
14.1. Basic Examples
The model-model is the fictional storage ring that could be used in a high energy proton collider. The machine is described in the primary BDSIM paper as an example.

The ring is racetrack shaped with two straight sections. One is used to create a small beam size that might be suitable for a collision and the other for large beam sizes to allow collimation. The model is prepared in MADX and custom collimator opening settings and materials are supplied in hand written text files. Example Python scripts are included to show how to merge this information together into a BDSIM model.

Several conversions of the model are made for different purposes. Linear optics, full non-linear optics and for a halo simulation.

- MADX models and input can be found in `bdsim/examples/model-model/madx`.
- BDSIM models can be found in `bdsim/examples/model-model/bdsim`.

Assuming the user is located in the `bdsim/examples/model-model` directory.

```
bdsim --file=bdsim/halo/h-main-h.gmad --outfile=t1 --batch
```

This will generate 20 events of up to 100 turns. The job is designed to simulate a halo beam impacting the first horizontal collimator.

### 14.1.6 Target

- Based on `bdsim/examples/target/target-p-cu.gmad`

A simple example of a BDSIM model. It has a drift followed by a closed collimator which forms a solid block of material, which is copper. This example is used in the worked example in detail - see *Analysis of Thin Target Products*.

How to run:

```
bdsim --file=target-p-cu.gmad --outfile=data
```

When the visualiser starts, the following command can be used:

```
/run/beamOn 5
```

As a lot of secondary particles are produced it is recommended to only run a small number of events.

### 14.2 Worked Examples

Detailed start-to-finish examples with explanation.

#### 14.2.1 Analysis of Thin Target Products

- Based on `bdsim/examples/target/target-p-cu.gmad`

Contents

- *Preparation*
- *Model Description*
- *Generating Data*
- *Questions Answered*
14.2. Worked Examples
Preparation

- BDSIM has been compiled and installed.
- The (DY)LD_LIBRARY_PATH and ROOT_INCLUDE_PATH environmental variables are set as described in Setup.
- ROOT can be imported in Python
- pybdsim has been installed.

Model Description

This example shows a high energy (6.5 TeV) proton beam hitting a 5cm copper target. This produces a large number of secondary particles. This examples illustrates the analysis and understanding of the products in an invisible ‘sampler’ plane after the target.

The model consists of a small 1 cm vacuum pipe followed by a target. The solid box of target material is created by creating a collimator (rcol) with no specified aperture (xsize and ysize) and so a closed collimator is built (a solid box of material). The width of the target is the default horizontalWidth for all elements, which is 0.5 m. See Common Options. The model looks like:

The input files for BDSIM can be found in <bdsim>/examples/target/.

- target-p-cu.gmad - model discussed here
- target-p-cu-sampler-extras.gmad - same model but with extra sampler output turned on.
- target-ion-w.gmad - tungsten target and 1 TeV carbon ion beam.
- target-ion-w-sampler-extras.gmad - similar but with extra sampler output.

The model was prepared by hand as it is quite simple. The input syntax is included below:
First we define the beam line elements we want and then the line (sequence) of them. The `use` command specifies which line to use. `sample, all;` tells BDSIM to put a sampler after each element. A sampler is an invisible 1 nm long plane that is 5 m wide by default that records the passage of any particle in any direction through it (so both forwards and backwards). We then specify a beam, the physics processes to use and some options.

Here we use a Geant4 reference physics list `g4FTFP_BERT` that is a standard high energy physics list suitable for LHC-like applications. See [Geant4 Reference Physics Lists](#).

We choose a default range cut to speed things up a little for the purposes of the example. The range cut is roughly a distance a secondary particle would have to travel and survive in that material. If it wouldn’t travel this distance, it isn’t produced and the energy is recorded at that location. This therefore approximately corresponds to an energy cut per particle species per material. This is much more accurate than a general energy cut, but care should be taken not to choose a length scale longer than the typical element length as you may see discontinuities in energy deposition close to boundaries. The default in Geant4 is 1 mm that roughly corresponds to ~keV energies for most particles.

Specifying a seed allows the user to reproduce the simulation exactly that’s demonstrated here.

The `elossHistoBinWidth` option specifies the binning along the beam line for the default energy deposition (“eloss”) histogram that’s created as BDSIM runs. The default is 1 m as BDSIM is typically used for large accelerators. In this case, our whole model would fit in one bin, which wouldn’t be so interesting. Therefore, we specify a smaller bin width. See [Common Options](#).

### Generating Data

Here, we generate a sample of data to be analysed. Run BDSIM with the input model and specify a number of events (i.e. number of primary beam particles) to simulate. We also specify what we want the output data file to be called.

```
bdsim --file=target-p-cu.gmad --outfile=data1 --batch --ngenerate=200
```

The `--batch` option means that no visualiser is used and the events are simulated and BDSIM finishes.

This creates an output ROOT file called `data1.root`. On the developer’s laptop, this took approximately 8 seconds to run in total.

The model can also be run interactively, but given the large number of secondary particles it is advisable to run a low number of events e.g. 1 to 10. In this case you would run BDSIM as follows:

```
bdsim --file=target-p-cu.gmad --outfile=data1
```

Once the visualiser starts, enter the following command in the session terminal prompt at the bottom. (see [Interactively](#)).
A visualisation of 10 events can be seen at the beginning of this example.

Some of the terminal print out (which is kept to be as minimal as possible in BDSIM) is worth noting. The beam print out shows the kinematics of the beam particle chosen.

---

Design particle properties:
Particle: "proton"
Mass: 0.938272 GeV
Charge: 1 e
Total Energy: 6500 GeV
Kinetic Energy: 6499.06 GeV
Momentum: 6500 GeV
Gamma: 6927.63
Beta: 1
Rigidity (Brho): 21681.7 T*m

There is then a large print out of physics processes from Geant4. However, at the end a “Table of Registered Couples” is of interest. This indicates the range cuts for basic particle types and the corresponding energies calculated. These energies do not vary linearly with range. We can use this as an idea of the approximate energy scale.

---

Table of registered couples

Index : 0 used in the geometry : Yes
Material : G4_AIR
Range cuts : gamma 3 cm e- 3 cm e+ 3 cm proton 3 cm
Energy thresholds : gamma 990 eV e- 43.5254 keV e+ 43.2339 keV proton
Region(s) which use this couple :
  DefaultRegionForTheWorld

Index : 1 used in the geometry : Yes
Material : G4_Galactic
Range cuts : gamma 3 cm e- 3 cm e+ 3 cm proton 3 cm
Energy thresholds : gamma 990 eV e- 990 eV e+ 990 eV proton 3 MeV
Region(s) which use this couple :
  DefaultRegionForTheWorld

Index : 2 used in the geometry : Yes
Material : vacuum
Range cuts : gamma 3 cm e- 3 cm e+ 3 cm proton 3 cm
Energy thresholds : gamma 990 eV e- 990 eV e+ 990 eV proton 3 MeV
Region(s) which use this couple :
  DefaultRegionForTheWorld

Index : 3 used in the geometry : Yes
Material : stainlesssteel
Range cuts : gamma 3 cm e- 3 cm e+ 3 cm proton 3 cm
Energy thresholds : gamma 109.289 keV e- 46.8866 MeV e+ 43.5457 MeV proton
Region(s) which use this couple :
  DefaultRegionForTheWorld

Index : 4 used in the geometry : Yes
Material : G4_Cu
Range cuts : gamma 3 cm e- 3 cm e+ 3 cm proton 3 cm
Energy thresholds : gamma 128.416 keV e- 52.2089 MeV e+ 48.4888 MeV proton
Region(s) which use this couple :
  DefaultRegionForTheWorld

(continues on next page)
Questions Answered

- **Question 1** What fraction of the beam makes it through the target?
- **Question 2** What spectrum of particles comes out after the target?
- **Question 3** How much energy is reflected backwards from the target?

**Question 1**

What fraction of the beam makes it through the target?

Here we want to know the fraction of primary particles after the target. To do this we look at the data recorded in the sampler after the target. The target was called “c1” in the input so there will be a sampler structure in the Event tree of the output called “c1”.

To get this answer we can make a histogram using rebdsim. This may seem an unintuitive approach but it includes all the correct event averaging and uncertainty calculations.

To analyse data and make histograms using rebdsim we use an analysis configuration text file. We start, as always, by copying an example from BDSIM that can be found in:

```
bdsim/examples/features/analysis/perEntryHistograms/analysisConfig.txt
```

We histogram any value of any particle recorded in the sampler in a 1 bin histogram with the filter (“selection”) that only primary particles are filled. The default histogramming is per event, i.e. normalised to the number of events. Below are two possible ways to achieve the same answer.

```
Histogram1D Event. Q1PrimaryFraction {2} {-0.5:1.5} c1.parentID==0 c1.

Histogram1D Event. Q1PrimaryFraction2 {1} {-2:2} c1.x c1.
```

This file for this example is provided in `bdsim/examples/target/analysisConfig.txt`. We run rebdsim with the following command:

```
rebdsim analysisConfig.txt data1.root data1-analysis.root
```

This produces an output ROOT file called `data1-analysis.root` that contains the desired histograms.

The first histograms a Boolean of whether the parentID (the track ID of the particle that created that one) is 0 or not. Only primary particles have `parentID==0` as they have no parent. This will happen for every particle recorded in the sampler including secondaries. We therefore add a “selection” (a filter) to only bin the particles where their parentID is 0. The Boolean will become a number when binned so it can either be a 0 or a 1. We choose histogram bins from 0.5 to 1.5 with two bins so that the centres are inside the bins.

This will produce a histogram with two bins centred on 0 and 1. The value of the second bin centred on 1 is the answer.

A second way is to histogram any one coordinate and apply the same filter of primaries only. In the above code we declare a 1 bin 1D histogram from -2 to +2 m to cover all values of x. The histogram contains one bin with the mean number of primaries per event that go through the sampler.

We can extract this number easily with pybdsim. In iPython (or Python):
>>> import pybdsim
>>> d = pybdsim.Data.Load("data1-analysis.root")
>>> d.histogramspy
{'Event/MergedHistograms/ElossHisto': <pybdsim.Data.TH1 at 0x129b37ad0>,
 'Event/MergedHistograms/ElossPEHisto': <pybdsim.Data.TH1 at 0x129b37a10>,
 'Event/MergedHistograms/PhitsHisto': <pybdsim.Data.TH1 at 0x129b37a50>,
 'Event/MergedHistograms/PhitsPEHisto': <pybdsim.Data.TH1 at 0x129b289d0>,
 'Event/MergedHistograms/PlossHisto': <pybdsim.Data.TH1 at 0x129b257f0>,
 'Event/MergedHistograms/PlossPEHisto': <pybdsim.Data.TH1 at 0x129b25780>,
 'Event/PerEntryHistograms/Q1PrimaryFraction': <pybdsim.Data.TH1 at 0x129b25d90>,
 'Event/PerEntryHistograms/Q2PrimaryFraction2': <pybdsim.Data.TH1 at 0x129b20410>}

>>> q1ha = d.histogramspy['Event/PerEntryHistograms/Q1PrimaryFraction']
>>> type(q1ha)
<pybdsim.Data.TH1 at 0x129b25790>

>>> q1ha.contents
array([0. , 0.755])
>>> q1ha.errors
array([0. , 0.03048807])

So here we see two ways to find the answer of $0.755 \pm 0.031$ of the proton beam goes through remaining an intact proton. Note, this doesn’t say whether it interacted or not, but just whether the primary made it through intact. If the proton had disintegrated then it would not be a primary anymore.

**Question 2**

- What spectrum of particles comes out after the target?

The ideal plot here would be histograms of different particle species for different energies, i.e. a line for each particle type. To do this, we again histogram the particles recorded in the sampler after the target. We histogram the energy for each particle species. The following analysis is used.
The particle IDs are the Particle Data Group IDs that can be found online at http://pdg.lbl.gov/2018/reviews/rpp2018-rev-monte-carlo-numbering.pdf.

After the target there is air as this is the default worldMaterial (see Common Options). Potentially, a (likely secondary) particle could bounce back off of the air and go through the sampler before hitting the target again. We could change to the world material option to “vacuum” to avoid this or add the filter of the z component of the momentum is positive - i.e. forwards travelling. This is why we have \( c1.zp > 0 \) in all of the selections above. We can quickly check if there are any backwards going particles at all by inspecting the data in a ROOT TBrowser (see Basic Data Inspection). Below is a screenshot of ROOT.

![ROOT Screenshot](image)

C1.zp.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entries</td>
<td>98,279</td>
</tr>
<tr>
<td>Mean</td>
<td>0.8875</td>
</tr>
<tr>
<td>Std Dev</td>
<td>0.2229</td>
</tr>
</tbody>
</table>

Fig. 1: Component of unit momentum recorded in sampler after target. Positive equates to forward going and negative to backwards going. Total number of particles across all particles in all events in the data.

There are indeed some, although only a small fraction. This is for all particles across all events. We therefore use the filter to get the correct answer.

This file for this example is provided in bdsim/examples/target/analysisConfig.txt. We run rebdsim with the following command:

```
rebsdsm analysisConfig.txt data1.root data1-analysis.root
```

We can then load and plot the data in iPython (or Python) using pybdsim. This requires us to make our own plot. We can look at pybdsim/pybdsim/Plot.py: Histogram1D() for inspiration and then make our own plot. Below is the Python code used to generate the plot. This is included with this example as plotSpectra.py.

```python
import matplotlib.pyplot as plt
import pybdsim
from OrderedDict import

def Spectra(filename, outputfilename='spectra', log=False):
    d = pybdsim.Data.Load(filename)
```

(continues on next page)
keys = OrderedDict(["All", "All"],
            ("ProtonsPrimary", "p (primary)"),
            ("ProtonsSecondary", "p (secondary)"),
            ("Neutrons", "n"),
            ("PiPlusMinus", "$\pi^\pm\$"),
            ("PiZero", "$\pi^0\$"),
            ("Electrons", "e$^-$"),
            ("Positrons", "e$^+$"),
            ("Gammas", "$\gamma\$"),
            ("Muons", "$\mu^\pm\$"))

_extra = "Log" if log else ""
for k, name in keys.iteritems():
    ho = d.histograms1dpy["Event/PerEntryHistograms/Q2"+extra+k]
    h = pybdsim.Data.PadHistogram1D(ho)
    _plt.errorbar(h.xcentres, h.contents, yerr=h.errors, drawstyle="steps-mid",
               label=name)

binWidth = d.histogramspys["Event/PerEntryHistograms/Q2"+extra="All"].xwidths[0]

if log:
    _plt.xscale("log")
    _plt.ylabel("Number / Proton / d$log(E)$ GeV")
    _plt.xlim(9,6700)
    _plt.yscale('log', nonposy='clip')
else:
    _plt.ylabel("Number / Proton / " + str(round(binWidth,0)) + " GeV")
    _plt.xlim(-50,6600)
    _plt.yscale('log', nonposy='clip')
    _plt.legend(fontsize="small")

if not outputfilename.endswith(".pdf"):
    outputfilename += ".pdf"
_plt.savefig(outputfilename)

We can use this as follows:

```python
>>> import plotSpectra

>>> plotSpectra.Spectra('data1-analysis.root')
```

This produces the following plot.

This doesn’t look so informative at first glance. We can generate more statistics but we can also make a logarithmically binned plot. We add more lines to the analysisConfig.txt for rebdsim. See Analysis Configuration File for more details. Here are the lines we add:

```
Histogram1DLog Event. Q2LogAll {100} {1:3.82} c1.energy c1.zp>0
Histogram1DLog Event. Q2LogProtonsPrimary {100} {1:3.82} c1.energy c1.zp>0&&
  c1.partID==2212&&c1.parentID==0
Histogram1DLog Event. Q2LogProtonsSecondary {100} {1:3.82} c1.energy c1.zp>0&&
  c1.partID==2212&&c1.parentID>0
Histogram1DLog Event. Q2LogNeutrons {100} {1:3.82} c1.energy c1.zp>0&&
  abs(c1.partID)==211
Histogram1DLog Event. Q2LogPiPlusMinus {100} {1:3.82} c1.energy c1.zp>0&&
  abs(c1.partID)==211
```

(continues on next page)
Fig. 2: Spectra of particles for 200 events through a 5cm target of copper.
We use the above plotting script in Python to make a logarithmically binned plot:

```python
>>> import plotSpectra
>>> plotSpectra.Spectra('data1-analysis.root', log=True)
```

This produces the following figure.

![Spectra of particles for 200 events through a 5cm target of copper.](image)

Fig. 3: Spectra of particles for 200 events through a 5cm target of copper.

This is more informative but still we are lacking statistics. Given the first generation of data took less than 10 seconds, we can rerun 3000, reanalyse the new data using rebdsim and make new plots. Below are such plots for 3000 events. On the developer’s computer took 90 seconds to run.

Note: The more histograms we add and the more filters we add, the slower the analysis will be. The analysis is actually very efficient for what it does. If the analysis becomes too long running, consider generating separate data files and analysing them separately, then combining the resultant histograms. For further details, see Scaling Up - Parallelising Analysis and Speed & Efficiency.
Fig. 4: Spectra of particles for 3000 events through a 5cm target of copper.
<table>
<thead>
<tr>
<th>Total Particle Energy (GeV)</th>
<th>Number / Proton / d log(E) GeV</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td></td>
</tr>
<tr>
<td>p (primary)</td>
<td></td>
</tr>
<tr>
<td>p (secondary)</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
</tr>
<tr>
<td>±</td>
<td></td>
</tr>
<tr>
<td>e</td>
<td></td>
</tr>
<tr>
<td>e+</td>
<td></td>
</tr>
<tr>
<td>±</td>
<td></td>
</tr>
</tbody>
</table>

Fig. 5: Spectra of particles for 3000 events through a 5cm target of copper.
**Question 3**

- How much energy is reflected backwards from the target?

Our sampler attached to the target is actually after the target. Samplers are only ever after an element, so we prepared the model with a drift on the front. If we really didn’t want to see the effect of scattering or energy deposition in the beam pipe, we could set the `apertureType="circularvacuum"` and `vacuumMaterial="air"` to set the material to air so it’s consistent with the surroundings. However, here we don’t mind. The `sample, all;` command means the drift will have a sampler at the end of it, so before the target. The sampler name will be the name of the drift, “d1”.

We therefore bin any variable on this sampler with the selection of backwards going and weight by the energy. The histogram definition is as follows:

```
Histogram1D Event. Q3BackwardsEnergy {1} {-2:2} d1.x d1.energy*d1.zp<0
```

The weighting by energy means that each entry will be filled with the energy instead of the count 1. The Boolean filter becomes a number in ROOT and is either 1 or 0.

This file for this example is provided in `bdsim/examples/target/analysisConfig.txt`. We run `rebdsim` with the following command:

```
rebdsim analysisConfig.txt data1.root data1-analysis.root
```

We inspect the histogram in `pybdsim` to get the value of the one bin. In iPython:

```python
>>> import pybdsim
d = pybdsim.Data.Load("data1-analysis.root")
h = d.histogramspy['Event/PerEntryHistograms/Q3BackwardsEnergy']
>>> h.contents
array([32.364])
>>> h.errors
array([1.44402409])
```

Therefore, we see that there is on average $32.364 \pm 1.444$ GeV reflected backwards per event, i.e. per 6.5 TeV proton hitting the target. This value was based on a sample of 3000 events.

### 14.2.2 Collimation

- Based on `bdsim/examples/collimation`

**Contents**

- Preparation
- Model Description
- Model Preparation
- Optics
- Losses
- Questions Answered

**Preparation**

- BDSIM has been compiled and installed.
- The `(DY)LD_LIBRARY_PATH` and `ROOT_INCLUDE_PATH` environmental variables are set as described in `Setup`.
• ROOT can be imported in Python
• *pybdsim* has been installed.

Model Description

This is an example to show how to turn on extra collimation output and how to analyse the data. This example shows information most relevant to a collimation system. A basic version of this example exists that shows general data exploration and analysis (see *Collimation*). This example focusses on more detailed collimation specific information.

The model consists of two collimators, followed by a triplet set of quadrupoles, a shielding wall and a third collimator. The collimators are made of successively denser material (carbon, copper and tungsten). This looks like:

Model Preparation

The model is fictional and designed to show the relevant features of BDSIM. As the beam line was relatively short, the model was written by hand without any automatic conversion.

The files can be found in *bdsim/examples/collimation*:

- collimation.gmad - model for beam losses and collimation
- collimationOptics.gmad - model for Gaussian beam for optics

Optics

To understand how this machine transports particles, it is useful to simulate a Gaussian beam that would nominally represent some ‘core’ beam that would be expected in the machine. Here we expect the losses to be very low. A specific *bdsim* input file is included that chooses a Gaussian distribution according to chosen Twiss parameters.
Secondary particles are stopped and the distribution is recorded after every beam line element with the `sample, all` command. Running this optics model takes around 10s to run 5000 particles on the developer’s computer:

```
bdsim --file=collimationOptics.qmad --outfile=o1 --batch --ngenerate=5000
```

**Note:** It is recommend to run at least 1000 particles for optical function evaluation and 10000 to 50000 if high accuracy is desired or a large energy spread is defined.

This produces an output file called “o1.root”. We can then calculate the optical functions and sizes of the beam after each element using the included rebdsimOptics tool:

```
rebdsimOptics o1.root o1-optics.root
```

This creates another file called “o1-optics.root” that contains only the optical function information. The Python utility `pybdsim` can be used to visualise the data:

```
ipython
>>> import pybdsim
>>> pybdsim.Plot.BDSIMOptics("o1-optics.root")
```

This produces a series of graphs showing, for example, the mean, sigma, divergence and dispersion of the beam. The sigma and dispersion are shown below.

**Losses**

Of course, more interesting than the optical functions is the possibility of beam losses. To illustrate this, we choose a beam distribution that is a circular ring of particles close to the edge of the collimator. Most will hit the first collimator but around 1/3 will make it through at the edges as the first collimator is square. We should generate some events to investigate the beam losses. The below command generates 2000 events (2000 primary particles), which takes approximately 30s on the developer’s computer:
bdsim --file=collimation.gmad --outfile=data1 --batch --ngenerate=2000

This produces an output file called “data1.root”, which is approximately 20Mb. Firstly, we might like to quickly see if there were any losses at all and if there was any energy deposition. This can be done by browsing the output data file as described in Basic Data Inspection, however, we’d like to look at the average energy loss and impacts quickly. Histograms of the primary particle impact and loss points as well as energy deposition are included by default per event in the output. A tool for convenience (rebdsimHistoMerge) allows averaging of these quickly as opposed to running rebdsim with an analysis configuration text file. This is run as follows:

rebdsimHistoMerge data1.root data1-histos.root

This produces an output file called “data1-histos.root” that contains the merged histograms. This too can also be viewed with a TBrowser in ROOT as described in Basic Data Inspection, however, here we will make more standardised plots using pybdsim in Python:

```python
>>> import pybdsim
>>> pybdsim.Plot.LossAndEnergyDeposition("data1-histos.root", hitslegendloc='upper center')
>>> pybdsim.Plot.EnergyDeposition("data1-histos.root")
```

In the case of the first command, the legend overlaps with an expected data point, so we move it - this is optional (limitation of plotting library). These produce the following plots of primary hits, losses and energy deposition and secondly just energy deposition.

As expected we see a large fraction of particle impact the first collimator and we see some energy deposition throughout. Now, we can perform a more advanced analysis to learn about these impacts and losses in the collimation system.

**Questions Answered**

- **Question 1** Where are particles absorbed that impact the first collimator?
- **Question 2** Where do particles impact that don’t impact the first collimator?
- **Question 3** What secondaries make it through the shielding wall created from impacts on the first collimator?

**Question 1**

- Where are particles absorbed that impact the first collimator?
We want to histogram the absorption point of the primary particle in each event but only for the events where the primary impact was in the first collimator. We always record the primary first hit point and the loss point, but here we make use of the collimator specific information. The first collimator is called “c1” and the collimator hits are stored under the “COLL_c1_0” branch of the Event tree.

Note: The name of the collimator is prefixed with “COLL_” to distinguish it from a sampler which would have the name “c1”. The suffix “_0” is because it’s the 0th placement of that collimator in the beam line.

In this collimator structure in the output there is a variable “primaryInteracted”. This is a Boolean which is true if the primary particle interacted with the material of the collimator on that event. We use this as a ‘selection’ in the histogram. We prepare an analysis configuration text file for rebdsim (see Preparing an Analysis Configuration File). We can start from an example in BDSIM and edit that one. An example can be found in bdsim/examples/features/analysis/perEntryHistograms/analysisConfig.txt.

The variables in the histogram specification must be exactly as you see in the output file so it’s useful to use a TBrowser in ROOT to browse the file while preparing the analysis configuration file. The following is the desired histogram specification:

```
# Object treeName Histogram Name #Bins Binning Variable
  Selection Histogram1D Event. C1ImpactLossLocation {96} {0:12} PrimaryLastHit.S COLL_c1_0.primaryInteracted
```

Note: Take note of the “.” in the variable names.

An example analysis configuration file is included in bdsim/examples/collimation/analysisConfig.txt that contains the histograms for this and subsequent questions.

This can be used with the following command:

```
rebdsim analysisConfig.txt data1.root data1-analysis.root
```

This produces a ROOT file called “data1-analysis.root” with the desired histograms. The histograms are by default made ‘per event’ (i.e. the histogram is made separately for each event, then these histograms are averaged), and the histogram “C1ImpactLossLocation” will be in the Event/PerEntryHistograms/ directory in the file. This histogram can be plotted with pybdsim:
>>> from matplotlib.pyplot import *
>>> import pybdsim
>>> d = pybdsim.Data.Load("data1-analysis.root")
>>> d.histogramspy
{'Event/MergedHistograms/CollElossPE': <pybdsim.Data.TH1 at 0x119f90ad0>,
 'Event/MergedHistograms/CollPInteractedPE': <pybdsim.Data.TH1 at 0x119f909d0>,
 'Event/MergedHistograms/CollPhitsPE': <pybdsim.Data.TH1 at 0x119f90b10>,
 'Event/MergedHistograms/CollPlossPE': <pybdsim.Data.TH1 at 0x119f90b90>,
 'Event/MergedHistograms/ElossHisto': <pybdsim.Data.TH1 at 0x119f90c10>,
 'Event/MergedHistograms/ElossPEHisto': <pybdsim.Data.TH1 at 0x119f90a10>,
 'Event/MergedHistograms/PhitsHisto': <pybdsim.Data.TH1 at 0x119f90a50>,
 'Event/MergedHistograms/PhitsPEHisto': <pybdsim.Data.TH1 at 0x119f82650>,
 'Event/MergedHistograms/PlossHisto': <pybdsim.Data.TH1 at 0x119f90910>,
 'Event/MergedHistograms/PlossPEHisto': <pybdsim.Data.TH1 at 0x119f90790>,
 'Event/PerEntryHistograms/AfterShielding': <pybdsim.Data.TH2 at 0x119f90c90>,
 'Event/PerEntryHistograms/C1ImpactLossLocation': <pybdsim.Data.TH1 at 0x119f90a00>,
 'Event/PerEntryHistograms/NoC1ImpactLossLocation': <pybdsim.Data.TH1 at 0x119f909d0>}
>>> pybdsim.Plot.Histogram1D(d.histogramspy['Event/PerEntryHistograms/
\rightarrow C1ImpactLossLocation'])
>>> yscale('log', nonposy='clip')
>>> xlabel('S (m)')
>>> ylabel('Fraction of Primary Particles')
>>> pybdsim.Plot.AddMachineLatticeFromSurveyToFigure(gcf(), d.model)

Note: The y axis here is fraction of total events, so the integral of this histogram is not 1 as not all particle impact the first collimator. This is however, the accurate fraction of the events simulated, so this is what is required to correctly scale to a correct rate of expected events for this beam distribution.

- The variable d.model is the beam line model included with each output file and automatically loaded with pybdsim.
- the "nonposy='clip'" argument to pyplot.yscale avoids gaps in the line of the histogram when plotting.
- The command d.histogramspy is used to print out the numpy-converted histograms loaded from the file by pybdsim so that the name can be copied and pasted into the next command.

This shows that the particles that interact with the first collimator are lost (in order)

1) just after the c1 collimator in the beam pipe (*))
2) before the c1 collimator in the beam pipe (from back-scattering)
3) c2 collimator
4) c3 collimator
5) throughout the machine

**Note:** (*) We should remember the binning in this histogram does not break at the element boundaries so particles stopping both in the collimator and just afterwards in the collimator could be in the same bin. We can always look at the ‘per element’ histogram from the merged histograms.

When the machine diagram is added to the figure, a searching feature is activated. Right-clicking anywhere on the plot will print out in the Python terminal the nearest beam line element to that point. Here, we can right-click on any of the peaks to get the names of these beam line elements.

**Question 2**

- Where do particles impact that don’t impact the first collimator?

Similarly, we want to histogram the impact location, so *PrimaryFirstHit.S*, but for only the events where the primary particle didn’t impact the first collimator. Again, we use a selection in the histogram specification:

```
# Object      treeName Histogram Name #Bins Binning Variable
Selection     Histogram1D Event. NoC1ImpactLossLocation {96} {0:12} PrimaryFirstHit.S COLL_
              → c1_0.primaryInteracted==0
```

This is included in the example analysis configuration `bdsim/examples/collimation/analysisConfig.txt` that contains the histograms for this and the other questions.

This can be used with the following command:

```
rebdsim analysisConfig.txt data1.root data1-analysis.root
```

Loading and plotting with `pybdsim`:

```
ipython
>>> from matplotlib.pyplot import *
>>> import pybdsim
>>> d = pybdsim.Data.Load("data1-analysis.root")
>>> d.histogramspy
{'Event/MergedHistograms/CollElossPE': <pybdsim.Data.TH1 at 0x119f90ad0>,
 'Event/MergedHistograms/CollPInteractedPE': <pybdsim.Data.TH1 at 0x119f909d0>,
 'Event/MergedHistograms/CollPhitsPE': <pybdsim.Data.TH1 at 0x119f90b10>,
 'Event/MergedHistograms/CollPlossPE': <pybdsim.Data.TH1 at 0x119f90b90>,
 'Event/MergedHistograms/ElossHisto': <pybdsim.Data.TH1 at 0x119f90c10>,
 'Event/MergedHistograms/ElossPEHisto': <pybdsim.Data.TH1 at 0x119f90a10>,
 'Event/MergedHistograms/PhitsHisto': <pybdsim.Data.TH1 at 0x119f90a50>,
 'Event/MergedHistograms/PhitsPEHisto': <pybdsim.Data.TH1 at 0x119f82650>,
 'Event/MergedHistograms/PlossHisto': <pybdsim.Data.TH1 at 0x119f90910>,
 'Event/MergedHistograms/PlossPEHisto': <pybdsim.Data.TH1 at 0x119f90790>,
 'Event/PerEntryHistograms/AfterShielding': <pybdsim.Data.TH2 at 0x119f90c90>,
 'Event/PerEntryHistograms/ClImpactLossLocation': <pybdsim.Data.TH1 at 0x119f90890>,
 'Event/PerEntryHistograms/NoC1ImpactLossLocation': <pybdsim.Data.TH1 at _
  0x119f7d710>}
>>> pybdsim.Plot.Histogram1D(d.histogramspy['Event/PerEntryHistograms/
  → NoC1ImpactLossLocation'])
>>> yscale('log', nonposy='clip')
>>> xlabel('S (m)')
```
Here we can see that particles that don’t impact the first collimator impact the second one and the third one. Some make it to the end of the beam line where they ‘hit’ the air of the world volume. Inspecting the raw data for Event.PrimaryFirstHit.S, we see some events with the value -1m. This is a value we put in the output when the impact was outside the curvilinear coordinate system, e.g., in the world volume away from the beam line. We can infer that the particles made it through the air of the world volume before reaching the boundary of the model.

Question 3

• What secondaries make it through the shielding wall created from impacts on the first collimator?

We could plot many quantities of the secondary particles coming through the shielding wall, but, here we suggest the 2D flux. We therefore have a sampler attached to the “s1” beam line element (the shielding wall) that records in the distribution of all particles after it. We plot the 2D distribution of these particles and then filter them. The filter includes:

• must be a secondary particle - parentID > 0
• primary impact must be in c1 collimator - COLL_c1_0.primaryInteracted is true

This is the line added to the example analysis configuration file:

```python
# Object treeName Histogram Name #Bins Binning Variable
Selection Histogram2D Event. AfterShielding {50,50} {-2.5:2.5,-2.5:2.5} s1.y:s1.x COLL_c1_0.primaryInteracted&&s1.parentID>0
```

**Note:** Our analysis configuration file is a relatively thin interface to TTree::Draw in ROOT and so we see the inconsistency in ROOT for the order of the variables to be histogrammed. All of our specifications are x, then y, then z if further dimensions are required. However, with ROOT, the variable to be histogrammed is 1D: x, 2D y vs x, 3D x vs y vs z. The 2D variables are y:x here. The number of bins and ranges are in x, y, z order always.

This histogram can be plotted with `pybdsim`:

```python
>>> from matplotlib.pyplot import *
>>> pybdsim.Plot.AddMachineLatticeFromSurveyToFigure(gcf(), d.model)
```
```python
>>> import pybdsim
>>> d = pybdsim.Data.Load("data1-analysis.root")
>>> d.histogramspy
{'Event/MergedHistograms/CollElossPE': <pybdsim.Data.TH1 at 0x119f90ad0>,
 'Event/MergedHistograms/CollPInteractedPE': <pybdsim.Data.TH1 at 0x119f909d0>,
 'Event/MergedHistograms/CollPhitsPE': <pybdsim.Data.TH1 at 0x119f90b10>,
 'Event/MergedHistograms/CollPlossPE': <pybdsim.Data.TH1 at 0x119f90b90>,
 'Event/MergedHistograms/ElossHisto': <pybdsim.Data.TH1 at 0x119f90c10>,
 'Event/MergedHistograms/ElossPHisto': <pybdsim.Data.TH1 at 0x119f90a10>,
 'Event/MergedHistograms/PhitsHisto': <pybdsim.Data.TH1 at 0x119f82a50>,
 'Event/MergedHistograms/PhitsPHisto': <pybdsim.Data.TH1 at 0x119f90910>,
 'Event/MergedHistograms/PlossHisto': <pybdsim.Data.TH1 at 0x119f909d0>,
 'Event/MergedHistograms/PlossPHisto': <pybdsim.Data.TH1 at 0x119f90790>,
 'Event/PerEntryHistograms/AfterShielding': <pybdsim.Data.TH2 at 0x119f90c90>,
 'Event/PerEntryHistograms/C1ImpactLossLocation': <pybdsim.Data.TH1 at 0x119f90890>,
 'Event/PerEntryHistograms/NoC1ImpactLossLocation': <pybdsim.Data.TH1 at _
 ...0x119f7d710>}
>>> pybdsim.Plot.Histogram2D(d.histogramspy['Event/PerEntryHistograms/
 ...AfterShielding'], logNorm=True)
>>> xlabel('X (m)')
>>> ylabel('Y (m)')
>>> tight_layout()

The value plotted is the number of particles per square bin size on average per particle simulated including all the filters.

**Note:** The samplers by default are 5m wide, but may be shrunk to avoid geometrical overlaps in the case of a model with very large angle bends. Of course, this does not apply to this model.
14.2.3 Accelerator Test Facility 2 - KEK, Japan

- Based on several models in bdsim/examples/atf2

Contents

- Preparation
- Model Description
- Input Preparation
- Input Inspection
- Conversion
- Optical Validation
- Adding to Model
- Changing Beam Distribution
- Halo Simulation
- Analysis
- Spectra at Plane

Preparation

- BDSIM has been compiled and installed.
- The (DY)LD_LIBRARY_PATH and ROOT_INCLUDE_PATH environmental variables are set as described in Setup.
- ROOT can be imported in Python
- pymadx and pybdsim have been installed.

Model Description

This is the 1.3GeV energy scaled test facility for the ILC final focus system. The real machine consists of an approximately 70m normal conducting linac, transfer line, racetrack damping ring and finally an extraction line. This model represents only the ~100m single-pass extraction line.

Input Preparation

A MAD-X job was used to prepare a Twiss table from MAD-X in TFS format. This is included in bdsim/examples/atf2/atf2-nominal-twiss-v5.2.tfs.tar.gz. The file was prepared by adding the following to the end of the MAD-X job:

```plaintext
select,flag-twiss, clear;
twiss,sequence=ATF2, file=atf2-nominal-twiss-v5.2.tfs;
```

As we don’t specify any columns, all columns are written out (~250). Whilst this may seem overkill, it ensures we don’t miss any of the required columns for conversion.

For convenience we compress this to save space. pybdsim and pymadx both work with a compressed TFS file without the need to decompress it.

```plaintext
tar -czf atf2-nominal-twiss-v5.2.tfs.tar.gz atf2-nominal-twiss-v5.2.tfs
```
Input Inspection

We can inspect the model as provided by MAD-X with pymadx in Python.

```python
>>> import pymadx
>>> a = pymadx.Data.Tfs("atf2-nominal-twiss-v5.2.tfs.tar.gz")
>>> a.ReportPopulations()
Filename > atf2-nominal-twiss-v5.2.tfs.tar.gz
Total number of items > 1032
Type........... Population
MULTIPOLE...... 516
DRIFT.......... 201
QUADRUPOLE..... 102
MARKER......... 78
MONITOR........ 64
SBEND.......... 24
SEXTUPOLE...... 18
HKICKER........ 15
VKICKER........ 14
```

We can also inspect the strengths of all the sextupoles for example

```python
>>> sextupoles = a.GetElementsOfType('SEXTUPOLE')
>>> len(sextupoles)
18
>>> type(sextupoles)
pymadx.Data.Tfs
>>> import matplotlib.pyplot as plt
>>> plt.hist(sextupoles.GetColumn("K2L")/sextupoles.GetColumn("L"))
```

This produces the following figure:
Conversion

The model can be converted to BDSIM’s GMAD syntax with the converter provided in \textit{pybdsim}.

\begin{verbatim}
> python
>>> import pybdsim
>>> a,b,c = pybdsim.Convert.MadxTfs2Gmad('atf2-nominal-twiss-v5.2.tfs.tar.gz',
˓→'atf2bdsim')
\end{verbatim}

The converter will automatically generate a Twiss beam distribution based on the first element of the lattice. If the first element is \textbf{not a marker} the beam will be wrong as the optical functions from MAD-X are typically at the end of each element (they can be set to the middle too, but not to the beginning). The user should check the distribution.

This converts the model as is. We can also prepare a linear only version of the model:

\begin{verbatim}
>>> a,b,c = pybdsim.Convert.MadxTfs2Gmad('atf2-nominal-twiss-v5.2.tfs.tar.gz',
˓→'atf2bdsimlinear', linear=True)
\end{verbatim}

Several gmad files are created:

\begin{verbatim}
> ls
atf2bdsimlinear.gmad
atf2bdsimlinear_beam.gmad
atf2bdsimlinear_components.gmad
atf2bdsimlinear_options.gmad
atf2bdsimlinear_sequence.gmad
\end{verbatim}

The components are defined in the file with \textit{components} suffix, the sequence, options and beam similarly. These GMAD files are included in the \textit{main} file \textit{atf2bdsimlinear.gmad}.

- No options are required by default to get a working model.
- Only tracking is provided by default - no physics processes are registered.
- By default, a sampler is attached to all items with the \textit{sample, all;} command in the main file.

Optical Validation

First we validate that the Twiss beam definition in the converted model is correct for our machine. This is the case as the first item in the lattice is a marker in the MAD-X job. The emittance and energy spread were also correctly specified in the MAD-X job and have therefore been converted correctly.

We run 1000 particles to validate the optics:

\begin{verbatim}
bdsim --file=atf2bdsimlinear.gmad --outfile=o1 --batch --ngenerate=1000
\end{verbatim}

This output file can then be analysed to calculate the beam size and optical functions:

\begin{verbatim}
rebsdsoptics o1.root optics.root
\end{verbatim}

We can now compare the optical functions using \textit{pybdsim}.

\begin{verbatim}
> python
>>> import pybdsim
>>> pybdsim.Compare.MadxVsBDSIM('atf2-nominal-twiss-v5.2-sige0.tfs', 'optics.root')
\end{verbatim}

This produces a series of plots comparing beam size and optical functions such as the following:

Note, with nonlinear optics (i.e. including sextupoles and higher) the emittance between each plane (horizontal, vertical) will be mixed and the calculated optical functions are not representative. A model converted with the ‘linear’ flag will however be valid.
Fig. 6: Beam size.

Fig. 7: Angular beam size.
Fig. 8: Beam centroid.

Fig. 9: Twiss $\beta$ function. Only the first part is shown due to the large variation.
This step verifies that the model has been prepared correctly and matches the model in the original program, MAD-X.

**Note:** The energy spread used in BDSIM beam definition must be the same as that in the Twiss output from MAD-X for the comparison to be valid.

**Note:** The errors are the statistical uncertainty associated with the calculation. It is possible depending on the number of particles for the model to agree but the original lie outside the error bars.

### Adding to Model

At this point, we can add more detail to the model. Here we place a GDML file containing the tunnel geometry around the beam line. This geometry was prepared externally and designed to have a hollow outermost ‘world’ volume so that it does not overlap with the beam line - both exist at the same level in the hierarchy. If the tunnel container were not hollow, the beam line would overlap with the tunnel geometry and tracking would be invalid.

In the main GMAD file, we define a placement of the geometry with the appropriate transform.

```
tun : placement, geometryFile="gdml:atf2_tunnel.gdml", x=-4.5*m, z=49*m;
```

The example GDML file (“atf2_tunnel.gdml”) is provided in bdsim/examples/atf2/. An example file including this geometry with the placement above is provided in bdsim/examples/atf2/nlsige/atf2-with-tunnel.gmad.

Care must be taken not to place geometry that overlaps with the beam line otherwise the tracking will be wrong. Using the option, checkOverlaps=1; option is recommended when placing the geometry for the first time. Once validated, this can be turned off for speed.

Geometry can be added for magnet yokes, placed alongside the beam line and placed in the beam line. See Externally Provided Geometry for more details.

Custom field maps could also be added to the yokes of particular magnets. A general field map for quadrupoles could also be added for example and auto-scaling used to scale the field map for each quadrupole it’s attached to. See Fields for more details.
One simple change is to specify a default aperture for all components.

```
option, aper1=1.5*cm,
beampipeThickness=1*mm;
```

The typical beam pipe width of the ATF2 is 30mm and the thickness ~1.5mm.

### Changing Beam Distribution

As the model stands, it is not very interesting. The default aperture of 5cm is much bigger than the typical sigma of the beam, which from the optics plots above can be seen to be of order 1mm. To experience even a few hits, would require billions of events to be simulated, which is of course not very efficient. We therefore specify a **halo** distribution of particles that are likely to hit the aperture. The halo distribution is described in *Beam Distributions* and specifically in *halo*. We define a halo distribution according to the normal Twiss parameters at the start of the lattice but with a much greater sigma.

Even if a Gaussian distribution is ultimately required, a common technique is to generate a uniform distribution of particles and then weight the events in analysis according to the Gaussian.

Here is an example halo distribution

```
beam, alfx=1.108024744,
alfy=-1.907222942,
betx=6.848560987*m,
bety=2.935758992*m,
distrType="halo",
emitx=2e-09*m,
emity=1.195785323e-11*m,
enenergy=1.282*GeV,
particle="e-",
```

(continues on next page)
sigmaE=0.0008,
haloNSigmaXInner = 30,
haloNSigmaXOuter = 80,
haloNSigmaYInner = 100,
haloNSigmaYOuter = 500,
haloPSWeightParameter = 1,
haloPSWeightFunction = "oneoverr";

To validate this distribution and visualise it, we can generate only the particles without performing the full simulation. We execute BDSIM with the --generatePrimariesOnly option. As the generation is very quick, we can afford to generate a large number of particles. Here 10000 were generated in approximately 10s.

```
bdsim --file=atf2-halo.gmad --generatePrimariesOnly --outfile=haloprimaries --
˓→batch --ngenerate=10000
```

We can then load and visualise the data using pybdsim. This is shown using a convenience function for the primary particle distribution:

```
> python
>>> import pybdsim
>>> pybdsim.Plot.PrimaryPhaseSpace('haloprimaries.root')
```

This produces the following figures. The user of course can create their own plots by loading the data.

![Figures showing primary particle distribution](image)

The raw data can be loaded from any sampler manually:

```
> python
>>> d = pybdsim.Data.Load("haloprimaries.root")
>>> psd = pybdsim.Data.PhaseSpaceData(d)
>>> allData = pybdsim.Data.SamplerData(d,0)
```

The “phase space data” is only the data required to make the above plots. The “sampler data” is all the data including weights, PDG ID, track ID etc.

The object “psd” here contains a member dictionary called “data” that has a numpy array for each key inside it.
Halo Simulation

As the model stands, no physics processes are registered so any particles hitting the machine will not interact with the matter and pass straight through. This is useful for efficient tracking and optical validation but not useful for a physics study. We therefore specify a physics list. For a 1.3GeV electron, the basic electromagnetic physics list from Geant4 as well as the decay physics and some muon specific processes are useful. The full set of physics lists are described in *Physics Processes*.

```python
>>> psd.data.keys()
['energy', 'T', 'yp', 'y', 'x', 'xp', 'z', 'zp']
>>> x = psd.data['x']
```

By default, samplers are attached to everything. Whilst suitable for optical comparison this produces a huge amount of data for a physics study. We turn this off by commenting it out with an exclamation mark.

```python
!sample, all;
```

We have now specified the halo distribution as described above, a default aperture and physics processes. One final step is to turn off sensitivity to the tunnel geometry as this is not required.

```python
tun : placement, geometryFile="gdml:../atf2_tunnel.gdml", x=-4.5*m, z=49*m, sensitive=0;
```

The input gmad file prepared is supplied in `bdsim/examples/atf2/nlsige/atf2-halo.gmad`.

We first run a small sample to gauge the length of the simulation and that the results are very roughly what we expect or want to see (before running a large number of particles).
This took approximately 10s to simulate and produced an output file “t1.root”. We perform a very quick and simple analysis now to investigate what happened in the simulation.

**Analysis**

The first simple analysis step is make a histogram of the mean energy deposition per event. BDSIM by default records a histogram of energy deposition per event. One could run the analysis tool `rebsdsm` with an input `analysis-Config.txt` specifying histograms. This would also merge (take the average of) the pre-made per event histograms. A utility is provided for merging only the histograms.

```bash
> rebdsimHistoMerge t1.root t1_ana.root
```

This loops over all events in the file and combines the per event histograms and writes them to a file called “t1_ana.root” here. To inspect this file, we load it in ROOT and browse it using a `TBrowser`.

```bash
> root -l t1_ana.root
> $> TBrowser tb;
```

This produces the following browser. We double click on the “t1_ana.root” file and then the folders inside. There is a folder for each Tree in the output and then per entry simple and merged histograms. We look inside and double click on the histogram to view it.

![Fig. 12: TBrowser in ROOT showing file structure from rebdsim / rebdsimHistoMerge.](image)

The energy deposition is in GeV / event. The horizontal axis is the curvilinear S coordinate in metres. The default binning is 1m and can be controlled with the option `elossHistoBinWidth=1*m;`.

As the level of energy deposition varies by many orders of magnitude, it is useful to view the histogram on a logarithmic scale. By right-clicking in the TBrowser close to the axis, the option “SetLogy” can be used.
Fig. 13: Setting log y axis in ROOT.
We can then repeat this simulation and simple analysis for a greater number of primary particles. The file examples/atf2/10khalo_ana.root is included from the analysis of 10000 particles. The simulation took 976s and produced a 178MB ROOT output file on the developer’s computer.

**Spectra at Plane**

To investigate the radiation at a plane at some point in the accelerator we can place a sampler on an element of interest. Here, we place a sampler on “B5FFB”, which is a dipole at the end of the long straight section in the lattice. In reality, cherenkov detectors were placed after this dipole in the past for detecting signal from experiments such as the laserwire experiment. The sampler is added via the command:

```
sample, range=B5FFB;
```

Sampler record the passage of any particle through them, even if it’s backwards or the same particle again. They are (by default) a 5m wide square plane that’s 1pm thick.

A simple analysis is to make a 2D histogram of the particle flux and the energy weighted particle flux at this plane. To do this we use the analysis tool rebdsim. This takes an input text file defining histograms. The syntax is described in *Preparing an Analysis Configuration File*. The analysisConfig.txt used is provided in examples/atf2/analysisConfig.txt.

```plaintext
InputFilePath 10k.root
OutputFileName 10khalo_ana.root
# Object  treeName  Histogram Name  # Bins  Binning
  → Variable  Selection
Histogram1D Event. XFlux  {40}  {-2:2}
  → B5FFB.x  1
Histogram1D Event. XFlux-Energy-Weighted  {40}  {-2:2}
  → B5FFB.x  B5FFB.energy
```

(continues on next page)
We can view the histograms as before, but we can also easily load them in Python and make our own plots.

```python
>>> import pybdsim
>>> d = pybdsim.Data.Load("10khalo_ana.root")
>>> d.ConvertToPybdsimHistograms(d.histograms1dpy)
>>> d.histograms1dpy
```

The `pybdsim` data loader automatically extracts the root histograms into Python dictionaries called “histogramsXd” where “X” is the number of dimensions. All exist in “histograms”. These are also automatically converted to numpy arrays and held in classes provided by `pybdsim` in the same members suffixed with “py” such as “d.histograms1dpy”. Calling these dictionaries shows the name of the histogram that is the full path inside the file.

```python
{'Event/MergedHistograms/ElossHisto': <ROOT.TH1D object ('ElossHisto') at 0x7f83a0cfba20>,
'Event/MergedHistograms/ElossPEHisto': <ROOT.TH1D object ('ElossPEHisto') at 0x7f83a1970000>,
'Event/MergedHistograms/PhitsHisto': <ROOT.TH1D object ('PhitsHisto') at 0x7f83a0cfa8e0>,
'Event/MergedHistograms/PhitsPEHisto': <ROOT.TH1D object ('PhitsPEHisto') at 0x7f83a1a00640>,
'Event/MergedHistograms/PlossHisto': <ROOT.TH1D object ('PlossHisto') at 0x7f83a0cfb310>,
'Event/MergedHistograms/PlossPEHisto': <ROOT.TH1D object ('PlossPEHisto') at 0x7f83a1a00a30>,
'Event/PerEntryHistograms/XElectrons': <ROOT.TH1D object ('XElectrons') at 0x7f83a0cd89d0>,
'Event/PerEntryHistograms/XFlux': <ROOT.TH1D object ('XFlux') at 0x7f83a0c9430>,
'Event/PerEntryHistograms/XFlux-Energy-Weighted': <ROOT.TH1D object ('XFlux-Energy-Weighted') at 0x7f83a0cd70f0>,
'Event/PerEntryHistograms/XPositrons': <ROOT.TH1D object ('XPositrons') at 0x7f83a0cd8320>,
'Event/PerEntryHistograms/YFlux': <ROOT.TH2D object ('YFlux') at 0x7f83a0cd8320>,
'Event/PerEntryHistograms/YFlux-Energy-Weighted': <ROOT.TH2D object ('YFlux-Energy-Weighted') at 0x7f83a0cd95a0>,
'Event/PerEntryHistograms/YFlux-Energy-Weighted': <ROOT.TH2D object ('YFlux-Energy-Weighted') at 0x7f83a0cd95a0>}
```

The Python versions can be easily plotted using `pybdsim`.

14.2. Worked Examples 251
These produce the following figures.

We leave it to the user to create the plots they desire. However, the primary particle impact, loss and associated energy deposition is a useful standard plot that is provided in pybdsim. The optional survey arguments allow a machine diagram to be added on top of the plot.

Just the energy deposition can be plotted.

```python
>>> pybdsim.Plot.LossAndEnergyDeposition('10khalo_ana.root', tfssurvey='../atf2-nominal-twiss-v5.2.tfs.tar.gz')
```

```python
>>> pybdsim.Plot.EnergyDeposition('10khalo_ana.root', tfssurvey='../atf2-nominal-twiss-v5.2.tfs.tar.gz')
```
Fig. 15: Primary particle impact points, losses and energy deposition from the simulation.

Fig. 16: Primary particle impact points, losses and energy deposition from the simulation.
14.3 Specific Machines

Converted examples of specific accelerators around the world.

14.3.1 Accelerator Test Facility 2 - KEK, Japan

This is the 1.3GeV energy scaled test facility for the ILC final focus system. The real machine consists of an approximately 70m normal conducting linac, transfer line, racetrack damping ring and finally an extraction line. This model represents only the ~100m extraction line.

The model was prepared in MADX and the Twiss output in MADX TFS format is provided (with tar gzip). Two copies of this are provided with only the energy spread in the header being different - in one case, the nominal 0.0008, and the other 0.

Three converted versions are provided. These are:

1) “linsige0” - linear optics, with fractional energy spread 0.
2) “linsige” - linear optics, with nominal fractional energy spread 0.0008.
3) “nlsige” - full nonlinear optics, with nominal fractional energy spread 0.0008.

How to run:

```
bdsim --file=linsige.gmad --outfile=test1
```

The full machine as visualised by default in BDSIM.
The full machine looking from the beginning to the end.

The full machine looking from the end to the beginning with perspective.

Comparison of beam size as calculated by rebdsimOptics from tracking 1000 particles through the full machine.

### 14.3.2 Diamond Light Source

The Diamond Light Source at Rutherford Appleton laboratory in Oxford, UK. The model is of the 560m storage ring. The model was prepared from a MADX job where the sextupoles throughout the lattice were treated as thin sextupoles. The Twiss table output in TFS format is included with this example and compressed for convenience (pybdsim and pymadx both work with compressed files).

The BDSIM model provided was converted directly without modification.

**Note:** This is a circular machine and should be executed with the `--circular` option to control the number of turns the particles will take.

```
bdsim --file=dia-nlsige.gmad --batch --circular --ngenerate=1000
```
This will run 1000 particles with a maximum of 1 turn. Below are sample visualisations and the beam size and dispersion as calculated using \textit{rebdsimOptics} and comparison with \textit{pybdsim}.

The full machine as visualised with BDSIM.

Part of the machine visualised with perspective.

The beam size in horizontal and vertical throughout a single turn of the ring with 5000 particles. Particles tracked in BDSIM, sizes calculated with \textit{rebdsimOptics} and comparison with \textit{pybdsim}.

The dispersion in both horizontal and dispersion throughout a single turn of the ring with 5000 particles. Particles tracked in BDSIM, sizes calculated with \textit{rebdsimOptics} and comparison with \textit{pybdsim}.

\textbf{14.3.3 Large Hadron Collider}

The Large Hadron Collider is approximately 27km in circumference proton proton collider at CERN, Geneva, Switzerland. It is the world’s highest energy particle accelerator and collider with a current centre of mass energy of 13TeV (6.5TeV per beam).

This is one of the largest BDSIM models. The model provided is of the 2017 optics with finite dispersion throughout.
Note: This is a circular machine and should be executed with the --circular option to control the number of turns the particles will take.

Example:

```
bdsim --file=lhc2017.gmad --batch --circular
```

This will run 5 particles with a maximum of 1 turn.

It is not useful to visualise this model as the ring consists of 0.5m wide dipoles in a ring approximately 8km across. Even if the visualiser resolution and computer display resolution supported 8000 pixels across, the magnet would be only one pixel. The viewpoint is automatically centred to the middle of the world so zooming in does not reveal anything as you zoom into the empty part of the ring.

It is possible to visualise, but the visualisation takes ~10mins and the visualiser commands should be used to move directly to a point of interest in global coordinates. The slowness is due to the Geant4 visualisation system.

Here, we show an example optical comparison. In all cases, rebdsimOptics was used to calculate the beam properties and pybdsim used to make the comparison plots.

Beam mean position (i.e. ‘orbit’) from 500 particles tracked for 1 turn.

Beam size from 500 particles tracked for 1 turn.

Dispersion from 500 particles tracked for 1 turn.

### 14.3.4 International Linear Collider

These were original models that came with BDSIM up until around 2011. A newer version of the International Linear Collider (ILC) lattice is being prepared.

This is a 250GeV (each beam) electron positron collider. Typically, both the old and new models will simulate the approximately 2km beam delivery system (BDS).

### 14.4 Features

These are tests which also act as examples of syntax.
14.4.1 Features

Geometry

Layout

1_layout.gmad

A simple machine with a few dipoles, drifts and quadrupoles. This is not matched or properly designed, but purely an example to show and test the layout of components.

How to run:

```
bdsim --file=1_layout.gmad
```

2_one_of_each.gmad

A small lattice with one of each component with default geometry options.

How to run:

```
bdsim --file=2_one_of_each.gmad
```

Transform3d

1_rotation.gmad

A drift, followed by a transform3d that rotates about the beam axis, then followed by another drift and a quadrupole. The drifts have rectangular beam pipes so the rotation caused by the transform3d can be seen. The quadrupole is therefore also rotated. A sampler placed at the end will also be rotated with respect to global coordinates as the transform3d permanently rotates the coordinate axes.

How to run:
bdsim --file=1_rotation.gmad

A small machine with some bends, quadrupoles, a sextupole and collimator. This is used to demonstrate the rotation of coordinates that transform3d has on the beam line and demonstrates a method to create vertical bends.

How to run:

bdsim --file=2_rotation_with_bends.gmad

2_rotation_with_bends.gmad

A small machine with some bends, quadrupoles, a sextupole and collimator. This is used to demonstrate the rotation of coordinates that transform3d has on the beam line and demonstrates a method to create vertical bends.

How to run:

bdsim --file=2_rotation_with_bends.gmad

Beam Pipes

1_circular.gmad

A 0.2m section of circular beam pipe - nothing particularly interesting.

How to run:

bdsim --file=1_circular.gmad
2_rectangular.gmad

A 0.2m section of rectangular beam pipe.

How to run:

```
bdsim --file=2_rectangular.gmad
```

3_elliptical.gmad

A 0.2m section of elliptical beam pipe. The definition of the drift overrides the default parameter of `beampipeThickness` here.

How to run:

```
bdsim --file=3_elliptical.gmad
```

4_lhc.gmad

A 0.2m section of lhc-style beam pipe. The definition of the drift overrides the default parameter of `beampipeThickness` here. Additionally, `aper1` in the definition of the drift $dI$ overrides the general (degenerate) `beampipeRadius` option in options.gmad.

How to run:

```
bdsim --file=4_lhc.gmad
```

5_lhcdetailed.gmad

Similarly to 4), a 0.2m section of lhc-style beam pipe but with the more detailed lhc aperture model.

How to run:
6_rectellipse.gmad

A 0.2m section of rectangular-ellipse beam pipe. This is composed of the intersection of a rectangle and an ellipse, unlike the lhc-style beam pipe that is the intersection of a rectangle with a circle.

How to run:

```
bdsim --file=5_lhcdetailed.gmad
```

7_racetrack.gmad

A small section of beam pipe with a MADX racetrack aperture style. This is a rectangle with circularly rounded corners.

How to run:

```
bdsim --file=7_racetrack.gmad
```

8_octagonal.gmad

A small section of beam pipe with an octagonal aperture style. This is a rectangle with flat cut corners.

How to run:
Circular vacuum doesn’t make any beam pipe and the vacuum is by default invisible so there is nothing to visualise. However, this is useful for visualising the trajectories of the beam.

How to run:
```
 bdsim --file=9.circularvacuum.gmad
```

10_clicpcl.gmad

A small section of CLIC post collision line beam pipe.

How to run:
```
 bdsim --file=10_clicpcl.gmad
```
Magnet Geometry

one_of_each_base.gmad

A base file that’s used (include one_of_each_base.gmad) by many examples in this directory. It’s loosely based on the one of each example in layout and contains one of each of the magnetic elements in BDSIM. Each example demonstrates a different geometry possible.

1_cylindrical.gmad

Key code:

```plaintext
magnetGeometryType="cylindrical";
```

This example uses the sequence of magnets defined in one_of_each_base.gmad and ensures that they have cylindrical geometry. This is a little redundant as the default is cylindrical.

How to run:

```
bdsim --file=1_cylindrical.gmad
```

2_poles_circular.gmad

Key code:

```plaintext
magnetGeometryType="polescircular";
```

As above but with poled geometry and a circular yoke.

How to run:

```
bdsim --file=2_poles_circular.gmad
```
3_poles_square.gmad

Key code:

```plaintext
magnetGeometryType="polessquare";
```

As above but with poled geometry and a square yoke.

How to run:

```plaintext
bdsim --file=3_poles_square.gmad
```

4_poles_face.gmad

Key code:

```plaintext
magnetGeometryType="polesfacet";
```

As above but with poled geometry and a square yoke rotated by 45 degrees.

How to run:

```plaintext
bdsim --file=4_poles_facet.gmad
```

5_poles_face_crop.gmad

Key code:

```plaintext
magnetGeometryType="polesfacetcrop";
```

As above but with poled geometry and a square yoke rotated by 45 degrees. Additionally, the corner edges are cropped giving the yoke, \( n \times 2 \) edges, where \( n \) is the number of poles the magnet has.

How to run:
bdsim --file=5_poles_facet_crop.gmad

6_lhcleft.gmad & 7_lhcright.gmad

Key code:

```plaintext
magnetGeometryType="lhcleft";
magnetGeometryType="lhcright";
```

LHC cryogenic magnet for dipoles and quadrupoles. Given the two beam nature of the LHC, the magnet is either offset to the right or two the left dictating the ‘active’ beam pipe.

8_none.gmad

Key code:

```plaintext
magnetGeometryType="none";
```

No magnet outer geometry is built around the beam pipe - only the beam pipe exists.

9_coils.gmad

A test with carefully split and spaced quadrupoles to test the automatic choice to place or not the coils at the end of a magnet.
An example / test to show the scaling of the circular outer geometry.

10_size_variation_circ_quad.gmad

An example / test to show the scaling of the circular outer geometry.

11_size_variation_circ_sext.gmad

An example / test to show the scaling of the circular outer geometry.

12_size_variation_circ_oct.gmad

An example / test to show the scaling of the circular outer geometry.

13_size_variation_circ_dec.gmad

An example / test to show the scaling of the circular outer geometry.
14.4. Features

14_size_variation_sq_quad.gmad

An example / test to show the scaling of the square outer geometry.

15_size_variation_sq_sext.gmad

An example / test to show the scaling of the square outer geometry.

16_size_variation_sq_oct.gmad

An example / test to show the scaling of the square outer geometry.
17_size_variation_sq_dec.gmad

An example / test to show the scaling of the square outer geometry.

18_size_variation_facet_quad.gmad

An example / test to show the scaling of facet outer geometry.

19_size_variation_facet_sext.gmad

An example / test to show the scaling of facet outer geometry.
20_size_variation_facet_oct.gmad

An example / test to show the scaling of facet outer geometry.

21_size_variation_facet_dec.gmad

An example / test to show the scaling of facet outer geometry.

22_size_variation_facetcrop_quad.gmad

An example / test to show the scaling of facetcrop outer geometry.
23_size_variation_facetcrop_sext.gmad
An example / test to show the scaling of facetcrop outer geometry.

24_size_variation_facetcrop_oct.gmad
An example / test to show the scaling of facetcrop outer geometry.

25_size_variation_facetcrop_dec.gmad
An example / test to show the scaling of facetcrop outer geometry.
26_size_variation_sbend_c.gmad

An example / test to show the scaling of sector bend outer geometry.

27_size_variation_rbend_c.gmad

An example / test to show the scaling of rectangular bend outer geometry.

28_size_variation_hkicker_c.gmad

An example / test to show the scaling of horizontal kicker outer geometry.

29_size_variation_vkicker_c.gmad

An example / test to show the scaling of vertical kicker outer geometry.
30_dipole_pole_face.gmad

An example/test to show sector bend with pole face rotation outer geometry.

Tunnel Geometry

For each tunnel test, there is an a and a b version. These are the same as the individual test but with elliptical and rectangular geometry respectively. A square tunnel section is really just a rectangular tunnel section, so this is tested as well.

1_long_straight.gmad

A drift section, with a thin collimator that’s offset so the beam definitely hits it, followed by another long drift section. The tunnel is cylindrical, and of a typical size of an accelerator and offset a little bit. The floor and soil are also built. This uses the newer modular physics lists.

This test builds one straight section of tunnel from start to finish irrespective of the beam line as controlled by buildTunnelStraight=1.

Key code:

```plaintext
option, buildTunnel=1,
    buildTunnelStraight=1,
    tunnelType="circular",
    tunnelThickness=1*m,
    tunnelSoilThickness=5*m,
    tunnelMaterial="concrete",
    soilMaterial="soil",
    buildTunnelFloor=1,
    tunnelFloorOffset=1.2*m,
    tunnelAper1=5*m,
    tunnelAper2=3*m,
    tunnelSensitive=1,
    tunnelVisible=1,
    tunnelOffsetX=0.4*m,
    tunnelOffsetY=-1.2*m;
```

How to run:

```plaintext
bdsim --file=1_long_straight.gmad
```

2_long_straight_following.gmad

This is the same as 1_long_straight.gmad but the tunnel building algorithm is allowed to follow the beam line. As it’s straight, it should result in a very similar outcome.

Key code:
option, buildTunnelStraight=0;

How to run:

bdsim --file=2_long_straight_following.gmad

3_initial_bend.gmad

This lattice has a relatively strong bend at the beginning of the lattice, followed by a long straight section. This tests the tunnel building algorithm’s ability to follow the beam line after a bend.

How to run:

bdsim --file=3_initial_bend.gmad

4_several_bends.gmad

This lattice has long straight sections with relatively sharp bends and this pattern is repeated several times.

How to run:

bdsim --file=4_several_bends.gmad
5_several_bends_back_and_forth

This example is much like 4_several_bends.gmad but also bends the other way (back and forth).

How to run:

```
bdsim --file=5_several_bends_back_and_forth.gmad
```

6_very_long_following.gmad

This example is much like 2_long_straight_following.gmad but longer and with not round number lengths. Being longer, the tunnel algorithm will split the tunnel sections up more than the single section produced in 2.

How to run:

```
bdsim --file=6_very_long_following.gmad
```

7_long_arc.gmad

This example contains more gradual bends and many of them separated by short drifts and is relatively long. This tests part of an arc in a collider.

How to run:

```
bdsim --file=7_long_arc.gmad
```

8_samplers.gmad

This example is roughly based on 7_long_arc.gmad (similar in form but not exactly) with the addition of samplers on every element, including a marker at the end as well as a superfluous one at the beginning. The tunnel geometry should break around these samplers leaving a 1 μm gap to avoid geometrical overlaps.

How to run:
Curvilinear Coordinates

1_s_offset.gmad

This demonstrates starting the initial beam distribution from a finite curvilinear S position along the line. This example / test is placed in a drift (ie straight section). The particle should finish at 0, 0 in curvilinear coordinates.

To run:

```bash
bdsim --file=1_s_offset.gmad
```

2_s_offset_in_strong_bend.gmad

Similar to 1_s_offset, but with the particle starting in the middle of a strong bend. Here, although the magnets are made from many short straight sections, the curvilinear trajectory is interpolated.
GDML

1_magnet_outer.gmad

A quadrupole flanked by two drifts. The outer geometry of the quadrupole uses a gdml file of a Hitachi magnet at the ATF2.

How to run:

```
bdsim --file=1_magnet_outer.gmad
```

2_placement.gmad

A placement of a piece of geometry (in GDML format) into the world in an arbitrary location with respect to the coordinate system origin and not the beam line.

How to run:

```
bdsim --file=2_placement.gmad
```
3_twogdmls.gmad

Example of using two different GDML files that contain objects of the same name. The Geant4 GDML loader would normally not load these correctly and use the already loaded geometry. BDSIM corrects this behaviour.

How to run:

```
bdsim --file=3_twogdmls.gmad
```

![Diagram showing two shapes alongside a beam pipe - a cuboid and a sphere.]

**Note:** Two shapes are visible alongside a beam pipe - a cuboid and a sphere.

Multiple Beam Lines

**multiplebeamlines.gmad**

This example shows several beam lines placed in the same model. Currently only one of these is suitable for tracking. This is mostly for display purposes and current development will render each beam line suitable for tracking.

How to run:

```
bdsim --file=multiplebeamlines.gmad
```

![Diagram showing multiple beam lines.]

**Crystals**

In each of the following examples one or more crystals are placed inside a piece of beam pipe through the `crystalcol` beam line element.

As the typical angles for crystal channelling are very low (in the miliradian to microradian regime), exaggerated examples are provided that are not suitable for channelling studies but are however useful to visualise the geometry involved. These exaggerated versions are suffixed with “strong”.

![Diagram showing crystals placed in beam pipes.]

280 Chapter 14. Examples & Tests
There is the possibility of one or two crystals. Models with “1” in the name have only 1 and with “2” in the name two.

**Note:** The crystal structure is independent of the geometry used and the geometry merely acts as a shape where the crystal applies.

### crystal-box

- crystal-box1-strong.gmad
- crystal-box1.gmad
- crystal-box2-strong.gmad

Crystals with the simple box geometry.

### crystal-cylinder

- crystal-cylinder1-strong.gmad
- crystal-cylinder1.gmad
• crystal-cylinder2-strong.gmad

Crystals with geometry that’s a section of a hollow cylinder.

---

**crystal-cylinder**

• crystal-cylinder1-strong.gmad
• crystal-cylinder1.gmad
• crystal-cylinder2-strong.gmad

Crystals with geometry that’s bent in both axes. This would similar to a small section of a hollow torus, hence the “torus” name. However, this is constructed as a G4ExtrudedSolid as it’s possible to choose angles that would result in an unphysical torus.

To construct the extruded solid, a cross-section is made at one (local, not machine coordinate frame) z plane and extruded along the (local) z axis with different offsets in x. Currently, 30 points are used along each dimension. The points are shown in the following figure.
Fields

Field Transforms

These are various examples that test the rotation of a field and its effect on the beam.

1_dipole_rotation

This contains two sector bend dipoles that are rotated using a `transform3d` instance so that they bend out of the global plane. This tests whether the dipole fields are correctly transformed for a given element.

How to run:

```
bdsim --file=1_dipole_rotation.gmad
```
All support issues can be submitted to our issue tracker.

15.1 BDSIM News Letter

For the latest updates about BDSIM and future releases, please register for our mailing list:
https://www.jiscmail.ac.uk/lists/BDSIM.html

15.2 Trouble Running BDSIM

For trouble with installation or running, see the Troubleshooting section. Bugs can be submitted to the issue tracker. Please have a look at the existing list of open bugs before submitting a new one.

15.3 Feature Request

Feature requests or proposals can be submitted to the issue tracker. Please have a look at the existing list of proposals before submitting a new one.

The development of BDSIM follows the needs of the developers and the users so please propose developments that will be useful and we can collaborate on them.
CHAPTER SIXTEEN

VERSION HISTORY

16.1 V1.3.2 - 2019 / 04 / 20

16.1.1 New Features

- Can now use any particle available through the physics list for a beam particle.
- Generic cross-section biasing can now be used for any particle, including ions.

16.1.2 General

- Tested with Geant4.10.5.p01
- Geometry navigators are reset at the beginning of a run now in a similar way to the start of an event to ensure independence between runs - future proofing.
- For Geant4.10.5, we now use the ‘low’ looping particle thresholds for tracking.
- The ‘vacuum’ field is now not applied to the container volume of a beam pipe. However, it is still applied to the vacuum and beam pipe volumes. This makes the tracking more robust against stuck particles in the extremely small gap between volumes.
- The yoke magnetic field now uses a wrapped G4ClassicalRK4 integrator. This wrapper acts as a drift for short (< 1um) steps. This makes tracking more robust for secondaries in the yoke.
- Improve testing for user bunch distribution for robustness.
- Increase transverse length safety margin between beam pipes and magnet volumes for safety.
- Translate bunch coordinates in global coordinates backwards by 1x length safety to avoid starting on a volume boundary at the start of the event. This is 1nm so will not affect tracking results. The local coordinates in the output are identically the same.

16.1.3 Bug Fixes

- Fix strong recreation when using user file supplied bunch distribution. The file was always read from the beginning in the past. Now the correct coordinates will be read and the event is correctly reproduced.
- Fix userinterface example given changes to sensitive detector manager - simple edit.
- Fix calculated phase offset for rfcavity in the beam line. This was peak at the end of the element rather at the middle.
- Fix possible segfault if event aborted due to extra collimator hit information.
- Fix user file beam loader for anomalous particle coordinates generated once at the end of a file if the end of the file was a blank line. It would result in all zero coordinates and beam energy x1000 for the first particle generated after the end of the file was reached.
- Fix abort of event if unknown particle ID specified in user file beam loader.
• Fix user file distribution file loading for comment lines, incomplete lines and empty (white space) lines.

• Fix phase offset calculation for rf cavities with respect to nominal value. Phase would have been smaller than intended. It was scaled to $1/2\pi$ instead of $2\pi$.

• Fix ambiguity in manual for rf cavities. Time is generally in seconds in BDSIM, however the rf cavity took nanoseconds. A time offset of $1*ns$ in the input gmad would result in double units.

• Fix warning when loading an output file with data loader class when the file was created without storing primary coordinates. The warning was related to the PrimaryGlobal branch.

• Fix warnings and artificial killing of particles by high looping particle thresholds for Geant4.10.5, which are default. Use the ‘low’ looping thresholds by default. Issue #268.

• Fix stuck particles by attaching the vacuum field in a beam pipe to every volume in the beam pipe apart from the container volume to avoid navigation problems in very thin gaps. Issue #268.

• Remove half-implemented integrator types in internal dictionaries.

• Fixed model-model example conversion Python scripts as these were specific to the developer’s computer.

• Fix coil end-piece placement with respect to main magnet body - now includes required length safety gap to avoid possible navigation issues with large sized models.

• Fix for exotic particle beams. Can now use any particle available in the physics list. Particle definitions now constructed earlier than in the regular physics list call.

• Fix bad tracking in undulators caused by nan values in field caused by querying the field at arbitrarily large positions. Limited the range of validity of the field to the beam pipe. The field is now no longer attached to the magnets outside the beam pipe.

• Biasing was not attached to components that were found to be unique in construction - i.e. an rbend back-to-back with another rbend will not have fringe fields in the middle at the join, so is considered a unique construction. This would result in these not having biasing attached.

16.1.4 Utilities

- pybdsim v2.1.0
- pymadx v1.7.1
- pymad8 v1.5.0
- pytransport v1.3.0

16.2 V1.3.1 - 2019 / 03 / 05

16.2.1 Bug Fixes

- Fix transform3d element where offsets were not working.

16.3 V1.3 - 2019 / 02 / 27

16.3.1 Expected Changes To Results

- The density of the surrounding air has changed very slightly to that of the standard G4_AIR one.

- Energy deposition in vacuum is now separated into a separate branch and is not mixed with general Eloss. Therefore, less energy deposition will be seen in the Eloss branch.
• The minimum kinetic energy option will now be respected when using a Geant4 reference physics list, whereas it wasn’t before.

• The range cuts can now be applied when using a Geant4 reference physics list if the option g4PhysicsUseBDSIMRangeCuts is turned on (=1). Previously, these had no effect with a Geant4 reference physics list.

16.3.2 New Features

• Support for Geant4.10.5.

• New environment script in `<bdsim-install-dir>/bin/bdsim.sh` to make running BDSIM easier.

• All Geant4 reference physics lists are now available.

• New beam pipe aperture for the CLIC post collision line.

• New jaw collimator element “jcol” with two blocks in the horizontal plane.

• New wire scanner element “wirescanner” with cylindrical wire in a beam pipe.

• Completed CMake to allow user applications based on BDSIM to easily link against it.

• New dump beam line element that is an infinite absorber. This prevents simulations running for a long time when particles may hit the air at the end of the beam line.

• BDSIM as a class for interfacing. Ability to add custom beam line components. See Interfacing.

• New samplerplacement object that defines an arbitrarily placed sampler in the world that may overlap with anything (see Output at an Arbitrary Plane - User Placed Sampler).

• New importance sampling implementation when using a user-supplied world geometry. (see Geometric Importance Sampling).

• New options:
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collimatorsAreInfiniteAbsorbers</td>
<td>When turned on, all particles that enter the material of a collimator ((rcol, ecol) and (jcol)) are killed and the energy recorded as deposited there.</td>
</tr>
<tr>
<td>geant4Macro</td>
<td>Fun an optional macro in the visualiser once it’s started.</td>
</tr>
<tr>
<td>g4PhysicsUseBDSIMCutsAndLimits</td>
<td>If on, the maximum step length will be limited to 110% of the component length - this makes the tracking more robust and is the default with a regular BDSIM physics list. The minimum kinetic option is also obeyed. Default off.</td>
</tr>
<tr>
<td>g4PhysicsUseBDSIMRangeCuts</td>
<td>If on, this will apply the BDSIM range cut lengths to the Geant4 physics list used. This is off by default.</td>
</tr>
<tr>
<td>ignoreLocalMagnetGeometry</td>
<td>If turned on, this option means that only the magnet geometry from options will be used. Similar to (\text{ignoreLocalAperture}).</td>
</tr>
<tr>
<td>importanceVolumeMap</td>
<td>File path for text file that maps importance values to volumes.</td>
</tr>
<tr>
<td>importanceWorldGeometryFile</td>
<td>File path for the externally provided geometry that will be used as the parallel world for the importance sampling.</td>
</tr>
<tr>
<td>physicsEnergyLimitLow</td>
<td>Control minimum energy for all physics models. (advanced)</td>
</tr>
<tr>
<td>physicsEnergyLimitHigh</td>
<td>Control maximum energy for all physics models. (advanced)</td>
</tr>
<tr>
<td>minimumKineticEnergyTunnel</td>
<td>Any particles below this energy (in GeV by default) will be artificially killed in all BDSIM-generated tunnel segments.</td>
</tr>
<tr>
<td>storeCollimatorInfo</td>
<td>Store collimator structure with primary hits per collimator.</td>
</tr>
<tr>
<td>storeCollimatorHitsAll</td>
<td>If (\text{storeCollimatorInfo}) is on and collimator hits are generated, hits will be generated for all particles interacting with the collimators whether primary or secondary and whether ion or not.</td>
</tr>
<tr>
<td>storeCollimatorHitsIons</td>
<td>If (\text{storeCollimatorInfo}) is on and collimator hits are generated, (islon, ionA) and (ionZ) variables are filled. Collimator hits will now also be generated for all ions.</td>
</tr>
<tr>
<td>storeCollimatorLinks</td>
<td>If (\text{storeCollimatorInfo}) is on and collimator hits are generated, extra information is stored for each collimator hit.</td>
</tr>
<tr>
<td>storeEloss</td>
<td>Ability to completely turn off generation of energy deposition hits to save memory usage and output file size. Default on.</td>
</tr>
<tr>
<td>storeElossModelID</td>
<td>Control whether the beam line index is stored in the energy loss output. More granular than (\text{storeElossLinks}).</td>
</tr>
<tr>
<td>storeElossTurn</td>
<td>Control whether energy deposition turn number is saved.</td>
</tr>
<tr>
<td>storeElossVacuum</td>
<td>Control whether energy deposition in the residual gas in the beam pipe ‘vacuum’ is recorded.</td>
</tr>
<tr>
<td>storeElossWorld</td>
<td>Turn on generation of energy deposition in the world volume (i.e. the air) as well as record energy leaving the simulation. Default off.</td>
</tr>
<tr>
<td>storeElossWorldContents</td>
<td>Turn on generation and storage of energy deposition in any included with the externally provided world geometry. Off by default but turned on automatically when using importance sampling. Allows the user to distinguish energy deposition in the air as stored in ElossWorld from the contents of the world.</td>
</tr>
<tr>
<td>storeGeant4Data</td>
<td>Control whether the basic particle data is stored in the output for all particles used or not.</td>
</tr>
<tr>
<td>storeSamplerAll</td>
<td>Conveniently store all optional sampler data with one option.</td>
</tr>
<tr>
<td>storeSamplerKineticEnergy</td>
<td>Store kinetic energy in the sampler output.</td>
</tr>
<tr>
<td>storeSamplerPolarCoords</td>
<td>Store the polar coordinates ((r, phi \text{ and } rp, phip)) in the sampler output.</td>
</tr>
<tr>
<td>tunnelsIsInfiniteAbsorber</td>
<td>When turned on, any BDSIM-generated tunnel segments will absorb and kill any particle of any energy. Used to speed up the simulation. Default off.</td>
</tr>
<tr>
<td>worldGeometryFile</td>
<td>External geometry file for world geometry.</td>
</tr>
</tbody>
</table>

- Access to data version in DataLoader in analysis.
- External geometry can be supplied as the world volume with the option \(\text{worldGeometryFile}\).
- New complete physics list for crystal channelling to achieve the correct result.
- New ability to specify a different beam particle that is different from the design particle used for magnetic field strength calculations \(\text{beamParticleName}\).
- Specify the particle assumed for the user file distribution that can be different from the design particle.
• New option to use a one turn map generated from MAD-X PTC to correct multi-turn tracking for circular machines.
• New option `geant4Macro` and executable option `--geant4Macro` to run an optional macro in the visualiser once it’s started.
• A warning will print if a user-defined material is more dense than 100g/cm³ as this is much higher than any naturally occurring material (on Earth). The simulation will still proceed.
• New optional collimator output structure in event made per collimator with prefix “COLL_”. Controlled by new option `collimatorInfo`.
• New mini-summary of collimators in Model tree when `collimatorInfo` option is used.
• New parameter for collimator elements `minimumKineticEnergy` that allows the user to kill particles below a certain kinetic energy in a collimator.

16.3.3 General

• All collimators now require a material to be specified and the default copper has been removed. This is because it strongly affects the results obtained and defaults should not be relied upon for this.
• The turn number for energy deposition hits is now automatically stored if a circular model is used.
• The `sensitiveBeamlineComponents` option has now been renamed to `sensitiveOuter` to better reflect its functionality. The old option is still accepted.
• The `tunnelSensitive` option has now been renamed to `storeElossTunnel` to be more consistent with the other sensitivity options. The old option is still accepted.
• The generic beam line element `element` now supports angle and the beam line will be curved by this amount.
• The world volume is now sensitive and can record energy deposition. Geant4.10.3 upwards is required to record both this information and the energy leaving the world as this requires G4MultiSensitiveDetector.
• New tests for testing backwards compatibility of analysis tool with previous data version.
• “Model Preparation” is now “Model Conversion” in the manual to be clearer.
• Visualisation now uses macro search path to look for visualisation macro in the installation directory then the build directory of BDSIM.
• In recreate mode, there is explicit print out about when the seed is set and if it was successfully loaded from the output file.
• The Cherenkov example has now been updated to show 3 materials (air, water, YAG).
• Fixes from static code analysis for virtual functions called in constructors of factories, shadow member variables and initialisation of crystal variables in parser.
• Significant reduction in use of the singleton pattern for beam pipe, magnet yoke, tunnel and geometry factories.
• Reduced memory usage for energy deposition hits by removing unused numbers stored each time.
• Reduced memory usage for energy deposition hits when not using extra variables such as the ‘links’.

16.3.4 Materials

• The materials construction in src/BDSMaterials.cc was checked through thoroughly.
• “air” is now G4_AIR instead of custom BDSIM air (similar composition). The old air is now “airbdsim”.
• The refractive index data for optical and cherenkov physics has been added on top of G4_AIR as well as “airbdsim”.
• “airbdsim” now has a density of 1.225mg/cm³.
• “bp_carbonmonoxide” material now has correct pressure (previously near infinite).
• Fixed double density for the following materials. They would have been extremely dense.
  – “berylliumcopper”
  – “stainless_steel_304L”
  – “stainless_steel_304L_87K”
  – “stainless_steel_316LN”
  – “stainless_steel_316LN_87K”
  – “tungsten Heavy alloy”
  – “fusedsilica”
  – “n-bk7”
  – “yag”
  – “pet”
  – “lhc_rock”
• “niobium” is now “niobium_2k” to better reflect the unusual temperature.
• “nbti” is now “nbti_4k” to better reflect the unusual temperature.
• “waterCkov” has been removed. “water” or “G4_W ATER” (the same) should be used. The refractive index data has been added to G4_W ATER material.

16.3.5 Developer Changes

• The BDSGeometryComponent base class now has the ability to specify which sensitive detector should be attached in a map using the BDSSDType enum. There is no default sensitive detector (previously general energy deposition) as the developer must be explicit about what sensitivity they want so nothing unexpected can happen.
• BDSBeamline can now return indices of beam line elements of a certain type.
• All sensitive detector classes have been renamed as have the accessor functions in BDSSDManager. This is to make the naming more consistent.

16.3.6 Bug Fixes

• Fixed reloading user file when reading more particles than defined in the file.
• Fixed “pt” column in user file for reading particle PDG IDs. The first particle would be read correctly and all subsequent particles would revert to the beam definition.
• Fixed infinite tracking from nans return from field map when BDSIM format field map file was lacking lower and upper limits.
• Fixed incorrect writing of optional sampler information.
• The sensitiveBeamPipe option now works and controls whether the beam pipe produces energy loss or not. This does not affect the physics, merely whether output information is generated or not.
• The sensitiveOuter (formerly sensitiveBeamlineComponents) option has been fixed and now controls whether the parts outside the beam pipe in an element record energy loss or not.
• Degrader and undulator did not record energy deposition.
• Energy deposition is now correctly recorded when tracks are artificially killed.
• Fix crystal channelling with cylindrical and torus shaped crystals. The crystal implementation only works along the local X direction of any solid. Fixed by using a G4DisplacedSolid to allow use of more advanced geometries than a box.

• Fix channelling physics for standard EM and hadronic processes as this requires process biasing.

• Fix A and Z being the wrong way around for ions in samplers.

• Charge now correctly recorded in primaries and in samplers for partially stripped ions.

• Solenoid tracking fixed. Fringes are constructed as appropriate according to integrator set.

• Fix possible nan values given to Geant4 tracking with miscalculated auto-scaling value for field maps.

• Fix setting default seed state for random number generator if using recreate mode and progressing beyond an event stored in the file.

• Fix setting the energy level of an ion - wasn’t set from input.

• SQL geometry factory didn’t clean up after repeated use. This geometry isn’t generally supported.

• Fixed a bug where very weak actions on particles in tracking would not be taken due to too stringent tests of finite numbers. This would result in particles with small offsets in magnets or particles with high momentum that would see only very small deviations being tracked as if it were a drift.

• Fixed segfault crash from ROOT with rebdsim when there were more dimensions in the variables than the declared number of dimensions. For example, “y:x” for Histogram1D.

• Fixed rare bug where segfault would occur in trying to account for energy deposition of artificially killed particles.

• Fix memory leak of sampler structures (relatively small).

• Fixed parsing of + or - symbols with ion definition. Now supports H- ion.

• Fixed very slow memory leak associated with the primary trajectory. only visible for very large numbers of events.

• Fixed dipole tracking for off-charge ions - reverts to backup integrator.

• Fixed Pytonic range iteration of Event tree when trying to look at Info branch. Conflicted with Info method of TObject. Now renamed to Summary.

• Fixed catching the construction of dipoles with too large an angle. Limit rbends and unsplit sbends to a maximum angle of pi/2, limit the maximum angle of all other dipoles to 2 pi.

16.3.7 Output Changes

• “Info” branch of the Event and Run trees are now “Summary”. This is to avoid conflict with ROOT TObject::Info() that could result in broken analysis or range iteration. The DataLoader class in analysis (used by pybdsim.Data.Load) is backwards compatible. In the case of loading older data with updated software, there will still be a member called Info that the data will be loaded into. Python range iteration cannot be used in this case.

• “TunnelHit” is now “EnergyLossTunnel” to be consistent. rebdsim and the analysis DataLoader class (both Python and ROOT) are backwards compatible and both TunnelHit and ElossTunnel are available. Only the correct one is filled with loaded data during analysis.

• Much more granular control of what is stored in the output. See new options in ‘new’ section above.

• Vacuum energy deposition separated from general energy deposition and now in its own branch.

• Memory usage (for Mac & Linux) added at the end of each event in event info. This is the memory usage of the whole program at that point including event independent quantities such as the model.

• Boolean flag store in even info as to whether the primary was absorbed in a collimator or not.

• New options to control level of output as described in table in new features.
• Tunnel energy deposition hits now respond to the `storeElossXXXX` options to control the level of detail with extra variables of their output.

• New class BDSOutputROOTEventLossWorld for a record of coordinates when a particle leaves a volume, use currently for exiting the world.

• New structures (“branches”) in the `Event` tree called `ElossWorld` and `ElossWorldExit` for energy deposition in the world material and energy leaving the world (and therefore the simulation) respectively.

• New members in `Event.Info` that are the integrated energy deposited in various parts for that event. These are for convenience and are the integrals of the various `Eloss` parts.

### 16.3.8 Output Class Versions

• Data Version 4.

<table>
<thead>
<tr>
<th>Class</th>
<th>Changed</th>
<th>Old Version</th>
<th>New Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDSOutputROOTEventBeam</td>
<td>Y</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>BDSOutputROOTEventCoords</td>
<td>N</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>BDSOutputROOTEventCollimator</td>
<td>Y</td>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>BDSOutputROOTEventCollimatorInfo</td>
<td>Y</td>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>BDSOutputROOTEventLossWorld</td>
<td>Y</td>
<td>NA</td>
<td>1</td>
</tr>
<tr>
<td>BDSOutputROOTEventHeader</td>
<td>N</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BDSOutputROOTEventHistograms</td>
<td>N</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BDSOutputROOTEventInfo</td>
<td>Y</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>BDSOutputROOTEventLoss</td>
<td>Y</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>BDSOutputROOTEventModel</td>
<td>Y</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>BDSOutputROOTEventOptions</td>
<td>Y</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>BDSOutputROOTEventRunInfo</td>
<td>N</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BDSOutputROOTEventSampler</td>
<td>Y</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>BDSOutputROOTEventTrajectory</td>
<td>N</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BDSOutputROOTEventTrajectoryPoint</td>
<td>N</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>BDSOutputROOTGeant4Data</td>
<td>N</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

### 16.3.9 Utilities

• pybdsim v2.0.0
• pymadx v1.7.0
• pymad8 v1.5.0
• pytransport v1.3.0

### 16.4 V1.2 - 2018 / 08 / 26

#### 16.4.1 Highlights

• `outerDiameter` is now `horizontalWidth` to better describe its function (backwards-compatible).
• Fixed dipole scaling with (the default) `bdsimmatrix` integrator set.
• Solenoid tracking fixed.
16.4.2 New Features

- New options to activate extra models in em_extra physics list.
- New `crystalcol` element for channelling crystals.
- New `crystal` definition in parser.
- New “channelling” physics list for Geant4 crystal channelling physics process.
- Field maps need not be in x, y, z, t order and lower dimension fields (i.e. 1D or 2D) can be made for any dimension, i.e. it is now possible to specify a 1D field along the z direction.
- Rebsdsm can now analyse a select range of events specified by “EventStart” and “EventEnd” options. Issue #240.
- Placements can now be made with respect to S,x, and y in the main beam line, with respect to a beam line element and lastly in global Cartesian coordinates.
- Samplers will no longer be automatically attached (with `sample, all;`) to dipoles with finite pole face rotations, as this may lead to tracking issues in Geant4. A developer option can force this on, although this is not recommended. Issue #241.
- `hkicker` and `vkicker` strength can be specified via the magnetic field $B$ instead of `hkick` or `vkick`.
- Support for dipole poleface curvature in tracking.
- Pole face rotations and fringe fields are now available for hkickers and vkickers, both thick and thin.
- New ability to specify the colour of any magnet and most elements through custom colour definition.
- Geant4’s DNA physics lists have been added.
- Solenoid fringe fields have been implemented and are on by default. They are controlled with the `include-FringeFields` option.

16.4.3 General

- `outerDiameter` is now `horizontalWidth` to better describe its function (backwards-compatible).
  This naming was from a time when BDSIM could only create cylindrical magnets and beam pipes. Given it can now create more complicated geometry, this name is not a good choice and so has been renamed. BDSIM still supports the old syntax.
- `vhratio` is now consistent with vkickers and refers to the vertical and horizontal ratio in the lab frame.
- The horizontal width of kickers is now taken from `outerDiameter`. Previously, `outerDiameter` corresponded to the height and `vhratio` was really the horizontal-to-vertical ratio in the lab frame.
- Synchrotron radiation is now disabled with em_extra physics list (use dedicated synchrad physics list). Avoids the double registration of the physics process.
- New CMake variable `ROOTSYS` to allow easy specification of a specific ROOT installation.
- Visualisation of trajectories significantly faster (~10x) due to different strategy with Geant4 visualisation system.
- “ang” unit is added to the parser for Angstroms.
- BDSIM will now exit if there is no space to make the necessary circular management objects (teleporter and terminator).
- `long int` used explicitly instead of `int` for event indices in analysis.
- Reimplemented primary first hit and last hit. Last hit is now the end point of the primary trajectory. No more linear memory usage with tracking time.
- Beam pipe extent calculation re-implemented and much less simplistic - used to check whether a pipe will fit inside a magnet.
• Mini-contents for syntax section of manual, as it’s grown to a large size.
• New rmatrix element (experimental).
• EM Dissociation is now applicable up to 100 TeV.
• Significantly improved aperture shape checking for whether beam pipe will fit inside a magnet.
• BDSIM now recognises all elements by chemical abbreviation. These are found in the Geant4 NIST database by automatically prefixing the name with “G4_”. Issue #236.
• circle distribution z and t distributions are now reversed to be consistent with all other generators. These were T0 - dt and are now T0 + dt. Distribution will be different for the same seed as compared to a previous version of BDSIM.
• square distribution now calls random number generator for each coordinate every time for consistency. Distribution will be different for the same seed as compared to a previous version of BDSIM.
• Memory usage for sampler hits has been significantly reduced with no affect to the output information stored.
• The “water” material in BDSIM is now the NIST G4_WATER material and no longer the one that was defined by BDSIM.
• New options for physics processes in em_extra.

16.4.4 Output Changes

• Data v3 incremented from v2.
• Options class (GMAD::optionsBase) number is incremented in output.
• New optional stepLength variable in Eloss part of Event Tree with option storeElossStepLength to use this data.
• New optional preStepKineticEnergy in Eloss part of Event Tree with option storeElossPreStepKineticEnergy to use this data.
• Energy Loss class (BDSOutputROOTEventLoss) number is increment in output.
• Tilt, offsetX, offsetY and material are added to the Model Tree output.
• Model class (BDSOutputROOTEventModel) number is incremented in output.
• Model information extended to provide everything necessary to make machine diagrams.
• New option storeModel to turn off model storage in the output.
• Even Info class (BDSOutputROOTEventInfo) number is incremented in output.
• Event.Info now has a Boolean of whether the primary particle hit something or not.
• Samplers are no longer placed next to elements with angled faces when using the bdsimtwo, geant4, or geant4dp integrator sets.
• Units are now written to the ASCII survey output for each column.
• New output class BDSOutputROOTEventCoords to store coordinates for primary global coordinates.
• New branch called “PrimaryGlobal” in Event tree that stores the coordinates used with Geant4 in the global Cartesian frame.
• Sampler name now stored in Orbit output from rebdsimOrbit to make sampler matching possible.
16.4.5 Bug Fixes

- Fixed solenoid tracking. The anti-spiralling code in the dipole integrator that is designed to stop infinite spiralling of low energy particles in strong fields was causing incorrect tracking in solenoids. This has been fixed with the re-implementation of the solenoid matrix and now includes the fringe effects. Issue #255.
- Fixed tracking bug where particle in very niche coordinates may reflect from a sampler at the end of a dipole with a very strongly angled pole face. #Issue 241.
- Fixed automatic tunnel building algorithm, which accumulated wrong variables, leading to problems when thin elements such as fringe fields or thin multipoles were included.
- Further improvements made to tunnel building algorithm for magnets with tilt. Issue #243.
- Fixed length check for very short elements. Small drifts would cause a crash from Geant4 due to invalid parameters - occurred as length check was wrong.
- Fixed non-positive definite warnings for no energy spread and time spread when using a Gaussian beam in a composite beam definition.
- Fixed Gauss beams used in composite distribution.
- Fixed the problem where no particles were being tracked when using a userfile bunch distribution with only one column.
- Fixed bug where last particle was missed from user bunch distribution file.
- Fixed corrupted example files for userfile bunch distribution. Issue #244.
- Fixed cutting planes in G4CutTubs constructor for tunnel in Geant up to Geant4.10.2.p02 from too short tunnel section.
- Reimplemented the method of finding primary first and last hit so BDSIM doesn’t need to save the whole trajectory for the primary. This fixes the behaviour of linearly growing unbounded memory usage when tracking for a long time in a ring. Issue #246, #242.
- Optical calculation now works for sub-relativistic positrons.
- ATF2 MAD-X output was not included in worked example as advertised - now included.
- Fixed scaling variable used when scaling a field map to a decapole magnet strength.
- Survey units for magnetic fields are now fixed from kT to T.
- Fixed issue where C-shaped vkickers and hkickers would ignore yokeOnInside. Issue #251.
- Fixed possible overlap in vkicker, hkicker, and h-style dipole geometry with highly asymmetric beam pipes.
- Fixed incorrect report that beam pipe wouldn’t fit in magnet for various aperture shapes. Issue #253.
- Fixed issue where the option writePrimaries = 0 would result in the hits for the first sampler being written to the primary sampler structure. Issue #245.
- Fixed lack of interaction with vacuum when processes biased - due to a specific Geant4 version. Issue #220.
- Fixed incorrect dipole scaling. Issue #239.
- Fixed E0 spread in ring beam distribution, which was finite in contrast to the description that it is always the central value.
- Fixed reproducibility for the ring distribution that didn’t use the same random number generator as every other distribution. Coordinates will be different for this distribution for the same seed now.
- Fixed inconsistency of t and z coordinate in square beam distribution.
- square beam distribution now varies with envelopeT.
- Fixed S coordinate in output. Issues #247 and #248.
- Fixed the setting of the sampler diameter where the user specifies a smaller one than that calculated from the minimum bending radius.
16.4.6 Utilities

- pybdsim v1.9.0
- pymadx v1.5.0
- pymad8 v1.4.1
- pytransport v1.2.1

16.5 V1.1 - 2018 / 05 / 23

16.5.1 New Features

- New visualiser command “/bds/beamline/goto name” to move visualiser to view a particular element.
- Check explicitly on incompatible physics lists that may cause a crash when used together.

16.5.2 General

- Global coordinates are now always stored for primary first hit.

16.5.3 Bug Fixes

- Fixed wrong transforms for finite $S0$ in composite beam distribution.
- Fixed crash when finite $S0$ was used with –generatePrimariesOnly executable option.
- Fixed units from mm to m for PrimaryFirstHit and PrimaryLastHit for $x$, $y$, $z$, $X$, $Y$, $Z$ positions.
- Fixed segfault for double deletion when ‘qgsp_bic’ and ‘qgsp_bert’ were attempted to be used together.

16.5.4 Utilities

- pybdsim v1.6
- pymadx v1.2
- pymad8 v1.1
- pytransport v1.0

16.6 V1.0 - 2018 / 04 / 10

16.6.1 Highlights

- Full support for dipole fringe fields and pole faces in tracking.
- Full low energy (sub-relativistic) tracking.
- Validation against PTC for sub-relativistic to high energy.
16.6.2 New Features

- Support for Geant4.10.4 - however, this version is unusable as G4ExtrudedSolid is broken and used in BDSIM. We recommend Geant4.10.4.p01.
- H-style dipoles controllable by default or per element with hStyle option.
- Control over dipole proportions with global and per element options vhRatio, coilWidthFraction and coil-HeightFraction.
- New options neutronTrackingTime, neutronKineticEnergyLimit and useLENDGammaNuclear (Geant4.10.4 onward) physics options.
- Support for new numerical integrator tracking algorithms in Geant 4.10.3 and 4.10.4.
- New integrator set “geant4dp” for Dormand Prince integrators (Geant 4.10.4 or higher required).
- Significantly improved analysis documentation.
- New component: ‘gap’. Creates a space in the beam line with no geometry.
- Ability to specify the world volume material with the worldMaterial option.
- Introduced minimumRange and minimumKineticEnergy user limits as provided by G4UserLimits.
- Ability to limit step size in user-specified fields.
- Ability to control turn number print out with printFractionTurns.
- Magnet yokes now have a general multipolar field.
- Sampler diameter is automatically reduced when high angle bends are used to prevent overlaps.
- New CMake option to disable event display for installation with ROOT EVE problems.
- Ability to combine rebdsim output files with correct statistical analysis, allowing high throughput analysis with scaling. New tool rebdsimCombine for this purpose.
- Parameter tests for all bunch distributions - BDSIM will exit with message if parameters are invalid.
- scaling parameter for each element allowing simple scaling of strengths.
- New program to convert PTC output in a TFS file to BDSIM output format (ptc2bdsim).

16.6.3 Output Changes

- The ROOT class definition has been incremented to “2” from “1”.
- Output files now have header structure with software versions.
- Output files now have ‘geant4 data’ that includes particle masses used in the simulation.
- “t” is now “T” in samplers and trajectory output to be consistent with naming convention of global and local coordinates.
- Samplers now have optional charge, mass, rigidity and ion A and Z (see Output Options for details).
- Classes in library can calculate on-the-fly for user analysis.
- Trajectory momentum is now in GeV.
16.6.4 Analysis Changes

- “librebdsimlib” has been changed to “librebdsim” to be more consistent for output loading.
- Support for logarithmic binning of histograms in rebdsim.
- “HistogramND” in rebdsim now creates **per-entry** histograms on the tree. This introduces the ability to create per-event histograms in analysis that were not previously possible. Older style histograms that are a sum across all events are now made with “SimpleHistogramND”.
- New option in rebdsim to turn off histogram merging (for speed).
- Analysis classes have member names changed to match those in the output files, i.e. “eloss” is now “Eloss” in *bdsim/analysis/Event.hh*.
- Rebdsim is now tolerant of a missing ‘.’ on the end of the Tree name (a ROOT subtlety).
- ‘orbit’ and ‘optics’ are now ‘Orbit’ and ‘Optics’ in the output.
- New executable option for *rebdsimOptics* --emittanceOnFly to calculate emittance at each sampler.

16.6.5 General

- Physics list names are now consistently named with ‘_’ between words. Old list names are still supported.
- *hadronic* and *hadronic_hp* physics lists have switched from *qgsp_bert* and *qgsp_bert_hp* to *ftfp_bert* and *ftfp_bert_hp* respectively, as these are recommended by Geant4 for high energy hadronic interactions.
- “bdsim” integrator set now maps to “bdsimtwo” integrator set.
- All objects in the parser can now be extended later rather than just elements.
- Tuned colours of hkicker and vkicker.
- Relative file paths are no longer padded excessively with slashes when translated to absolute paths.
- More efficient file IO in rebdsim improves analysis speed.
- The world material is now air instead of G4_Galactic.
- *printModuloFraction* is now *printFractionEvents*.
- *includeIronMagFields* option is now *yokeFields*.
- Vacuum volumes now visible in --vis_debug mode.
- Only forward moving particles are now considered for the optical function calculation.
- Updated examples for ATF2, LHC. New Diamond machine example.

16.6.6 Bug Fixes

- Fixed magnetic field strength for AWAKE dipole using pure dipole field.
- User limits are now applied to external geometry.
- Fixed bug where some visualisation settings wouldn’t be applied to all logical volumes in external geometry.
- Fixed bug where some file paths may not be translated to absolute paths correctly.
- Fixed a bug where recreate mode would fail with the new Beam structure in the output.
- Prevent segfault when analysing wrong type of file with rebdsim (use ‘backwardsCompatible’ option to analyse old files without the new header structure).
- The *stopTracks* option has been removed as it did not function as intended. Use *stopSecondaries*.
- *thresholdCutCharged* and *thresholdCutPhotons* have been removed as they did not function as intended.
• Fixed bug where the world size would be too small with placements of large externally provided geometry with strong rotations.

• When no energy was specified in the userfile bunch distribution, the central beam energy was 1000x too low - fixed - missing factory of GeV.

• Number of turns simulated is now number desired and not number+1.

• Limits to control maximum step length and tracking time were not attached to magnet yoke geometry.

• Fixed crash when using octagonal or racetrack apertures in a magnet with pole geometry.

• Fixed issue where ~10% of particles may appear offset in samplers in large ring models.

• Fixed some very small overlaps not reported by Geant4 in magnet yoke geometry.

• Fixed issue where drift in magnet was always 1pm shorter than expected.

• Fringe fields are no longer built in between two dipoles against each other.

• Fixed Gauss Twiss bunch generator for finite dispersion in \( x, xp, y, yp \).

• Fixed bug where overlapping dipole end pieces would be produced.

• Fixed GDML preprocessing for parameterised variables.

• Tracking limits are now attached to magnet yokes.

• Fixed central value of \( T0 \) not being set for circle, gauss, gausstwiss, gaussmatrix, halo and square distributions.

16.6.7 Utilities

• pybdsim v1.4
• pymadx v1.1
• pymad8 v1.0
• pytransport v1.0

16.7 V0.993 - 2017 / 12 / 11

16.7.1 New Features

• Executable option –writeseedstate is now –writeSeedState.

• Executable option –exportgeometryto is now –exportGeometryTo.

• Executable option –distrfile is now –distrFile.

• Redefined simpler syntax for halo distribution. Please see manual for new parameters.

• Support for all EM physics lists included with Geant4.10.3.p03.

• Support for an ion as the primary beam particle.

• Support for ion physics lists.

• Ability to load two GDML files with degenerate object names correctly compensating for deficiency in Geant4 GDML parser.

• Local copy of GDML schema included. No longer require internet access to load GDML files. Custom local schema still supported.

• Support for Geant4.10.4 - however, we don’t recommend using this until p01 is used, as there is a fault with G4ExtrudedSolid that is used for all poled magnet geometry.
16.7.2 Bug Fixes

- Compilation fixes for compilers with XCode 9.
- Fixed possible compiler-dependent tracking bug where particle would get stuck in dipole.
- Cherenkov radiation in water fixed with specification of optical refractive index.
- Fixed ATF2 example input syntax and updated model.
- Removed temporary files created in current working directory.

16.7.3 Output Changes

- Options split into options and beam - beam contains all information related to beam definition.
- Associated output classes and analysis classes added for Beam in output.
- Removed older output format.

16.7.4 General

- Updated automatic Geant4 from AFS to version 10.3.p01 (latest available).
- Updated automatic ROOT from AFS to version 6.06.08.
- Remove support for ROOT v5 and require v6 onward.

16.7.5 Utilities

- Python utilities now use a setup.py compatible with PIP.
  - pybdsim v1.3
  - pymadx v1.0
  - pymad8 v0.8
  - pytransport v0.1 (new)

16.8 V0.992 - 2017 / 07 / 26

16.8.1 New Features

- Preparsing for GDML for pygeometry generated geometry - overcomes Geant4’s GDML parser deficiency of only allowing one GDML file to be loaded in the whole program.
- Visualisation of multiple beam lines.
- Option to use first sampler emittance as assumed emittance throughout lattice in optical function calculation.
- Additional materials for LHC tunnel and geometry.

16.8.2 Bug Fixes

- Fixed uncaught exception in analysis DataLoader class construction depending on optional arguments.
- BDSIM installation no longer dependent on build files - ROOT analysis dictionaries would use build headers. Issues #197 and #199.
- Fixed magnitude of B-field in rfcavity that resulted in extraordinarily strong B-fields.
- Fixed rf E- and pill-box fields ignoring phase and being a factor of \(2\pi\) too low in frequency.
- Fixed for crash when particle was at exactly 0 in a quadrupole.
- Fixed compiler warnings for Clang 8.1.
- Fixed all variable shadowing throughout codebase and reintroduced compiler warning if present.
- Fixed field transform for tilted and offset magnets. This fixes incorrect tracking for titled magnets when using the “bdsimtwo” integrator set for dipoles.

16.8.3 General

- Separated field and geometry in rf cavity element.
- Revised implementation of rf cavity construction for greater flexibility.
- RF cavity phase now automatically calculated based on location in lattice.
- Removal of old ROOT analysis scripts for very old data format.
- Revised construction to allow construction of multiple beam lines dynamically.

16.8.4 Utilities

- pymadx v0.9
- pybdsim v1.2
- pymad8 v0.7

16.9 V0.991 - 2017 / 07 / 04

16.9.1 Bug Fixes

- Fixed simple histogram weighting with Boolean expressions in rebdsim.
- Fixed comparator always failing.
- Fixed loading default event information in analysis.
- Fixed executing BDSIM from outside the directory containing the main gmad file.
- Fixed charge dependence of thin multipole and dipole fringe integrators.
- Scaled dipole fringe and thin multipole by momentum of particle.
- Fixed for loading geometry files prefixed with superfluous ‘.’.
- Fixed for duplicate run beam on icon in visualiser for Geant4.10.3 onward.

16.9.2 General

- Separated Doxygen cmake into own macro.

16.9.3 Utilities

- pymadx v0.9
- pybdsim v1.1
- pymad8 v0.7
16.10 V0.99 - 2017 / 06 / 09

16.10.1 New Features

- Parser will expand values from structures when printing. Issue #167.
- Optical physics example and test. Issue #156.
- Improved parser error messages. Issue #170.
- Support for compressed input coordinate files for beam distributions using tar and gz.
- Switch entirely to CMake labels instead of naming convention for tests.
- AWAKE experiment code refactored into module.
- New shield element that is a drift surrounded by rectangle of material.
- New placement element that allows an object with geometry to be placed independent of the beam line.
- maximumTrackLength option to limit any track in the geometry.
- Ability to offset beam line w.r.t. world coordinates at start.
- Check for required Geant4 environment variables.
- Thin horizontal and vertical (and combined) kickers.
- Thin multipoles.
- Compatibility with Geant4.10.3.

Fields & Integrators

- Complete refactorisation of field classes and construction.
- Centralised construction of fields.
- 1-4D BDSIM format field map loading.
- 2D Poisson SuperFish SF7 format field map loading.
- 1-4D nearest neighbour, linear and cubic interpolators for field maps.
- Support for compressed field maps using tar and gz.
- Ability to choose integrator sets for all elements via parser.
- Removal of all individual magnet classes - centralised construction in BDSMagnet.
- New executable - “bdsinterpolator” - allows loaded and interpolated field to be queried and written out.
- Rewritten dipole integrator using Geant4’s helical stepper.
- All integrators tested for low energy spiralling particles.
- Introduction of visualisation commands.

Geometry

- Rewritten external geometry loading.
- Ability to overlay externally provided geometry on magnets (except sbend).
- Automatically generated tight-fitting containers for externally loaded GDML geometry.
- circularvacuum beam pipe geometry that allows no geometry for the beam pipe; only vacuum.
- Magnet colours tweaked slightly - pybdsim now matches BDSIM colour-wise.
• Additional curvilinear bridge world to ensure continuous curvilinear coordinates.

Output & Analysis

• Protection against invalid sampler names that would cause ROOT branching errors.
• 1x 3D histogram in default output that can be placed along the beam line.
• Support for 3D histograms in rebdsim.
• All magnet strength components written out to survey.
• Change of syntax in rebdsim analysis file to specify dimensions of histogram.
• Stricter parsing of analysisConfig.txt for syntax checking.
• New executable rebdsimOrbit to extract single orbit from sampler data.

16.10.2 Bug Fixes

• ASCII seed state can be loaded properly. Issue #163.
• rfcavity can be created without a cavitymodel instance. Issue #165.
• Memory leak in comparator event tree comparison fixed. Issue #169.
• Zero angle bend with finite field can be created. Issue #176.
• Samplers are compared properly in comparator. Issue #177.
• Sampler names in Model tree now match those exactly in the Event tree.
• Missing virtual keyword from destructors fixed through to fix leaks at the end of the program.
• GFlash parameterisation is only loaded if specified in physics list.
• Fixed geometry construction errors that may occur due to dynamic tolerances for physically large models.
• Fixed infinite loop events if the primary vertex starts outside the world volume.
• Regions and biases set correctly to components in BDSLine class.
• Circle distribution did not have central value offsets.
• Fix double registration of pion decay as well as some others for muons when using muon physics list.
• Particles from physics list are now constructed correctly allowing more particles to be used in the beam definition.
• Removal of Cherenkov radiation from muon physics significantly reducing simulation time.
• Fix double registration of pion decay with muon physics list.
• Issue #134 - samplers cause tracking warning.
• Long running events due to spiralling particles. Issues #178, #132, #187.

16.10.3 General

• Sampler_ prefix was removed from all samplers in rootevent output.
• Sampler thickness reduced from 40 nm to 10 pm.
• Removal of unnecessary step length limit to half the length of an element.
• Revised region construction allowing arbitrary number to be constructed.
• Revised bend construction with reduced volume count in some cases.
16.10.4 Utilities

- pymadx v0.8
- pybdsim v1.0
- pymad8 v0.7

16.11 V0.95 - 2016 / 11 / 07

16.11.1 New Features

- Comparator program introduced for statistical comparison against reference results.
- rebdsim analysis examples and tests added.
- ROOT examples and tests added for analysis code usage in ROOT.
- Discrete optics only program (rebdsimOptics) added.
- Update CLHEP in AFS build to 2.3.1.0 for apple and 2.3.3.0 for RHL6.
- Reduced compilation time.

16.11.2 Bug Fixes

- Fixed geometry tolerance issue that would cause Geant4 run-time errors for regular geometry for some particularly large spatial size models.
- Fixed for linker error with ZLIB and gzstream. Issues #9, #155, #158.
- Fixed NaN errors in certain circumstances when calculating optical functions.
- Fixed shadowing compilation warnings.
- Fixed geometry overlaps in rf cavity geometry. Issue #136.
- Coverity fixes for uninitialised variables / small memory leaks. Issues #152, #156.
- Fixed potential magnet geometry errors when creating very thin components.
- Fixed negative interaction warnings due to biasing. Issue #141.

16.11.3 General

- Deprecate BDSIM Plank scattering, laserwire calorimeter and Bremsstrahlung lead particle biasing.

16.11.4 Utilities

- pymadx v0.7
- pybdsim v0.9
- pymad8 v0.6
- robdsim v0.7
16.12 V0.94 - 2016 / 09 / 13

16.12.1 New Features

Analysis

- Analysis class has been refactored to allow analysis on any tree in the BDSIM ROOT event output format using rebdsim.

Geometry

- Tilted dipoles are now supported.
- The Read-Out geometry has been moved to the Geant4 parallel world scheme.
- The parallel sensitive geometry for tunnel hits has been deprecated and the functionality now provided by the new parallel read-out geometry.
- The read-out geometry construction has moved out of BDSAcceleratorComponent to its own geometry factory.
- Beam pipes are now constructed with arbitrary 3-vector surface normals, rather than angled faces described by an angle only in the x-z plane.
- The side of the yoke of a dipole with poled geometry can now be controlled with the `yokeOnLeft` option.
- New interfaces to the auxiliary navigator have been written that use the mid-point of a step to much more robustly locate the required volume in the parallel curvilinear geometry.
- Overlap checking between adjacent dipoles with pole face rotations (and even tilts) prevents overlaps in geometry.

Output

- Tunnel hits are now of the same type as general energy loss hits.
- The track ID can now be optionally written out to energy deposition hits.

Parser

- The option `modularPhysicsListsOn` has been deprecated.
- New per-element parameter `yokeOnLeft`.

Physics

- The modular physics lists are now compulsory and the old physics construction has been deprecated.
- The existing BDSIM laserwire Compton scattering process construction was moved to the modular physics list scheme.

16.12.2 Bug Fixes

Geometry

- Extent inheritance with BDSGeometryComponent was fixed resolving rare overlaps.
- Poled geometry variable clean up fixed to ensure components from factories aren’t related to each other - could cause rare crash on exit.
• Fixed extents and possible overlaps in tunnel ‘rectaboveground’ geometry.

Output

• Energy deposition coordinates are randomly chosen along the step of the deposition. The x,y,z coordinates now match the random point, whereas they were the post step point previously.

Physics

• Fixed an issue that would cause infinite loops with strong process biasing.

General

• Initialisation of variables fixed throughout.
• Significantly improved compilation speed.
• Removed executable permission on all source files.

16.12.3 Utilities

• pymadx v0.6
• pybdsim v0.8
• pymad8 v0.5
• robdsim v0.7

16.13 V0.93 - 2016 / 08 / 24

16.13.1 New Features

Analysis

• New analysis tool ‘rebdsim’ replaces robdsim.
• Analysis directory with event, model, event info and options analysis.
• Histogram merging with correct statistical uncertainties.
• Deprecated root utilities to analysis/old/.
• Rewritten optical function calculation with validated calculation.

Build

• Require CMake 2.8.12 or higher.
• Require Geant4 compiled with external CLHEP - ensures strong reproducibility.
• Start of bootstrapping scripts in depend/.
• Factorisation of BDSIM’s cmake package finding into cmake/.
Geometry

- Coil geometry introduced to generic library magnets.
- Overlap checking between magnets with pole face rotations.
- Collimator colour can now be controlled.
- End pieces for coils also introduced
- Default poled dipole geometry is now a C-shaped magnet with yoke on inside of bend.

Output

- Switched to rootevent as default and **recommended** format.
- Include full set of options used in simulation in output.
- Include software version in output.
- Store seed state per event.
- Store histograms per event.
- Run and event durations stored in output.
- Output written in event of a crash.
- Refactor of trajectory information.
- Write out primary trajectory points.

Parser

- Factorised options into optionsBase that is simple structure for easy saving.

Physics

- “Modular physics” list is now the default.
- Use geant4 helper class for physics lists construction to ensure correct order.
- Ability to provide a default bias to all types of volumes (vacuum, accelerator, all).
- Attribute energy deposition (uniformly) randomly along the step where it occurred for more accurate energy deposition - currently only s, not x,y,z - they represent before, after.
- ‘solid’ air materials for cross-section validation.
- Seed states are saved and restored in the primary generator action rather than event action.

Tracking

- Geant4 Runge-Kutta stepper for quadrupole and sextupole for increased robustness.

General

- Strong recreation for an event by setting seed state issue (#118, #139).
- A BDSAcceleratorComponent can own an associated end piece(s) (before and after).
- A BDSAcceleratorComponent can have a input and output angled face.
- Halo bunch distribution developed significantly.
• Revised executable options for recreation / using a seed state.
• Signal handling improved.
• Templated user bunch file - can now use gzip compressed files.
• Improved default options for more realistic geometry.

16.13.2 Bug Fixes

Geometry
• Fixed loading for multiple GDML files having conflicting ‘world’ volumes.
• Reimplementation of pole geometry fixes gaps in poles (issue #110).

Parser
• Fixed issue of parser python interface (issue #133).

Physics
• Modular physics lists are truly modular - fixes segfaults (issue #130).

Tracking
• Fixes for cavity field values (issue #124).
• Fixed field value transform in sextupoles and above for global / local coordinates.
• Auxiliary navigator used more routinely for transforms with optional caching.

General
• Issues #115, #127, #129, #131

16.13.3 Utilities
• pymadx v0.6
• pybdsim v0.7
• pymad8 v0.41
• robdsim v0.7

16.14 V0.92 - 2016 / 03 / 29

16.14.1 New Features
• Samplers are attached at the exit instead of the entrance of an element.
• Poleface rotations for bends are implemented (issue #100).
• Geant4 9.6.x versions support has been dropped (issue #111).
• DUMP element removed (issue #116).
Geometry

- Samplers are no longer placed in the physical world but in a parallel sampler world.
- Above ground ‘tunnel’ geometry implemented.
- Introduced new RF cavity geometry and fields (still in development).

Output

- ROOT version 6 support (issue #114).
- Option to fill ROOT with double or float precision.

Parser

- Support for string variable (issue #126).

Physics

- Modular physics lists are default (issue #121).
- Use Geant4 provided synchrotron radiation instead of BDSIM one.

16.14.2 Bug fixes

Geometry

- Tunnel geometry fixes (issues #88 and #89).

Parser

- Fixed fast list insertion (issue #113).
- Support for tildes in path names (issue #119).

Physics

- Old physics code cleanup (issue #123).
- Physics biasing properly initialised (issue #84).

General

- PDF Manual builds on Ubuntu (issue #85).
- 1D Histogram class significantly faster for uneven bin width histograms.

16.14.3 Utilities

- pymadx v0.4
- pybdsim v0.5
- pymad8 v0.3
- robdsim v0.5
16.15 V0.91 - 2015 / 12 / 17

16.15.1 New Features

- New tests for file IO, coordinate transforms, aperture models, extra optical lattice patterns and general ring examples.

**Geometry**

- Race track and octagonal aperture models introduced.
- New wedged energy degrader component introduced.

**Output**

- Optional reduced number of variables in ROOT output - formats now “root” and “rootdetailed” (issue #107)
- Forced dependency on ROOT

**Parser**

- Parser warns for redefined variable, and exits for usage of undeclared variables (issue #98)
- Parser reorganised to C++ class structure (issue #77)
- Command line options more flexible (issue #105)
- Ability to define Geant4 regions as objects in parser.
- Can attach samplers to all elements of one type (i.e. collimaters).

**Physics**

- Can attach biasing to any part of any element from parser.

**Tracking**

- Ability to start bunch from any S-position along accelerator, rather than just at beginning.

16.15.2 Bug fixes

**Geometry**

- Fixed for LHC detailed geometry when beam shield is rotated.
- Consolidation and improvement of aperture parameter validity testing.
- Fixed for femtometre occasional overlaps in magnet outer geometry.
- Fixed placement overlaps in rbend.
- Fixed segfault with RfCavity at end of run.
- Fixed crashes with zero angle sector bends.
Parser

• Multiple command line arguments without space will now be recognised and highlighted.

Physics

• Made required version of Geant4 consistent across biasing code.

Tracking

• Fields only constructed if non-zero strength used - avoids tracking errors for zero strength components.
• Fixed several issues with vertical and horizontal kicker construction and tracking.
• Broken external magnet fields disabled by default.
• Circular turn counting bugs fixed.
• Particles no longer killed with circular flag on if starting slightly behind starting midpoint.
• Particles no longer stepped by teleporter at beginning of 1st turn if starting behind starting midpoint.
• Fixed teleporter tracking for backwards travelling particles that would get stuck in a loop.

General

• Add CMake protection against Geant4 built with multithreading on (issue #103).

16.15.3 Utilities

• pymadx v0.3
• pybdsim v0.4
• pymad8 v0.2
• robdsim v0.4

16.16 V0.9 - 2015 / 11 / 10

16.16.1 New Features

• Physics biasing with ability to change cross-section for individual particles and processes, as well as attach to a variety of objects
• Decapole magnet
• Robdsim analysis package as separate executable for testing
• Tracking tester
• Improved C++11 use and iterator implementation across containers
• Can fill histogram with energy hit over a range covering several bins
• Introduced a separate auxiliary G4Navigator to avoid accidentally moving the particle during tracking when querying global-to-local transforms.
• Transform for curvilinear coordinates to global coordinates so primaries in those coordinates can be injected from anywhere (issue #63)
• Parser put in GMAD namespace
- New executable options for writing out geometry coordinates as built by BDSIM
- Magnets now have tightly fitting invisible container volumes, as opposed to large boxes before.
- Changed return type of magnet outer geometry factories to new BDSMagnetOuter class. This is because the container construction is now delegated to the magnet outer factory for tight-fitting container volumes.
- Extended examples and tests
- Move entirely to Geant4 visualisation manager supporting all available visualisers available with the local Geant4 installation

### 16.16.2 Bug fixes

#### Geometry

- Fixed bug where the read-out coordinates would also be offset by the offset of the element.
- Fixed overlaps in read out geometry.
- Reduced duplication in magnet outer factories.
- Fixed overlaps in rbend geometry (issue #64).
- Increased tolerance for sector bends (issue #73).
- Protected against zero angle sector bends (issue #74).
- Fixed overlaps in GDML geometry (issue #81).
- Geometry fixes (issues #76, 94, 95)

#### Physics

**Parser**

- Occasional material parser segfault fixed (issue #25)
- Improved syntax checking and to not ignore unknown keywords (issue #71)
- Element extension fixed (issue #87)

#### Tracking

- Dipole uses local coordinates and can bend in any direction (issue #78)

#### General

- Samplers can be attached to occurrence of a duplicated element (issue #47).
- Output survey updated and fixed (issue #60)
- Check for Geant4 environment variables (issue #62)
- Consistent policy for overwriting output files (issue #65)
- Improved memory and CPU for output writing (issue #86)
16.16.3 Utilities

- pymadx v0.2
- pybdsim v0.3
- pymad8 v0.2
- robdsim v0.3

16.17 V0.8 - 2015 / 08 / 10

16.17.1 New Features

- Tunnel geometry and flexible tunnel factories for different styles
- Tunnel read out geometry introduced for coordinates along tunnel axis
- C++11 adopted (required)
- \textit{stopSecondaries} option
- Removed dependency on boost (issue #57)
- Restructured examples directory - top level contains only full machines and sub-directories contain features
- Example documentation in manual and in place beside each example with example screenshots
- Updated python utilities \textit{pybdsim v0.1, pymadx v0.1, pymad8 v0.1 and robdsim v0.2}
- Repeated components are not duplicated in memory - previously, they would be repeatedly constructed. Reduced memory footprint.
- Component information comes from Physical Volumes instead of Logical Volumes
- Improved manual documentation
- Improved Doxygen documentation
- Rubbish collection for all objects, rather than relying on only one run and Geant4 (partial) rubbish collection.
- String representation of enum types leading to more readable output
- Introduced ability to switch to new modular physics lists with flexible construction and addition of physics lists without hard-coded names for each combination - the user must turn this on explicitly

16.17.2 Bug fixes

Geometry

- Geometry overlaps (issues #55 and #58)
- Transform3d fix (issue #54)
- Fixed placement of objects outside x,z global plane - rotation bug, similarly for read-out geometry placement.
- Fixed broken circular control - bug was introduced in v0.7 - (issue #51).
- Strict checking of read out geometry construction to avoid invalid solids that would cause Geant4 to exit and BDSIM to crash.
- Strict checking on teleporter volume construction for circular machines that would cause Geant4 to exit and BDSIM to crash.
- Fixed calculation of length of sector bend magnet that would cause it to be slightly short - introduced in v0.7.
• Removed stored axes of rotation due to better implementation in BDSBeamline, avoiding duplication of information.
• Fixed issue of zero angle rbends causing a crash (issue #44).
• Event number print-out is now dynamic and based on the number of events to be generated. It is also controllable with the printModuloFraction option.
• Protected against bad user-specified values of lengthSafety to avoid geometry overlaps.
• Improved parser speed.

**Physics**

• SR radiation fixed in dipole (issue #53).
• Removed continuous synchrotron radiation, as it traps particles in low step-size infinite loop.
• Removal of poorly set deltaIntersection, chordStepMinimum and lengthSafety variables from examples - these should be left unset unless the user knows their purpose.

**Output**

• Changed all transverse output units to **metres** - manual updated accordingly.
• Changed z in ASCII output to **global Z** instead of local z.
• Recorded energy in output is now unweighted, but energy recorded in convenient energy loss histogram is. Could have lead to double weighting previously.
• Fixed global coordinates being written out as local coordinates in ROOT output.
• Random number generator seed state not written out when no output is specified.

**Parser**

• Return error if superfluous arguments are present (issue #56).
• Make parser more robust against duplicate element names (issue #43).
• Fixed warnings about compiling c as c++ being deprecated behaviour.

**General**

• Fixed wrong print out warning due to logic error (issue #51).
• Fixed boundary effects of energy deposition (issue #52).
• Fixed large memory leak for events with large number of particles - was due to accumulation of BDSTrajectory objects.

**16.18 V0.702 2015 / 07 / 28 - Hotfix**

• Fixes for physics production range cuts were not obeyed in simulation.

**16.19 V0.701 2015 / 07 / 02 - Hotfix**

• Fix for global X-coordinate not written to output for energy deposition.
16.20  V0.7 - 2015 / 06 / 30

16.20.1 New Features

- Ability to write no output
- New magnet geometry factories introduced with seven possible magnet types.
- Introduction of –vis_debug flag to see container volumes without debug build.
- Revised magnet colours (same base colour, just prettier variant)
- New manual using sphinx documentation system
- Default visualiser provided - no requirement for a vis.mac by the user
- Nicer visualisation GUI by default
- Improved visualisation for GDML geometry
- Support for all Geant4 visualisers introduced (issue #11).

16.20.2 Bug fixes

- Fixed overlapping volumes and tracking errors in beam pipes.
- Fixes for wrong transverse coordinates for geometry other than cylindrical magnets (issue #30).
- Histograms are now written to disk in case of crash or kill signal (issue #38).
- Fix for uncontrolled memory consumption for synchrotron radiation (issue #36).
- Fixed syntax error in parser on windows end of line character (issue #40).
- Follow user paths properly (issue #24).
- Parser can end on commented line (issue #41).
- Introduction of more flexible and weighted halo bunch distribution.
- Significant tidy of BDSAcceleratorComponent base class and derived classes.
- Fix LHC magnet geometry overlaps and improve efficiency as well as more flexible with different beam pipes.
- New BDSBeamline class used for component placement consistently in code.

16.21  V0.65 - 2015 / 04 / 10

- New base class for any geometrical object BDSGeometryComponent
- New interchangeable beam pipes with six possible beam pipe shapes
- New sensitive detector manager to hold single instance of sd classes
- Introduction of G4Galactic material for ‘empty’ volumes rather than beam pipe vacuum
- Possibility to write to multiple output formats at once
- Extensive removal of unnecessary headers throughout
- Updated python utilities
- Fix for muon spoiler magnetic field (thanks to B. Pilicer)
- Fix for invisible cylinder of iron surrounding drifts previously
16.22 V0.64 - 2015 / 02 / 16

- New histogram manager and factorisation of histograms from outputs
- Extra per-element histograms
- Basic implementation of valid solenoid

16.23 V0.63 - 2015 / 02 / 06

- Large angle sbends split into multiple sbends, based on aperture error tolerance - currently 1mm.
- New geometry construction and placement for sbends and rbends - no overlapping volumes and simpler / increased performance
- Proper building under c++11 if available
- Introduction of composite bunch distribution
- Drop support for Geant4 versions 9.5 and older

16.24 V0.62 - 2014 / 08 / 07

16.25 V0.61 - 2014 / 08 / 05

- Geant4 version 10 support

16.26 v0.6 - 2013 / 12 / 02
16.27 v0.5 - 2008 / 11 / 08
16.28 v0.4 - 2008 / 02 / 26
16.29 v0.3 - 2007 / 01 / 26
16.30 v0.2 - 2006 / 05 / 18
16.31 v0.1 - 2006 / 02 / 22
16.32 beta - 2005 / 05 / 01
17.1 Purpose of Developer Documentation

This section of the documentation explains the how and the why, rather than how to use BDSIM. It is a place where the choice of algorithms or the mathematics behind them can be explained. It also details how the function of BDSIM is implemented.

17.2 Style Guide

This section describes the agreed style and conventions used when editing and writing source code in BDSIM.

17.2.1 C++

General

• All variables shall be namespaced and not global.
• The asterisk is attached to the object, not the variable name.

```cpp
BDSClassName* anInstance;
```

Naming

• Underscores are to be avoided, as they are hard to type and read.
• HEADERGUARDS_H are like this
• Classes start with “BDS”.
• Classes use UpperCamelCaseForNaming.
• Member functions use UpperCamelCase as well.
• Member variables have no prefix (such as _variable or m_variable).
• Member variables use lowerCamelCase.
• Energy loss is “ELoss” in code and “Eloss” in options and output for backwards compatibility even this generally breaks our lower camel case rule.

Indentation & Spacing

• Two spaces per level of indentation.
• Tabs should not be used.
• Spaces should be between operators.

**Braces**

• Explicit braces should always be used, even for one line if statements.

```c++
if (a > 4)
    {G4cout << a << G4endl;}
```

• Braces should be on a new line - makes scope easier to determine.

```c++
if (a > 4)
{
    G4cout << a << G4endl;
    G4cout << "This is a test" << G4endl;
}
```

• The above style is preferred (indented block), but the following is also fine.

```c++
if (a > 4)
{
    G4cout << a << G4endl;
    G4cout << "This is a test" << G4endl;
}
```

**In-Code Documentation**

• Every single class should have doxygen documentation in the header.
• Obviously comments are strongly encouraged, as well as notes in this documentation.
• Avoid documenting the purpose of functions (i.e. outside the function) in the source code - document the header as that is the interface that people should use when developing.

**17.2.2 Python**

The Python modules are developed with the intention that they be used and discovered interactively in ipython, therefore, the naming convention described should make tab completion easy to understand and docstrings should allow the use of the ? for help on any class or object.

**General**

• Docstrings must be provided for all modules, classes and functions.
• General packages such as numpy should be imported in a hidden fashion by renaming.

```python
import numpy as _np
```

**Naming**

• Classes use UpperCamelCaseForNaming.
• Member functions use UpperCamelCase as well.
• Member variables have no prefix (such as _variable or m_variable).
• Member variables use lowerCamelCase.
17.3 Release Checklist

Things to update immediately before a release (i.e. from a release candidate branch):

1. README - update at the top and the version history.
2. CMakeLists.txt - change major, minor and patch version at the very top.

For each submodule:

- Update version in setup.py.
- Update version in setup.cfg.
- Update version in docs/source/conf.py in two places.
- Update version history in submodule manual.
- Generate submodule documentation (html + latexpdf); copy pdf to docs dir.
- Update main __init__.py / __version__ number to match.
- Commit pdf of documentation to each submodule.
- Tag submodule version.
- Upload submodule manual to website.
- Upload to pypi.

3. Tag submodule repository versions.
4. Update submodules in bdsim repository.
5. If the data format has changed increment the data version in output header structure.
   - Check data version in analysis.
   - Check data version in analysis/DataLoader.cc initialiser list.
   - Regenerate data samples in examples/features/data/.
   - Regenerate data sample examples/features/beam/userfile-sample.root.
6. Update version history (including submodule and data versions) in manual source.
   - Submodule versions.
   - Data version.
   - Data class versions.

7. If BDSColours has changed, run BDSIM with DEBUGOUTPUT build and copy print out of colours to manual (already in correct format).

10. In develop, put back README, CMakeLists.txt to new version.develop.
11. Merge release candidate branch into master then delete.
12. Check all tests complete locally given merge before pushing.
13. Tag master branch for version number.
17.4 Program Layout

BDSIM progresses in the following general steps:

1. Parser reads and parses input text files that prepare structures representing input - all in GMAD namespace.
2. A Geant4 model (geometry, fields, sensitivity) is created based on information from the parser structures.
3. Special Geant4 user actions are registered.
4. The simulation proceeds either interactively or automatically for a given number of events.
5. Some analysis is performed at run time, output is written and post-run analysis can be performed.

17.4.1 Parser

The parser uses Flex and Bison to automatically generate the C++ code required given a scripted syntax. This defines a set of rules for the syntax that can be matched.

The parser is described in much more detail in Parser.

17.4.2 Geant4 Model

The use of Geant4 falls into two general parts: the model and the user action classes. The model is built using the parser beam line structure through many factories. The details are described in Geometry, Fields, Tracking Algorithms, and Sensitivity, Output & Analysis.

17.4.3 Geant4 User Action Classes

Apart from the geometry and the fields, the Geant4 managed simulation is largely controlled through user action classes. Geant4 provides base classes (some virtual, others not) that the developer can register that allow actions to be taken at specific points, such as the beginning and end of event actions.

These are discussed in detail in Geant4 User Action Classes.

17.4.4 Analysis

Run time analysis is also discussed in the aforementioned Sensitivity, Output & Analysis.

Post run time (outside BDSIM) analysis is discussed in Analysis Suite.

17.4.5 Specific Run Time Order

This is a rough description of the first few actions in BDSIM (bdsim.cc) that have to be in this order.

1. BDSExecOptions
2. Check if Geant4 is available in the current environment (env variables checked) - cannot proceed without this.
3. Construct parser and parser input.
4. Construct required Geant4 materials, as they’re required by BDSGlobalConstants.
5. Force instantiation of BDSGlobalConstants singleton based on GMAD::Options.
6. Initialise the pseudo-random number generator engine.
17.5 Build System & Testing

17.5.1 Build System

The build system is based on CMake. The build options and variables are described in the user’s manual. Some additional options for developers:

- make dist: This archives the git repository in a file bdsim-0.9.develop.tar.bz2
- make copy-deps: This copies all dependent libraries to the build/devs/ directory
- BUILD_MACOSX_APP: CMake variable that builds an MacOS application at installation time

17.5.2 Test System

The test system is based on CTest, which works in combination with CMake. Tests are added to the CMakeLists.txt file beside each example declaring the main gmad file as well as any optional command line arguments.

Useful ctest commands (can all be combined):

- ctest -N: print tests without execution
- ctest -R <regexp>: execute tests that match regexp
- ctest -LE LONG: execute tests that are not LONG (more than about a minute)
- ctest -I 5,5: only execute test 5
- ctest -VV: print output to screen

For example, to find the test command for the sextupole component test:

cctest -R Component-Sextupole -VV -N

17.5.3 Test Server

The test server is based on CDash. Submissions to the BDSIM test server (only from inside the RHUL domain) can be done with instructions from the BDSIM-cdash repository.

17.5.4 Package System

The package system is based on CPack, which works in combination with CMake. One can package the binaries and libraries with cpack. A packed .tar.gz and a Mac OS .dmg is created. The source can be packed with make dist. It is recommended to pack the libraries after each release version and put the binaries and libraries on the BDSIM Download page.

17.5.5 Static Code Analysis

The code of BDSIM is analysed by the static analysis tool Coverity. Its analysis is available with CERN credentials at CERN’s Coverity site.

17.6 Parser

The parser for the GMAD language is independent from the rest of BDSIM and can be found in the parser directory. The main parser interface can be found in parser.h. The parser is currently a singleton (only one instance with global scope).
The GMAD language is a LALR (Look-Ahead LR) parser language generated by Bison (in the file `parser.y`) with a lexical analyser generated by flex (from `parser.l`). GMAD provides basic arithmetic and Boolean operators. GMAD also provides predefined units, constants in SI units and some common mathematical functions, like the trigonometric ones, by binding them to actual C-variables and functions. It has a global scope only.

**Note:** GMAD is designed to be an extension of the syntax used by MAD, which means that any MAD syntax should be supported.

### 17.6.1 Parser Classes

All options for a BDSIM run are contained in an instance of the Options class. This is passed to BDSIM. The Options class is built out of a struct-like `OptionsBase` and a layer of self-inspection provided by the `Published` class. The struct-like `OptionsBase` is needed to output the options to the `rootevent` output.

All other parser classes like `Element`, `Region`, etc. also have this layer of self-inspection.

The `Parameter` class is a temporary storage class of the parameters before the actual `Elements` are created.

The `enum ElementType` has a list of all elements currently in BDSIM.

The `Array` class is an array representation for `bison` and either strings or doubles.

The `python.h` provides the Python interface to the parser.

The `Symtab` class represents a parser variable. All variables are stored in a map.

### 17.6.2 Bison

This section gives a brief overview of Bison. For more comprehensive reading, a manual is recommended.

The `parser.y` file contains the typical four main sections:

```yacc
{%
C declarations
%
Bison token and types declarations
%%
Grammar rules
%%
Additional C code
%
```

The `gmad` keywords are translated to bison tokens in the `library` file `parser.l`.

#### C-declarations

The C-declarations are a few global variables.

#### Bison Tokens

Bison tokens (translated directly with the library) and types (more general variables) are from a union and these can be one of the following types:

- `double`
- `int` (for the enum class `ElementType`)
- `std::string*` (a pointer so its size can fit in the union; its memory is stored in the `Parser` class)
The union type of the tokens are defined in the Bison declaration section of `parser.y`, for example:

```
%token <str> STR
%type <dval> aexpr
```

`STR` is a token of type string, and `aexpr` is general number of type double.

Tokens can also have no value attached to it at all:

```
%token MATERIAL
```

### Grammar Rules

The grammar rules define a syntax tree. Bison is a bottom-up parser. It tries, by shifts and reductions, to reduce the entire input down to a single grouping whose symbol is the grammar's start symbol, which in our case is `input`:

```
// every statement ends in a semicolon
input :
  | input stmt ';'
```

This rule is split into two parts:

- Input can be empty (indicated by no text after the colon)
- It is a recursive rule, where it breaks the input into statements (`stmt`) ending with a semicolon.

A rule can be split into as many parts as possible.

Another example are the atomic statements (single lines without if constructs):

```
// atomic statements can be an mathematical expression, a declaration or a command
atomic_stmt :
  | expr { if(ECHO_GRAMMAR) printf("atomic_stmt -> expr\n"); }
  | command { if(ECHO_GRAMMAR) printf("atomic_stmt -> command\n"); }
  | decl { if(ECHO_GRAMMAR) printf("atomic_stmt -> decl\n"); }
```

The part inside the brackets is the actual C-code, which is only debug printout in this case. The rules for `expr`, `command` and `decl` are defined elsewhere.

Rules can be tokens and types as well, and can have a value. For example, the rule for addition looks like:

```
aexpr | aexpr '+' aexpr { $$ = $1 + $3;}
```

`aexpr` is a variable of type double. The rule reduces the syntax “number + number” to a single number. The new value (indicated with $$) will be the value of the first token ($1) plus the third token ($3). Note that the second token is ‘+’.

### Debugging

Since adding or changing Bison rules can often have unforeseen consequences, it is strongly recommended that when extending the GMAD language, first write a test case for it and check that it fails. There are many GMAD CMake tests in the `parser/test` directory.

Often the compiler will complain when the rules are inconsistent and the CMake tests cover many syntax cases which all should still work. For debugging there are several options in `parser.y`, all of which need recompilation:

- The variables ECHO_GRAMMAR and INTERACTIVE can be switched on for extra output.
• Compile Bison with “-t” flag. This is automatically done when CMAKE_BUILD_TYPE equals Debug.
• Uncomment the line with %debug. This will print out the token stack after each step.

17.7 Geant4 User Action Classes

Geant4 provides several base classes the developer can inherit, instantiate and register with the Geant4 run manager as a way of taking various actions defined by the developer at different stages of the simulation. These are described at Geant4 user actions. They provide actions at each level of granularity in the simulation. The actions BDSIM takes in each of these are described in the following sections.

17.7.1 G4UserRunAction

A ‘Run’ is the largest unit of simulation in Geant4 parlance. It is a simulation where all the geometry, fields and physics are fixed and the same throughout. It typically contains many ‘Events’.

BDSRunAction::GenerateRun

This is not used, as BDSIM does not define its own Run object and uses the G4Run one.

BDSRunAction::BeginOfRunAction

• This histograms for the simulation (Run) are prepared and registered with the BDSAnalysisManager.
• The start time is printed out.

BDSRunAction::EndOfRunAction

• The stop time is printed out.
• All histograms are written to the output.
• Clean up

17.7.2 G4UserEventAction

Warning: Note, the Primary Generator action is called before this method at the start of each event!

BDSEventAction::BeginOfEventAction

• Sampler hits collections are created and registered.

BDSEventAction::EndOfEventAction

• Primary vertex coordinates are recorded.
• Sampler, Cylindrical Sampler, Energy Deposition, Tunnel Energy Deposition hits are recorded in that order.
• Primary impact points and loss points are recorded.
• Output is written.
• Trajectories matching filtering criteria are prepared and written to output.
17.7.3 G4UserStackingAction

This allows the developer to fiddle the priority of particles (primary and secondary) based on their properties.

BDSStrackingAction::ClassifyNewTrack

• All neutrinos are killed by default to save time on tracking.
• If specified in GMAD::Options stop tracks, the following secondary particles are killed:
  – electrons, photons, positrons and protons/antiprotons
• If stop secondaries is specified, all secondaries are killed.
• If the track is beyond the limit of tracks per event, it is killed.
• If a particle is flagged as killed and it’s in a volume that has sensitivity an attempt is made to process the sensitivity by dynamically casting the sensitive detector to a variety of BDSIM ones.

17.7.4 G4UserTrackingAction

The tracking action happens when a track is created (a new particle) and when it is finished tracking.

BDSUserTrackingAction::PreUserTrackingAction

• Trajectories are stored if requested or as required for visualisation.
• If no trajectories are required (turned off and in batch mode) a special reduced trajectory is created for the primary particle only that marks its first impact and its loss location.

BDSUserTrackingAction::PostUserTrackingAction

• The primary particle loss point is flagged as in a collimator or not.

17.7.5 G4UserSteppingAction

Only UserSteppingAction is implemented and only to provide verbose output if required.

17.7.6 G4VUserPrimaryGeneratorAction

BDSPrimaryGeneratorAction::GeneratePrimaries

This uses a single particle gun and uses BDSBunch derived classes to generate various distributions of single primary particles.

Strong Reproducibility

Note, the primary generator action is the first thing to be called per event that advances the random number generator. Therefore, the seed state is saved and restored here. If strong reproduction is requested using the executable flag \(--recreate=<file>\), the options will be loaded from the output file, including the bunch distribution, and all will be recreated.
17.8 Geometry

In `bdsim.cc`, an instance of `BDSDetectorConstruction` is created and registered to the Geant4 run manager.

The geometry is dynamically built based on information from the parser instance. BDSIM is designed to build a model of an accelerator and, as such, creates a single beam line in order, element by element. Each element is created using a component factory (`BDSComponentFactory`) to instantiate the correct class and it is then placed in a holder (`BDSBeamline`) that calculates the cumulative coordinates of the element in the world given the already created ones. It also keeps track of the extent of the model. Optionally, the tunnel is built with respect to the beamline. Only after these stages can an appropriately sized world volume be created. Each element in the beam line is then placed into the world volume. Ultimately, the fully constructed world volume (and therefore all of its contents - the accelerator model) is returned to Geant4, which then handles it for the simulation.

17.8.1 Beam Line Calculations

As well as being a vector of the beam line elements, when each `BDSAcceleratorComponent` is added to the beam line, the coordinates that should be placed in the world that represent that element’s position in the beam line are calculated. The rotation matrices and positions for the beginning, middle and end are stored along with the `BDSAcceleratorComponent` instance in a `BDSBeamlineElement` instance. A further subtlety is that any one element can be offset or tilted with respect to the accelerator curvilinear reference (‘design’) trajectory, so the originals are stored under the name ‘reference’ and the final positions (incorporating tilts and offsets) are recorded without name.

Assumptions About Geometry

The coordinate calculation is simplified to a degree, with a few basic assumptions about how any one component affects the reference (‘design’) trajectory.

- A `BDSAcceleratorComponent` advances the reference trajectory by a length \( l \).
- A `BDSAcceleratorComponent` may change the outgoing angle of the reference trajectory by angle \( \alpha \) in the horizontal \((x - z)\) plane and this is assumed to be a single smooth change.
- Any offset in the reference trajectory at the end of a `BDSAcceleratorComponent` is due to the change in angle through the component.
- It is not possible for the outgoing trajectory to be offset but with zero angle, i.e. a slalom or S shape.

A Few Important Points

- Geant4 uses the right-handed coordinate system.
- Euler angles are used to rotate frames of reference and offsets are applied first.
- \( l \) is not used for length in the code - only `chordLength` or `arcLength` to be explicit.
- The chord length and arc length are supplied or calculated in `BDSAcceleratorComponent`.

A schematic of the chord and arc length for a `BDSAcceleratorComponent` with a finite bend angle is shown below.

17.8.2 Component Factory

17.8.3 Beam Pipe / Aperture Factories

17.8.4 Magnet Factories

The magnet geometry is built in factories with virtual base class `BDSMagnetOuterFactoryBase`. Many factories inherit this by implementing the virtual methods (one for each magnet type) and provide various styles of
magnet geometry. In this way, a new magnet style can be added easily, or a factory made that mixes and matches others by calling other factories. All factories are singletons as there need only be one of them - although this isn’t strictly required.

17.8.5 Angles of Bends, and Faces

Bending Angle Convention

The two images below show the direction of bending for positive and negative angles:

- A positive angle will bend toward negative X in the x-z plane (local coordinates).
- A negative angle will bend toward positive X in the x-z plane (local coordinates).
- Note: X-axis is red, z-axis is blue, y-axis is green and points towards you.

Angles of Rectangular Bend Faces

To accommodate both normal bends and those with pole face rotations, the angle of the input face and the angle of the output face are specified. If no pole face rotation angles are specified, half the bend angle is given as the face angles for sbends. For rbends without pole face rotation, the end faces of the magnet are parallel, therefore the half-bend angle is instead applied to the elements preceding/succeeding it.

A sequence of consecutive rbends can also be defined, however, rather than split up a single magnet into multiple segments, the result would be similar to the sequence shown in the figure below.

To split an rbend into multiple segments to create a straight final magnet, you would require an indefinite look ahead (beyond the current one element look ahead), to determine the total length and angle. This would then be followed by a rotation of each segment and a lateral offset to form the line. The current implementation would become more prominent for a larger total angle (especially if the magnet length was short), however, given the rarity of this, the current method can suffice for now.
Angles of Sector Bend Faces

Sbends can be easily broken up into smaller consecutive sbends if needed. If multiple sbends are defined as such, the input pole face angle ($e_1$) for an sbend must be -1 times the output pole face angle ($e_2$) of the previous sbend. This is purely to avoid overlaps between elements. This doesn’t apply to the input angle and output angle of the first and last sbends respectively; these are effectively the pole face rotations for the whole sequence.

Irrespective of any splitting from the user, all sbends are split into an odd number of segments. This is calculated in `CalculateNSBendSegments`. Each segment has the equal length along the reference trajectory, and the number of segments it is split into is determined by the angle and length of the whole magnet. Shown in the figure below is a diagram of the reference system for pole face rotations with an example sbend. Two-thirds of the sbend segments are shown as partially transparent to highlight the changes in the face angles.

When there is no pole face angle specified, each sbend segment will have the same input and output angle of 0.5 times the total bending angle, divided by the number of sbends. With a finite pole face angle(s), the input and output face angle of each segment increases or decreases as appropriate from the first wedge (with the user specified $e_1$) until the central wedge is reached. (This is why the number of sbend segments must always be odd, as the angle algorithm always works towards/away from a central wedge). This central wedge has the face angles equal to that if no pole face angles were supplied. From the middle wedge, the face angles are then increased/decreased as appropriate until the final wedge is created with the user specified $e_2$.

There are multiple reasons for this implementation. Without the change in face angle for each segment, if a large $e_1$ is specified when the length of each segment is short, the projected length of the first segment would overlap with the next segment, as indicated by the red triangle in the left figure below. Another reason is that each segment has to be rotated slightly in order for them to sit correctly on the sbend reference trajectory. As such, when a non-zero pole face angle is specified, the input face angle of a segment cannot be the opposite sign of the output face angle of the previous element. Therefore the input and output faces have to be increased/decreased differently.
In certain circumstances, the situation may occur where the angles of both input and output face angles are such that they cause the faces to intersect within the magnet radius. For any segment that is created, the radial distance where the faces overlap is compared to the magnet radius, and exits if it is larger. (This check is performed to avoid a Geant4 exit with unclear errors). The radius is calculated in `CalculateFacesOverlapRadius` in `BDSUtilities`. This is outlined in the above right figure. It works by taking the input and output face angles, and calculating their normal vectors (green arrows in the above diagram). These are then rotated as appropriate so that both unit vectors are in the planes of their respective faces (red arrows). The vector to where these two lines intercept is then calculated (black arrow), and the x-component taken as the radius. It should be noted that for non-cylindrical magnet geometries, the limit for the interception radius is (arbitrarily) 1.25 times smaller. This is due to their geometries being transversely smaller than their cylindrical container volumes.

**Poled Dipole Geometry Code Conventions**

The C++ code to generate the poled dipole geometry is fairly complex by necessity. The code exists in `BDSMagnetOuterFactoryPolesBase.cc` and what common code there is is grouped together. The code must produce both H- and C-style dipoles with a variety of scalings and optionally with poles and coils if there is room. When poles are created, the shapes are typically created by creating a vector of 2D boundary points for an extruded solid.

In the case of vertical kickers, this geometry should be rotated. We cannot simply rotate the whole magnet, as the transforms for a field would be incorrect. However, the points for the extruded solid can be generated as normal and then a 2D rotation applied before construction of the `G4ExtrudedSolid`.

For sanity, the calculations for the various parameters and whether or not the poles and coils will be built are performed **always in the horizontal**. So, even for a vkick, the geometry is constructed as a horizontal one. The `vhRatio` is inverted in this case, `outerDiameter` is always the horizontal width.

The following diagram illustrates the variable meaning for the calculation. Only the horizontal case is shown, as the calculations are only performed in the horizontal orientation.

**17.8.6 Specific Element Details**

**Rectangular Bend**

**17.9 Tracking Algorithms**

**17.9.1 Background On How Geant4 Tracking Works**

Geant4 provides the ability track a variety of particles through space in the 3D geometry model. Importantly, it provides the ability to track the motion of particles in electromagnetic fields. As Geant4 provides a large library of particle types, the tracking software must be capable of tracking particles of different masses and charges.

This is achieved by factorising certain functionality into different sets of C++ classes. The user must provide a function that will return the electric and magnetic field vectors for a given set of \((x, y, z, t)\) coordinates.
Specifically, this is a class that inherits \texttt{G4Field} and provides an implementation of the pure virtual method \texttt{GetFieldValue(position)} where \texttt{position} is \(x, y, z, t\).

As the user specifies the field (and it is essentially unknown), numerical integration techniques must be used to solve the equation of motion to calculate the trajectory of a given particle. Geant4 provides a variety of different numerical integrators that offer various capabilities and trade-offs in accuracy and computational speed. The field and numerical integrator classes are combined with a few other necessary Geant4 classes to create a complete “field” capable of calculating the trajectory of a particle that would represent the physical motion in the given field. After this, the complete field may be attached to a \texttt{G4LogicalVolume} instance. A logical volume has not just a shape, but also material, colour, field, sensitivity etc. Even though only one logical volume object may be created, it may placed multiple times in the 3D geometry model.

As described in \textit{Fields}, BDSIM provides a variety of C++ classes that represent typical accelerator (pure) magnetic fields. These can be attached to the relevant vacuum volumes in the geometry along with Geant4 numerical integrators to achieve particle tracking in an accelerator.

**Linear Fields**

For both a uniform dipole field and quadrupolar field (linear fields), there exist analytical solutions to the equations of motion. These solutions provide a more accurate representation of the particle’s motion in the field and may offer significant computational advantage over numerical integration techniques. Primarily, for reasons of accuracy, these are provided in BDSIM for the dipole and quadrupole.

**Non-Linear Fields**

BDSIM provides an integrator for higher-order fields that more accurately conserves energy when calculating the particle trajectory (symplecticity) as well as being competitive computationally. The routine provided is a second-order Euler integration algorithm. More will be added in future.

**17.9.2 Integrator Sets**

When using BDSIM, the user can select a set of integrators (i.e. tracking routines) with the following syntax:

\begin{quote}
\texttt{option, integratorSet=“bdsimtwo”;
}\end{quote}

This choice affects the computation time and accuracy of the simulation, but each set may be suited to different scenarios. As more integration algorithms are added to BDSIM, more sets can be added that mix and match routines as required.

- “bdsimmatrix” is the default and recommended.
- “bdsimtwo” is acceptable if no fringe dipole pole faces are present.

All sets apart from “geant4” make use of custom BDSIM integrators for accelerator tracking. These integrators ignore the supplied field and use a strength parameter (such as \(k1\) for a quadrupole) instead. The field is always present and should a backwards or non-paraxial particle be used, these BDSIM integrators resort to a \texttt{G4ClassicalRK4} algorithm. This allows all particles to be tracking in all directions over all momentum ranges, but with the accuracy and speed of accelerator tracking for paraxial particles.

The specific details are described in Integrator Algorithms.

The integrator set may be one of the following (case-insensitive):
### Magnetic Field Type and Integrator Settings

<table>
<thead>
<tr>
<th>Set</th>
<th>Magnetic Field Type</th>
<th>Integrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdsimmatrix</td>
<td>Solenoid</td>
<td>BDSIM Solenoid</td>
</tr>
<tr>
<td></td>
<td>Dipole</td>
<td>BDSIM Dipole Matrix</td>
</tr>
<tr>
<td></td>
<td>Dipole with K1</td>
<td>BDSIM Dipole Matrix</td>
</tr>
<tr>
<td></td>
<td>Quadrupole</td>
<td>BDSIM Quadrupole</td>
</tr>
<tr>
<td></td>
<td>Sextupole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Octupole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Decapole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Thick Multipole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Muon Spoiler</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF Cavity (EM)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF (E only)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>General Default</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Quadrupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Sextupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Octupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Decapole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Dipole Fringe</td>
<td>BDSIM Dipole Fringe</td>
</tr>
<tr>
<td></td>
<td>Thin Multipole</td>
<td>BDSIM Thin Multipole</td>
</tr>
<tr>
<td></td>
<td>Multipole Outer</td>
<td>G4NystromRK4</td>
</tr>
<tr>
<td></td>
<td>Thin RMatrix</td>
<td>BDSIM Thin RMatrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Magnetic Field Type</th>
<th>Integrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdsimmatrixfringescaling</td>
<td>Solenoid</td>
<td>BDSIM Solenoid</td>
</tr>
<tr>
<td></td>
<td>Dipole</td>
<td>BDSIM Dipole Matrix</td>
</tr>
<tr>
<td></td>
<td>Dipole with K1</td>
<td>BDSIM Dipole Matrix</td>
</tr>
<tr>
<td></td>
<td>Quadrupole</td>
<td>BDSIM Quadrupole</td>
</tr>
<tr>
<td></td>
<td>Sextupole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Octupole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Decapole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Thick Multipole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Muon Spoiler</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF Cavity (EM)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF (E only)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>General Default</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Quadrupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Sextupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Octupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Decapole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Dipole Fringe</td>
<td>BDSIM Dipole Fringe Scaling</td>
</tr>
<tr>
<td></td>
<td>Thin Multipole</td>
<td>BDSIM Thin Multipole</td>
</tr>
<tr>
<td></td>
<td>Multipole Outer</td>
<td>G4NystromRK4</td>
</tr>
<tr>
<td></td>
<td>Thin RMatrix</td>
<td>BDSIM Thin RMatrix</td>
</tr>
<tr>
<td>Set</td>
<td>Magnetic Field Type</td>
<td>Integrator</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>bdsimtwo</td>
<td>Solenoid</td>
<td>BDSIM Solenoid</td>
</tr>
<tr>
<td></td>
<td>Dipole</td>
<td>BDSIM Dipole Rodrigues 2</td>
</tr>
<tr>
<td></td>
<td>Dipole with K1</td>
<td>BDSIM Dipole Matrix</td>
</tr>
<tr>
<td></td>
<td>Quadrupole</td>
<td>BDSIM Quadrupole</td>
</tr>
<tr>
<td></td>
<td>Sextupole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Octupole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Decapole</td>
<td>BDSIM Euler</td>
</tr>
<tr>
<td></td>
<td>Thick Multipole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Muon Spoiler</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF Cavity (EM)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF (E only)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>General Default</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Quadrupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Sextupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Octupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Decapole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Dipole Fringe</td>
<td>BDSIM Dipole Fringe</td>
</tr>
<tr>
<td></td>
<td>Thin Multipole</td>
<td>BDSIM Thin Multipole</td>
</tr>
<tr>
<td></td>
<td>Multipole Outer</td>
<td>G4NystromRK4</td>
</tr>
<tr>
<td></td>
<td>Thin RMatrix</td>
<td>BDSIM Thin RMatrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Magnetic Field Type</th>
<th>Integrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>bdsim</td>
<td>Solenoid</td>
<td>BDSIM Solenoid</td>
</tr>
<tr>
<td></td>
<td>Dipole</td>
<td>BDSIM Dipole Rodrigues</td>
</tr>
<tr>
<td></td>
<td>Quadrupole</td>
<td>BDSIM Quadrupole</td>
</tr>
<tr>
<td></td>
<td>Sextupole</td>
<td>BDSIM Sextupole</td>
</tr>
<tr>
<td></td>
<td>Octupole</td>
<td>BDSIM Octupole</td>
</tr>
<tr>
<td></td>
<td>Decapole</td>
<td>BDSIM Decapole</td>
</tr>
<tr>
<td></td>
<td>Thick Multipole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Muon Spoiler</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF Cavity (EM)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF (E only)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>General Default</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Quadrupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Sextupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Octupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Decapole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Dipole Fringe</td>
<td>BDSIM Dipole Fringe</td>
</tr>
<tr>
<td></td>
<td>Thin Multipole</td>
<td>BDSIM Thin Multipole</td>
</tr>
<tr>
<td></td>
<td>Multipole Outer</td>
<td>G4NystromRK4</td>
</tr>
<tr>
<td></td>
<td>Thin RMatrix</td>
<td>BDSIM Thin RMatrix</td>
</tr>
<tr>
<td>Set</td>
<td>Magnetic Field Type</td>
<td>Integrator</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>geant4</td>
<td>Solenoid</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Dipole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Dipole with K1</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Quadrupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Sextupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Octupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Decapole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Thick Multipole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Muon Spoiler</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF Cavity (EM)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>RF (E only)</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>General Default</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Quadrupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Sextupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Octupole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Skew Decapole</td>
<td>G4ClassicalRK4</td>
</tr>
<tr>
<td></td>
<td>Dipole Fringe</td>
<td>BDSIM Dipole Fringe</td>
</tr>
<tr>
<td></td>
<td>Thin Multipole</td>
<td>BDSIM Thin Multipole</td>
</tr>
<tr>
<td></td>
<td>Multipole Outer</td>
<td>G4NystromRK4</td>
</tr>
<tr>
<td></td>
<td>Thin RMatrix</td>
<td>BDSIM Thin RMatrix</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
<th>Magnetic Field Type</th>
<th>Integrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>geant4dp*</td>
<td>Solenoid</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Dipole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Dipole with K1</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Quadrupole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Sextupole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Octupole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Decapole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Thick Multipole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Muon Spoiler</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>RF Cavity (EM)</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>RF (E only)</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>General Default</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Skew Quadrupole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Skew Sextupole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Skew Octupole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Skew Decapole</td>
<td>G4DormandPrince745</td>
</tr>
<tr>
<td></td>
<td>Dipole Fringe</td>
<td>BDSIM Dipole Fringe</td>
</tr>
<tr>
<td></td>
<td>Thin Multipole</td>
<td>BDSIM Thin Multipole</td>
</tr>
<tr>
<td></td>
<td>Multipole Outer</td>
<td>G4NystromRK4</td>
</tr>
<tr>
<td></td>
<td>Thin RMatrix</td>
<td>BDSIM Thin RMatrix</td>
</tr>
</tbody>
</table>

Note: *“geant4dp” is only available when BDSIM is compiled against Geant 4.10.4 or higher.

Note: Both dipole, fringe and thin multipole fields are thin elements and have no thick equivalent. Therefore, they have no field that Geant4 can use and can only use the BDSIM integrators.
17.9.3 Integrator Algorithms

BDSIM currently only provides integrators for magnetic fields, i.e. not electric or electromagnetic fields. For these types of fields, Geant4 integrators are used.

Common Magnetic Field Interface From Geant4

The magnetic field integrators provided by BDSIM inherit `G4MagIntegratorStepper`. This is constructed with respect to a `G4EquationOfMotion` object, which is a `G4Mag_UsualEqRhs` instance for BDSIM integrators. This equation of motion provides the partial differential of the motion at a given location - i.e. the field is found at that location and the vector potential calculated.

An integrator derived from `G4MagIntegratorStepper` must implement a method:

```cpp
virtual void Stepper( const G4double y[],
   const G4double dydx[],
   G4double h,
   G4double yout[],
   G4double yerr[] ) = 0;
```

This is responsible for calculating the coordinates of a trajectory given the input point `y[]` (which is `[x, y, z, p_x, p_y, p_z, t]`) for a step length of `h`. The output coordinates are written to `yout[]` (also `[x, y, z, p_x, p_y, p_z, t]`), along with the associated absolute uncertainty for each parameter to `yerr[]`. The differentials at the initial location are given by `dydx`. These are calculated in `G4Mag_UsualEqRhs.cc` as follows:

\[
A = \frac{\text{charge} \cdot e}{|\mathbf{p}|} (\mathbf{p} \times \mathbf{B})
\]

\[
dydx[0] = \frac{p_x}{|\mathbf{p}|}
\]

\[
dydx[1] = \frac{p_y}{|\mathbf{p}|}
\]

\[
dydx[2] = \frac{p_z}{|\mathbf{p}|}
\]

\[
\]

\[
\]

\[
\]

(There are other factors in the code for units that aren’t shown here.)

**Note:** Field calls: Geant4 will sample the field to give to the equation of motion to calculate `A`. Getting the field value is generally considered an expensive operation, as it may often involve geometry lookup for transforms, applying transforms or indexing a large array along with interpolation. In the case of BDSIM, the majority of fields require a geometry lookup and transform but are simple equations.

**Note:** Time: Geant4 magnetic integrators do not integrate time and therefore copy the initial value of time to the output coordinates. BDSIM integrators follow this behaviour. The time is handled by Geant4 at a higher level, as the magnetic integrators are specified to be only integrating over six variables.

**Coordinate Convention**

- Units are not explicitly mentioned here. In code there are factors to convert to Geant4 units.
- `q` is used to represent a 3-vector for spatial coordinates `(x, y, z)`.
- `p` is used to represent a 3-vector for the momentum `(p_x, p_y, p_z)`.
• The subscript "\(i\)" is used to denote input coordinates.
• The subscript "\(o\)" is used to denote what will be output coordinates after the step.
• \(h\) is used to describe the spatial step length requested. This would be along the curved trajectory through a field the particle would take.

**BDSIM Drift**

This algorithm transports a particle through free space with no external force acting on it. This is provided here, although provided generally by Geant4, as it is required by other BDSIM integrators under various circumstances. It exists in the `BDSIntegratorBase::AdvanceDrift` base class for the majority of BDSIM integrators.

\[
q_{\text{out}} = q_{\text{in}} + h \hat{p}_{\text{in}} \\
p_{\text{out}} = p_{\text{in}}
\]

**Note:** The drift element in BDSIM is not assigned a field or BDSIM-provided tracking algorithm. The tracking is handled by Geant4.

**BDSIM Dipole Rodrigues**

• **Class name:** `BDSIntegratorDipoleRodrigues`

This integrator is constructed with its own strength parameter and ignores the field information provided by Geant4. The field value (already multiplied by `CLHEP::tesla`) is assumed to be entirely along local \(\hat{y}\), i.e. the field vector is \(B = (0, B, 0)\). The algorithm progresses as follows:

• If the field value is zero or the particle is neutral, the coordinates are advanced as a drift.

Otherwise continue as follows:

• Calculate bending radius \(\rho\) as:

\[
\rho = \frac{\|p_{\text{in}}\|}{B \cdot \text{charge}}
\]

• Convert coordinates from global to local (curvilinear) frame of reference.

• Calculate local change of coordinates.

\[
\theta = \frac{h}{\rho} \\
\hat{f} = \hat{p} \times \hat{y} \\
\text{CT} = \cos^2(\theta/2) - \sin^2(\theta/2) \\
\text{ST} = 2 \cos(\theta/2) \sin(\theta/2) \\
q_{\text{out}} = q_{\text{in}} + \rho \left[ \text{ST} \hat{p}_{\text{in}} + (1 - \text{CT}) \hat{f} \right] \\
p_{\text{out}} = \hat{p}_{\text{in}} \text{CT} + \hat{f} \text{ST}
\]

• If \(\rho\) is less than a minimum radius of curvature (5 cm by default), reduce the magnitude of the momentum by two percent to induce artificial spiralling.

• Convert to global coordinates.

This was the original dipole algorithm included with BDSIM until v0.96, however, this is limited to dipole fields aligned with \(\hat{y}\) only and often caused tracking warnings with very low momenta particles in strong magnetic fields. A more flexible integrator that works in 3D was written to improve upon this and is described in BDSIM Dipole2.
BDSIM Dipole Rodrigues 2

- **Class name**: `BDSIntegratorDipoleRodrigues2`

This routine makes use of the tracking routine provided in Geant4 for a pure magnetic field. This is provided in the `G4MagHelicalStepper` class, which provides the tracking routine for a single step through a pure magnetic field, but not the other functionality required for a suitable integrator. This BDSIM class that inherits it provides the rest of the required functionality, as well as special treatment for particles that may spiral indefinitely.

- The field $\mathbf{B}$ is queried at $q_{in}$.
- A full step along the trajectory is calculated.
- If the radius of curvature is less than the minimum radius of curvature (5 cm by default), use the explicit spiralling algorithm.

Otherwise:

- Calculate the motion through two half-steps (includes sampling the field at the half-step point).
- Calculate the error on the output coordinates as the difference between two half-steps and one full-step.

The spiralling algorithm artificially advances the helix of the particle along the field direction more quickly than it would naturally by step length $h$, even if it had no momentum component along the field direction. This ensures that a particle that spirals in a strong magnetic field without ever hitting a boundary will terminate in timely manner and not dominate tracking time. The minimum radius of curvature is chosen to be approximately the radius of the typical aperture throughout the model (specified in the options). As the magnetic field does no work, a spiralling particle could spiral for a very long time and cause an event to run almost indefinitely. Given most dipoles in accelerators induce only a few milliradians of deflection, such a particle must be of a much lower momentum than the design momentum of the dipole and would not progress far from the magnet in reality.

This artificial behaviour terminates particles in the approximate location by moving them more quickly to a boundary.

The routine provided by Geant4 in `G4MagHelicalStepper` is as follows:

\[
\begin{align*}
\mathbf{p}_\parallel &= \|\mathbf{B}\| (\hat{\mathbf{B}} \cdot \hat{\mathbf{p}}_{in}) \\
\mathbf{p}_\perp &= \hat{\mathbf{p}}_{in} - \mathbf{p}_\parallel \\
R &= -\frac{\|\mathbf{B}\| \text{charge}}{\mathbf{p}_{in}} \\
\theta &= \frac{h}{R}
\end{align*}
\]

- If $\|\theta\| < 0.005$:
  \[
  \begin{align*}
  ST &= \sin \theta \\
  CT &= \cos \theta
  \end{align*}
  \]

- Else:
  \[
  \begin{align*}
  ST &= \theta - \frac{1}{6} \theta^3 \\
  CT &= 1 - \frac{1}{2} \theta^2 + \frac{1}{24} \theta^4
  \end{align*}
  \]

The final coordinates are calculated as:

\[
\begin{align*}
q_{out} &= q_{in} + R \left[ ST \mathbf{p}_\perp + (1 - CT) (\hat{\mathbf{B}} \times \hat{\mathbf{p}}_{in}) \right] + h \mathbf{p}_\parallel \\
p_{out} &= \hat{\mathbf{p}}_{in} \left[ CT \mathbf{p}_\perp + ST (\hat{\mathbf{B}} \times \hat{\mathbf{p}}_{in}) \right] + \mathbf{p}_\parallel
\end{align*}
\]

The distance from the chord and arc of the true path are also calculated by Geant4 and the algorithm is as follows.

- If the angle of the curve is in the range $0 \leq \theta \leq \pi$:
  \[
  \Delta_{\text{chord}} = R \left[ 1 - \cos \left( \frac{\theta}{2} \right) \right]
  \]
• Else if $\pi < \theta < 2\pi$:
  \[
  \Delta_{\text{chord}} = R \left[ 1 + \cos \left( \frac{2\pi - \theta}{2} \right) \right]
  \]

• Else:
  \[
  \Delta_{\text{chord}} = 2 \, R
  \]

**BDSIM Quadrupole**

• Class name: BDSIntegratorQuadrupole

The field gradient is calculated upon construction of the integrator as:

\[
B' = \frac{dB_y}{dx} = B_p \left( \frac{1}{B_p} \frac{dB_y}{dx} \right) = B_p \, k_1
\]

For each usage:

• Calculates strength parameter $\kappa$ w.r.t. a given particle rigidity:

\[
\kappa = \frac{\text{charge} \cdot c}{\|p\|} \frac{dB_y}{dx}
\]

If $\|\kappa\| < 10^{-20}$, use the drift integrator. Else, continue as:

• Convert to local curvilinear coordinates.

If $\hat{p}_z,\text{local} < 0.9$, the particle is considered non-paraxial and the backup integrator from BDSIntegratorMag is used. Else, proceed with thick matrix transportation. In this case, the following factors are calculated:

\[
rk = \sqrt{\|\kappa\| \, p_z}
\]

\[
rkh = h \, p_z \, rk
\]

For $\kappa > 0$, the focusing thick matrix is used (in the local curvilinear frame):

\[
M_{\text{quad},+\kappa} = \begin{pmatrix}
\cos(\rkh) & \frac{1}{\rk} \sin(\rkh) & 0 & 0 \\
-\|\kappa\| \frac{1}{\rk} \sin(\rkh) & \cos(\rkh) & 0 & 0 \\
0 & 0 & \cosh(\rkh) & \frac{1}{\rk} \sinh(\rkh) \\
0 & 0 & -\|\kappa\| \frac{1}{\rk} \sinh(\rkh) & \cosh(\rkh)
\end{pmatrix}
\]

and for $\kappa < 0$, the defocusing thick matrix is used (again, in the local curvilinear frame):

\[
M_{\text{quad},-\kappa} = \begin{pmatrix}
\cosh(\rkh) & \frac{1}{\rk} \sinh(\rkh) & 0 & 0 \\
-\|\kappa\| \frac{1}{\rk} \sinh(\rkh) & \cosh(\rkh) & 0 & 0 \\
0 & 0 & \cos(\rkh) & \frac{1}{\rk} \sin(\rkh) \\
0 & 0 & -\|\kappa\| \frac{1}{\rk} \sin(\rkh) & \cos(\rkh)
\end{pmatrix}
\]

These are used as follows (again in the local curvilinear frame):

\[
\begin{pmatrix}
q_{x,\text{out}} \\
p_{x,\text{out}} \\
q_{y,\text{out}} \\
p_{y,\text{out}}
\end{pmatrix} = M_{\text{quad},\pm} \begin{pmatrix}
q_{x,\text{in}} \\
p_{x,\text{in}} \\
q_{y,\text{in}} \\
p_{y,\text{in}}
\end{pmatrix}
\]

$p_{z,\text{out}}$ is calculated by conserving momentum.

\[
p_{z,\text{out}} = \sqrt{1 - p_{x,\text{out}}^2 - p_{y,\text{out}}^2}
\]

$q_{z,\text{out}}$ is calculated as:

\[
q_{z,\text{out}} = \sqrt{\left| h^2 \left( 1 - \frac{h^2}{12 \, R^2} \right) - (dq_x^2 + dq_y^2) \right|}
\]
where \( dq_{x,y} \) are the changes in local \( x \) and \( y \) respectively. \( R \) is:

\[
R = \frac{1}{\|R''\|}
\]

\[
R'' = \begin{pmatrix}
-p_{z,in} q_{x,in} \\
p_{z,in} q_{y,in} \\
q_{x,in} p_{z,in} - q_{y,in} p_{y,in}
\end{pmatrix}
\]

The distance from the chord and arc of the true path are estimated as:

\[
\Delta_{\text{chord}} = \frac{h^2}{8R}
\]

**BDSIM Euler**

- **Class name**: BDSIntegratorEuler
- Calculates the halfway position along step length \( h \) if the particle were to drift:

\[
q_{\text{half}} = q_{\text{in}} + \frac{\hat{p}_{\text{in}} h}{2}
\]

- Calculates the vector potential \( A \) w.r.t. \( q_{\text{half}} \) but with \( p_{\text{in}} \) (the original momentum - so as if the particle truly drifted to that point). Uses the equation of motion method RightHandSide. This invokes one query of the field.

- Calculates the new coordinates:

\[
q_{\text{out}} = q_{\text{in}} + \frac{\hat{p}_{\text{in}} h}{2} + A \frac{h^2}{2 \| \hat{p}_{\text{in}} \|}
\]

\[
p_{\text{out}} = p_{\text{in}} + A h
\]

**BDSIM Sextupole**

- **Class name**: BDSIntegratorSextupole

This integrator is constructed with \( k_2 \) (originally calculated w.r.t. the nominal beam rigidity higher up in BDSIM). It uses this to give a notion of a sextupolar field whilst calculating the magnetic vector potential in the local curvilinear coordinate frame. The input coordinates must therefore be converted to local curvilinear ones.

In comparison to the BDSIM Euler integrator, this has one extra transform for the coordinates but one fewer for the field and so has roughly the same performance. The algorithm is as follows:

- If \( \| k_2 \| \leq 10^{-12} \), track as a drift.
- Convert coordinates from global to local curvilinear frame.
- A point halfway along the step length \( h \) is calculated using a drift algorithm (“mid”).
- This position is used to calculate the vector potential as:

\[
A = \frac{k_2}{2!} \begin{pmatrix}
\hat{p}_{z,in} (q_{x,mid}^2 - q_{y,mid}^2) \\
-2 \hat{p}_{z,in} q_{x,mid} q_{y,mid} \\
\hat{p}_{x,in} (q_{x,mid}^2 - q_{y,mid}^2) - 2 \hat{p}_{y,in} q_{x,mid} q_{y,mid}
\end{pmatrix}
\]

**Note**: This can be viewed as the cross product between the unit momentum vector and the sextupolar field, whilst assuming that the \( B_z \) component is always zero and so some terms of the cross product can be omitted.

- The output coordinates are calculated with the communal BDSIM Old Euler Common algorithm.
BDSIM Octupole

- **Class name:** BDSIntegratorOctupole

This integrator is constructed with $k_3$ (originally calculated w.r.t. the nominal beam rigidity higher up in BDSIM).

- If $\|k_3\| < 10^{-20}$, track as a drift.
- Convert coordinates from global to local curvilinear frame.
- A point halfway along the step length $\ell$ is calculated using a drift algorithm ("$\ell_{mid}$").
- This position is used to calculate the vector potential as:

$$
A = \frac{k_3}{3!} \begin{pmatrix}
-p_{z,in} (q^3_{x,mid} - 3 q^2_{y,mid} q_{x,mid}) \\
-p_{z,in} (q^3_{x,mid} - 3 q^2_{y,mid} q_{y,mid}) \\
\hat{p}_{x,in} (q^3_{x,mid} - 3 q^2_{y,mid} q_{x,mid}) - \hat{p}_{y,in} (q^3_{x,mid} - 3 q^2_{y,mid} q_{y,mid})
\end{pmatrix}
$$

**Note:** This can be viewed as the cross product between the unit momentum vector and the octupolar field, whilst assuming that the $B_z$ component is always zero and so some terms of the cross product can be omitted.

- The output coordinates are calculated with the communal BDSIM Old Euler Common algorithm.

BDSIM Decapole

- **Class name:** BDSIntegratorDecapole

This integrator is constructed with $k_4$ (originally calculated w.r.t. the nominal beam rigidity higher up in BDSIM).

- If $\|k_4\| < 10^{-20}$, track as a drift.
- Convert coordinates from global to local curvilinear frame.
- A point halfway along the step length $\ell$ is calculated using a drift algorithm ("$\ell_{mid}$").
- This position is used to calculate the vector potential as:

$$
A = \frac{k_4}{4!} \begin{pmatrix}
\hat{p}_{z,in} (q^4_{x,mid} - 6 q^2_{x,mid} q^2_{y,mid} + q^4_{y,mid}) \\
-p_{z,in} [4 q_{x,mid} q_{y,mid} (q^2_{x,mid} - q^2_{y,mid})] \\
\hat{p}_{x,in} [q^4_{x,mid} - 6 q^2_{x,mid} q^2_{y,mid} + q^4_{y,mid}] - \hat{p}_{y,in} [4 q_{x,mid} q_{y,mid} (q^2_{x,mid} - q^2_{y,mid})]
\end{pmatrix}
$$

**Note:** This can be viewed as the cross product between the unit momentum vector and the decapolar field, whilst assuming that the $B_z$ component is always zero and so some terms of the cross product can be omitted.

- The output coordinates are calculated with the communal BDSIM Old Euler Common algorithm.

BDSIM Solenoid

- **Class name:** BDSIntegratorSolenoid

This integrator is constructed with a field strength. $k_s$ is calculated from this field strength for the nominal rigidity.

The particle motion for a solenoid is calculated for the body of the solenoid only and the edge effects are provided via thin elements using the rmatrix integrator.

The thick matrix for the solenoid body is:

$$
\begin{pmatrix}
1 & \frac{\sin(2KL)}{2K} & 0 & \frac{1-\cos(2KL)}{2K} \\
0 & \frac{\cos(2KL)}{2K} & 0 & \frac{\sin(2KL)}{2K} \\
0 & \frac{1-\cos(2KL)}{2K} & 1 & \frac{\sin(2KL)}{2K} \\
0 & -\frac{\sin(2KL)}{2K} & 0 & \cos(2KL)
\end{pmatrix}
$$
The solenoid fringes are constructed as thin RMATRIX elements either end of the solenoid body. If the fringes are constructed, the length of the solenoid body is reduced by the thin element length in order to conserve the total element length. In this case the solenoid strength is scaled accordingly. For the fringes, the following matrix is used

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & \pm K & 0 \\
0 & 0 & 1 & 0 \\
\pm K & 0 & 0 & 1
\end{pmatrix}
\]

where

\[K = \frac{B}{2B\rho}\]

The plus minuses are flipped for the exit fringe.

If the the x and y components of the unit curvilinear momentum are greater than 0.1 or the z component is less than 0.9, the fall-back G4ClassicalRK4 integrator is used.

**BDSIM Old Euler Common**

- **Class name:** BDSIntegratorMag

The Euler integration part of the original BDSIM integrators for higher order fields exists in one place in BDSIntegratorMag::AdvanceChord(). This takes the step length \(h\), the local position, momentum and vector potential. The algorithm is as follows:

- If \(\|A\| = 0\), advance as a drift.
- Else, proceed as:

\[
q_{x, out} = q_{x, in} + p_{x, in} h + \frac{A_x h^2}{2}
\]

\[
q_{y, out} = q_{y, in} + p_{y, in} h + \frac{A_y h^2}{2}
\]

The output z-coordinate is calculated as:

\[
q_{z, out} = q_{z, in} + \sqrt{h^2 \left(1 - \frac{h^2 \|A\|^2}{12}\right) - \delta_x^2 - \delta_y^2}
\]

The momentum is calculated as:

\[p_{out} = p_{in} + h A\]

The delta chord intersection is calculated as:

\[
\Delta_{chord} = \frac{h^2 \|A\|}{8}
\]

The error is not calculated here.

**BDSIM Dipole Fringe**

- **Class name:** BDSIntegratorDipoleFringe

This integrator provides only a change in momentum that represents both the edge effect of a dipole with a pole face rotation and dipole pole face curvature. The effect of pole face curvature is applied using the thin multipole integrator with a sextupole strength of:

\[K_{3l} = \frac{h}{\rho \cos^3(\theta)}\]
where $h$ is the pole face curvature and $\theta$ is the pole face rotation angle.

The pole face curvature effect is applied first, but only if the pole face curvature is finite. The function for applying the momentum kick converts to curvilinear coordinates, calls the thin multipole stepper function that applies the kick, and finally converts back to global coordinates.

As the thin fringe element has finite length, a small dipole kick must be applied to conserve the magnetic length of the dipole. This class inherits BDSIntegratorDipoleRodrigues2 for the dipole component of the motion. After that, the small change in momentum is applied.

- If the step length is longer than 1 mm, the kick is not applied (i.e. not a thin dipole edge element).
- The input coordinates are converted to the local curvilinear frame. This is required only for this algorithm and not for that in BDSIntegratorDipoleRodrigues2.
- If $\hat{p}_z,\text{local} < 0.9$, the particle is considered non-paraxial and no change in momentum is applied.

The thin matrix in the local curvilinear frame is:

$$
\begin{pmatrix}
  x_1 \\
x'_1 \\
y_1 \\
y'_1 \\
l_1 \\
\delta \\
\end{pmatrix} = \begin{pmatrix}
  1 & 0 & 0 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 & 0 \\
  0 & 0 & -\frac{1}{\rho} \tan (\theta - \text{corr.}) & 1 & 0 & 0 \\
  0 & 0 & 0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 0 & 0 & 1 \\
\end{pmatrix} \begin{pmatrix}
  x_0 \\
x'_0 \\
y_0 \\
y'_0 \\
l_0 \\
\delta \\
\end{pmatrix}
$$

The resulting momentum change will therefore be:

$$
dp_x = \frac{q_x,\text{in}}{\rho} \tan(\theta) \\
dp_y = \frac{q_y,\text{in}}{\rho} \tan(\theta - \text{corr.})
$$

Where “corr.” is the fringe field correction term. The calculation of the fringe field correction term is split into two terms, which are calculated separately in two namespace functions BDS::FringeFieldCorrection() and BDS::SecondFringeFieldCorrection() upon class instantiation. These functions calculate:

$$
corr. = f_{\text{int}} \frac{2 h_{\text{gap}}}{\rho} \frac{(1 + \sin^2 \theta)}{\cos \theta} \text{corr}_2.
$$

and:

$$
corr_2. = 1 - f_{\text{int}} f_{\text{intk2}} \frac{2 h_{\text{gap}}}{\rho} \tan \theta
$$

respectively, where $f_{\text{int}}$ is an input parameter but described by:

$$
f_{\text{int}} = \int_{-\infty}^{\infty} B_y(s) \left( B_0 - B_y(s) \right) \frac{2 h_{\text{gap}} B^2_0}{2 h_{\text{gap}} B^2_0} ds
$$

Here, $h_{\text{gap}}$ is also an input parameter that specifies the half-distance between the dipole poles. Fintk2 is a second fringe parameter with a default of zero, meaning the corr term equals 1 by default.

- The correction term.

The change in momentum:

$$
dp = \begin{pmatrix}
+dp_x \\
-dp_y \\
0 \\
\end{pmatrix}
$$

- This $dp$ vector is converted to the global frame.

The output momentum (from the dipole integrator) is adjusted as:

$$
p_{\text{out}} = p_{\text{in}} + dp
$$
BDSIM Thin Multipole

- **Class name:** BDSIntegratorMultipoleThin

This integrator applies a thin multipole kick to forward going paraxial particles. This is normally attached to a box or disc that is very thin (Geant4 requires finite dimensions) but sufficiently small that only one step is taken through it. Typically, a length of 1 pm is used along $S$. It is not possible to control how many steps a particle takes through a given volume in Geant4 tracking, as many physics processes can propose different step lengths. However, by choosing such a short length of volume and by filling it with vacuum, no other process will force a step in the middle of the volume. If more than one step were taken, the integrator would be used multiple times, resulting in stronger kicks than are correct.

- Convert coordinates from global to local curvilinear frame.
  
  If $\hat{p}_{z,\text{local}} < 0.9$, the particle is considered non-paraxial and the backup integrator from BDSIntegratorMag is used. Else proceed with thin kick.
  
  The output position remains the same.
  
  $$q_{\text{out}} = q_{\text{in}}$$

  The momentum is modified as:
  
  $$q_c = q_{x,\text{in}} + i q_{y,\text{in}}$$
  
  $$dp_n = \sum_{j=1}^{12} \frac{k_{n,j}}{j!} q_c^j$$
  
  $$dp_s = \sum_{j=1}^{12} \frac{k_{s,j}}{j!} q_c^j$$

  Where $q_c$ is the complex number formed from the horizontal and vertical positions in the local curvilinear frame and the subscripts “n” and “s” represent normal and skew multipole components respectively. The output momentum is therefore:
  
  $$p_{x,\text{out}} = p_{x,\text{in}} - \text{Re}(dp_n) - \text{Im}(dp_s)$$
  
  $$p_{y,\text{out}} = p_{y,\text{in}} + \text{Im}(dp_n) + \text{Im}(dp_s)$$
  
  $$p_{z,\text{out}} = \sqrt{1 - p_{x,\text{out}}^2 - p_{y,\text{out}}^2}$$
  
  $$\mathbf{p}_{\text{out}} = \begin{pmatrix} p_{x,\text{out}} \\ p_{y,\text{out}} \\ p_{z,\text{out}} \end{pmatrix}$$

BDSIM Dipole Matrix

- **Class name:** BDSIntegratorDipoleQuadrupole

This integrator is constructed with its own strength parameter and ignores the field information provided by Geant4. The field value (already multiplied by CLHEP::tesla) is assumed to be entirely along local $\hat{y}$, i.e. the field vector is $\mathbf{B} = (0, B, 0)$.

Upon construction of the integrator, the following are calculated:

- The nominal bending radius $\rho$:
  
  $$\rho = \frac{L}{\theta}$$

  The bending radius is not calculated using the magnetic field, as the field can be set to purposefully underpower or overpower the magnet.

- The quadrupolar component, the field gradient:
\[ B' = \frac{dB_y}{dx} = B \rho \left( \frac{1}{B \rho} \frac{dB_y}{dx} \right) = B \rho \ k_1 \]

For each usage, the strength parameter \( \kappa \) is calculated w.r.t. a given particle rigidity:

\[ \kappa = \frac{\text{charge} \cdot c}{\| p \|} \frac{dB_y}{dx} \]

- The ratio of supplied magnetic field to nominal magnetic field:

\[ \text{fieldRatio} = \frac{B \cdot \rho}{B \rho} \]

Where \( B \) is the magnetic field strength, \( \rho \) is the nominal bending radius, and \( B \rho \) is the nominal magnetic rigidity for the magnet, which is cached upon construction. The field ratio is used to calculate the curvilinear transform angle. If \( \text{fieldRatio} = 1 \), then proceed using the nominal bending angle \( \theta \), otherwise if \( \text{fieldRatio} \neq 1 \):

\[ \theta = \text{fieldRatio} \cdot \theta \]

As this integrator will ultimately use particle coordinates in the curvilinear frame, the bending actually occurs in the curvilinear transforms. As a dipole can be underpowered or overpowered by specifying both the field and angle in the input component definition, the transforms must be supplied with the correct bending angle to ensure the particles will be transformed onto the correct trajectory.

The algorithm progresses as follows:

- If the field value is zero, the particle is neutral. For a very small step length \( h < 10^{-12} m \), the coordinates are advanced as a drift.

Otherwise continue as follows:

The distance from the chord and arc of the true path is calculated by taking a single step taken using the backup dipole stepper BDSIntegratorDipoleRodrigues2. This integrator provides access to the chord-arc distance which is then used in this integrator. We assume the dipole component will provide a bigger effect than the quadrupole component.

- For small step length \( h < 10^{-7} m \), the coordinates are advanced using the full backup stepper.

The radius of curvature is also taken from the aforementioned single step in the backup integrator. If the chord-arc distance is \( > 0.3 \) times the radius of curvature, the particle is assumed to be spiralling and subsequently the full backup stepper is used.

- Convert to local curvilinear coordinates.

If \( \hat{p}_{z, \text{local}} < 0.9 \), the particle is considered non-paraxial and the backup integrator from BDSIntegratorMag is used. Else, proceed with thick matrix transportation.

- Thick dipole matrix:

The matrix implemented is the RMatrix from Particle Accelerator Physics (3rd Edition) by Wiedemann, chapter five. For the case of a focusing magnet, \( \kappa \geq 0 \):

\[
\begin{pmatrix}
  x_1 \\
x'_1 \\
y_1 \\
y'_1 \\
l \delta
\end{pmatrix} =
\begin{pmatrix}
  \cos \Theta_x & \sin \Theta_x & 0 & 0 & 0 & \frac{1-\cos \Theta_x}{\sqrt{K_0}} \\
-\sqrt{K_x} \sin \Theta_x & \cos \Theta_x & 0 & 0 & 0 & 0 \\
0 & 0 & \cosh \Theta_y & \sinh \Theta_y & 0 & 0 \\
0 & 0 & \sqrt{K_y} \sinh \Theta_y & \cosh \Theta_y & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x_0 \\
x'_0 \\
y_0 \\
y'_0 \\
l_0 \\
\delta
\end{pmatrix}
\]

Where:

\[ \Theta_x = \sqrt{K_x} \ h = \sqrt{\| \kappa + \kappa_0 \|} \ h \]

\[ \Theta_y = \sqrt{\| K_y \|} \ h = \sqrt{\| \kappa \|} \ h \]

\[ \kappa_0 = \frac{1}{\rho} \]
For the case of a defocussing magnet $\kappa < 0$:

$$
\begin{bmatrix}
    x_1 \\
    x'_1 \\
    y_1 \\
    y'_1 \\
    l_1 \\
    \delta
\end{bmatrix} =
\begin{bmatrix}
    \cosh \Theta_x & \frac{\sinh \Theta_x}{\sqrt{K_x}} & 0 & 0 & 0 & \frac{1 - \cosh \Theta_x}{\sqrt{K_x}} \\
    \sqrt{K_x} \sinh \Theta_x & \frac{\cosh \Theta_x}{\sqrt{K_x}} & 0 & 0 & 0 & \frac{\sinh \Theta_x}{\sqrt{K_x}} \\
    0 & 0 & \cos \Theta_y & \frac{\sin \Theta_y}{\sqrt{K_y}} & 0 & 0 \\
    0 & 0 & -\sqrt{K_y} \sin \Theta_y & \cos \Theta_y & 0 & 1 \\
    0 & 0 & 0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_0 \\
    x'_0 \\
    y_0 \\
    y'_0 \\
    l_0 \\
    \delta
\end{bmatrix}
$$

In the case where $\kappa = 0$, the matrix simplifies to:

$$
\begin{bmatrix}
    x_1 \\
    x'_1 \\
    y_1 \\
    y'_1 \\
    l_1 \\
    \delta
\end{bmatrix} =
\begin{bmatrix}
    \cos \frac{h}{\rho} & \rho \sin \frac{h}{\rho} & 0 & 0 & 0 & \rho \left(1 - \cos \frac{h}{\rho}\right) \\
    -\rho \sin \frac{h}{\rho} & \cos \frac{h}{\rho} & 0 & 0 & 0 & \sin \frac{h}{\rho} \\
    0 & 0 & 1 & h & 0 & 0 \\
    0 & 0 & 0 & 1 & 0 & 0 \\
    0 & 0 & 0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x_0 \\
    x'_0 \\
    y_0 \\
    y'_0 \\
    l_0 \\
    \delta
\end{bmatrix}
$$

The $z$ terms are not calculated via the matrix method, rather the $z$-position is simply the addition of the step length, and the $z$-momentum is calculated from the $x$- and $y$-momentum to ensure momentum conservation. Note that these matrices are incomplete; there are terms for the calculation of the $l$ parameter which are not needed in this stepper.

### 17.9.4 Validation of BDSIM Integrators

- Comparison with PTC
- Comparison with RK4

### 17.9.5 BDSIM Integrator Response to Non-Paraxial Particles

- Use Geant4 RK4
- Treat as drift for very low energy

### 17.10 Fields

BDSIM provides the magnetic fields typical to an accelerator, as well as the ability to import field maps and overlay them on geometry. Practically, this is accomplished through several objects, but the main two that the user or developer must select are the field and the integrator. The first describes the field itself; the latter how the particle position and momentum is affected by the field.

The integrators are described in Tracking Algorithms.

#### 17.10.1 Coordinate System

The accelerator is modelled, following convention, in curvilinear coordinates that follow the beam line. However, in practice, Geant4 requires that the fields and coordinates be supplied and calculated in global Cartesian coordinates. The simplest solution in Geant4 is to get the transformation from the local coordinates to the volume being queried for the field and tracking. However, if the field is ‘attached’ to not just a simple single shape or volume, but a nested set of volumes, the local coordinates of that volume are not necessarily the same as the accelerator curvilinear coordinates. To get around this, a parallel geometry is built with simple shapes whose local coordinates are degenerate with the curvilinear coordinate system. This geometry is used to find the transforms so that when their local coordinates are queried, they represent the curvilinear coordinates of the beam line.
Generally, to query a point in the geometry, one should use a G4Navigator instance. There is the singleton G4Navigator from Geant4 available to the developer, but this must never be used. Querying a point in the geometry with this navigator changes the state of the navigator and therefore the perceived location in the geometry hierarchy of the particle in question from that point on. To avoid this, an extra navigator is created and used. Whilst these are not large objects in memory, a single static extra navigator (member of BDSAuxiliaryNavigator) is used. An instance of a G4Navigator can perform a search relative to the last queried point, which is usually significantly faster than a fresh query from the top of the hierarchy. By using a static auxiliary navigator, we take advantage of the relative search, because although many fields and integrators may have query separate BDSAuxiliaryNavigator instances, the underlying static navigator is used. In practice, the queries are generated by the progress of a particle, so they’re likely to be close to each other in the geometry.

Utility methods for conversion are provided in BDSAuxiliaryNavigator. The developer should design their class to inherit this one if they wish to convert to curvilinear coordinates.

Pure magnetic fields are provided that don’t inherit BDSAuxiliaryNavigator to avoid the requirement of ‘closed’ Geant4 geometry and a parallel curvilinear world. This greatly simplifies things if the developer wants to simply make use of (or test alone for that matter) a single field class.

**Important Points**

- Field classes don’t use BDSAuxiliary navigators and therefore don’t require a full Geant4 run.
- The fields constructed for the BDSIM model are wrapped in an instance of BDSFieldMagGlobal that provides the necessary local-to-global transforms.
- Using BDSAuxiliaryNavigator requires an accelerator to be built, i.e. it requires a world volume, read out world, contents in both, the geometry to be ‘closed’ by Geant4 and a valid run manager instantiated. One may generally use the field classes, but without the auxiliary navigator in this case.

**Notes About Plots**

The plots provided here are generated by surveying each field class at various points in both a uniform Cartesian grid and also in a radial grid. The vector field plots are generated automatically using matplotlib. Whilst these look pretty spiffy, occasionally the vectors pass through inflection points (such as the origin) that we know they should not. This is purely a numerical artefact and the plots are only to give a rough impression of the field shape. The fields are generated by the equations described and are correct even at inflection points.

**17.10.2 Pure Magnetic Fields From Equations**

Described here are a list of typical magnetic fields that are described by equations, rather than an interpolated field map. These are used for the majority of the accelerator components. Described here is the pure version without global to curvilinear transformations. These classes are wrapped when used with general BDSAcceleratorComponent instances.

**Dipole**

The dipole field is constructed with a magnitude $|B|$ and a unit vector $\hat{b}$. It is constant with position and the default unit vector is $(0, 1, 0)$ - unit y.

$$B = \hat{b} \cdot |B|$$
Fig. 4: Example field map of a dipole with $B = 1.3$ T, and $B_\rho = 4.333$.

**Quadrupole**

The quadrupole field is constructed with strength parameter $k_1$ and with respect to a nominal rigidity $B_\rho$. Although the rigidity is included in $k_1$, it is required to calculate the field gradient internally.

$$k_1 = \frac{1}{B_\rho} \frac{\partial B_y}{\partial x}$$

The field is described by

$$B_x = \frac{\partial B_y}{\partial x} y$$
$$B_y = \frac{\partial B_y}{\partial x} x$$
$$B_z = 0$$

**Sextupole**

The sextupole field is constructed with strength parameter $k_2$ and with respect to a nominal rigidity $B_\rho$.

$$k_2 = \frac{1}{B_\rho} \frac{\partial^2 B_y}{\partial x^2}$$

The field is described by

$$B_x = \frac{1}{2!} \frac{\partial^2 B_y}{\partial x^2} 2xy$$
$$B_y = \frac{1}{2!} \frac{\partial^2 B_y}{\partial x^2} (x^2 - y^2)$$
$$B_z = 0$$
Fig. 5: Example field map of a quadrupole with $k_1 = 0.34$, and $B\rho = 4.333$.

Fig. 6: Example field map of a sextupole with $k_2 = 3.91$, and $B\rho = 4.333$. 
Octupole

The octupole field is constructed with strength parameter $k_3$ and with respect to a nominal rigidity $B\rho$.

$$k_3 = \frac{1}{B\rho} \frac{\partial^3 B_y}{\partial x^3}$$

The field is described by

$$B_x = \frac{1}{3!} \frac{\partial^3 B_y}{\partial x^3} (3x^2y - y^3)$$

$$B_y = \frac{1}{3!} \frac{\partial^3 B_y}{\partial x^3} (x^3 - 3xy^2)$$

$$B_z = 0$$

![Octupole field map](image)

Fig. 7: Example field map of an octupole with $k_3 = 12.56$, and $B\rho = 4.333$.

Decapole

The decapole field is constructed with strength parameter $k_4$ and with respect to a nominal rigidity $B\rho$.

$$k_4 = \frac{1}{B\rho} \frac{\partial^4 B_y}{\partial x^4}$$

The field is described by

$$B_x = \frac{1}{4!} \frac{\partial^4 B_y}{\partial x^4} 4xy(x^2 - y^2)$$

$$B_y = \frac{1}{4!} \frac{\partial^4 B_y}{\partial x^4} (x^4 - 6x^2y^2 + y^4)$$

$$B_z = 0$$
Skewed Versions

All of the above magnets (dipole, quadrupole, sextupole, octupole and decapole) are also available as their skew counterparts. With BDSIM, it is trivial to create a skew component by simply creating a normal component and applying the appropriate tilt to it. However, should one want the field skewed but not the component - say, the correct upright square aperture - these fields can be used.

A wrapper class is provided that is instantiated with an angle (hard coded in BDSFieldFactory). When the field is queried, the coordinates being queried are rotated by the angle. The returned field vector is then anti-rotated to give the correct skew field at the original location.

\[
\begin{align*}
B_{\text{skew}}(x, y) &= R(-\theta)B(x', y') \\
\begin{bmatrix}
x'' \\
y'' \\
z''
\end{bmatrix} &= R(\theta)
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix} =
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\end{align*}
\]

Example field maps are shown below.

Skew Quadrupole

Skew Sextupole

Skew Octupole

Skew Decapole

Multipole

A general multipole field is also provided. The field is calculated in cylindrical coordinates, then converted to Cartesian. The field is calculated using an array of strength parameters \(k_1, k_2, \ldots, k_{12}\) and the skewed strength parameters \(ks_1, ks_2, \ldots, ks_{12}\) with respect to a nominal rigidity \(B\rho\).
Fig. 9: Example field map of a skew quadrupole with $k_1 = 0.34$, and $B_\rho = 4.333$.

Fig. 10: Example field map of a skew sextupole with $k_2 = 3.92$, and $B_\rho = 4.333$. 
Fig. 11: Example field map of a skew octupole with $k_3 = 12.56$, and $B\rho = 4.333$.

Fig. 12: Example field map of a skew decapole with $k_4 = 45567.32$, and $B\rho = 4.333$. 
Note: Currently the dipole component is not implemented. $k_1$ is the quadrupole strength, $k_2$ is the sextupole strength, etc.

$$r = \sqrt{x^2 + y^2}$$

$$B_r(\text{normal}) = \frac{1}{B\rho} \sum_{i=1}^{12} \frac{k_i}{i!} r^i \sin(i\phi)$$

$$B_\phi(\text{normal}) = \frac{1}{B\rho} \sum_{i=1}^{12} \frac{k_i}{i!} r^i \cos(i\phi)$$

$$B_r(\text{skewed}) = \frac{1}{B\rho} \sum_{i=1}^{12} \frac{k_{si}}{i!} r^i \cos(i\phi)$$

$$B_\phi(\text{skewed}) = \frac{1}{B\rho} \sum_{i=1}^{12} \frac{k_{si}}{i!} r^i \sin(i\phi)$$

$$B_x = B_r \cos\phi - B_\phi \sin\phi$$

$$B_y = B_r \sin\phi + B_\phi \cos\phi$$

Fig. 13: Example field map of a multipole with $\{k_1, k_2, k_3, k_4, k_5\} = \{0.12, 0.02, -0.003, 0.0004, -0.00005\}$, and $B\rho = 4.333$. 

17.10. Fields
### Undulator

The undulator field is constructed with the peak field strength $B$ and the undulator period $\lambda$. The field, according to Wiedemann pg. 103, is described by

\[
B_x = 0 \\
B_y = B \cdot \cos \left( \frac{2\pi z}{\lambda} \right) \cosh \left( \frac{2\pi y}{\lambda} \right) \\
B_z = -B \cdot \sin \left( \frac{2\pi z}{\lambda} \right) \sinh \left( \frac{2\pi y}{\lambda} \right)
\]

### Muon Spoiler

A muon spoiler field is provided that gives a constant toroidal field. It is constructed with field strength $B$ (T). The field is calculated according to

\[
r = \sqrt{x^2 + y^2} \\
B_x = \frac{y}{r} B \\
B_y = -\frac{x}{r} B \\
B_z = 0
\]

![muonspoilerRadialFieldMap.png](attachment:muonspoilerRadialFieldMap.png)

Fig. 14: Example field map of a muon spoiler with field $B = 1.3$ (T). Note, the variation shown in the graph is only numerical differences. The field is constant and this is purely due to the plotting vector field algorithm.

### Dipole Yoke Field 3D

For the outer part of a dipole, as described by a uniform field in 3D $B$, a pure dipole field at position $r$ from the origin is provided according to

\[
\mathbf{B}_{\text{dipole}}(r) = \frac{3r(m \cdot r)}{r^5} - \frac{m}{r^3}
\]
where \( \mathbf{m} \) is a unit vector along the pure dipole field direction. The field value is scaled to the field at the pole tip. For positions within a radial distance of the origin of pole tip radius, the uniform field vector is used. At the transition, a sigmoid function is used to smoothly vary (weight \( w \)) between the uniform field vector \( \mathbf{B_u} \) according to

\[
w = \frac{1}{2} \left[ \tanh \left( 3r - \frac{\|0.5 \cdot r_{\text{poletip}}\|}{1\text{cm}} \right) + 1 \right]
\]

\[
\mathbf{B}(r) = w \mathbf{B}_{\text{dipole}}(r) + (1 - w) \mathbf{B_u}
\]

An example is shown below for \( \mathbf{B} = (0.23, 0.56, 0) \) T and a pole tip radius of 40mm.

**General Yoke Multipole**

For the outside of magnets, a generalised multipolar field is provided. This is an approximate field for outside the beam pipe and does not take into account the permeability of the iron. We suggest overlaying a field map for your own magnets if greater accuracy is desired.

The field is described by the linear sum of infinitely long current sources along \( \pm z \) (in curvilinear coordinates). Each current source is placed exactly in between each pole at a distance of pole tip radius \( r_{\text{poletip}} \). The field is normalised to the field sampled from the interior field at a pole tip.

Wire locations:

\[
\mathbf{c}_i = \begin{bmatrix} x \\ y \end{bmatrix}_i = \begin{bmatrix} 0 \\ r_{\text{poletip}} \end{bmatrix} \begin{bmatrix} \cos \theta_i \\ -\sin \theta_i \\ \sin \theta_i \\ \cos \theta_i \end{bmatrix}
\]

\[
\theta_i = \left\{ \frac{i \cdot 2\pi}{n_{\text{poles}}} \right\} \text{ for } i = \{0 \ldots n_{\text{poles}}\}
\]

The field value as a function of position \( \mathbf{r} = (x, y) \) is

\[
\mathbf{B}(\mathbf{r}) = \sum_{i=1}^{i=n_{\text{poles}}} (-1)^i \frac{(\mathbf{r} - \mathbf{c}_i)_\perp}{\|\mathbf{r} - \mathbf{c}_i\|}
\]

These are provided for dipole through to decapole- including their skew counterparts. A few examples are presented below.
Multipole Yoke - Dipole

Multipole Yoke - Quadrupole

Multipole Yoke - Sextupole

Multipole Yoke - Octupole
17.10. Fields

multipoleoutersextupole_radial field map

Field (T)

multipoleouteroctupole_radial field map

Field (T)
17.10.3 Electric Fields From Equations

Sinusoidal Electric Field

This field provides an electric field along local $z$ direction with an amplitude $E$ that does not vary with position $(x, y, z)$, but only varies sinusoidally with time ($t$). A cosine is used so when the default phase is zero, a maximum
acceleration is provided. Aside from the field amplitude \( E \), the frequency \( f \) (Hz) along with the phase \( \phi \) are used.

\[
E_z = E \cos(2\pi ft + \phi)
\]

The 3D Cartesian field vectors are therefore:

\[
\mathbf{B} = (0, 0, 0) \\
\mathbf{E} = (0, 0, E_z)
\]

### 17.10.4 Electromagnetic Fields From Equations

#### Pill-Box Cavity

The pill-box cavity field is constructed with an electric field amplitude \( E \), a frequency \( f \), phase \( \psi \) and cavity radius. The cavity radius is used to normalise the Bessel function so that the field drops to zero at this point. The field is time-dependent and the \( E_z \) and \( B_\phi \) components are calculated and then returned in 3D Cartesian coordinates. The cavity radius is used to calculate a normalised radius \( r_n \) with respect to the first zero of the zeroth Bessel:

\[
r_n = r \sqrt{2.404825557695772768622 / \text{cavity radius}}
\]

The electric field is calculated as:

\[
E_z = E J_0(r) \cos(2\pi ft + \psi)
\]

The B-field amplitude is calculated from the E-field amplitude.

\[
H = \frac{E_z}{Z_0} \\
B = \mu_0 H
\]

where \( Z_0 \) is the impedance of free space. To calculate B, a vacuum is assumed and therefore only the vacuum permeability is used to calculate B from H.

The radial magnetic field in the pill-box field is:

\[
B_\phi = \frac{E \mu_0}{Z_0} J_1(r) \sin(2\pi ft + \psi)
\]

The 3D Cartesian field vectors are therefore:

\[
\mathbf{B} = (B_\phi \cos(\phi), B_\phi \sin(\phi), 0) \\
\mathbf{E} = (0, 0, E_z)
\]

Where \( \phi \) is the polar coordinate.

### 17.10.5 Field Map File Formats

#### BDSIM Field Format

The field should be in an ASCII text file with the extension .dat. A compressed file using tar and gzip may also be used. The tar should contain only one file that is the field. In this case, the file should have .tar.gz extension. Below is an example of the required format in each 1D, 2D, 3D and 4D case.

The pybdsim utility may be used to prepare fields in the correct format in Python if a Python numpy array is provided. If the user has a custom field format, it would be advisable to write a script to load this data into a Python numpy array and use the provided file writers in pybdsim.

Generally:

- A series of keys define the dimensions of the grid.
• The keys at the beginning do not have to be in any order.
• Empty lines will be skipped.
• A line starting with ! denotes the column name definition row (there can be only one of these).
• The order in the file must be 1) keys, 2) column name definition row, 3) data.
• A line starting with # will be ignored as a comment line.
• The order of the data must loop in the **lowest** dimension first and then the upper, so the order should be \( x, y, z, t \).
• Python classes are provided to write numpy arrays to this format.
• Any lines beyond the amount of data specified by the dimensions will be ignored.
• One **cannot** put a comment after the data in the line.
• \( \text{loopOrder} > \text{zyxt} \) may optionally be defined in the header to indicate the order of looping of variables in the file to the loader. The default is \( \text{xyzt} \).

**Note:** The units are \( \text{cm} \) for spatial coordinates and \( \text{s} \) for temporal.

**Note:** If a 1,2 or 3D field is required that is not along \( x, x, y, x : y : z \) respectively, the user should label the columns appropriately (i.e. \( 'X' \) and \( 'Z' \)) and use the correct key names in the file (i.e. \( 'xmin' \) and \( 'zmin' \)) and the field will be automatically constructed along the desired direction. It is assumed the field is constant in the other dimensions.

There are python scripts in bdsim/examples/features/fields/4_bdsimformat called Generate1D.py etc., that were used to create the example data sets there that have sinusoidally oscillating data.

**Warning:** The dimension parameters \( (x, y, z, t) \) are used in order here for 1,2,3 and 4D fields, but other combinations are possible. See **BDSIM Field Format Different Dimensions**.

**BDSIM Field Format 1D**

For a field that varies in \( x \).

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmin</td>
<td>The lower spatial coordinate in ( x ) associated with the field map</td>
</tr>
<tr>
<td>xmax</td>
<td>The upper spatial coordinate in ( x ) associated with the field map</td>
</tr>
<tr>
<td>nx</td>
<td>Number of elements in ( x ) - 1 counting</td>
</tr>
</tbody>
</table>

Example syntax is shown below and there is an example in bdsim/examples/features/fields/4_bdsimformat/1dexample.tar.gz. The complete example field is specified here:

```
xmin> -30.0
nx> 8
xmax> 22.5
!
x| Fx| Fy| Fz
-3.00000000E+01 -2.94957486E+00 -2.82240016E-01 -1.16825503E+00
-2.25000000E+01 -9.08808379E-01 -1.55614639E+00 -7.42211878E-01
-1.50000000E+01 1.44943102E+00 -1.99498997E+00 -2.99500250E-01
-7.50000000E+00 3.30134246E+00 -1.36327752E+00 1.49937508E+00
0.00000000E+00 4.00000000E+00 0.00000000E+00 5.96007992E-01
7.50000000E+00 3.30134246E+00 1.36327752E+00 1.02869342E+00
```

(continues on next page)
The same field could be specified along \( z \) with the following start:

```
zmin> -30.0
nz> 8
zmax> 22.5
! Z  Fx  Fy  Fz
```

### BDSIM Field Format 2D

All of the 1D parameters, plus:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ymin</td>
<td>The lower spatial coordinate in ( y ) associated with the field map</td>
</tr>
<tr>
<td>ymax</td>
<td>The upper spatial coordinate in ( y ) associated with the field map</td>
</tr>
<tr>
<td>ny</td>
<td>Number of elements in ( y ) - 1 counting</td>
</tr>
</tbody>
</table>

Example syntax is shown below and there is an example in `bdsim/examples/features/fields/4_bdsimformat/2dexample.tar.gz`. Only the first small part of the file is reproduced here:

```
ymax> 22.6
nx> 8
ny> 11
xmin> -30.0
ymin> -25.0
! X  Y  Fx  Fy  Fz
-3.00000000E+01 -2.50000000E+01 1.76523839E+00 1.08228603E+00 2.12211605E+01
-2.44000000E+01 -2.50000000E+01 8.90617540E-01 1.48727104E+00 1.03093724E+00
-1.88000000E+01 -2.50000000E+01 -1.59784082E-01 1.59871406E+00 1.76936408E+00
-1.32000000E+01 -2.50000000E+01 -1.17864919E+00 1.39461962E+00 2.36997669E+00
-7.60000000E+00 -2.50000000E+01 -1.96488486E+00 9.15269759E-01 2.68815749E+00
-2.00000000E+00 -2.50000000E+01 -2.36331212E+00 2.55273528E-01 9.85012500E+00
3.60000000E+00  -2.50000000E+01 -2.29529355E+00 -4.55105921E+00 9.45153108E+00
9.20000000E+00  -2.50000000E+00 -1.77425397E+00 -1.07566133E+00 2.18815749E+00
1.48000000E+00  -2.50000000E-01 -9.03030699E-01 -1.48391395E+00 2.75405668E-01
-2.04000000E+00 -2.50000000E+01  1.46423320E-01 -1.59928717E+00 1.50000000E+00
2.60000000E+00  -2.50000000E+00  1.16697784E+00 -1.39900982E+00 8.02496486E-01
-1.32000000E+01 -2.50000000E+00 -1.90859292E+00 4.29334082E+00 2.12211605E-01
-1.82000000E+01 -2.50000000E+00  2.85845993E+00 3.33182089E-01 2.12211605E-01
-2.44000000E+01 -2.50000000E+00  1.44218172E+00 4.57856850E-01 1.03093724E+00
-1.88000000E+01 -2.50000000E+00 -2.58739215E-01 4.92164617E-01 1.76936408E+00
-7.60000000E+00 -2.50000000E+00  1.16697784E+00 -1.39900982E+00 8.02496486E-01
-1.32000000E+01 -2.50000000E+00 -1.90859292E+00 4.29334082E+00 2.12211605E-01
```

(continues on next page)
BDSIM Field Format 3D

All of the 1D and 2D parameters, plus:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>zmin</td>
<td>The lower spatial coordinate in ( z ) associated with the field map</td>
</tr>
<tr>
<td>zmax</td>
<td>The upper spatial coordinate in ( z ) associated with the field map</td>
</tr>
<tr>
<td>nz</td>
<td>Number of elements in ( z ) - 1 counting</td>
</tr>
</tbody>
</table>

Example syntax is shown below and there is an example in `bdsim/examples/features/fields/4_bdsimformat/3dexample.tar.gz`. Only the first small part of the file is reproduced here:

```
zmax> 29.0
ymax> 18.2
zmin> -35.0
nx> 9
ny> 7
nz> 10
xmax> 24.9
xmin> -30.0
ymin> -25.0

! X Y Z Fx Fy
    -3.00000000E+01 -2.50000000E+01 -3.50000000E+01 -3.32347616E+01 7.
    -1.0822860E+01 -2.97096247E+00 -1.54145628E+01 7.
    -2.39000000E+01 -2.50000000E+01 -3.50000000E+01 -3.41989531E+01 7.
    -1.78000000E+01 -1.88969342E+00 -2.68008252E+00 7.
    -1.17000000E+01 -1.40000000E+01 -1.35218479E+00 7.
    -5.60000000E+01 -2.50000000E+01 -3.50000000E+01 -3.71576482E+01 7.
    -3.64Fz
```

(continues on next page)
<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.27000000E+01</td>
<td>-2.50000000E+01</td>
<td>-3.50000000E+01</td>
<td>-2.51221541E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>6.86380434E+01</td>
<td>-3.38130113E+01</td>
<td>-3.50000000E+01</td>
<td>-3.52612368E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>1.88000000E+01</td>
<td>-2.50000000E+01</td>
<td>-3.50000000E+01</td>
<td>-2.67620595E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>9.40516920E+01</td>
<td>-1.91091320E+01</td>
<td>-3.50000000E+01</td>
<td>-2.84575134E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>2.49000000E+01</td>
<td>1.10000000E+01</td>
<td>-2.70000000E+01</td>
<td>-2.85845999E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>4.48795940E+01</td>
<td>-3.20843230E+01</td>
<td>-2.70000000E+01</td>
<td>-2.98548459E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>1.88000000E+01</td>
<td>1.10000000E+01</td>
<td>-2.70000000E+01</td>
<td>-2.98548459E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>4.48795940E+01</td>
<td>-2.20188533E+01</td>
<td>-2.70000000E+01</td>
<td>-2.98548459E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>1.78000000E+01</td>
<td>1.82000000E+01</td>
<td>-2.70000000E+01</td>
<td>-2.98548459E+01</td>
<td>5.00000000E+00</td>
</tr>
<tr>
<td>4.48795940E+01</td>
<td>-1.82000000E+01</td>
<td>-2.70000000E+01</td>
<td>-2.98548459E+01</td>
<td>5.00000000E+00</td>
</tr>
</tbody>
</table>

(continues on next page)
### BDSIM Field Format 4D

All of the 1D, 2D and 3D parameters, plus:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tmin</td>
<td>The lower spatial coordinate in t associated with the field map</td>
</tr>
<tr>
<td>tmax</td>
<td>The upper spatial coordinate in t associated with the field map</td>
</tr>
<tr>
<td>nt</td>
<td>Number of elements in t - 1 counting</td>
</tr>
</tbody>
</table>

There is an example in `bdsim/examples/features/fields/4_bdsimformat/tdexample.tar.gz`. 

```plaintext
-1.17000000E+01 1.82000000E+01 -2.70000000E+01 -2.47012207E+01 5.
-43971730E+01 -2.70122716E+01 -5.60000000E+00 1.82000000E+01 -2.35061087E+01 5.
-39802747E+01 -3.2398755E+01 5.00000000E-01 1.82000000E+01 -2.31266642E+01 5.
-37514900E+01 -3.06517719E+01 6.60000000E+00 1.82000000E+01 -2.36514705E+01 5.
-35807213E+01 -2.60843230E+01 1.27000000E+01 1.82000000E+01 -2.49584916E+01 5.
-35498480E+01 -1.03700517E+01 2.49000000E+01 1.82000000E+01 -2.67412608E+01 5.
-35078354E+01 -1.43505942E+01 1.88000000E+01 1.82000000E+01 -2.8584168E+01 5.
-3459480E+01 -1.90086356E+01 -3.00000000E+01 -2.50000000E+01 -1.72347616E+01 3.
-90822860E+01 -2.19120640E+01 -2.39000000E+01 -2.50000000E+01 -1.81989531E+01 3.
-95850542E+01 -2.1576482E+01 -9099195E+01 -8.36790554E+00 -1.93501533E+01 3.
-92901497E+01 -2.04196083E+01 -95099195E+01 -8.36790554E+00 -1.93501533E+01 3.
-79359256E+01 -2.13919737E+01 -95850542E+01 -8.36790554E+00 -1.93501533E+01 3.
-71927569E+01 -2.1067802E+01 -79359256E+01 -2.79287545E+01 -1.90000000E+01 3.
-64012859E+01 -2.11576482E+01 -71927569E+01 -2.1569769E+01 -1.90000000E+01 3.
-65377567E+01 -2.02610291E+01 -64012859E+01 -1.8556347E+01 -1.90000000E+01 3.
-82805617E+01 -2.061291E+01 -66380434E+01 -1.83556347E+01 -1.90000000E+01 3.
-3.00000000E+01 -1.78000000E+01 -1.78000000E+01 -1.7690671E+01 -1.95723220E+01 3.
-83914175E+01 -1.6147359E+01 -82805617E+01 -8.36790554E+00 -1.95723220E+01 3.
-84108947E+01 -1.43505942E+01 -83914175E+01 -8.36790554E+00 -1.95723220E+01 3.
```
BDSIM Field Format Different Dimensions

**Warning:** Only for BDSIM format field map files.

Different dimensions can be used but they must be in order. Below is a list of the allowable alternate dimensions for various field maps.

- **4D field:**
  - \(x, y, z, t\)

- **3D field:**
  - \(x, y, z\)
  - \(x, y, t\)
  - \(x, z, t\)
  - \(y, z, t\)

- **2D field:**
  - \(x, y\)
  - \(x, z\)
  - \(x, t\)
  - \(y, z\)
  - \(y, t\)
  - \(z, t\)

- **1D field:**
  - \(x\)
  - \(y\)
  - \(z\)
  - \(t\)

### 17.10.6 BDSIM Field Map File Preparation

The Python BDSIM utility `pybdsim` may be used to prepare a BDSIM format field map file from a Python numpy array.

The `pybdsim` field classes are fully documented in the [pybdsim documentation](http://www.pp.rhul.ac.uk/bdsim/pybdsim/).

### 17.10.7 Field Map Interpolators

A variety of interpolators are provided with BDSIM. Example data sets in 1D and 2D were generated with simple \(x, y, z\) field vector components that have different amplitudes and phased sinusoids shown below.

**Nearest Neighbour**

The nearest neighbour algorithm returns the field value of the closest defined point in the map and returns that value. Therefore, the interpolated map contains only the values of the original map. This only serves the purpose of being able to query the map at any set of coordinates and provides a ‘pixelated’ appearance and sharp discontinuities halfway between points in the map. This is intended only for completeness and debugging.
Fig. 15: Example 1D field value components.

Fig. 16: Example 2D field value components.
Fig. 17: Example 1D field value components with nearest neighbour interpolation.

Fig. 18: Example 2D field value components with nearest neighbour interpolation.
Linear

In this case, the interpolated value lies on a straight line between two given points. The field value $f$ at point $x_i$ lying between $x_a$ and $x_b$ is given by

$$xd = \frac{(x_i - x_a)}{(x_b - x_a)}$$

$$f(x_i) = f(x_a) (1 - xd) + f(x_b) xd$$

Here, $xd$ will lie in the range $[0, 1]$. This is, of course, a 1D equation and a version of linear interpolation. See Linear & Cubic Higher Dimension Interpolation for further details for 2, 3 and 4D interpolation.

![Fig. 19: Example 1D field value components with linear interpolation.](image1)

![Fig. 20: Example 2D field value components with linear interpolation.](image2)

Cubic

In this case, the surrounding four map entries of any given point are used in combination to give a small section of a cubic polynomial. For a given point $x_i$, the closest point which is on the lower-valued side is called $m_1$ (m for map), and the closest point which is on the higher-valued side is called $m_2$. Points further outside these
(in a 1D case) are called \(m_0\) and \(m_3\) respectively. (On a linear number scale from low to high they would be \(m_0, m_1, m_2, m_3\).) The field value \(f(x_i)\) is given by

\[
x_d = \frac{(x_i - x_a)}{(x_b - x_a)}
\]

\[
f(x_i) = m_1 + \frac{1}{2} x_d (m_2 - m_0) + x_d (2m_0 - 5m_1 + 4m_2 - m_3 + x_d (3(m_1 - m_2) + m_3 - m_0))
\]

Here, \(x_d\) will lie in the range \([0, 1]\).

This is, of course, a 1D equation and version of cubic interpolation. See [Linear & Cubic Higher Dimension Interpolation](#) for further details for 2,3 and 4D interpolation. One could of course cache the gradient at each point, but here it is calculated dynamically. This allows the 1D interpolation case to be used in different dimensions for different gradients and is not prohibitively slow.

![Example 1D field value components with cubic interpolation.](#)

**Note:** Although the \(x, y, z\) components are shown individually, they are in fact part of a 3-vector class that is used for interpolation, i.e. the components are not interpolated individually.
To interpolate both in a cubic polynomial and linear at greater than one dimension, the 1D interpolator can be used iteratively. In the case of 2D interpolation this would be called bilinear and bicubic, and in the case of 3D, trilinear and tricubic interpolation. Below is a diagram of a cube representing a point $C$ at an arbitrary point inside the eight corners that represent the closest values of the regular field map. The diagram shows this approximately in the centre of the cube, but it could lie anywhere inside the eight points.

$$\text{Fig. 23: Field map value coordinates for 3D interpolation.}^1$$

$C_{00}$ can be found by interpolating between $C_{000}$ and $C_{100}$. $C_{10}$, $C_{01}$, $C_{11}$ can be found in a similar manner with each of their edges. $C_0$ and $C_1$ can be found by then interpolating between $C_{00}$ and $C_{10}$ for example (in the case of $C_0$). $C$ can then be found by interpolating between $C_0$ and $C_1$, giving the desired value.

One may interpolate the dimensions in any order and arrive at the same result. By doing it in such a way, the 2D interpolator can use the 1D interpolator; the 3D interpolator can use the 2D interpolator etc. By ensuring the 1D case is correct, there is a much lower likelihood of implementation faults occurring for higher dimensional interpolators.

**Implementation Specifics**

To implement this iterative algorithm, $C$ arrays are used, as sub-arrays can be easily passed around, due to their underlying pointer nature in $C$. A small section of code from bdsim/src/BDSInterpolatorRoutines.cc is shown below:

```cpp
BDSFieldValue BDS::Cubic1D(const BDSFieldValue p[4],
const Gddouble x) {
}

BDSFieldValue BDS::Cubic2D(const BDSFieldValue p[4][4],
const Gddouble x, const Gddouble y) {
    BDSFieldValue arr[4];
    arr[0] = BDS::Cubic1D(p[0], y);
    arr[1] = BDS::Cubic1D(p[1], y);
    arr[2] = BDS::Cubic1D(p[2], y);
    arr[3] = BDS::Cubic1D(p[3], y);
    return BDS::Cubic1D(arr, x);
}
```

$^1$ Marmelad Cubic Diagram Wikipedia.
17.11 Beam Generation

BDSIM uses the CLHEP random number generator and associated classes to generate pseudo-random particle distributions for tracking.

17.11.1 Gauss Twiss Covariance Matrix

The beam in an accelerator can be described as a 6D distribution of particles. The six dimensions are three canonical conjugate coordinate pairs in three planes, and can be represented by the vector \( \mathbf{x} = (x, x', y, y', t, \delta) \). The means of that distribution are the beam centroids. The 6x6 matrix of second order moments of the distribution is referred to as the sigma matrix and can be represented compactly as \( \Sigma_{ij} = \langle x_i x_j \rangle \) with the angle brackets denoting a central moment. More explicitly for a particle beam, this is:

\[
\Sigma = \begin{pmatrix}
\langle x^2 \rangle & \langle x x' \rangle & \langle x y \rangle & \langle x y' \rangle & \langle x t \rangle & \langle x \delta \rangle \\
\langle x' x \rangle & \langle x'^2 \rangle & \langle x' y \rangle & \langle x' y' \rangle & \langle x' t \rangle & \langle x' \delta \rangle \\
\langle y x \rangle & \langle y x' \rangle & \langle y^2 \rangle & \langle y y' \rangle & \langle y t \rangle & \langle y \delta \rangle \\
\langle y' x \rangle & \langle y' x' \rangle & \langle y'^2 \rangle & \langle y' y' \rangle & \langle y' t \rangle & \langle y' \delta \rangle \\
\langle t x \rangle & \langle t x' \rangle & \langle t y \rangle & \langle t y' \rangle & \langle t^2 \rangle & \langle t \delta \rangle \\
\langle \delta x \rangle & \langle \delta x' \rangle & \langle \delta y \rangle & \langle \delta y' \rangle & \langle \delta t \rangle & \langle \delta^2 \rangle 
\end{pmatrix}
\]

The beam is typically parameterised in an accelerator using the Twiss functions. \( \beta, \alpha, \gamma \) and dispersions \( \eta_x, \eta'_x, \eta_y, \eta'_y \). These are used in combination with beam parameters \( \epsilon_x, \epsilon_y, \sigma_\delta, \sigma_t \). The combination of machine and beam parameters gives the physical observables such as beam size.

Please note that \( \sigma_\delta \) is used here for clarity, but its definition is equivalent to \( \sigma_E \) defined in Model Description - Input Syntax.

For linear optics, and in the absence of dispersion or energy spread, the three 2x2 submatrices along the diagonal of the sigma matrix fully represent each plane and can be treated independently of each other. For example, the horizontal plane is parameterised as:

\[
\begin{pmatrix}
\langle x^2 \rangle & \langle x x' \rangle \\
\langle x' x \rangle & \langle x'^2 \rangle 
\end{pmatrix}
= \epsilon_x \begin{pmatrix}
\beta_x & -\alpha_x \\
-\alpha_x & \gamma_x 
\end{pmatrix},
\]

When dispersion and energy spread are present, numerous couplings are introduced, which appear as off-diagonal elements in the sigma matrix. In this case, the energy dependence of the coordinates results in both longitudinal-transverse and transverse-transverse couplings.

To be a valid covariance matrix, the sigma (capital sigma) matrix, must be positive definite. This ensures a positive variance in each dimension.

To derive each term of the covariance matrix, we consider the effect of dispersion on single coordinates. For example, for an individual particle, the horizontal position is affected as follows:

\[
x_i = x_{\beta i} + \eta_x \delta_i,
\]

where \( x_i \) is the observed coordinate, \( x_{\beta} \) is the betatron component and \( \eta_x \delta_i \) reflects the change due to the particle’s relative energy offset in a dispersive region.

Because \( x_{\beta} \) and \( \delta_i \) are random variables and are independent by construction, the second order moment in \( x_{\beta} \) can be factorised using the properties of the variance

\[
\langle x^2 \rangle = \text{var}[x_{\beta} + \eta_x \delta] = \langle x_{\beta}^2 \rangle + \eta_x^2 \langle \delta^2 \rangle = \epsilon_x \beta_x + \eta_x^2 \sigma_\beta^2,
\]

where the last form is obtained by applying the Courant-Snyder equations and rewriting the variance in terms of standard deviation.

The covariance of the horizontal position and the energy offset can also be obtained as:

\[
\langle x \delta \rangle = \text{cov}[x_{\beta} + \eta_x \delta, \delta] = \langle x_{\beta} \delta \rangle + \eta_x \langle \delta^2 \rangle = \eta_x \langle \delta^2 \rangle,
\]

as \( \langle x_{\beta} \delta \rangle = 0 \) by construction as explained above.
Finally, correlations that involve two different variables affected by dispersion have the same form. For example:

\[
\langle xy \rangle = \text{cov}[x_\beta + \eta_x \delta, x'_\beta + \eta_{x'} \delta]
\]
\[
= \langle x_\beta x'_\beta \rangle + \eta_x \langle x_\beta \delta \rangle + \eta_{x'} \langle x'_\beta \delta \rangle + \eta_x \eta_{x'} \langle \delta^2 \rangle
\]
\[
= \langle x_\beta x'_\beta \rangle + \eta_x \eta_{x'} \langle \delta^2 \rangle
\]
\[
= -\epsilon_x \alpha_x + \eta_x \eta_{x'} \sigma_\delta^2
\]

All other couplings are done by analogy.

Some useful relations:

\[
\langle XY \rangle \equiv \text{cov}[X, Y],
\]
\[
\langle X^2 \rangle \equiv \text{cov}[X, X] \equiv \text{var}[X] \equiv \sigma_X^2,
\]
\[
\text{var}[X + Y] = \text{var}[X] + \text{var}[Y] \quad \text{if} \; X, Y \; \text{are independent},
\]
\[
\text{cov}[aX + bY, cW + dV] = ac\text{cov}[X, W] + ad\text{cov}[X, V] + bc\text{cov}[Y, W] + bd\text{cov}[Y, V].
\]

### 17.12 Sensitivity, Output & Analysis

#### 17.12.1 Information From Geant4

Geant4 allows the developer to gather information at every stage of the simulation, but is by default a quiet simulation with no output generated.

BDSIM collects information broadly in two ways. Firstly, through sensitive detector (“SD”) classes that are attached to individual volumes and make a ‘hit’, which is a snapshot of some desired information. Secondly, through trajectory information. Trajectories are created for the passage of a single particle throughout the whole geometry, whereas SDs only generate information for particles taking steps in the volumes they are attached to.

**Sensitive Detectors**

The role of a sensitive detector is to generate output information from steps taken in a volume. It is registered to a ‘logical volume’ and generates hits when particles pass through only the logical volume(s) it’s attached to.

These ‘hits’ are then stored in a ‘collection’ (i.e. vector) that can be processed at the end of an event.

BDSIM provides several SD classes to prepare different types of information. The two main sensitive detector classes in BDSIM are `BDSSDEnergyDeposition` and `BDSSDSampler`. These are responsible for generating energy deposition hits in accelerator components and for recording information in sampler volumes respectively. The hits are translated and put into output structures in the end of event action in BDSEventAction.

There is usually only one instance of the sensitive detector class even though it is attached to many logical volumes. These are held in the singleton class BDSSDManager.

- Each instance of a sensitive detector produces one collection.
- Hits should contain only the required information as there will be many (millions) of these objects.
- The energy deposition hit (`BDSHitEnergyDeposition`) is split in two parts - the default part and the optional part. This is to save transient memory by around 80% for these hits when only using basic energy deposition.

Collimators have their own sensitivity as extra collimator specific information can be stored. The sensitive detector is really two on top of each other. The regular energy deposition one is used then the collimator specific one. This is done to ensure consistent information between the two (the randomly chosen point along the step where the deposition ‘happens’). The collimator hits always require the full energy deposition hits, so the energy deposition part of the collimator hit is in a different collection of ‘full’ energy deposition hits. These are mixed (stored) with the simple energy deposition hits at the end to give the same information but with reduced transient memory usage.
Note: The developer must ensure consistent use of output options from global constants in both BDSSDManager and BDSOutput. i.e. the options should be used to control both the generation and the storage of hits. Only what’s needed for output should be generated in the first place.

Attachment of Sensitive Detectors

All beam line objects in BDSIM generally attach the sensitive detector they want at construction time. This is done through the BDSGeometryComponent base class that keeps a map of G4LogicalVolume* to BDSSDType, a class enumeration for which sensitive detector class to use.

During the placement of the beam lines in BDSDetectorConstruction, the SD classes are retrieved from BDSSDManager and attached to the G4LogicalVolumes. At this stage the user options are applied and if the hits are not required because some output options are turned off, the SDs will not be attached saving CPU time and memory during the simulation.

- Only SDs that are required are attached based on the options.

Trajectory Creation & Storage

Trajectory objects are created for each new track in the Geant4 user tracking action hook (in BDSTrackingAction). If turned on for all particles, this would record every piece of available information for the full simulation - an unmanageable amount of data in large studies. Therefore, this is by default only turned on for primary particles. The trajectory information is used by the Geant4 visualisers to view the tracks, so it must be turned on in this case too.

If the user turns on trajectory storage, trajectories are created for every particle in the event and then filtered to store only the ones of interest in the output in the end of event action in BDSEventAction.

17.12.2 Output

BDSIM writes currently to one format which is ROOT (we call it “root event” format). The code is designed so that other formats can be added in future without affecting the current structure of the program or output formats.

Generally, we keep all the information we might want to store in the output in a base class called BDSOutputStructures. This contains only the basic information that will be stored. BDSOutput inherits this class and is an abstract interface for any specific output format. BDSOutput helps fill the objects in BDSOutputStructures as well as define the interface for all output.

Any output format is represented by a class that inherits BDSOutput, such as BDSOutputROOT for example. We use ROOT objects and classes in BDSIM as they are convenient for preparing the output information from a particle physics study. BDSOutputROOT is relatively simple as it is relatively simple to store ROOT objects in a ROOT file. Any other output format that is written would have access to the information in the base class BDSOutputStructures and can translate and write that as required. We also define ROOT dictionaries and create shared libraries with all the classes used in the output so that they can be easily loaded again.

- Generally, we use SI units + GeV and ns in the output.

How A ROOT File Works

The ROOT file system works by have an instance of whatever you want to store that exists in the current memory. The file is laid out pointing to these local objects (roughly SetBranchAddress(‘name in output’, &localObject )). The developer then sets the values of the local object, i.e. to the data they want to store. The Fill function (on a ROOT “Tree”) is then called which copies the data to the file.

Loading the file is done in reverse. First a local empty object is created, the file is attached to it and GetEntry() is called which loads one entry from the output into the local object, which can then be read by the user just as if they’d created it then and there.
A ROOT file can store data as one of objects in the file (such as a histogram), but the most common usage is with a ‘Tree’ (TTree class), that is really equivalent to a vector. Whatever structure the tree has is duplicated for each ‘entry’. In a tree, there can be single objects or ‘branches’ with ‘leaves’ (so a maximum number of dimensions of 2). These objects may be basic C++ types or ROOT objects, or classes defined by the user. The ROOT file secretly stores a template of all classes stored in it, so even if a user class is used to write a file and later on, the software is lost, the data can still be read.

In BDSIM, the main output tree is called “Event” and each entry in that tree represents one event in the simulation that starts with one primary particle fired into the accelerator. Everything you see in the Event tree is replicated for each event but with different data of course.

### 17.13 Analysis Suite

#### 17.13.1 Executables

<table>
<thead>
<tr>
<th>Executable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rebdsim</td>
<td>Main analysis tool. Reads histogram definition file.</td>
</tr>
<tr>
<td>rebdsimCombine</td>
<td>Combines output from multiple runs of rebdsim</td>
</tr>
<tr>
<td>rebdsimHistomerge</td>
<td>Loops over events and calculates mean of histograms already store in output only. A quick way to get a summary of data.</td>
</tr>
<tr>
<td>rebdsimOptics</td>
<td>Calculates the optics only from a BDSIM ROOT file</td>
</tr>
<tr>
<td>rebdsimOrbit</td>
<td>Extracts the primary particle hit in each sampler for a given event</td>
</tr>
<tr>
<td>edbdsim</td>
<td>Event display</td>
</tr>
</tbody>
</table>

#### 17.13.2 Source Libraries

The classes stored in the output are compiled into the BDSIM library, however, they are also copied into a separate library called `libBdsimRootEvent` without any Geant4 types of linkages. These classes are `bdsim/include/BDSOutputROOTEvent*`.

A series of classes for loading the output in the structure as it is in the output files are included in `bdsim/analysis`. These are compiled into a library called `librebdsim`.

The user must load both libraries to fully inspect the data.

**Note:** No Geant4 types are exposed to the output as this would increase the complexity of the interface with ROOT and ROOT’s inspection of the types for reflection.

#### 17.13.3 Dynamic Loading

Loading all of the data at once results in slow performance and is often unnecessary. For the event tree where the majority of the data is stored, it is advantageous to only load the ‘branches’ required for a particular analysis. Turning off branches means that when `GetEntry(n)` is called on the tree, nothing is copied out of the file onto the local objects. We therefore turn ‘on’ the branches we need for a particular analysis.

The analysis tool `rebdsim` works out the branches required when parsing histogram definitions and these are stored in a map of branch names for each tree. The `DataLoader` class takes this as an optional argument and turns on the appropriate branches.

The default arguments for the `DataLoader` class are such that all data is loaded. It is non-trivial and requires a lot of typing to define the map necessary for efficient loading, which hinders a user when inspecting data interactively. Therefore, all data is on when loading interactively.
17.13.4 Interactive Usage

The code used for analysis must be maintained in such a way that is both useful for the compiled analysis tools, but also simple to use interactively.
• genindex
• modindex
• search