
BDSIM Documentation

Release 0.9.2

BDSIM Collaboration

March 29, 2016

CONTENTS

1	Licence & Disclaimer	3
1.1	Licence	3
1.2	Disclaimer	3
1.3	Geant4 Acknowledgement	3
1.4	Funding Acknowledgements	3
2	Authorship	5
2.1	Current Authors	5
2.2	Past Authors	5
3	Introduction	7
3.1	Purpose of BDSIM	7
3.2	General Simulation Steps	7
3.3	A Little More Detail	8
4	Installation	9
4.1	Obtaining BDSIM	9
4.2	Requirements	9
4.3	Setting Up	10
4.4	Troubleshooting	16
5	Running BDSIM	19
5.1	Interactively	19
5.2	In Batch Mode	20
5.3	Examples	21
6	Model Description - Input Syntax	23
6.1	GMAD Syntax	23
6.2	Coordinates & Units	23
6.3	Useful Commands	24
6.4	Lattice Description	25
6.5	Lattice Elements	25
6.6	Aperture Parameters	37
6.7	Magnet Geometry Parameters	37
6.8	Offsets & Tilts - Component Misalignment	44
6.9	Lattice Sequence	45
6.10	Samplers - Output	45
6.11	Physics Processes	46
6.12	Physics Biasing	48
6.13	Options	48
6.14	Beam Parameters	50
6.15	Tunnel Geometry	55
6.16	Material and Atoms	56
6.17	Regions	58

7	Extended Geometry	59
7.1	gmad	59
7.2	Mokka	61
7.3	GDML	67
7.4	LCDD	68
8	Model Preparation	69
8.1	Manual Preparation	69
8.2	MADX Conversion	69
8.3	MAD8 Conversion	70
8.4	Python Builder	70
9	Output	71
9.1	Output Format Differences	71
9.2	Histograms	72
9.3	Samplers	72
9.4	Primary Coordinates	73
10	Output Analysis	75
10.1	ROOT Output (robdsim)	75
10.2	Converting ROOT trees as numpy arrays	76
10.3	ROOT classes and structure	76
10.4	Extending ROOT analysis	76
10.5	Examples	76
10.6	ASCII Output	77
11	Python Packages	79
11.1	Installing Python Packages	79
11.2	Documentation	79
12	Visualisation	99
12.1	Default and Custom Visualisation	99
12.2	Visualisation Features	100
13	Examples & Tests	103
13.1	Features	103
13.2	Accelerator Test Facility 2 - KEK, Japan	114
13.3	International Linear Collider	114
13.4	Large Hadron Collider	114
13.5	Simple Machine	114
14	Support	117
14.1	Trouble Running BDSIM	117
14.2	Feature Request	117
15	BDSIM Version History	119
15.1	V0.92 - 2016 / 03 / 29	119
15.2	V0.91 - 2015 / 12 / 17	120
15.3	V0.9 - 2015 / 11 / 10	122
15.4	V0.8 - 2015 / 08 / 10	123
15.5	V0.702 2015 / 07 / 28 - Hotfix	125
15.6	V0.701 2015 / 07 / 02 - Hotfix	125
15.7	V0.7 - 2015 / 06 / 30	125
15.8	V0.65 - 2015 / 04 / 10	126
15.9	V0.64 - 2015 / 02 / 16	126
15.10	V0.63 - 2015 / 02 / 06	126
15.11	V0.62 - 2014 / 08 / 07	126
15.12	V0.61 - 2014 / 08 / 05	126
15.13	v0.6 - 2013 / 12 / 02	127

15.14 v0.5 - 2008 / 11 / 08	127
15.15 v0.4 - 2008 / 02 / 26	127
15.16 v0.3 - 2007 / 01 / 26	127
15.17 v0.2 - 2006 / 05 / 18	127
15.18 v0.1 - 2006 / 02 / 22	127
15.19 beta - 2005 / 05 / 01	127
16 Developer Documentation	129
16.1 Purpose of Developer Documentation	129
16.2 Style Guide	129
16.3 Release Checklist	131
16.4 Program Layout	131
16.5 Build System & Testing	132
16.6 Parser	132
16.7 Geant4 User Action Classes	133
16.8 Geometry	134
16.9 Fields	137
16.10 Tracking Algorithms	137
16.11 Sensitivity, Output & Analysis	138
16.12 Analysis Suite	138
16.13 Indices and tables	138
Python Module Index	139
Index	141

Contents:

LICENCE & DISCLAIMER

Beam Delivery Simulation (BDSIM) Copyright (c) Royal Holloway, University of London, 2001 - 2015.

1.1 Licence

Beam Delivery Simulation (BDSIM) - this software - is copyright to and provided by Royal Holloway, University of London. All rights reserved.

1.2 Disclaimer

This software is provided “AS IS” and an express or limit warranties, including, but not limited to, implied warranties of merchantability, of satisfactory quality, and fitness for a particular purpose or use are disclaimed.

1.3 Geant4 Acknowledgement

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

1.4 Funding Acknowledgements

BDSIM has received funding from the following sources:

- John Adams Institute

AUTHORSHIP

BDSIM was originally started G.A. Blair in around 2001 and has since been developed and maintained by a large group, largely based at Royal Holloway, University of London.

2.1 Current Authors

- Laurie Nevay (*RHUL, lead developer*)
- Jochem Snuverink (*RHUL, lead developer*)
- Stewart Boogert (*RHUL, lead developer*)
- Lawrence Deacon (*RHUL, CERN, UCL*)
- William Shields (*RHUL*)
- Hector Morales (*RHUL, CERN*)
- Regina Kwee-Hinzmann (*RHUL, CERN*)
- Stephen Gibson (*RHUL*)
- Stuart Walker (*RHUL*)
- Andrey Abramov (*RHUL*)

2.2 Past Authors

- Grahame Blair (*RHUL*)
- Ilya Agapov (*RHUL*)
- John Carter (*RHUL*)
- Stephen Malton (*RHUL*)
- Robert Ainsworth (*RHUL*)
- Daniel Brewer (*RHUL*)
- Helmut Burkhardt (*CERN*)
- Olivier Dadoun (*LAL*)
- G. Marchiori (*INFN*)
- Will Parker (*RHUL*)
- Jaime Van Oers (*RHUL*)

INTRODUCTION

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.

3.1.1 What BDSIM is suitable for

- Single particle Monte-Carlo simulations of particle accelerators
- Simulating beam loss in a particle accelerator
- 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 SixTrack or PTC

3.1.3 Example Applications

- LHC beam loss simulation
- CLIC muon backgrounds studies
- Laserwire signal to background ratio
- ILC collimator efficiency study and detector backgrounds

3.2 General Simulation Steps

1. Create an text input **.gmad** lattice for BDSIM by converting a **MADX** or **MAD8** twiss file.
2. Run BDSIM with core beam distribution for validation.
3. Analyse data from 2) to reproduce optical functions of lattice as validation of a correct model
4. Run BDSIM with desired input distribution and physics processes either as a single instance or on a farm.
5. Analyse output data as desired.

3.3 A Little More Detail

BDSIM uses **ASCII** text input with a syntax designed to be very similar to **MAD8 / MADX**. The user prepares a representation of the accelerator lattice they wish to simulate by defining magnets or various accelerator components and the sequence they should appear in. Additionally, the user may set options describing, for example, energy tracking cuts, which physics processes are of importance and at which locations to record output.

BDSIM can then use the input file to simulate the passage of the desired number of particles and how they interact with the accelerator components themselves. Should a particle hit the beampipe, the physics processes of Geant4 will be used to calculate the interaction with the beampipe and secondary particles that may be produced. Particles are recorded at user specified ‘sampling’ planes and energy deposition through the accelerator is recorded in both discrete ‘hits’ and a histogram as a function of distance along the accelerator.

One may generally write their own C++ program to simulate the setup of geometry and magnetic fields they like, however, in the case of an accelerator the typical geometry is highly repetitive and usually consists of either a beampipe or a beampipe with a magnet surrounding it. BDSIM generates Geant4 geometry automatically and provides integrators for linear magnetic fields in place of the normal Runge-Kutta integrators used by Geant4. Equations of motion describing particle motion in magnetic fields such as that of a quadrupole or sector-bend have analytical solutions that can be used in place of numerical integration. BDSIM uses these to provide fast and accurate thick lens tracking in vacuum.

BDSIM provides a library of fairly generic magnet geometry styles that will cover most cases. Should a more detailed geometry be required, the user may supply this in various formats as well as magnetic field maps. Alternatively, if the user wishes to write their own Geant4 geometry for a particular component this can be integrated into BDSIM relatively easily.

INSTALLATION

4.1 Obtaining BDSIM

BDSIM may be obtained either from the BDSIM website from the git repository (<https://www.bitbucket.org/jairhul/bdsim>). The user must compile it on their system and must have Geant4 already present (or access to AFS).

4.1.1 From the GIT Repository

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

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

This will create a directory called `bdsim`, inside which all the code, examples and documentation is provided. To obtain the python utilities that come with BDSIM, use the following commands:

```
cd bdsim
git submodule init
git submodule update
```

4.1.2 From precompiled sources

BDSIM may also be downloaded from pre-compiled sources. These are available on: <http://www.pp.rhul.ac.uk/bdsim/download>

4.1.3 AFS

With AFS connection you can get the latest released `bdsim` version from:

```
/afs/cern.ch/user/j/jsnuveri/public/bdsim
```

The latest develop version (updated daily) is available under:

```
/afs/cern.ch/user/j/jsnuveri/public/bdsim-develop
```

As usual the Geant4 environment script needs to be loaded:

```
source /afs/cern.ch/user/j/jsnuveri/public/geant4.10.2-setup.sh
```

4.2 Requirements

0. A recent compiler. Proven compiler versions are gcc 4.9 or higher, or clang 6 or higher.

1. [Geant4](#) installed or access to [AFS](#)¹. Version 4.10 or higher. See [Geant4 Installation Guide](#)
2. [CMake](#) 2.6.4 or higher (Geant4.10.2 - the latest - requires [CMake](#) 3.3 or higher.)
3. [Flex](#) 2.5.37 or higher
4. [Bison](#) 2.3 or higher
5. [CLHEP](#) 2.1.3.1 or higher, see also [CLHEP Installation Guide](#)
6. [ROOT](#) framework for output analysis.

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

```
> echo $G4 <tab>
$G4ABLADATA      $G4NEUTRONHPDATA    $G4RADIOACTIVEDATA
$G4LEDDATA       $G4NEUTRONXSDATA    $G4REALSURFACEDATA
$G4LEVELGAMMADATA $G4PIIDATA             $G4SAIDXSDATA
```

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
```

Note, if Geant4 is not installed, please see [Geant4 Installation Guide](#).

4.3 Setting Up

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

- [Mac OSX](#)
- [Linux](#)
- [Linux with AFS Access](#)

4.3.1 Mac OSX

We recommend obtaining [required packages](#) using [MacPorts](#) package manager, although they can be obtained both through other package managers and by manually downloading, compiling and installing the source for each.

After this, [Building](#) can be started.

4.3.2 Linux

Install the [required packages](#) preferably with a package manager.

Older version 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
```

After this, [Building](#) can be started.

¹ Note, the use of [AFS](#) with the Mac OSX build of BDSIM is not supported as there is no compatible version of Geant4 available on AFS.

4.3.3 Linux with AFS Access

When the machine has AFS connection, the latest stable release binary is available:

```
/afs/cern.ch/user/j/jsnuveri/public/bdsim
```

Before using the binary you must source the geant4 setup:

```
source /afs/cern.ch/user/j/jsnuveri/public/geant4.10-setup.sh
```

When compiling BDSIM from source, the dependent packages like Geant4 can be taken from AFS and don't need to be compiled and installed locally. The same compiler version needs to be used for BDSIM as the one that was used for Geant4. The following scripts must be sourced before using CMake.

For the versions 0.61 and onwards:

```
source /afs/cern.ch/user/j/jsnuveri/public/gcc49-setup.sh
source /afs/cern.ch/user/j/jsnuveri/public/geant4.10-setup.sh
```

For version 0.6 and older:

```
source /afs/cern.ch/user/j/jsnuveri/public/gcc46-setup.sh
source /afs/cern.ch/user/j/jsnuveri/public/geant4.9.6-setup.sh
```

After this, *Building* can be started.

4.3.4 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 as 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 6.0.0.6000056
-- The CXX compiler identification is AppleClang 6.0.0.6000056
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Configuring BDSIM 0.8
-- Build Type RelWithDebInfo
-- Compiler supports C++11
-- Looking for CLHEP... - found
-- Found CLHEP 2.2.0.5 in /opt/local/lib/CLHEP-2.2.0.5/../../include
-- Looking for ROOT...
-- Found ROOT 5.34/32 in /opt/local/libexec/root5
-- GDML support ON
-- Looking for XML2... - found
```

```
-- LCDD support ON
-- Geant4 Use File: /Users/nevay/physics/packages/geant4.10.00.p02-install
  /lib/Geant4-10.0.2/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: 10.1.1
-- Found Doxygen: /opt/local/bin/doxygen (found version "1.8.9.1")
-- Found BISON: /opt/local/bin/bison (found version "3.0.4")
-- Found FLEX: /opt/local/bin/flex (found version "2.5.37")
-- Configuring ROBDSIM 0.3.develop
-- Build Type RelWithDebInfo
-- Compiler supports C++11
-- Looking for ROOT...
-- Found ROOT 5.34/32 in /opt/local/libexec/root5
-- Found Sphinx: /opt/local/bin/sphinx-build-2.7
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/nevay/physics/repos/bdsim-build
```

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:

```
> make
```

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

```
> 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.

Note: This step is not strictly necessary. It is possible to create an alias to the executable `bdsim` that exists in the build directory in your shell profile. This is common practice for developers who may wish to have a debug build of the code as well as the normal release build.

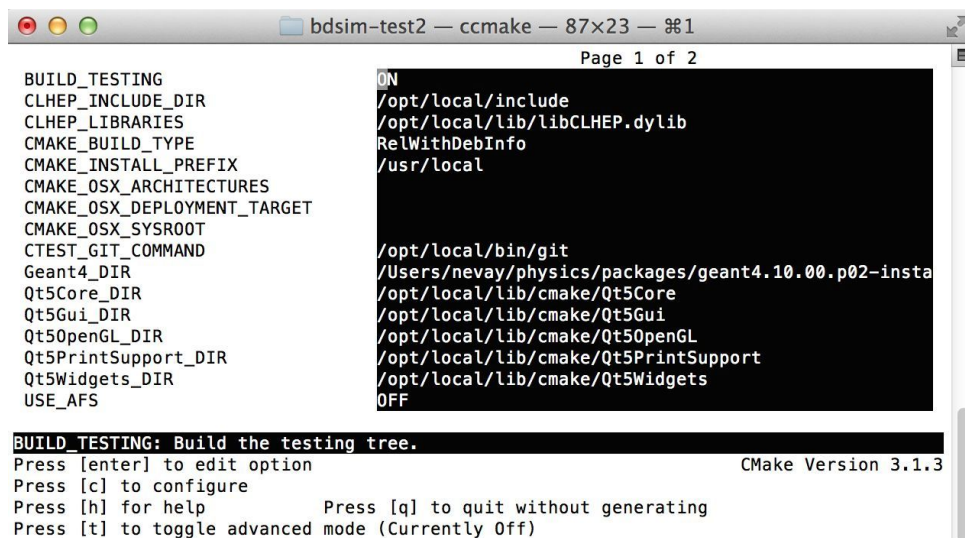
From the build directory you can verify your installation using a series of tests included with BDSIM (excluding long tests):

```
> ctest -E LONG
```

4.3.5 Configuring the BDSIM Build with CMake

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

```
> cmake .
```



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.

4.3.6 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 pdf_{latex} from the build directory using the following command:

```
> make manual
```

to make the HTML manual in the folder `manual/html`. Similarly:

```
> make manual-pdf
```

will make the pdf Manual in the folder `manual/latex`.

Note: This requires the sphinx documentation system to be installed and all utility python packages to be available in python from any directory. The latexpdf build requires a full installation of pdf_{latex} to be available as well.

4.3.7 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:

```
> make doc
```

to make the Doxygen documentation in a folder called Doxygen.

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

4.3.8 Geant4 Installation Guide

As of version 0.6, BDSIM builds with the most recent versions of Geant4 (version 4.10 onwards). If not built with **MacPorts** then download the 4.10.2 version or an older version from the Geant archive. Move and unpack to a suitable place

```
> tar -xzf geant4.10.2.tar.gz
> ls
geant4.10.2
```

Make a build and installation directory **outside** that directory

```
> mkdir geant4.10.2-build
> mkdir geant4.10.2-install
```

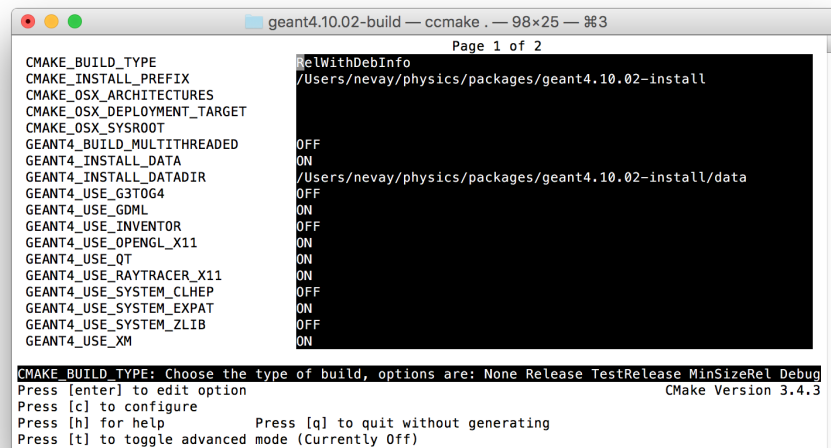
Configure Geant4 using CMake

```
> cd geant4.10.2-build
> cmake ../geant4.10.2
```

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](#).

```
> cmake .
```

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



Option	Description
CMAKE_INSTALL_PREFIX	Useful to specify to a known folder you make.
GEANT4_BUILD_CXXSTD	- For ROOT version 6 (and gcc compiler).
GEANT4_BUILD_MULTITHREADED	OFF - THREADED does not support this yet.
GEANT4_INSTALL_DATA	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 then.
GEANT4_INSTALL_DATA_DIR	Useful to specify to a known folder you make. Typically whatever CMAKE_INSTALL_PREFIX / data.
GEANT4_USE_GDML	ON - for external geometry import.
GEANT4_USE_OPENGL	ON - basic visualiser.
GEANT4_USE_QT	ON - the best and most interactive visualiser. Needs Qt to be installed
GEANT4_USE_RAYTRACE	ON - most accurate visualiser, but relatively slow and not interactive. Useful for promotional materials.
GEANT4_USE_XM	ON - similar to Qt and the one to use if Qt isn't available. Needs motif to be installed.

Make sure **GEANT4_BUILD_MULTITHREADED** is off since this is currently not supported. 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 computer². 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/geant4.10.2-install/bin/geant4.sh
```

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

4.3.9 CLHEP Installation Guide

If not installed with a package manager, download the [CLHEP-2.3.1.1](#) or 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:

² If your computer supports hyper-threading, you can use twice the number of cores with the `make -jN` command. If 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.

```
> cmake .
```

Make and install:

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

4.4 Troubleshooting

Below are 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:
ASCIITree (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, OGLS_FALLBACK, OGLSQt_FALLBACK)
RayTracer (RayTracer)
RayTracerX (RayTracerX)
VRML1FILE (VRML1FILE)
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. Error from OpenGL:

```
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>

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 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/lib/./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. Easiest might be to not use this part of BDSIM by switching off the CMake variable USE_LCDD (in ccmake).

RUNNING BDSIM

BDSIM can be executed in a terminal with extra arguments to specify various inputs. The angular brackets should not be used.

<code>-file=<file></code>	specify the input gmad file
<code>-batch</code>	batch mode - no graphics
<code>-circular</code>	assume circular machine - turn control
<code>-exportgeometryto=<file></code>	export the geometry to a file extension determines format where possible extensions are ("gdml")
<code>-generatePrimariesOnly</code>	generate primary particle coordinates only then exit without simulating anything
<code>-gflash=<N></code>	whether or not to turn on gFlash fast shower parameterisation.
<code>-gflashemax=<N></code>	maximum energy for gflash shower parameterisation in GeV.
<code>-gflashemin=<N></code>	minimum energy for gflash shower parameterisation in GeV.
<code>-help</code>	display this message.
<code>-materials</code>	list materials included in BDSIM by default
<code>-ngenerate=N</code>	the number of primary events to simulate overrides the ngenerate option in input file
<code>-output=<fmt></code>	output format "root", "ascii" (default), "combined" or "none"
<code>-outfile=<file></code>	output file name. Will be appended with _N where N = 0, 1, 2, 3...
<code>-seed=<N></code>	seed for the random number generator
<code>-seedstate=<file></code>	file containing CLHEP::Random seed state NB - this overrides other seed value
<code>-survey=<file></code>	print survey info to <file>
<code>-verbose</code>	display general parameters before run
<code>-verbose_event</code>	display information for every event
<code>-verbose_step</code>	display tracking information after each step
<code>-verbose_event_num=<N></code>	display tracking information for event N
<code>-vis_debug</code>	display all volumes in visualiser
<code>-vis_mac=<file></code>	file with the visualisation macro script, default provided by BDSIM: openGL (OGLSQt)

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. Alternatively, BDSIM can be run in batch mode, where no visualiser is used and the specified number of primary events is simulated and feedback 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 - ie where a particle hits.

5.1 Interactively

Features:

- default option

- interactive visualisation of accelerator model
- ability to view and rotate accelerator model
- ability to run individual events
- 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: Qt visualiser, extra convenient buttons). Once executed a window such as this will appear (depending on the visualiser you use).

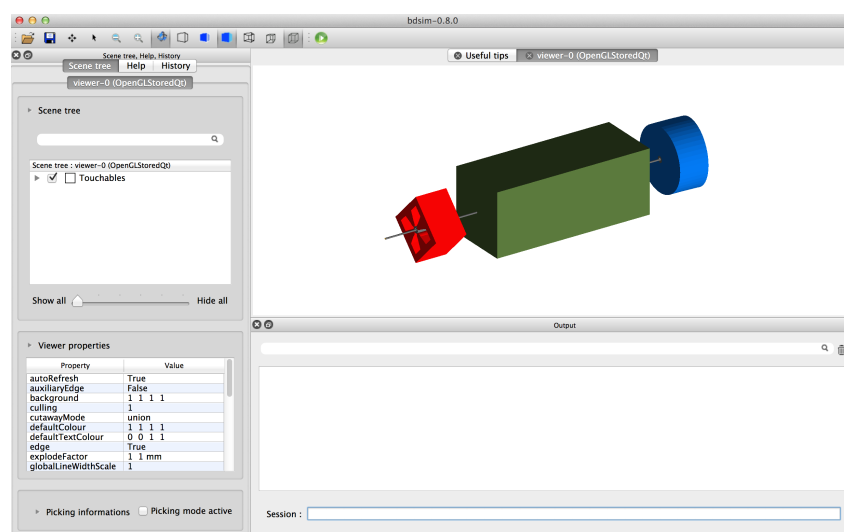


Fig. 5.1: BDSIM running interactively with OpenGL Qt visualiser from Geant4.

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.

More details can be found in [Visualisation](#).

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.

5.2 In Batch Mode

Features:

- no interactive visualiser
- faster
- number of events run then program quits
- no user input
- typical use for a “job” on a farm

To execute BDSIM in batch mode, simply use the `--batch` execution option.

5.3 Examples

```
bdsim --file=atf2.gmad --output=root --outfile=test1 --batch --seed=123
```

This executes BDSIM for the ATF2 example with ROOT output to a file name “test1” in batch mode with a seed value of 123. The simulation runs the number of events specified by the `ngenerate` options parameter in the input gmad file.

```
bdsim --file=sm.gmad --outfile=test2
```

This executes BDSIM for the simpleMachine example with ASCII output (default) to a file named “test2”. The program is run interactively and the window in Figure appears. From here, the user types:

```
/run/beamOn 1
```

In the visualiser terminal to run one event and visualise it.

MODEL DESCRIPTION - INPUT SYNTAX

6.1 GMAD Syntax

GMAD is a language specifically for BDSIM but is made to be human readable and very similar to MADX.

- arithmetic expressions can be defined
- binary operators +, -, *, /, ^ are valid
- unary operators +, -, are valid
- boolean operators <, >, <=, >=, <>, == are valid
- every expression **must** end with a semi-colon;
- no variable name can begin with a number

The following functions are provided

- sqrt
- cos
- sin
- tan
- exp, e
- log
- acos
- asin
- abs

Examples:

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

6.2 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

Name	Units
length	[m] (metres)
time	[s] (seconds)
angle	[rad] (radians)
quadrupole coefficient	[m ⁻²]
multipole coefficient 2n poles	[m ⁻ⁿ]
electric voltage	[MV] (Megavolts)
electric field strength	[MV/m]
particle energy	[GeV]
particle mass	[GeV/c ²]
particle momentum	[GeV/c ²]
beam current	[A] (Amperes)
particle charge	[e] (elementary charges)
emittances	[pi m mrad]
density	[g/cm ³]
temperature	[K] (Kelvin)
pressure	[atm] (atmosphere)
frequency	[Hz] (Hertz)
mass number	[g/mol]

Some useful predefined values / units are:

Name	Value
pi	3.14159265358979
GeV	1
eV	10 ⁻⁹
keV	10 ⁻⁶
MeV	10 ⁻³
TeV	10 ³
MV	1
Tesla	1
rad	1
mrad	10 ⁻³
urad	10 ⁻⁶
clight	2.99792458 × 10 ⁸
km	10 ³
m	1
cm	10 ⁻²
mm	10 ⁻³
um	10 ⁻⁶
nm	10 ⁻⁹
pm	10 ⁻¹²
s	1
ms	10 ⁻³
us	10 ⁻⁶
ns	10 ⁻⁹
ps	10 ⁻¹²
Hz	1
kHz	10 ³
MHz	10 ⁶
GHz	10 ⁹

For example, one can write either 100*eV or 0.1*keV to specify an energy in GMAD and both are equivalent.

6.3 Useful 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 `option`
- `print, parameter;` prints the value of `parameter`, where `parameter` could be your own defined parameter
- `length = d1["l"];` way to access properties of elements, in this case length of element `d1`.
- `stop;` or `return;` exists parser
- `if () {};` if construct

6.4 Lattice Description

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

1. Component definition
2. Sequence definition (of the already defined components)
3. Which sequence to use
4. Where to record output (samplers)
5. A beam distribution
6. Options, including which physics lists, number to simulate etc.

These are described in the following sections

6.5 Lattice Elements

Any element in BDSIM is described with the following pattern:

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

Note: Notice the ‘:’, the 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*
- *vkick*
- *hkick*
- *rf*

- *rcol*
- *ecol*
- *degrader*
- *muspoiler*
- *solenoid*
- *laser*
- *transform3d*
- *element*
- *marker*

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 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 MADX 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, MADX treats all particles as positively charged for tracking purposes.

New in version 0.7: BDSIM currently treats *k* absolutely so to convert a MADX lattice for negatively particles, the MADX *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 MADX.

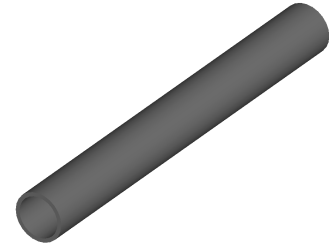
6.5.3 drift

drift defines a straight beam pipe with no field.

parameter	description	default	required
<i>l</i>	length [m]	0	yes
<i>vacuumMaterial</i>	the vacuum material to use, can be user defined	vacuum	no

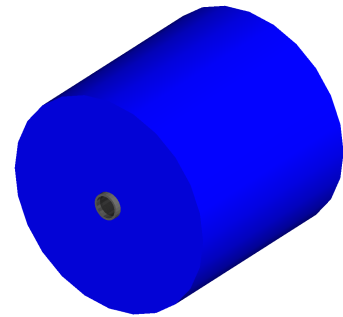
- The *aperture parameters* may also be specified.

Examples:



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

6.5.4 rbend



rbend defines a rectangular bend magnet. Either the total bending angle, *angle* for the nominal beam energy can be specified or the magnetic field, *B* in Tesla. *B* overrides *angle*. The faces of the magnet are normal to the chord of the input and output point. Pole face rotations can be applied to both the input and output faces of the magnet, based upon the reference system shown in the above image.

parameter	description	default	required
<i>l</i>	length [m]	0	yes
<i>angle</i>	angle [rad]	0	yes, or <i>B</i>
<i>B</i>	magnetic field [T]	0	yes
<i>e1</i>	input poleface angle [rad]	0	no
<i>e2</i>	output poleface angle [rad]	0	no
<i>material</i>	magnet outer material	Iron	no

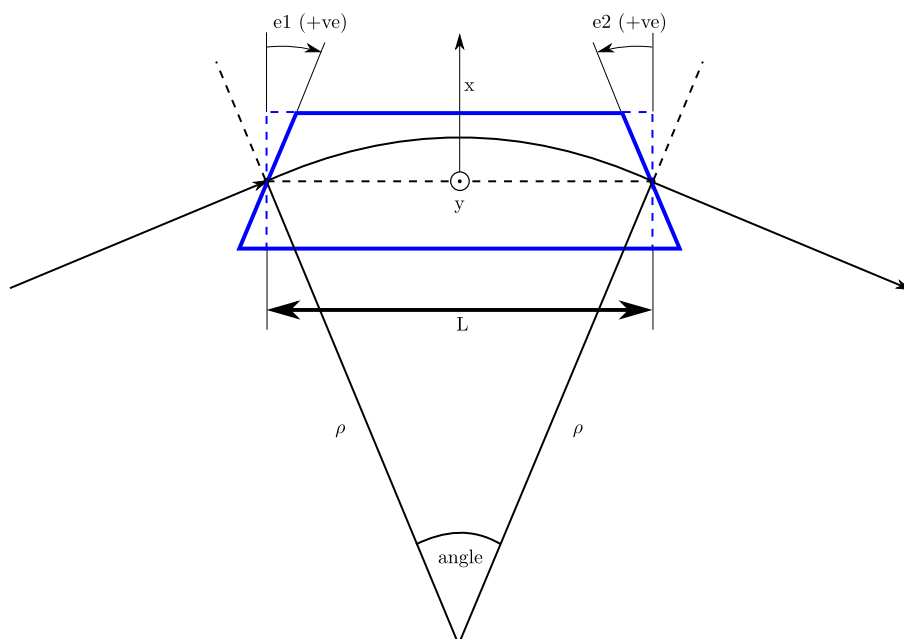
- The *aperture parameters* may also be specified.
- The *magnet geometry parameters* may also be specified.

Note: 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.

Note: As of v0.64 a combined quadrupole component is not possible, but is under development

Note: The poleface rotation angle is limited to $\pm\pi/4$ radians.

Note: If a non-zero poleface rotation angle is specified, the element preceding / succeeding the rotated magnet

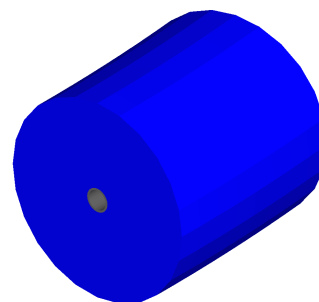


face must either be a drift or an rbend with opposite rotation (e.g. an sbend with $e2 = 0.1$ can be followed by an sbend with $e1 = -0.1$). The preceding / succeeding element must be longer than the projected length from the rotation, given by $2 \tan(eX)$.

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
```

6.5.5 sbend

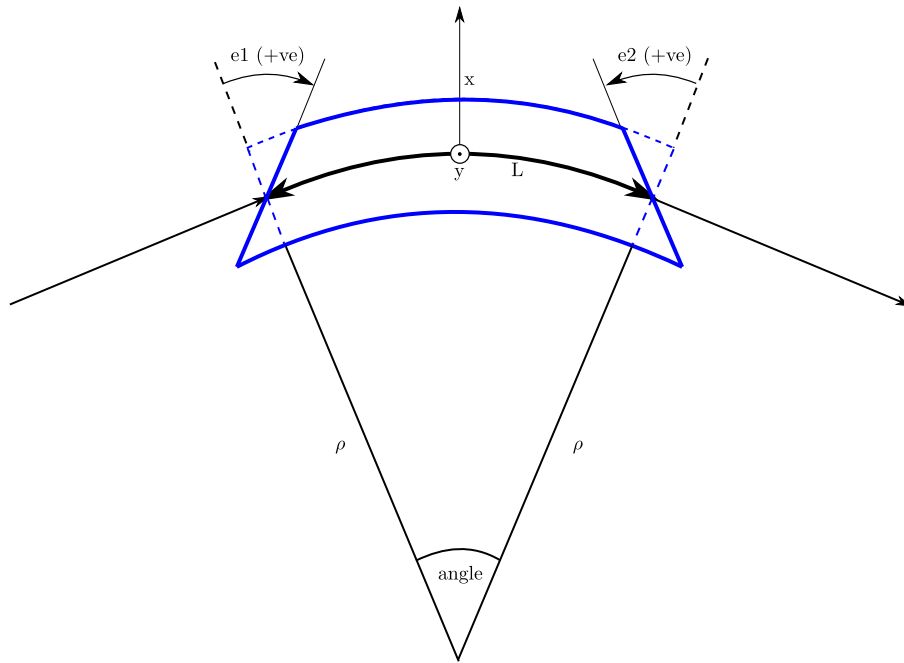


sbend defines a sector bend magnet. Either the total bending angle, *angle* for the nominal beam energy can be specified or the magnetic field, *B* in Tesla. *B* overrides *angle*. The faces of the magnet are normal to the curvilinear coordinate system. *sbend* magnets are made of a series of straight segments. If the specified (or calculated from *B* field) bending angle is large, the *sbend* is automatically split such that the maximum tangential error in the aperture is 1 mm. Sbend magnets are typically split into several co-joined *sbend* magnets, the number depending on the magnet length and bending angle. Pole face rotations can be applied to both the input and output faces of the magnet, based upon the reference system shown in the above image.

parameter	description	default	required
l	length [m]	0	yes
$angle$	angle [rad]	0	yes, or B
B	magnetic field [T]	0	yes
$e1$	input poleface angle [rad]	0	no
$e2$	output poleface angle [rad]	0	no
$material$	magnet outer material	Iron	no

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

Note: As of v0.64 a combined quadrupole component is not possible, but is under development



Note: The poleface rotation angle is limited to $\pm\pi/4$ radians.

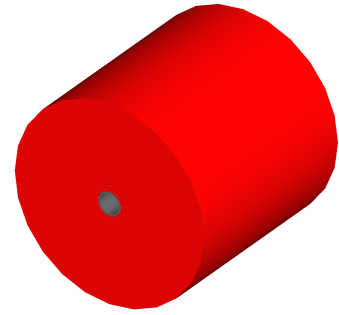
Note: If a non-zero poleface rotation angle is specified, the element preceding / succeeding the rotated magnet face must either be a drift or an rbend with opposite rotation (e.g. an sbend with $e2 = 0.1$ can be followed by an sbend with $e1 = -0.1$). The preceding / succeeding element must be longer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

Examples:

```
s1: sbend, l=14.5*m, angle=0.005, magnetGeometryType="lhcrigh";
mb201x: sbend, l=304.2*cm, b=1.5*Tesla;
SB17A: sbend, l=0.61*m, angle=0.016, e1=-0.05, e2=0.09
```

6.5.6 quadrupole

quadrupole defines a quadrupole magnet. The strength parameter $k1$ is defined as $k1 = 1/(B\rho) dB_y / dx [m^{-2}]$.



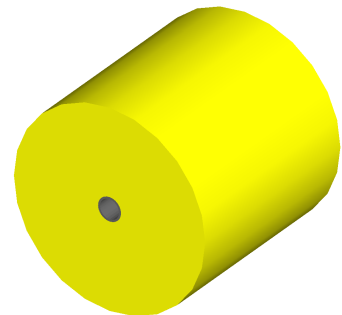
parameter	description	default	required
l	length [m]	0	yes
$k1$	quadrupole coefficient	0	yes
<i>material</i>	magnet outer material	Iron	no

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

Examples:

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

6.5.7 sextupole



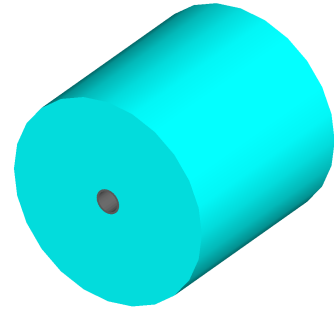
sextupole defines a sextupole magnet. The strength parameter $k2$ is defined as $k2 = 1/(B\rho) dB_y^2 / dx^2 [m^{-3}]$.

parameter	description	default	required
l	length [m]	0	yes
$k2$	sextupole coefficient	0	yes
<i>material</i>	magnet outer material	Iron	no

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

Examples:

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



6.5.8 octupole

octupole defines an octupole magnet. The strength parameter $k3$ is defined as $k3 = 1/(B\rho) dB_y^3 / dx^3 [m^{-4}]$.

parameter	description	default	required
l	length [m]	0	yes
$k3$	octupole coefficient	0	yes
<i>material</i>	magnet outer material	Iron	no

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

Examples:

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

6.5.9 decapole

decapole defines a decapole magnet. The strength parameter $k4$ is defined as $k4 = 1/(B\rho) dB_y^4 / dx^4 [m^{-5}]$.

parameter	description	default	required
l	length [m]	0	yes
$k4$	decapole coefficient	0	yes
<i>material</i>	magnet outer material	Iron	no

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

Examples:

```
MXDEC3: decapole, l=0.3*m, k4=32.9;
```

6.5.10 multipole

multipole defines a general multipole magnet. The strength parameter knl is a list defined as $knl[n] = 1/(B\rho) dB_y^n / dx^n [m^{-(n+1)}]$ starting with the quadrupole component. The skew strength parameter ksl is a list representing the skew coefficients.

parameter	description	default	required
l	length [m]	0	yes
knl	list of normal coefficients	0	no
ksl	list of skew coefficients	0	no
<i>material</i>	magnet outer material	Iron	no

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

Examples:

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

6.5.11 vkick

vkick or *vkicker* defines a vertical dipole magnet and has the same parameters as *sbend*.

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

Examples:

```
KX15v: vkick, angle=0.01*mrad;
```

6.5.12 hkick

hkick or *hkicker* defines a horizontal dipole magnet and has the same parameters as *sbend*.

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

Examples:

```
KX17h: hkick, angle=0.01;
```

6.5.13 rf

rf or *rfcavity* defines an rf cavity

parameter	description	default	required
<i>l</i>	length [m]	0	yes
<i>gradient</i>	field gradient [MV/m]	0	yes
<i>material</i>	outer material	Iron	no

- The *aperture parameters* may also be specified.

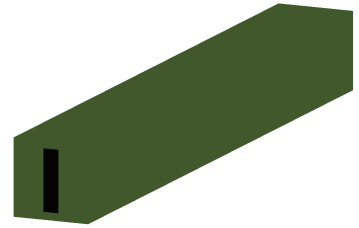
Note: Be careful with the sign of the gradient with respect to the sign of the primary particle

Examples:

```
RF4f: rf, l=3*m, gradient=10*MV;
```

6.5.14 rcol

rcol defines a rectangular collimator. The aperture is rectangular and the eternal volume is square.



parameter	description	default	required
<i>l</i>	length [m]	0	yes
<i>xsize</i>	horizontal half aperture [m]	0	yes
<i>ysize</i>	vertical half aperture [m]	0	yes
<i>xsizeOut</i>	horizontal exit half aperture [m]	0	no
<i>ysizeOut</i>	vertical exit half aperture [m]	0	no
<i>material</i>	outer material	Iron	no
<i>outerDiameter</i>	outer full width [m]	global	no

Note: *rcol* and *ecol* do not currently implement tilt, so if an angled collimator is required, a *transform3d* should be before and afterwards in the sequence to rotate the coordinate frame before and afterwards. See [transform3d](#) for further details and examples.

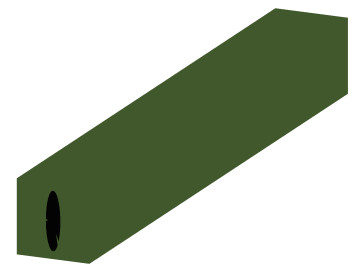
The collimator can be tapered by specifying an exit aperture size with *xsizeOut* and *ysizeOut*, with the *xsize* and *ysize* parameters then defining the entrance aperture.

Examples:

```
! 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;
```

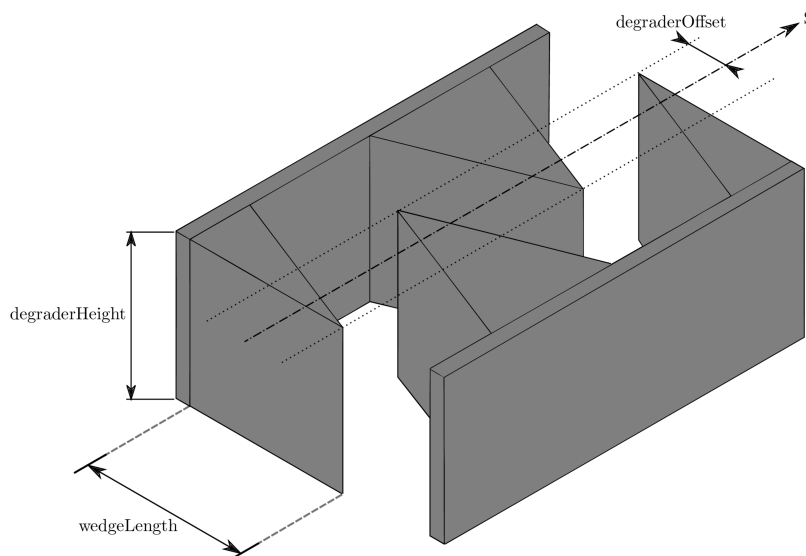
6.5.15 ecol



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.

6.5.16 degrader

degrader defines an interleaved pyramidal degrader that decreases the beam's energy.



parameter	description	default	required
<i>l</i>	length [m]	0	yes
<i>numberWedges</i>	number of degrader wedges	1	yes
<i>wedgeLength</i>	degrader wedge length [m]	0	yes
<i>degraderHeight</i>	degrader height [m]	0	yes
<i>materialThickness</i>	amount of material seen by the beam [m]	0	yes/no*
<i>degraderOffset</i>	horizontal offset of both wedge sets	0	yes/no*
<i>material</i>	degrader material	Carbon	yes
<i>outerDiameter</i>	outer full width [m]	global	no

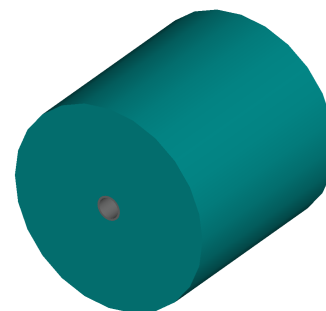
Note: Either *materialThickness* or *degraderOffset* can be specified to adjust the horizontal lateral wedge position, and consequently the total material thickness the beam can propagate through. If both are specified, *degraderOffset* will be ignored.

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=1
DEG2: degrader, l=0.25*m, material="carbon", numberWedges=5, wedgeLength=100*mm, degraderHeight=1
```

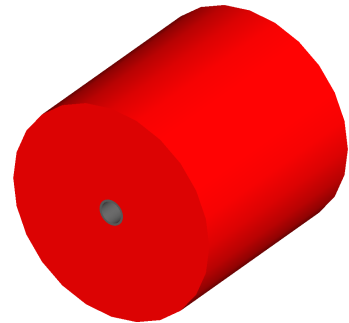
6.5.17 muspoiler



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.

parameter	description	default	required
<i>l</i>	length [m]	0	yes
<i>B</i>	magnetic field [T]	0	yes
<i>material</i>	outer material	Iron	no
<i>outerDiameter</i>	outer full width [m]	global	no

6.5.18 solenoid



solenoid defines a solenoid magnet. This utilises a thick lens transfer map with a hard edge field profile so it is not equivalent to split a single solenoid into multiple smaller ones. **This is currently under development.** The strength parameter *ks* is defined as $ks =$.

parameter	description	default	required
<i>l</i>	length [m]	0	yes
<i>ks</i>	solenoid strength []	0	yes
<i>material</i>	outer material	Iron	no
<i>outerDiameter</i>	outer full width [m]	global	no

- See [Magnet Strength Polarity](#) for polarity notes.

Examples:

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

6.5.19 laser

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

parameter	description	default	required
<i>l</i>	length of drift section [m]	0	yes
<i>x, y, z</i>	components of laser direction vector (normalised)	(1,0,0)	yes
<i>waveLength</i>	laser wavelength [m]	532*nm	yes

Examples:

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

6.5.20 transform3d

transform3d defines an arbitrary 3-dimensional transformation of the the curvilinear coordinate system at that point in the beam line sequence. This is often used to rotate components by a large angle.

parameter	description	default	required
<i>x</i>	x offset	0	no
<i>y</i>	y offset	0	no
<i>z</i>	z offset	0	no
<i>phi</i>	phi Euler angle	0	no
<i>theta</i>	theta Euler angle	0	no
<i>psi</i>	psi Euler angle	0	no

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

Examples:

```
rclrot: transform3d, psi=pi/2;
```

6.5.21 element

element defines an arbitrary element that's defined by external geometry and magnetic field maps. 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 size of diameter.

parameter	description	default	required
<i>geometry</i>	filename of geometry	NA	yes
<i>l</i>	length	NA	yes
<i>outerDiameter</i>	diameter of component [m]	NA	yes
<i>bmap</i>	filename of magnetic field map	NA	no

geometry and *bmap* require the input string to be of the format *format:filename*, where *format* is the geometry format being used (*gdml* | *gmad* | *mokka*) and filename is the filename of the geometry file.

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 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.

Examples:

```
detector: element, geometry="gdml:atlasreduced.gdml", outerDiameter=10*m, l=44*m;  
detec: element, geometry="mokka:qq.sql", bmap = "mokka:qq.bmap", l=5*m, outerDiameter=0.76*m;
```

For specific details on the geometry format, see [Extended Geometry](#)

6.5.22 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 [samplers - output](#)) 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:

```
m1: 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 MADX convention and have four parameters and 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.

Aperture Model	# of parameters	<i>aper1</i>	<i>aper2</i>	<i>aper3</i>	<i>aper4</i>
<i>circular</i>	1	radius	NA	NA	NA
<i>rectangular</i>	2	x half width	y half width	NA	NA
<i>elliptical</i>	2	x semi-axis	y semi-axis	NA	NA
<i>lhc</i>	3	x half width of rectangle	y half width of rectangle	radius of circle	NA
<i>lhcdetailed</i>	3	x half width of rectangle	y half width of rectangle	radius of circle	NA
<i>rectellipse</i>	4	x half width of rectangle	y half width of rectangle	x semi-axis of ellipse	y semi-axis of ellipse
<i>racetrack</i>	3	horizontal offset of circle	vertical offset of circle	radius of circular part	NA
<i>octagonal</i>	4	x half width	y half width	x point of start of edge	y point of start of edge

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. 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 MADX aperture definitions. The required parameters and their meaning are given in the following table.

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/](#).

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.

parameter	description	de- fault	re- quired
<i>magnetGeometryType</i>	The style of magnet geometry to use. One of: <i>cylindrical</i> , <i>polescircular</i> , <i>polessquare</i> , <i>polesfacet</i> , <i>polesfacetcrop</i> , <i>lhleft</i> , <i>lhcrigh</i> and <i>none</i>	<i>cylindrical</i>	no
<i>outerDiameter</i>	full horizontal width of the magnet (m)	1 m	no
<i>outerMaterial</i>	material of the magnet	“iron”	no

Example:

```
option, magnetGeometryType = "polesfacetcrop",
      outerDiameter = 0.5*m;
```

New in version 0.7: *magnetGeometryType* parameter allows different generic magnet geometry libraries to be used. Before, only cylindrical geometry was available. Examples of other geometry types are described below.

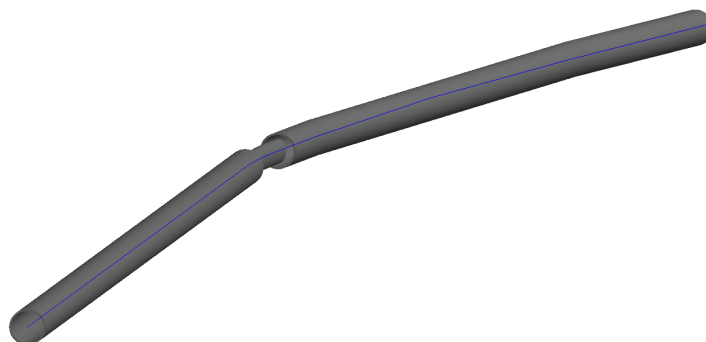
Deprecated since version 0.65: *boxSize* - this is still accepted by the parser for backwards compatibility but users should use the *outerDiameter* keyword where possible.

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 processes.

6.7.1 No Magnet Outer Geometry - “none”

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

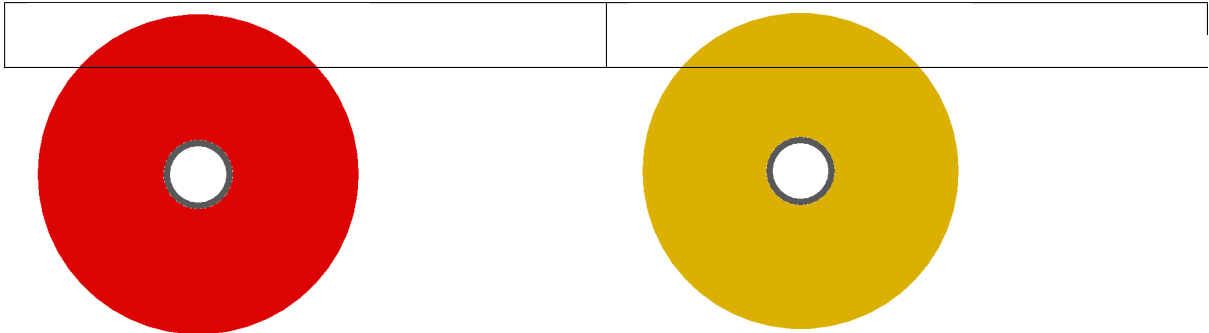


6.7.2 Cylindrical (Default) - “cylindrical”

The beam pipe is surrounded by a cylinder of material (the default is iron) whose outer diameter is controlled by the *outerDiameter* 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 the default and 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.

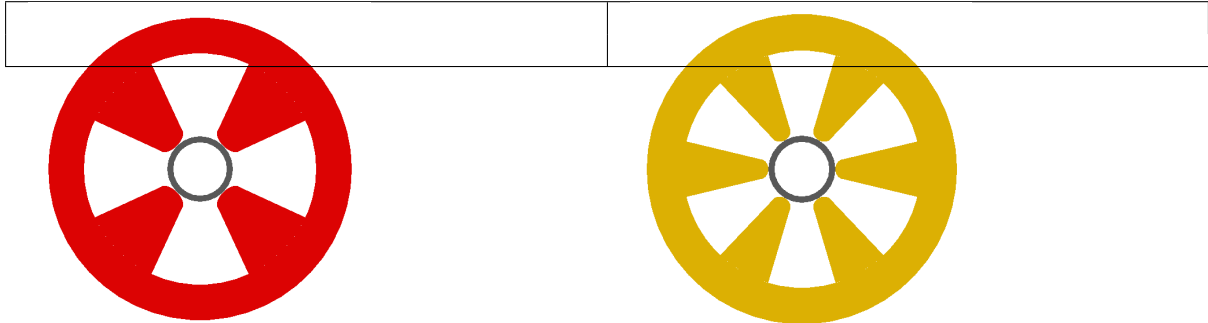
This geometry will be selected by **not** specifying any *option, magnetGeometryType*. If however, another magnet geometry is used as *option, magnetGeometryType*, the *magnetGeometryType* keyword can be used to override this on a per element basis.



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.

New in version 0.7.

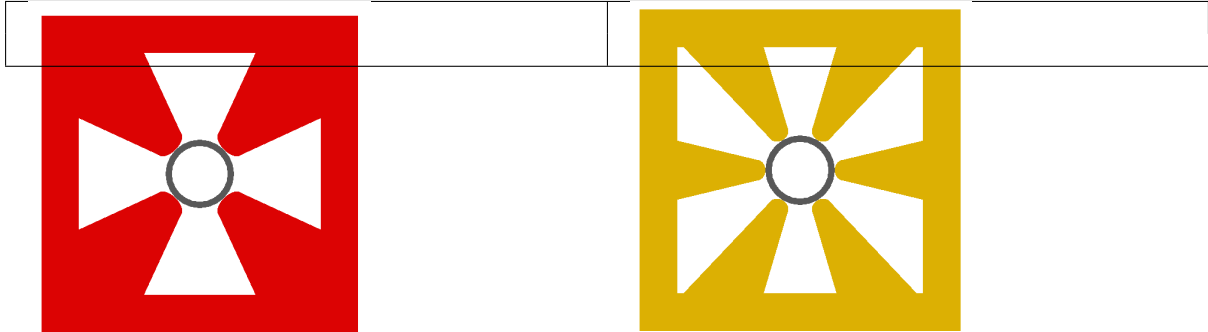


6.7.4 Poles Square - “*polessquare*”

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.

New in version 0.7.

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

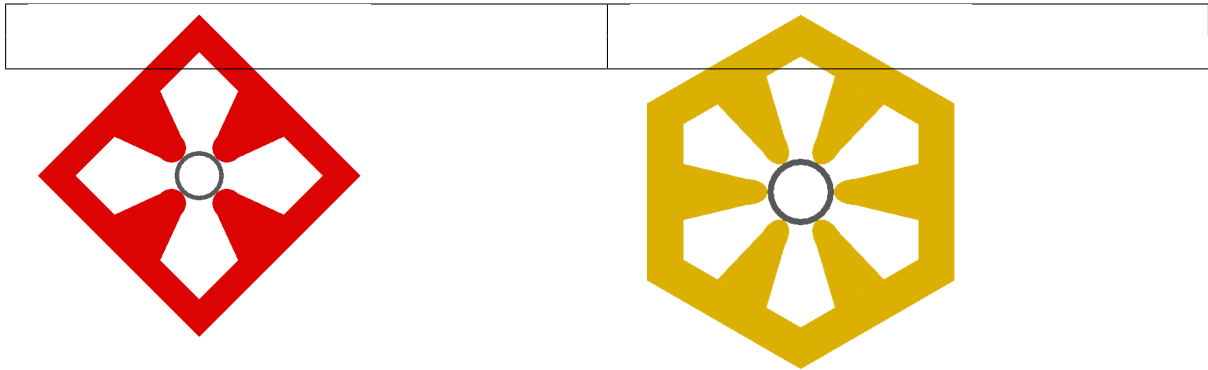


6.7.5 Poles Faceted - “*polesfacet*”

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 regard to the beam pipe size.

New in version 0.7.

outerDiameter is the full width as shown in the figure.

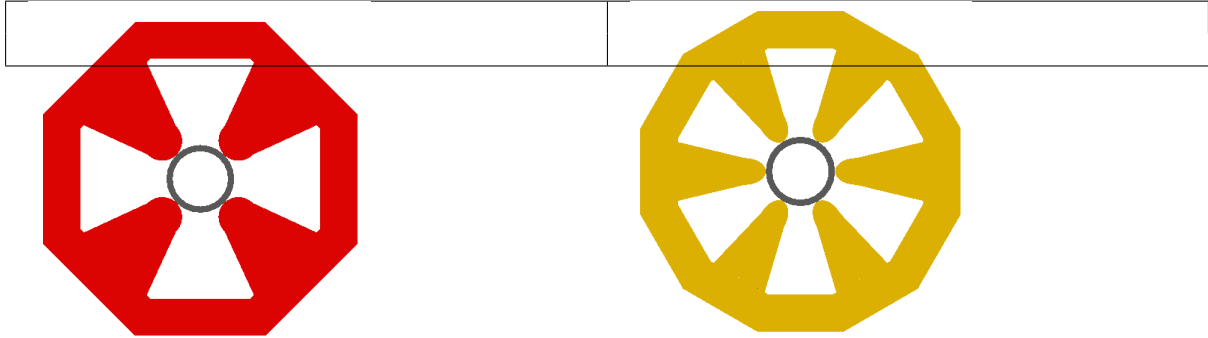


6.7.6 Poles Faceted with Crop - “*polesfacetcrop*”

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

New in version 0.7.

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



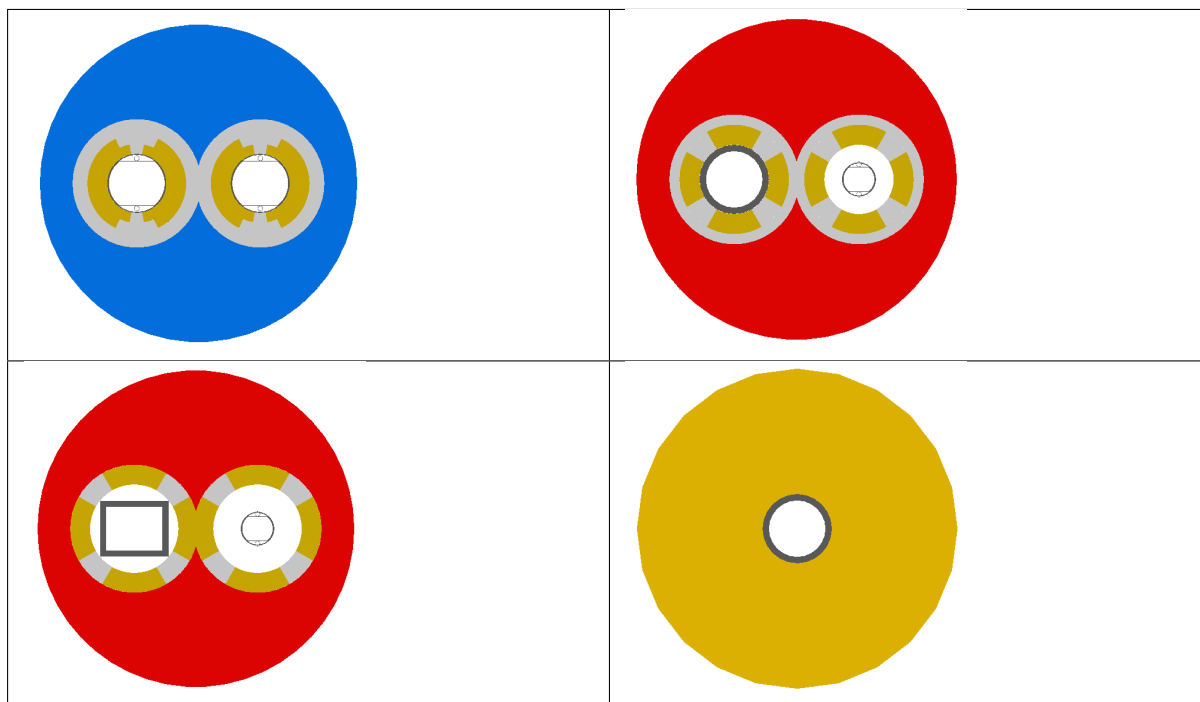
6.7.7 LHC Left & Right - “*lhleft*” | “*lhright*”

New in version 0.7.

lhleft and *lhright* 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, *lhleft* 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 *lhright* 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.



6.8 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 vertical offsets as horizontal offsets and rotations may lead to overlapping geometry. This limitation will be addressed in possible future releases, but necessitates significant changes to the geometry construction.

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

The misalignments can be controlled through the following parameters

Parameter	Default value
<i>offsetX</i>	horizontal displacement of the component [m]
<i>offsetY</i>	vertical displacement of the component [m]
<i>tilt</i>	rotation of component clockwise facing in the direction of the beamline z [rad]

Examples:

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

6.9 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.

```
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);
```

6.9.1 line

line defines a sequence of elements.

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

where *element* can be any element or line. Lines can also be reversed using

```
line_name : line=-(line_2)
```

or within another line by:

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

Reversing a line also reverses all nested lines within.

6.9.2 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:

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

Examples:

```
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;
```

6.10 Samplers - Output

Normally, the only output BDSIM would produce is the various particle loss histograms, as well as the coordinates of energy deposition hits. To observe the particles at a point in the beam lattice a *sampler* can be used. Samplers are attached to an already defined element and record all the particles passing through a plane at the *exit* to that element. They are defined using the following syntax:

```
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 (ROOT output only).

To place a sampler before an item, attach it to the previous item. If however, you wish to record the coordinates with another name, you must define a marker, place it in the sequence and then define a sampler that uses that marker:

```
d1: drift, l=2.4*m;
endoftheline: marker;
l1: line=(d1,d1,d1,d1,endoftheline);
use,period=l1;

sample, range=endoftheline;
```

When an element is defined multiple times in the line, samplers will be attached to all instances. If you wish to sample only one specific instance, the following syntax can be used:

```
sample, range=<element_name>[index];
```

To attach samplers to all elements (except the first one):

```
sample, all;
```

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

```
sample, <type>;
```

e.g.:

```
sample, quadrupole;
```

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.

6.11 Physics Processes

BDSIM can exploit all the physics processes that come with Geant4. As with any Geant4 program and simulation it is very useful to define the physical processes that should be simulated so that the simulation is both relevant and efficient. Rather than specify each individual process for every individual particle, a series of “physics lists” are provided that are a predetermined set of physics process suitable for a certain applications. BDSIM follows the Geant4 ethos in this regard.

Note, using extra physics processes that are not required will slow the simulation and produce many orders of magnitude more particles, which in turn slow the simulation further. Therefore, only use the minimal set of physics processes required.

The physics list can be selected with the following syntax (delimited by a space):

```
option, physicsList="physicslistname anotherphysicslistname";
option, physicsList="em optical";
```

Note: The strings for the physics list are case-insensitive.

New in version 0.92: Note, the physics lists changed from BDSIM produced physics lists to using the Geant4 modular physics lists in version 0.92. This also introduced the space-delimited syntax slight changes to the physics list names.

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

More details can be found in the Geant4 documentation:

6.11.1 Physics Lists In BDSIM

- [Reference Physics Lists](#)
- [Physics Reference Manual](#)

String to use	Description
	Transportation of primary particles only - no scattering in material.
em	Transportation of primary particles, ionisation, bremsstrahlung, Cerenkov, multiple scattering. Uses <i>G4EmStandardPhysics</i> .
em_extra	This provides extra electromagnetic models including, muon nuclear processes, bertini electro-nuclear model and synchrotron radiation (not in material). Provided by <i>G4EmPhysicsExtra</i> .
em_low	The same as <i>em</i> but using low energy electromagnetic models. Uses <i>G4EmPenelopePhysics</i> .
synchrad	BDSIM synchrotron radiation process.
optical	Optical physics processes including absorption, Rayleigh scattering, Mie scattering, optical boundary processes, scintillation, cherenkov. This uses <i>G4OpticalPhysics</i> class.
hadronic_elastic	Elastic hadronic processes. This is provided by <i>G4HadronElasticPhysics</i> .
hadronic	A shortcut for <i>QGSP_BERT</i> .
hadronic_hp	A shortcut for <i>QGSP_BERT_HP</i> .
qgsp_bert	Quark-Gluon String Precompound Model with Bertini Cascade model. This is based on <i>G4HadronPhysicsQGSP_BERT</i> class and includes hadronic elastic and inelastic processes. Suitable for high energy (>10 GeV). This includes and uses <i>G4EmStandardPhysics</i> .
qgsp_bert_hp	Similar to <i>QGSP_BERT</i> but with the addition of data driven high precision neutron models to transport neutrons below 20 MeV down to thermal energies. This includes and uses <i>G4EmStandardPhysics</i> . This is provided by <i>G4HadronPhysicsQGSP_BERT_HP</i> .
qgsp_bic	Like <i>QGSP</i> , 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 in interactions of protons and neutrons with nuclei. This is provided by <i>G4HadronPhysicsQGSP_BIC</i> .
qgsp_bic_hp	Similar to <i>QGSP_BIC</i> but with the high precision neutron package. This is provided by <i>G4HadronPhysicsQGSP_BIC_HP</i> .
ftfp_bert	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 <i>G4HadronPhysicsFTFP_BERT</i> . All FTF physics lists require <i>G4HadronElasticPhysics</i> to work correctly.
ftfp_bert_hp	Similar to <i>FTFP_BERT</i> but with the high precision neutron package. This is provided by <i>G4HadronPhysicsFTFP_BERT_HP</i> .
decay	Provides radioactive decay processes using <i>G4DecayPhysics</i> .
muon	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. Provided by BDSIM physics builder (a la Geant4) <i>BDSMuonPhysics</i> .

6.12 Physics Biasing

A physics biasing process can be defined with the keyword **xsecbias**.

Note: This only works with Geant4 version 10.1 or higher.

parameter	description
name	biasing process name
particle	particle that will be biased
proc	process(es) to be biased
flag	flag which particles are biased for the process(es) (1=all, 2=primaries, 3=secondaries)
xsecfact	biasing factor(s) for the process(es)

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 to the vacuum volume or everything outside the vacuum respectively:

```

q1: quadrupole, l=1*m, material="Iron", biasVacuum="biasDef1 biasDef2"; ! uses the process biasDe
q2: quadrupole, l=0.5*m, biasMaterial="biasDef2";

```

6.13 Options

Various simulation details can be controlled through the *option* command. Options are defined using the following syntax:

```
option, <option_name>=<value>;
```

If the value is a string and not a number, it should be enclosed in “double inverted commas”. 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.

6.13.1 options in BDSIM

Below is a full list of all options in BDSIM. If the option is boolean, 1 (true) or 0 (false) can be used as their value.

Option	Function
Common Parameters	
beampipeRadius	default beam pipe inner radius [m]
beampipeThickness	default beam pipe thickness [m]
beampipeMaterial	default beam pipe material
boxSize	default accelerator component full width [m]
randomSeed	the integer seed value for the random number generator
ngenerate	number of primary particles to simulate

Continued on

Table 6.1 – continued from previous page

Option	Function
eLossHistoBinWidth	the width of the histogram bins [m]
physicsList	the physics list to use
thresholdCutCharged	the minimum energy above which to simulate electron and positrons - any below this energy will be killed
thresholdCutPhotons	the minimum energy above which to simulate photons - any below this energy will be killed
stopSecondaries	whether to stop secondaries or not (default = false)
stopTracks	whether to stop tracks after interaction (default = false)
circular	whether the accelerator is circular or not
printModuloFraction	the fraction of events to print out (default 0.1)
Geometry Parameters	
samplerDiameter	diameter of samplers (default 5 m) [m]
includeIronMagFields	whether to include magnetic fields in the magnet poles
sensitiveBeamlineComponents	whether all beam line components record energy loss
sensitiveBeamPipe	whether the beam pipe records energy loss
vacuumMaterial	the material to use for the beam pipe vacuum
vacuumPressure	the pressure of the vacuum gas [bar]
Tracking Parameters	
deltaChord	chord finder precision
deltaIntersection	boundary intersection precision
chordStepMinimum	minimum step size
lengthSafety	element overlap safety (caution!)
minimumEpsilonStep	minimum relative error acceptable in stepping
maximumEpsilonStep	maximum relative error acceptable in stepping
deltaOneStep	set position error acceptable in an integration step
Physics Processes Parameters	
synchRadOn	whether to use synchrotron radiation processes
srTrackPhotons	whether to track synchrotron radiation photons
srLowX	minimum synchrotron radiation energy as a fraction of $E_{critical}$ ($0 > srLowX > 1$)
srLowGamE	lowest synchrotron photon energy to track
srMultiplicity	a factor multiplying the number of synchrotron photons
prodCutPhotons	standard overall production cuts for photons
prodCutPhotonsP	precision production cuts for photons
prodCutElectrons	standard overall production cuts for electrons
prodCutElectronsP	precision production cuts for electrons
prodCutPositrons	standard overall production cuts for positrons
prodCutPositronsP	precision production cuts for positrons
prodCutProtons	standard overall production cuts for protons
prodCutProtonsP	precision production cuts for protons
turnOnCerenkov	whether to produce cerenkov radiation
defaultRangeCut	the default predicted range at which a particle is cut (default $1e-3$) [m]
gammaToMuFe	the cross-section enhancement factor for the gamma to muon process
annihiToMuFe	the cross-section enhancement factor for the electron-positron annihilation to muon process
eetoHadronsFe	the cross-section enhancement factor for the electron-positron annihilation to hadrons process
useEMLPB	whether to use electromagnetic lead particle biasing (default = 0)
LPBFraction	the fraction of electromagnetic process in which lead particle biasing is used ($0 < LPBFraction$)
Output Parameters	
storeTrajectories	whether to store trajectories in the output
storeMuonTrajectories	whether to store muon trajectories in the output
storeNeutronTrajectories	whether to store neutron trajectories in the output
trajCutGTZ	global z position cut (minimum) for storing trajectories
trajCutLTR	radius cut for storing trajectories (maximum)
nperfile	number of evens to record per output file
nlinesIgnore	number of lines to ignore when reading user bunch input files

- For **Tunnel** parameters, see, [Tunnel Geometry](#).

6.14 Beam Parameters

To specify the input particle distribution to the accelerator model, the *beam* command is used ¹. This also specifies the particle species and **reference energy**, which is the design energy of the machine. This is used along with the particle species to calculate the momentum of the reference particle and therefore the magnetic field of dipole magnets if only the *angle* parameter has been specified.

Note: A design energy can be specified and in addition, the central energy, of say a bunch with a Gaussian distribution, can be specified.

The user **must** specify at least *energy* and the *particle* type. 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 in *GeV* by default. The particle may be one of the following:

- *e-*
- *e+*
- *proton*
- *gamma*
- *mu-*
- *mu+*

Many particles can be used and are taken from the Geant4 particle table directly.

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

6.14.1 Beam Distributions

The following beam distributions are available in BDSIM

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

¹ Note, the *beam* command is actually currently equivalent to the *option* command. The distinction is kept for clarity, and this might be changed in the future.

6.14.2 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.

These parameters also act as central parameters for all other distributions. For example, a Gaussian distribution may be defined with the *gauss* parameters but *X0* set to offset the centroid of the Gaussian with respect to the reference trajectory.

Option	Description	Default
<i>X0</i>	Horizontal position [m]	0
<i>Y0</i>	Vertical position [m]	0
<i>Z0</i>	Longitudinal position [m]	0
<i>T0</i>	Longitudinal position [s]	0
<i>Xp0</i>	Horizontal canonical momentum	0
<i>Yp0</i>	Vertical canonical momentum	0

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;
```

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

6.14.3 gaussmatrix

Uses the N dimensional gaussian generator from *CLHEP*, *CLHEP::RandMultiGauss*. The generator is initialised by a 6×1 means vector and 6×6 sigma matrix.

- All parameters from *reference* distribution as used as centroids.

Option	Description
<i>sigmaNM</i>	Sigma matrix element (N,M)

Examples:

```
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;
```

6.14.4 gauss

Uses the *gaussmatrix* beam generator but with simplified input parameters 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

$$\begin{aligned}\sigma_{11} &= \sigma_x^2 \\ \sigma_{22} &= \sigma_x'^2 \\ \sigma_{33} &= \sigma_y^2 \\ \sigma_{44} &= \sigma_y'^2 \\ \sigma_{55} &= \sigma_T^2 \\ \sigma_{66} &= \sigma_E^2.\end{aligned}$$

- All parameters from *reference* distribution as used as centroids.

Option	Description
<i>sigmaX</i>	Horizontal gaussian sigma [m]
<i>sigmaY</i>	Vertical gaussian sigma [m]
<i>sigmaXp</i>	Sigma of the horizontal canonical momentum
<i>sigmaYp</i>	Sigma of the vertical canonical momentum
<i>sigmaE</i>	Relative energy spread
<i>sigmaT</i>	Sigma of the temporal distribution [s]

6.14.5 gausstwiss

The beam parameters are defined by the usual α , β and γ from which the usual beam σ -matrix is calculated, using the following equations

$$\begin{aligned}\sigma_{11} &= \epsilon_x \beta_x \\ \sigma_{12} &= -\epsilon_x \alpha_x \\ \sigma_{21} &= -\epsilon_x \alpha_x \\ \sigma_{22} &= \epsilon_x \gamma_x \\ \sigma_{33} &= \epsilon_y \beta_y \\ \sigma_{34} &= -\epsilon_y \alpha_y \\ \sigma_{43} &= -\epsilon_y \alpha_y \\ \sigma_{44} &= \epsilon_y \gamma_y \\ \sigma_{55} &= \sigma_T^2 \\ \sigma_{66} &= \sigma_E^2\end{aligned}$$

- All parameters from *reference* distribution as used as centroids.

Option	Description
<i>emitx</i>	Horizontal beam core emittance [m]
<i>emity</i>	Vertical beam core emittance [m]
<i>betax</i>	Horizontal beta function [m]
<i>betay</i>	Vertical beta function [m]
<i>alfx</i>	Horizontal alpha function
<i>alfy</i>	Vertical alpha function

6.14.6 circle

Beam of randomly distributed particles with a uniform distribution within a circle in each dimension dimension of phase space - x & xp ; y & yp , T & E with each uncorrelated. Each parameter defines the maximum absolute extent in that dimension. Ie, the possible values range from *-envelopeX* to *envelopeX* for example.

- All parameters from *reference* distribution as used as centroids.

Option	Description
<i>envelopeR</i>	Maximum position
<i>envelopeRp</i>	Maximum canonical momentum
<i>envelopeT</i>	Maximum time offset [s]
<i>envelopeE</i>	Maximum energy offset [GeV]

6.14.7 square

This distribution has similar properties to the *circle* distribution with the exception that the particles are randomly uniformly distributed within a square.

- All parameters from *reference* distribution as used as centroids.

Option	Description
<i>envelopeX</i>	Maximum position in X [m]
<i>envelopeXp</i>	Maximum canonical momentum in X
<i>envelopeY</i>	Maximum position in Y [m]
<i>envelopeYp</i>	Maximum canonical momentum in Y
<i>envelopeT</i>	Maximum time offset [s]
<i>envelopeE</i>	Maximum energy offset [GeV]

6.14.8 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 - ie *xp*, *yp* etc.

- All parameters from *reference* distribution as used as centroids.

Option	Description
<i>Rmin</i>	Minimum radius in <i>x</i> and <i>y</i> [m]
<i>Rmax</i>	Maximum radius in <i>x</i> and <i>y</i> [m]

6.14.9 eshell

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

- All parameters from *reference* distribution as used as centroids.

Option	Description
<i>shellX</i>	Ellipse semi-axis in phase space in horizontal position [m]
<i>shellXp</i>	Ellipse semi-axis in phase space in horizontal canonical momentum
<i>shellY</i>	Ellipse semi-axis in phase space in vertical position [m]
<i>shellYp</i>	Ellipse semi-axis in phase space in vertical momentum
<i>shellXWidth</i>	Spread of ellipse in phase space in horizontal position [m]
<i>shellXpWidth</i>	Spread of ellipse in phase space in horizontal canonical momentum
<i>shellYWidth</i>	Spread of ellipse in phase space in vertical position [m]
<i>shellYpWidth</i>	Spread of ellipse in phase space in vertical momentum

6.14.10 halo

The halo distribution is effectively a flat phase space with the central beam core removed at ϵ_{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 so $\epsilon_{SP} < \epsilon_{core}$ the particle is rejected. *haloPSWeightFunction* is a string that selects the function $f_{haloWeight}(\epsilon_{SP})$ which is 1 at the ellipse defined by ϵ_{core} . The weighting functions are either *flat*, one over emittance *oneoverr* or exponential *exp* so

$$\begin{aligned} f_{haloWeight}(\epsilon_{SP}) &= 1 \\ f_{haloWeight}(\epsilon_{SP}) &= \left(\frac{\epsilon_{core}}{\epsilon_{SP}} \right)^p \\ f_{haloWeight}(\epsilon_{SP}) &= \exp \left(-\frac{\epsilon_{SP} - \epsilon_{core}}{p\epsilon_{core}} \right) \end{aligned}$$

- All parameters from *reference* distribution as used as centroids.

Option	Description
<i>emitx</i>	Horizontal beam core emittance [m] $\epsilon_{core,x}$
<i>emity</i>	Vertical beam core emittance [m] $\epsilon_{core,y}$
<i>betax</i>	Horizontal beta function [m]
<i>betay</i>	Vertical beta function [m]
<i>alfx</i>	Horizontal alpha function
<i>alfy</i>	Vertical alpha function
<i>envelopeX</i>	Horizontal position maximum [m]
<i>envelopeY</i>	Vertical position maximum [m]
<i>envelopeXp</i>	Horizontal angle maximum [m]
<i>envelopeYp</i>	Vertical angle maximum [m]
<i>haloPSWeightFunction</i>	Phase space weight function [string]
<i>haloPSWeightParameter</i>	Phase space weight function parameters []

6.14.11 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 as used as centroids.

Option	Description
<i>xDistrType</i>	Horizontal distribution type
<i>yDistrType</i>	Vertical distribution type
<i>zDistrType</i>	Longitudinal distribution type

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

Examples:

```
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,
emitx = 1e-9,
emity = 1e-9,
sigmaE = 0.00008836,
sigmaT = 0.000000000001,
shellX = 150*um,
shellY = 103*um,
shellXp = 1.456e-6,
shellYp = 2.4e-5,
shellXWidth = 10*um,
shellYWidth = 15*um,
shellXpWidth = 1e-9,
shellYpWidth = 1d-9;

```

6.14.12 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.

Option	Description
<i>distrFile</i>	File path to ASCII data file
<i>distrFileFormat</i>	A string that details the column names and units

Examples:

```

beam, particle = "e-",
energy = 1*GeV,
distrType = "userfile",
distrFile = "9_UserFile.dat",
distrFileFormat = "x[mum]:xp[mrad]:y[mum]:yp[mrad]:z[cm]:E[MeV] ";

```

The corresponding *9_UserFile.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

```

6.14.13 ptc

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

Option	Description
<i>distrFile</i>	PTC output file

6.15 Tunnel Geometry

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

1. The tunnel follows the beamline, 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 and 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*.

Tunnel Parameters	
buildTunnel	whether to build a tunnel (default = 0)
buildTunnel-Straight	whether to build a tunnel ignoring the beamline and just in a straight line (default = 0)
buildTunnelFloor	whether to add a floor to the tunnel
tunnelType	which style of tunnel to use - one of: <i>circular</i> , <i>elliptical</i> , <i>square</i> , <i>rectangular</i> (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.

Note: 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.

6.16 Material and Atoms

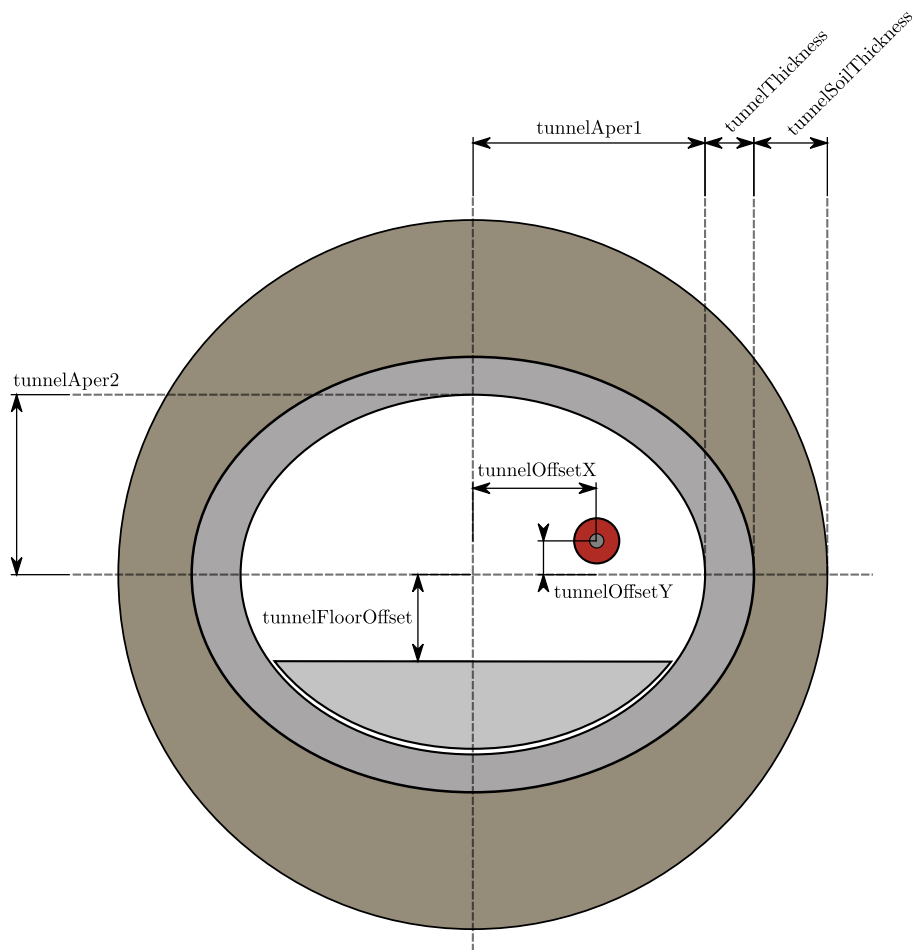
Materials and atoms can be added via the parser, just like lattice elements.

If the material is composed by a single element, it can be defined using the **matdef** command with the following syntax:

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

parameter	description	default
Z	atomic number	
A	mass number [g/mol]	
density	density in [g/cm3]	
T	temperature in [K]	300
P	pressure [atm]	1
state	"solid", "liquid" or "gas"	"solid"

Example:



```
iron : matdef, Z=26, A=55.845, density=7.87;
```

If the material is made up by several components, first of all each of them must be specified with the **atom** keyword:

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

parameter	description
Z	atomic number
A	mass number [g/mol]
symbol	atom symbol

The compound material can be specified in two manners:

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

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

parameter	description
density	density in [g/cm3]
components	list of symbols for material components
componentsWeights	number of atoms for each component in material unit

Example:

```
niobium : atom, symbol="Nb", Z=41, A=92.906;
titanium : atom, symbol="Ti", Z=22, A=47.867;
NbTi : matdef, density=5.6, T=4.0, components=["Nb","Ti"], componentsWeights={1,1};
```

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

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

parameter	description
components	list of symbols for material components
componentsFractions	mass fraction of each component in material unit

Example:

```
samarium : atom, symbol="Sm", Z=62, A=150.4;
cobalt : atom, symbol="Co", Z=27, A=58.93;
SmCo : matdef, density=8.4, T=300.0, components=["Sm","Co"], componentFractions = {0.338,0.662};
```

The second syntax can be used also to define materials which are composed by other materials (and not by atoms).
Nb: Square brackets are required for the list of element symbols, curly brackets for the list of weights or fractions.

6.17 Regions

In Geant4 it is possible to drive different *regions* each with their own production cuts and user limits. In BDSIM three different regions exist, each with their own user defined production cuts (see *Physics*). These are the default region, the precision region and the approximation region. Beamline elements can be set to the precision region by setting the attribute *precisionRegion* equal to 1. For example:

EXTENDED GEOMETRY

The element with user-defined physical geometry (see also *element*) is defined by:

```
<element_name> : element, geometry=format:filename, attributes
```

where *format* must be set to *gmad*, *mokka*, *gdml* or *lcdd* and *filename* must point to a file that contains the geometry descriptions. For example:

```
colli : element, geometry="gmad:colli.geo"
```

7.1 gmad

gmad is a simple format used as G4geometry wrapper. It can be used for specifying more or less simple geometries like 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:

7.1.1 Box

parameter	description
x0	x origin
y0	y origin
z0	z origin
x	x size
y	y size
z	z size
phi	Euler angle for rotation
theta	Euler angle for rotation
psi	Euler angle for rotation
material	Material name

7.1.2 Cons

parameter	description
x0	x origin
y0	y origin
z0	z origin
rmin1	inner radius at start
rmax1	outer radius at start
rmin2	inner radius at end
rmax2	outer radius at end
z	z size
phi	Euler angle for rotation
theta	Euler angle for rotation
psi	Euler angle for rotation
phi0	angle for start of sector
dphi	angle swept by sector
material	Material name

7.1.3 Tubs

parameter	description
x0	x origin
y0	y origin
z0	z origin
rmin	inner radius
rmax	outer radius
z	z size
phi	Euler angle for rotation
theta	Euler angle for rotation
psi	Euler angle for rotation
material	Material name

7.1.4 Trd

parameter	description
x0	x origin
y0	y origin
z0	z origin
x1	half length at wider side
x2	half length at narrower side
y1	half length at wider side
y2	half length at narrower side
z	z size
phi	Euler angle for rotation
theta	Euler angle for rotation
psi	Euler angle for rotation
material	Material name

7.2 Mokka

As well as using the GMAD format to describe user-defined physical geometry it is also possible to use a Mokka style format. 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 to setting up the Mokka geometry:

- *Describing the geometry*
- *Creating a geometry list*
- *Defining a Mokka Element*

7.2.1 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 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 attributes set is shown below. The first box is a simple vacuum cube whilst the second is an iron box with length $x = 10\text{mm}$, length $y = 150\text{mm}$, length $z = 50\text{mm}$, positioned at $x=1\text{m}$, $y=0$, $z=0.5\text{m}$ 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),  
ROTTHEA DOUBLE(10,3),  
ROTPHI DOUBLE(10,3)  
);  
  
INSERT INTO mytable_BOX VALUES("a_box","vacuum", 50.0, 50.0, 50.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0);  
INSERT INTO mytable_BOX VALUES("another_box","iron", 10.0, 150.0, 50.0, 1000.0, 0.0, 500.0, 0.0, 0.0, 0.0);
```

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 center of the parent object.
- **INHERITSTYLE**
Variable type: VARCHAR(32)
Optional parameter to be used with PARENTNAME.
If set to "SUBTRACT" then the 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 0.

- 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 0.

- 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 to 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 0.

- 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 are from the position in mm used to place the object in the component volume. POSX and POSY are defined with respect to the center 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 center of the parent object.

- ROTPSI, ROTTHETA, ROTPHI

Variable type: DOUBLE(10,3)

Optional parameters

They 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 displayment of the object. “S” produces a shaded solid and “I” leaves the object invisible. The default is set to be solid.

- **FIELDX, FIELDY, FIELDZ**

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 then 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 then 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 then 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 then this value will be used to specify the delta angle of the cone. The default value is 2π .

'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 then 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 center of the torus ring to the center 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 then this value will be used to specify the starting angle of the torus. The default value is zero.

- DELTAPHI

Variable type: DOUBLE(10,3)

Optional parameter

If set then this value will be used to specify the delta swept angle of the torus. The default value is 2π .

'Polycone' Solid Types

Append _POLYCONE 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, 3 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, 3 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, 3 z-planes will require that ROUTER1, ROUTER2, and ROUTER3 are all set up.

- STARTPHI

Variable type: DOUBLE(10,3)

Optional parameter

If set then 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 then this value will be used to specify the delta angle of the polycone. The default value is 2π .

'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 = XSEMIAXIS * (LENGTHZ - u) / u * Cos(v)
Y = YSEMIAXIS * (LENGTHZ - u) / u * Sin(v)
Z = u
```

where v is between 0 and $2 * \pi$ and u between 0 and h respectively.

7.2.2 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:

```
# '#' symbols can be used for commenting out an entire line
/directory/boxes.sql
/directory/cones.sql
```

7.2.3 Defining a Mokka element

The Mokka element can be defined by the following command:

```
collimator : element, geometry=mokka:coll_geomlist.sql
```

7.3 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 we refer to the [GDML website](#) and [manual](#).

7.4 LCDD

The LCDD (Linear Collider Detector Description) is based on [GDML](#). The syntax and usage is described in this [SLAC paper](#).

MODEL PREPARATION

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 to that of BDSIM - gmad. Additionally, the python tools can be used to programatically create an accelerator lattice of your own design, which is described in *Python Builder*.

8.1 Manual Preparation

An input gmad (text) file can be prepared manually in your favourite text editor easily by defining:

1. Individual elements
2. Define 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 MADX Conversion

A MADX 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 madx containing all twiss table information
2. converting the tfs file to gmad using pybdsim

The twiss file can be prepared by appending the following MADX syntax to the end of your MADX script:

```
select, flag=twiss, clear;  
twiss, sequence=SEQUENCENAME, file=twiss.tfs;
```

where *SEQUENCENAME* is the name of the sequence in madx. 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.

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

8.4 Python Builder

OUTPUT

Output from BDSIM can be in various formats.

Format	Syntax	Description
None	-output=none	No output is written
ASCII	-output=ascii	A series of text files in a folder are written
ROOT	-output=root	A root file with a tree for each sampler is written with float precision
ROOT	-output=rootdouble	A root file with a tree for each sampler is written with double precision
ROOT (detailed)	- output=rootdetailed	Similar to the above ROOT format but with extra variables for more detail
ROOT (detailed)	- output=rootdetaileddouble	Similar to the above ROOT format but with extra variables for more detail with double precision
ROOT (per event)	-output=rootevent	A root file with a per event structure format
Multiple	-output=combined	All formats written at once

Note: Where double precision is used, the data is typically 2x as big. This is only recommended where variable precision is extremely important - ie comparing different particle coordinates for tracking accuracy. Histograms are stored to double precision irrespective.

Details about each one are listed below in [output format differences](#). As a general guideline, the following naming conventions are used:

Short Name	Meaning
phits	primary hits
ploss	primary losses
eloss	energy loss
PE	per element

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

9.1 Output Format Differences

9.1.1 ROOT

With the ROOT format, everything is recorded in one single file. If the number of events simulated exceeds `nperfile` a new file will be started. The chosen filename will be suffixed with `_N.root` where N is an integer.

- Histograms are stored as TH1F objects within the file

- Each sampler has its own Tree

9.1.2 ASCII Output

With ASCII output, a folder is created with the given output name. Inside this histograms and sampler output are produced in different text files.

- Histograms are suffixed with `.hist.txt`.
- The file with only `.txt` is the main output from all samplers
- The sampler output is recorded in simulation order, not spatial order

The ASCII output is relatively limited compared to the root output.

- The main `filename.txt` file contains hits on samplers in the order they happened and are not grouped by sampler.
- In the energy loss, primary hits and primary loss files, x' and y' are always 0 as they are undefined in these cases.
- In all files, local x and y are used whereas global Z is used.

9.2 Histograms

BDSIM produces six histograms by default during the simulation. These are: primary hits per bin width; primary losses per bin width; energy loss per metre (GeV); primary hits per element; primary losses per element; and Energy loss per element.

The per element histograms are integrated across the length of each element so they will have a different bin width. The other histograms are evenly binned according to the option `elossHistoBinWidth` (in metres).

Note: Histograms are only written to disk once all events have been simulated.

9.3 Samplers

Samplers record the particle position at the start of each element. The following variables are recorded:

Variable Name	Units	Meaning
E0	GeV	Initial primary energy
x0	m	Initial primary global x position
y0	m	Initial primary global y position
z0	m	Initial primary global z position
xp0	rad	Initial primary global x' position
yp0	rad	Initial primary global y' position
zp0	rad	Initial primary global z' position
t0	ns	Initial primary global time
E	GeV	Total current energy
x	m	Local x position
y	m	Local y position
z	m	Local z position
xp	rad	Local x' position
yp	rad	Local y' position
zp	rad	Local z' position
t	ns	Time of flight
X	m	Global x position
Y	m	Global y position
Z	m	Global z position
Xp	rad	Global x' position
Yp	rad	Global y' position
Zp	rad	Global z' position
s	m	Curvilinear S
weight	NA	Weight
partID	NA	PDG ID number
nEvent	NA	Event number
parentID	NA	Parent ID (0 means primary)
trackID	NA	Track ID
turnnumber	NA	Turns completed

Note: *rad* is not strictly correct for the prime units but is used in the small angle approximation. The prime is the differential of that position

9.4 Primary Coordinates

The primary coordinates for each event are recorded in a similar fashion to the samplers in their own file / tree.

Warning: A common issue is 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, putting a sampler at the beginning of a beam line should be avoided to avoid confusion. The primary output (either separate file in ASCII or as a tree in root) records all primary coordinates before they enter the tracking in the geometry, so it always contains all primary particles.

OUTPUT ANALYSIS

10.1 ROOT Output (robdsim)

To use the ROOT analysis (and the Python interface):

```
export ROBDSIM=<bdsim-build-dir>/utils/robdsim
export PATH=$PATH:$ROBDSIM
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$ROBDSIM (Linux only)
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$ROBDSIM (mac only)
export PYTHONPATH=$PYTHONPATH:$ROBDSIM
```

Note that several ROOT analysis output (output of robdsimAnal) files can be combined into one file with the command:

```
robdsimComb combined.root output_0.root output_1.root output_2.root output_3.root
```

Using the robdsim library from within ROOT:

```
> root
root [0] gSystem->Load("<bdsim-build-dir>/utils/robdsim/librobdsim.so")
root [1] r = new RobdsimAnalysis("analysisConfig.txt")
```

Using the robdsim library from within Python:

```
> pylab
In [1]: import robdsim
In [2]: r = robdsim.RobdsimAnalysis("analysisConfig.txt")
```

From the command line with the executable (containing the ROOT output files and analysisConfig.txt):

```
> robdsimAnal analysisConfig.txt
```

Typically analysis is performed with a simple configuration file (analysisConfig.txt, in the two examples above) that defined the histograms that should be created from trees etc. An example structure follows:

Debug	0		
InputFilePath	./output.root		
OutputFileName	./analysis.root		
CalculateOpticalFunctions	1		
CalculateOpticalFunctionsFileName	./output.dat		
Histogram	Sampler_sampler4	elec_x	x
Histogram	Sampler_sampler4	elec_y	y
Histogram	Sampler_sampler4	elec_xy	y:x
Histogram	Sampler_sampler4	elec_E	E
Histogram	Sampler_sampler4	photon_x	x
Histogram	Sampler_sampler4	photon_y	y
Histogram	Sampler_sampler4	photon_xy	y:x
Histogram	Sampler_sampler4	photon_E	E
Histogram	Sampler_sampler4	photon_Spec	log10(E)

10.2 Converting ROOT trees as numpy arrays

First root_numpy needs to be installed (from scratch):

```
wget https://pypi.python.org/packages/source/r/root_numpy/root_numpy-4.3.0.tar.gz
tar -zxf root_numpy-4.3.0.tar.gz
cd root_numpy-4.3.0
python2.7 setup.py build
sudo python2.7 setup.py install
```

Or using PIP (mac):

```
sudo port install py27-pip
sudo pip-2.7 install root_numpy
```

Extracting data from ROOT file:

```
> pylab
In [1]: import ROOT
In [2]: import root_numpy
In [3]: f = ROOT.TFile("analysis.root")
In [4]: t = f.Get("Sampler1")
In [5]: a = root_numpy.tree2rec(t)
```

10.3 ROOT classes and structure

- RobdsimAnalysis
- RobdsimOutput
- AnalysisConfig

Then for each tree type (Eloss, PrecisionRegionElossPhits, Sampler) * Tree * TreeAnalysis

10.4 Extending ROOT analysis

10.5 Examples

10.5.1 Example: Plot Optical Functions with pybdsim

```
import pybdsim
import pymadx
import robdsim

# load ROOT output file into robdsim
r=robdsim.RobdsimOutput("output.root")
# calculate optics and write to output file "optics.dat"
r.CalculateOpticalFunctions("optics.dat")
# load optics file into pybdsim
a=pybdsim.Data.Load("optics.dat")

f = figure()
# Plot sqrt Beta_x
plot(a.S(), sqrt(a.Beta_x()), '-.', label='BDSIM')

# compare with MadX file:
b = pymadx.Tfs("madx.tfs")
s = b.GetColumn("S")
```

```
betx = b.GetColumn("BETX")
plot(s, sqrt(betx), '-.', label='MadX')

# labels and legend:
xlabel('$s$ [m]')
ylabel('$\sqrt{\beta_x}$ [m]')
legend(loc=0)

# add machine lattice to figure (optional):
pybdsim.Plot.AddMachineLatticeToFigure(f,b)

# save figure
f.savefig("sqrtbetax.png")
```

10.6 ASCII Output

PYTHON PACKAGES

Several python packages are provided to aid preparation and conversion of models to a from BDSIM's gmad format and are described the following sections.

- pybdsim - conversion and building of input as well as ASCII data analysis
- pymadx - file loader for MADX Tfs file format
- pymad8 - file loader for MAD8 file format

11.1 Installing Python Packages

To access (import) the supplied packages from anywhere on your system, the *utils* directory should be added to your *PYTHONPATH* environmental variable:

```
export PYTHONPATH=$PYTHONPATH:~/physics/rep/bdsim/utils
```

This should allow you to import the python utilities without error from any location:

```
cd ~/
python
```

```
>>> import pybdsim
>>> # no errors
```

11.2 Documentation

In the following sections is individual documentatino for each python package. These are also available when using ipython by typing ? after any module or class.

11.2.1 pybdsim

Module contents

pybdsim - python tools for bdsim

Dependencies:

package - minimum version required

numpy - 1.7.1

matplotlib - 1.3.0

Modules:

Builder - create generic accelerators for bdsim

Convert - convert other formats into gmad

Data - read the bdsim output formats

Gmad - create bdsim input files - lattices & options

ModelProcessing - tools to process existing BDSIM models and generate other versions of them.

Options - methods to generate bdsim options

Plot - some nice plots for data

Root - functions to convert ROOT histograms into matplotlib (not explicitly imported)

Visualisation - help locate objects in the BDSIM visualisation, requires a BDSIM survey file

Classes:

Analysis - encapsulates functions & plots for a single file

Beam - a beam options dictionary with methods

XSecBias - a cross-section biasing object

pybdsim.Beam module

```
class pybdsim.Beam.Beam (particletype='e-', energy=1.0, distrtype='reference', *args, **kwargs)
```

Bases: dict

ReturnBeamString ()

SetDistributionType (distrtype='reference')

SetEnergy (energy=1.0, unitsstring='GeV')

SetParticleType (particletype='e-')

SetT0 (t0=0.0, unitsstring='s')

SetX0 (x0=0.0, unitsstring='m')

SetXP0 (xp0=0.0)

SetY0 (y0=0.0, unitsstring='m')

SetYP0 (yp0=0.0)

SetZ0 (z0=0.0, unitsstring='m')

SetZP0 (zp0=0.0)

pybdsim.Builder module

```
class pybdsim.Builder.Machine (verbose=False)
```

A class represents an accelerator lattice as a sequence of components. Member functions allow various lattice components to be append to the sequence of the machine. This class allows the user to programatically create a lattice and write the BDSIM gmad representation of it.

Example:

```
>>> a = Machine()
>>> a.AddDrift('mydrift', l=1.3)
>>> a.Write("lattice.gmad")
```

Caution: adding an element of the same name twice will result the element being added only to the sequence again and not being redefined - irrespective of if the parameters are different. If verbose is used (True), then a warning will be issued.

AddBeam (*beam=None*)

Assign a beam instance to this machine. If no Beam instance is provided, a reference distribution is used.

AddBias (*biasobject*)

AddDipole (*category='sbend'*)

category - 'sbend' or 'rbend' - sector or rectangular bend

AddDrift (*name='dr', length=0.1, **kwargs*)

Add a drift to the beam line

AddECol (*name='ec', length=0.1, xsize=0.1, ysize=0.1, **kwargs*)

AddFodoCell (*basename, magnetlength, driftlength, kabs, **kwargs*)

basename - the basename for the fodo cell beam line elements magnetlength - length of magnets in metres driftlength - length of drift segment in metres kabs - the absolute value of the quadrupole strength - alternates between magnets

**kwargs are other parameters for bdsim - ie material='Fe'

AddFodoCellMultiple (*basename='fodo', magnetlength=1.0, driftlength=4.0, kabs=0.2, ncells=2, **kwargs*)

AddFodoCellSplitDrift (*basename, magnetlength, driftlength, kabs, nsplits, **kwargs*)

basename - the basename for the fodo cell beam line elements magnetlength - length of magnets in metres driftlength - length of drift segment in metres kabs - the absolute value of the quadrupole strength - alternates between magnets nsplits - number of segments drift length is split into

Will add qf quadrupole of strength +kabs, then drift of l=driftlength split into nsplit segments followed by a qd quadrupole of strength -kabs and the same pattern of drift segments.

nsplits will be cast to an even integer for symmetry purposes.

**kwargs are other parameters for bdsim - ie aper=0.2

AddFodoCellSplitDriftMultiple (*basename='fodo', magnetlength=1.0, driftlength=4.0, kabs=0.2, nsplits=10, ncells=2, **kwargs*)

AddHKicker (*name='hk', length=0.1, angle=0.0, **kwargs*)

AddMarker (*name='mk'*)

Add a marker to the beam line.

AddMultipole (*name='mp', length=0.1, knl=0, ksl=0, tilt=0.0, **kwargs*)

AddOctupole (*name='oc', length=0.1, k3=0.0, **kwargs*)

AddOptions (*options=None*)

Assign an options instance to this machine.

AddQuadrupole (*name='qd', length=0.1, k1=0.0, **kwargs*)

AddRCol (*name='rc', length=0.1, xsize=0.1, ysize=0.1, **kwargs*)

AddRFCavity (*name='arreff', length=0.1, gradient=10, **kwargs*)

AddSampler (**elementnames*)

AddSextupole (*name='sx', length=0.1, k2=0.0, **kwargs*)

AddSolenoid (*name='sl', length=0.1, ks=0.0, **kwargs*)

AddTransform3D (*name='t3d', **kwargs*)

AddVKicker (*name='vk', length=0.1, angle=0.0, **kwargs*)

Append (*object*)

GetIntegratedAngle ()

Get the cumulative angle of all the bends in the machine. This is therefore the difference in angle

between the entrance and exit vectors. All angles are assumed to be in the horizontal plane so this will not be correct for rotated dipoles.

GetIntegratedLength()

Get the integrated length of all the components.

Write(filename, verbose=False)

Write the machine to a series of gmad files.

next()

class pybdsim.Builder.**Line**(name, *args)

Bases: list

A class that represents a list of Elements

Provides ability to print out the sequence or define all the components.

Example:

```
>>> d1 = Element("drift1", "drift", l=1.3)
>>> q1 = Element("q1", "quadrupole", l=0.4, k1=4.5)
>>> a = Line([d1,q1])
```

DefineConstituentElements()

Return a string that contains the lines required to define each element in the Line.

Example using predefined Elements name 'd1' and 'q1':

```
>>> l = Line([d1,q1])
>>> f = open("file.txt", "w")
>>> f.write(DefineConstituentElements())
>>> f.write(l)
>>> f.close()
```

class pybdsim.Builder.**Element**(name, category, **kwargs)

Bases: pybdsim.Builder.ElementBase

Element - an element / item in an accelerator beamline. Very similar to a python dict(ionary) and has the advantage that built in printing or string conversion provides BDSIM syntax.

Element(name,type,**kwargs)

```
>>> a = Element("d1", "drift", l=1.3)
>>> b = Element("qx1f", "quadrupole", l=(0.4,'m'), k1=0.2, aper1=(0.223,'m'))
>>> print(b)
qx1f: quadrupole, k1=0.2, l=0.4*m, aper1=0.223*m;
>>> str(c)
qx1f: quadrupole, k1=0.2, l=0.4*m, aper1=0.223*m\n;
```

A beam line element must ALWAYS have a name, and type. The keyword arguments are specific to the type and are up to the user to specify - these should match BDSIM GMAD syntax.

The value can be either a single string or number or a python tuple where the second entry must be a string (shown in second example). Without specified units, the parser assumes S.I. units.

An element may also be multiplied or divided. This will scale the length and angle appropriately.

```
>>> c = Element('sb1', 'sbend', l=(0.4,'m'), angle=0.2)
>>> d = c/2
>>> print(d)
sb1: sbend, l=0.2*m, angle=0.1;
```

This inherits and extends ElementBase that provides the basic dictionary capabilities. It adds the requirement of type / category (because 'type' is a protected keyword in python) as well as checking for valid BDSIM types.

pybdsim.Convert

Module for various conversions.

```
pybdsim.Convert._MadxTfs2Gmad.MadxTfs2Gmad(input, outputfilename, startname=None,
                                             stopname=None, stepsize=1, ignorezerolengthitems=True,
                                             thinmultipoles=False, samplers='all', aperturedict={},
                                             collimatordict={}, userdict={}, beampiperadius=5.0,
                                             verbose=False, beam=True, flipmagnets=False, usemadxaperture=False,
                                             defaultAperture='circular', biasVacuum=None, biasMaterial=None)
```

MadxTfs2Gmad convert a madx twiss output file (.tfs) into a gmad input file for bdsim

inputfile-name	path to the input file
outputfile-name	requested output file
startname	the name (exact string match) of the lattice element to start the machine at this can also be an integer index of the element sequence number in madx tfs.
stopname	the name (exact string match) of the lattice element to stop the machine at this can also be an integer index of the element sequence number in madx tfs.
stepsize	the slice step size. Default is 1, but -1 also useful for reversed line.
ignoreze-rolength-items	nothing can be zero length in bdsim as real objects of course have some finite size. Markers, etc are acceptable but for large lattices this can slow things down. True allows to ignore these altogether, which doesn't affect the length of the machine.
thinmulti-poles	will convert thin multipoles to ~1um thick finite length multipoles with upscaled k values - experimental feature
samplers	can specify where to set samplers - options are None, 'all', or a list of names of elements (normal python list of strings). Note default 'all' will generate separate outputfilename_samplers.gmad with all the samplers which will be included in the main .gmad file - you can comment out the include to therefore exclude all samplers and retain the samplers file.
aperture-info	aperture information. Can either be a dictionary of dictionaries with the the first key the exact name of the element and the daughter dictionary containing the relevant bdsim parameters as keys (must be valid bdsim syntax). Alternatively, this can be a pymadx.Aperture instance that will be queried.
collimator-dict	a dictionary of dictionaries with collimator information keys should be exact string match of element name in tfs file value should be dictionary with the following keys: "bdsim_material" - the material "angle" - rotation angle of collimator in radians "xsize" - x full width in metres "ysize" - y full width in metres
userdict	A python dictionary the user can supply with any additional information for that particular element. The dictionary should have keys matching the exact element name in the Tfs file and contain a dictionary itself with key, value pairs of parameters and values to be added to that particular element.
beampipera-dius	In metres. Default beam pipe radius and collimator setting if unspecified.
verbose	Print out lots of information when building the model.
beam	True False - generate an input gauss Twiss beam based on the values of the twiss parameters at the beginning of the lattice (startname) NOTE - we thoroughly recommend checking these parameters and this functionality is only for partial convenience to have a model that works straight away.
flipmagnets	True False - flip the sign of all k values for magnets - MADX currently tracks particles agnostic of the particle charge - BDSIM however, follows their manual definition strictly - positive k -> horizontal focussing for positive particles therefore, positive k -> vertical focussing for negative particles. Use this flag to flip the sign of all magnets.
use-madxaperture	True False - use the aperture information in the TFS file if APER_1 and APER_2 columns exist. Will only set if they're non-zero.
defaultA-perture	The default aperture model to assume if none is specified.
biasVacuum	Optional list of bias objects to written into each component definition that are attached to the vacuum volume.
biasMaterial	Optional list of bias objects to written into each component definition that are attached to volumes outside the vacuum.

Example:

```
>>> a,o = pybdsim.Convert.MadxTfs2Gmad('twiss.tfs', 'mymachine')
```

In normal mode: Returns Machine, [omittedItems]

In verbose mode: Returns Machine, Machine, [omittedItems]

Returns two `pybdsim.Builder.Machine` instances. The first desired full conversion. The second is the raw conversion that's not split by aperture. Thirdly, a list of the names of the omitted items is returned.

```
pybdsim.Convert._MadxTfs2Gmad.TfsHasRequiredColumns (tfsinstance)
```

Test the tfs file to check it has everything we need.

pybdsim.Constants module

```
pybdsim.Constants.GetPDGInd (particlename)
```

```
pybdsim.Constants.GetPDGName (particleid)
```

pybdsim.Data module

Output

Read bdsim output

Classes: Data - read various output files

```
class pybdsim.Data.BDSAsciiData (*args, **kwargs)
```

Bases: list

Filter (booleanarray)

Filter the data with a booleanarray. Where true, will return that event in the data.

Return type is BDSAsciiData

IndexFromNearestS (S)

IndexFromNearestS(S)

return the index of the beamline element closest to S

Only works if "SStart" column exists in data

MatchValue (parametername, matchvalue, tolerance)

This is used to filter the instance of the class based on matching a parameter withing a certain tolerance.

```
>>> a = pybdsim.Data.Load("myfile.txt")
>>> a.MatchValue("S", 0.3, 0.0004)
```

this will match the "S" variable in instance "a" to the value of 0.3 within +- 0.0004.

You can therefore used to match any parameter.

Return type is BDSAsciiData

NameFromNearestS (S)

```
pybdsim.Data.Load (filepath)
```

pybdsim.Gmad module

Survey() - survey a gmad lattice, plot element coords

Loader() - load a gmad file using the compiled bdsim parser

GmadFile() - modify a text based gmad file

```
class pybdsim.Gmad.GmadFile (fileName)
```

Class to determine parameters and gmad include structure

```
class pybdsim.Gmad.GmadFileBeam (fileName)
```

Class to load a gmad options file to a buffer and modify the contents

class pybdsim.Gmad.**GmadFileComponents** (*fileName*)

Class to load a gmad components file to a buffer and modify the contents

Example : python> g = pybdsim.Gmad.GmadFileComponents("./atf2_components.gmad") python> g.change("KEX1A","I","10") python> g.write("./atf2_components.gmad")

change (*element, parameter, value*)

Edit element dictionary

elementNames ()

Make a list of element names, stored in self.elementNameList

findElement (*elementName*)

Returns the start and end (inclusive location of the element lines as a tuple (start,end)

getParameter (*element, parameter*)

Edit element dictionary

getType (*element*)

parseElement (*elementString*)

Create element dictionary from element

write (*fileName*)

class pybdsim.Gmad.**GmadFileOptions** (*fileName*)

Class to load a gmad options file to a buffer and modify the contents

class pybdsim.Gmad.**Lattice** (*filename=None*)

BDSIM Gmad parser lattice.

Use this class to load a bdsim input file using the BDSIM parser (GMAD) and then interrogate it. You can use this to regenerate a lattice with less information for example

```
>>> a = Lattice("filename.gmad")
```

or

```
>>> a = Lattice()
>>> a.Load("filename.gmad")
>>> a # this will tell you some basic details
>>> print(a) # this will print out the full lattice
```

GetAllNames ()

GetAngle (*index*)

GetAper (*index*)

GetAperX (*index*)

GetAperY (*index*)

GetElement (*i*)

GetIndexOfElementNamed (*elementname*)

GetKs (*index*)

GetLength (*index*)

GetName (*index*)

GetType (*index*)

Load (*filename*)

Load the BDSIM input file and parse it using the BDSIM parser (GMAD).

ParseLattice ()

Put lattice data into python data structure

Print (*includeheaderlines=True*)

PrintZeroLength (*includeheaderlines=True*)
 Print elements with zero length with s location
next ()

class pybdsim.Gmad.**Survey** (*filename=None*)
 Survey - load a gmad lattice and have a look

Example:

```
>>> a = Survey()
>>> a.Load('mylattice.gmad')
>>> a.Plot()
```

CompareMadX (*fileName*)
FinalDiff ()
FindClosestElement (*coord*)
Load (*filename*)
Plot ()
Step (*angle, length*)

pybdsim.Joinhistograms module

pybdsim.Joinhistograms.**JoinRootHistograms** (*inputdir='./',* *outputfile-*
name='output.root')

pybdsim.ModelProcessing module

ModelProcessing

Tools to process existing BDSIM models and generate other versions of them.

pybdsim.ModelProcessing.**GenerateFullListOfSamplers** (*inputfile, outputfile*)
 inputfile - path to main gmad input file

This will parse the input using the compiled BDSIM parser (GMAD), iterate over all the beamline elements and generate a sampler for every elements. Ignores samplers, but may include already defined ones in your own input.

pybdsim.ModelProcessing.**WrapLatticeAboutItem** (*maingmadfile, itemname, outputfile-*
name)

pybdsim.Options module

class pybdsim.Options.**Editor** (*fileName*)
 pybdsim.Options.**ElectronColliderOptions** ()
 pybdsim.Options.**MinimumStandard** ()
class pybdsim.Options.**Options** (**args, **kwargs*)
 Bases: dict
ReturnOptionsString ()
SetBLMLength (*length=50, unitsstring='cm'*)
SetBLMRadius (*radius=5, unitsstring='cm'*)
SetBeamPipeRadius (*beampiperadius=5, unitsstring='cm'*)
SetBeamPipeThickness (*bpt, unitsstring='mm'*)

SetBuildTunnel (*tunnel=False*)

SetBuildTunnelFloor (*tunnelfloor=False*)

SetCherenkovOn (*on=True*)

SetChordStepMinimum (*csm=1, unitsstring='nm'*)

SetDefaultRangeCut (*drc=0.7, unitsstring='mm'*)

SetDeltaChord (*dc=0.001, unitsstring='m'*)

SetDeltaIntersection (*di=10, unitsstring='nm'*)

SetDeltaOneStep (*dos=10, unitsstring='nm'*)

SetELossHistBinWidth (*width*)

SetEMLeadParticleBiasing (*on=True*)

SetEPAnnihilation2HadronEnhancementFactor (*ef=2*)

SetEPAnnihilation2MuonEnhancementFactor (*ef=2*)

SetGamma2MuonEnhancementFactor (*ef=2*)

SetIncludeIronMagField (*iron=True*)

SetLPBFraction (*fraction=0.5*)

SetLengthSafety (*ls=10, unitsstring='um'*)

SetMaximumEpsilonStep (*mes=1, unitsstring='m'*)

SetMinimumEpsilonStep (*mes=10, unitsstring='nm'*)

SetNGenerate (*nparticles=10*)

SetNLinesIgnore (*nlines=0*)

SetNPerFile (*nperfile=100*)

SetOuterDiameter (*outerdiameter=2, unitsstring='m'*)

SetPhysicsList (*physicslist=''*)

SetPipeMaterial (*bpm*)

SetProductionCutElectrons (*pc=100, unitsstring='keV'*)

SetProductionCutPhotons (*pc=100, unitsstring='keV'*)

SetProductionCutPositrons (*pc=100, unitsstring='keV'*)

SetRandomSeed (*rs=0*)

SetSRLowX (*lowx=True*)

SetSRMultiplicity (*srm=2.0*)

SetSamplerDiameter (*radius=10, unitsstring='m'*)

SetSensitiveBeamPipe (*on=True*)

SetSensitiveBeamlineComponents (*on=True*)

SetSenssitiveBLMs (*on=True*)

SetSoilMaterial (*sm*)

SetSoilThickness (*st=4.0, unitsstring='m'*)

SetStopTracks (*stop=True*)

SetStoreMuonTrajectory (*on=True*)

SetStoreNeutronTrajectory (*on=True*)

```

SetStoreTrajectory (on=True)
SetSynchRadiationOn (on=True)
SetThresholdCutCharged (tcc=100, unitsstring='MeV')
SetThresholdCutPhotons (tcp=1, unitsstring='MeV')
SetTrackSRPhotons (track=True)
SetTrajectoryCutGTZ (gtz=0.0, unitsstring='m')
SetTrajectoryCutLTR (ltr=10.0, unitsstring='m')
SetTunnelFloorOffset (offset=1.0, unitsstring='m')
SetTunnelMaterial (tm)
SetTunnelOffsetX (offset=0.0, unitsstring='m')
SetTunnelOffsetY (offset=0.0, unitsstring='m')
SetTunnelRadius (tunnelradius=2, unitsstring='m')
SetTunnelThickness (tt=1.0, unitsstring='m')
SetVacuumMaterial (vm)
SetVacuumPressure (vp)
    Vacuum pressure in bar

```

```
pybdsim.Options.ProtonColliderOptions()
```

pybdsim.Plot module

Useful plots for bdsim output

```

pybdsim.Plot.AddMachineLatticeFromSurveyToFigure (figure, surveyfile)
pybdsim.Plot.AddMachineLatticeToFigure (figure, tfsfile)
pybdsim.Plot.CompareBDSIMSurveyWithMadXTfs (tfsfile, bdsfile, title='', outputfile-
                                             name=None)
pybdsim.Plot.MadxGmadComparison (tfsfile, gmadfile, title='', outputfilename=None)
pybdsim.Plot.MadxTfsBeta (tfsfile, title='', outputfilename=None)
pybdsim.Plot.MadxTfsBetaSimple (tfsfile, title='', outputfilename=None)

```

pybdsim.Root module

```

pybdsim.Root.GetDataFromROOTHist (hist)
pybdsim.Root.GetMetaDataFromROOTHist (hist)
pybdsim.Root.PlotTH1Bar (hist, edgecolor='none', color='b', label='', newFigure=True)
pybdsim.Root.PlotTH1Hist (hist, edgecolor='none', color='b', label='', newFigure=True)

```

pybdsim.Visualisation module

```

class pybdsim.Visualisation.Helper (surveyFileName)
    To help locate objects in the BDSIM visualisation, requires a BDSIM survey file

    draw ()
        Quick survey drawing for diagnostic reasons

```

findComponentCoords (*componentName*)

Returns the XYZ coordinates of a component relative to the centre

getWorldCentre (*type='linear'*)

Returns the center in world coordinates of the centre of the visualisation space

11.2.2 pymadx

Module contents

pymadx - Madx TFS file parser

Authors: L. Nevay, S. Boogert 2014 11 14

Tfs - a loader class Visualisation - visualisation of the loaded lattice

class pymadx.**Tfs** (*filename=None, **kwargs*)

Bases: object

MADX Tfs file reader

```
>>> a = Tfs()
>>> a.Load('myfile.tfs')
>>> a.Load('myfile.tar.gz') -> extracts from tar file
```

or

```
>>> a = Tfs("myfile.tfs")
```

a has data members:

header - dictionary of header items

columns - list of column names

formats - list of format strings for each column

data - dictionary of entries in tfs file by name string

sequence - list of names in the order they appear in the file

nitems - number of items in sequence

NOTE: if no column “NAME” is found, integer indices are used instead

See the various methods inside a to get different bits of information:

```
>>> a.ReportPopulations?
```

Examples:

```
>>> a['IP.1'] #returns dict for element named "IP.1"
>>> a[:30]    #returns list of dicts for elements up to number 30
>>> a[345]    #returns dict for element number 345 in sequence
```

Clear ()

Empties all data structures in this instance.

ColumnIndex (*columnstring*)

Return the index to the column matching the name

REMEMBER: excludes the first column NAME 0 counting

GetColumn (*columnstring*)

Return a numpy array of the values in columnstring in order as they appear in the beamline

GetColumnDict (*columnstring*)
 return all data from one column in a dictionary
 note not in order

GetElementNamesOfType (*typename*)
 GetElementNamesOfType(typename)

Returns a list of the names of elements of a certain type. Typename can be a single string or a tuple or list of strings.

Examples: >>> GetElementsOfType('SBEND') >>> GetElementsOfType(['SBEND','RBEND'])
 >>> GetElementsOfType(('SBEND','RBEND','QUADRUPOLE'))

GetElementsOfType (*typename*)

Returns a Tfs instance containing only the elements of a certain type. Typename can be a single string or a tuple or list of strings.

This returns a Tfs instance with all the same capabilities as this one.

GetRow (*elementname*)
 Return all data from one row as a list

GetRowDict (*elementname*)
 Return a dictionary of all parameters for a specific element given by element name.
 note not in order

GetSegment (*segmentnumber*)

IndexFromName (*namestring*)
 Return the index of the element named namestring

IndexFromNearestS (*S*)
 IndexFromNearestS(S)
 return the index of the beamline element closest to S

InterrogateItem (*itemname*)
 Print out all the parameters and their names for a particular element in the sequence identified by name.

Load (*filename*)

```
>>> a = Tfs()
>>> a.Load('filename.tfs')
```

Read the tfs file and prepare data structures. If 'tar' or 'gz' are in the filename, the file will be opened still compressed.

NameFromIndex (*integerindex*)
 return the name of the beamline element at index

NameFromNearestS (*S*)
 NameFromNearestS(S)
 return the name of the beamline element closest to S

Plot (*filename='optics.pdf'*)

PlotSimple (*filename='optics.pdf'*)

ReportPopulations ()
 Print out all the population of each type of element in the beam line (sequence)

next ()

class pymadx.**TfsArray** (*filename=None*)

Clear ()
Clear() empty all internal variables

ColumnIndex (*columnstring*)
ColumnIndex(columnname):

return the index to the column matching the name

REMEMBER: excludes the first column NAME 0 counting

GetColumn (*colname*)

GetElementsOfType (*typename*)
GetElementsOfType(typename)

Returns a list of the names of elements of a certain type

typename can be a single string or a tuple or list of strings

ie GetElementsOfType('SBEND') GetElementsOfType(['SBEND','RBEND']) GetElementsOfType(('SBEND','RBEND','QUADRUPOLE'))

GetRow (*rowindex*)

GetSegment (*segmentindex*)

InterrogateItem (*index*)
Print out all the parameters and their names for a particular element in the sequence identified by name

Load (*filename*)
Load('filename.tfs') read the tfs file and prepare data structures

NameFromIndex (*index*)

ReportPopulations ()
Print out all the population of each type of element in the beam line (sequence)

next ()

class pymadx.**Beam** (*particletype='e-', energy=1.0, distrtype='reference', *args, **kwargs*)
Bases: dict

ReturnBeamString ()

ReturnPtcString ()

SetDistributionType (*distrtype='reference'*)

SetEnergy (*energy=1.0, unitsstring='GeV'*)

SetParticleType (*particletype='e-'*)

SetT0 (*t0=0.0, unitsstring='s'*)

pymadx.**MadxTfs2Ptc** (*input, outputfilename, ptcfile, startname=None, stopname=None, ignorezerolengthitems=True, samplers='all', beampiperadius=0.2, beam=True*)

class pymadx.**PtcPlot** (*ptcInput=None, ptcOutput=None*)

class pymadx.**PtcAnalysis** (*ptcInput=None, ptcOutput=None*)
Bases: object

CalculateOpticalFunctions (*output*)
Calculates optical functions from a PTC output file

output - the name of the output file

SamplerLoop ()

pymadx.Beam module

```
class pymadx.Beam.Beam (particletype='e-', energy=1.0, distrtype='reference', *args, **kwargs)
    Bases: dict

    ReturnBeamString ()

    ReturnPtcString ()

    SetDistributionType (distrtype='reference')

    SetEnergy (energy=1.0, unitsstring='GeV')

    SetParticleType (particletype='e-')

    SetT0 (t0=0.0, unitsstring='s')
```

pymadx.Plot module

Plotting script for madx TFS files using the pymadx Tfs class

```
pymadx.Plot.AddMachineLatticeToFigure (figure, tfsfile)

pymadx.Plot.PlotTfsBeta (tfsfile, title='', outputfilename=None, dispersion=False)
    Plot sqrt(beta x,y) as a function of S as well as the horizontal dispersion.

    Also adds machine lattice at the top of the plot

pymadx.Plot.PlotTfsBetaSimple (tfsfile, title='', outputfilename=None)
    Plot the sqrt(beta x,y) as a function of S
```

pymadx.Ptc module

```
class pymadx.Ptc.FlatGenerator (mux=0.0, widthx=0.001, mupx=0.0, widthpx=0.001, muy=0.0,
                                widthy=0.001, mupy=0.0, widthpy=0.001)
    Bases: object

    Simple ptc inray file generator - even distribution

    Generate (nToGenerate=100, fileName='inrays.madx')
        returns an Inrays structure

class pymadx.Ptc.GaussGenerator (gemx=1e-10, betax=0.1, alfx=0.0, gemy=1e-10, betay=0.1,
                                  alfy=0.0, sigmat=1e-12, sigmapt=1e-12)
    Bases: object

    Simple ptx inray file generator

    Generate (nToGenerate=1000, fileName='inrays.madx')
        returns an Inrays structure

class pymadx.Ptc.Inray (x=0.0, px=0.0, y=0.0, py=0.0, t=0.0, pt=0.0)
    Class for a madx ptc input ray x : horizontal position [m] px : horizontal canonical momentum p_x/p_0 y :
    vertical position [m] py : vertical canonical momentum p_y/p_0 t : c*(t-t0) [m] pt : (delta-E)/(pc)

    use str(Inray) to get the representation for file writing

class pymadx.Ptc.Inrays
    Bases: list

    Class based on python list for Inray class

    AddParticle (x=0.0, px=0.0, y=0.0, py=0.0, t=0.0, pt=0.0)

    Clear ()

    Plot ()

    Statistics ()
```

Write (*filename*)

`pymadx.Ptc.LoadInrays` (*fileName*)

Load input rays from file *fileName* : `inrays.madx` return : Inrays instance

`pymadx.Ptc.PlotInrays` (*i*)

Plot Inrays instance, if input is a sting the instance is created from the file

`pymadx.Ptc.WriteInrays` (*fileName*, *inrays*)

`pymadx.PtcAnalysis` module

`class pymadx.PtcAnalysis.PtcAnalysis` (*ptcInput=None*, *ptcOutput=None*)

Bases: `object`

CalculateOpticalFunctions (*output*)

Calulates optical functions from a PTC output file

output - the name of the output file

SamplerLoop ()

`pymadx.PtcPlot` module

`class pymadx.PtcPlot.PtcPlot` (*ptcInput=None*, *ptcOutput=None*)

11.2.3 pymad8

Module contents

`pymad8.Converter` module

`class pymad8.Converter.Mad8` (*fileName*)

getKeys (*keylist*, *l*)

parseFile (*mom=1300000000.0*, *line='ATF2'*)

parseLine (*l*)

readFile (*fileName*)

Readfile into long uncontinued lines

`pymad8.Mad8` module

`class pymad8.Mad8.Chrom`

Bases: `pymad8.Mad8.General`

Chromaticity data structure data : numpy array of data keys : key to data

getData (*index*)

`class pymad8.Mad8.Common`

Bases: `pymad8.Mad8.General`

containsEnergyVariation ()

Method to determine if the energy is constant in the lattice Required if there is 1) RfCavities

getApertures (*raw=True*)

getColumn (*colName*)

```

    getData (index)
    getRowByIndex (index)
    getRowByName (name)
    keys = {'hmonitor': {'note': 10, 'aper': 9, 'E': 11, 'I': 0}, 'lcav': {'aper': 9, 'E': 11, 'lag': 7, 'I': 0, 'note': 10, 'volt': 6}
    makeLocationList (elementNames=[])
class pymad8.Mad8.Envelope
    Bases: pymad8.Mad8.General
    Beam envelope data structure data : numpy array of data keys : key to data
    getData (index)
    keys = {'s43': 20, 's54': 27, 's42': 19, 's13': 2, 's12': 1, 's11': 0, 's36': 17, 's31': 12, 's16': 5, 's15': 4, 's14': 3, 's35': 1}
class pymad8.Mad8.General
    General list of accelerator component infomation
    addElement (type, name, data)
    findByName (name)
    findByType (type)
    getColumn (key)
    getIndex (name)
    getNElements ()
    getNames (ind)
    getRowByIndex (index)
    getRowByName (name)
    makeArray ()
    plotXY (xkey, ykey)
    subline (start, end)
class pymad8.Mad8.OutputReader
    Class to load different Mad8 output files Usage : o = Mad8.OutputReader() [c,
    s] = o.readFile('./survey.tape','survey') [c, r] = o.readFile('./rmat.tape','rmat') [c, t]
    = o.readFile('./twiss.tape','twiss') [c, c] = o.readFile('./chrom.tape','chrom') [c, e] =
    o.readFile('./envelope.tape','envel')
    c : Common data r : Rmat object t : Twiss object c : Chrom object e : Envelope object
    readChromFile (f=None)
    readEnvelopeFile (f=None)
    readFile (fileName='', type='twiss')
        read mad8 output file
    readRmatFile (f=None)
    readSurveyFile ()
    readTwissFile (f=None)
class pymad8.Mad8.Rmat
    Bases: pymad8.Mad8.General
    Rmatrix data structure data : numpy array of data keys : key to data
    getData (index)
    keys = {'r16': 5, 'r14': 3, 'r15': 4, 'r12': 1, 'r13': 2, 'r11': 0, 'r42': 19, 'r45': 22, 'r44': 21, 'r61': 30, 'r46': 23, 'r41': 1}

```

```
class pymad8.Mad8.Survey
```

```
    Bases: pymad8.Mad8.General
```

```
    Survey data structure data : numpy array of data keys : key to data
```

```
    keys = {'phi': 5, 'psi': 6, 'suml': 3, 'theta': 4, 'y': 1, 'x': 0, 'z': 2}
```

```
class pymad8.Mad8.Twiss
```

```
    Bases: pymad8.Mad8.General
```

```
    Twiss data structure data : numpy array of data keys : key to data
```

```
    keys = {'bety': 6, 'betx': 1, 'suml': 14, 'dx': 3, 'dy': 8, 'alfy': 5, 'alfx': 0, 'px': 11, 'py': 13, 'muy': 7, 'mux': 2, 'dpy': 13}
```

```
    plotAlf()
```

```
    plotBeta()
```

```
    plotEta()
```

```
    plotEtaPrime()
```

```
    plotMu()
```

```
pymad8.Mad8.getValueByName(name, key, common, table)
```

pymad8.Mad8Converter module

```
class pymad8.Mad8Converter.Mad8Converter(fileName, ngenerate=1000)
```

```
    convert()
```

pymad8.Plot module

```
pymad8.Plot.apertures(name='ebds1')
```

```
pymad8.Plot.drawMachineLattice(mad8c, mad8t)
```

```
pymad8.Plot.energy(name='ebds1')
```

```
pymad8.Plot.linearOptics(name='ebds1')
```

```
pymad8.Plot.phaseAdvance(name='ebds1')
```

```
pymad8.Plot.setCallbacks(figure, axm, axplot)
```

```
pymad8.Plot.survey(name='ebds1')
```

pymad8.Saveline module

```
class pymad8.Saveline.Loader(fileName)
```

```
    expand_file()
```

```
    fileAnalysis()
```

```
    flatten_elements(element_name)
```

```
    loadFile(fileName)
```

```
pymad8.Saveline.SavelineTest(file_name)
```

pymad8.Sim module

```
class pymad8.Sim.Track (common, rmat)
```

```
    generate ()
```

```
    trackParticle (p)
```

```
    trackParticles (nparticle)
```

```
pymad8.Sim.testTrack (rmatFile, nparticle=10)
```

pymad8.Visualisation module

```
class pymad8.Visualisation.OneDim (common, survey, debug)
```

```
    drawBend (c, s, suml, colour=True)
```

```
    drawElement (elem, colour=True)
```

```
    drawElements (type, colour=True)
```

```
    drawHkic (c, s, suml, colour=True)
```

```
    drawInst (c, s, suml, colour=True)
```

```
    drawMark (c, s, suml, colour=True)
```

```
    drawMoni (c, s, suml, colour=True)
```

```
    drawMult (c, s, suml, colour=True)
```

```
    drawProf (c, s, suml, colour=True)
```

```
    drawQuad (c, s, suml, colour=True)
```

```
    drawSext (c, s, suml, colour=True)
```

```
    drawVkic (c, s, suml, colour=True)
```

```
    drawWire (c, s, suml, colour=True)
```

```
    plot (colour=True)
```

```
class pymad8.Visualisation.TwoDim (common, survey, debug=False, annotate=False,  
                                   fancy=False)
```

```
    drawBend (c, s, x, y, z)
```

```
    drawElement (elem)
```

```
    drawElements (type)
```

```
    drawMark (c, s, x, y, z)
```

```
    drawMoni (c, s, x, y, z)
```

```
    drawQuad (c, s, x, y, z)
```

```
    plot (event=None)
```

```
    plotUpdate (event)
```

```
pymad8.Visualisation.testOneDim ()
```

```
pymad8.Visualisation.testTwoDim ()
```

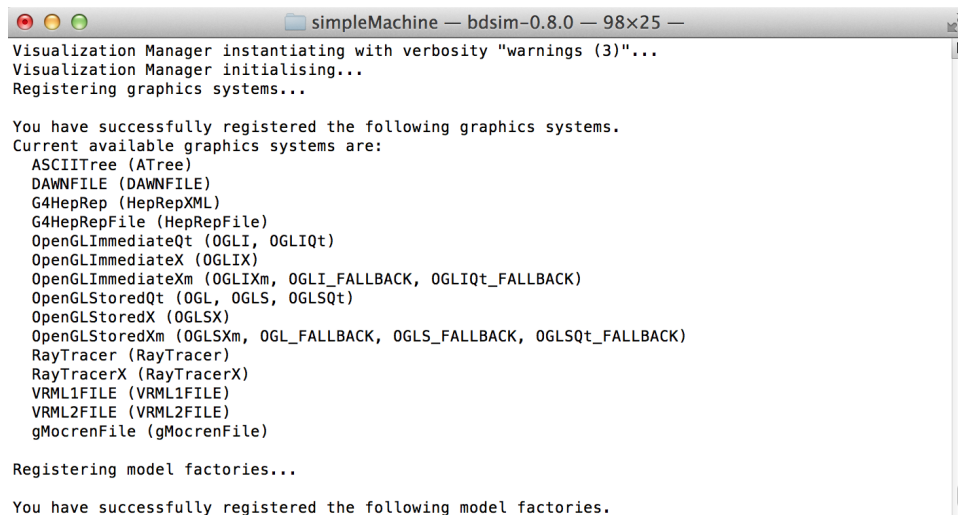
```
pymad8.Visualisation.transformedPoly (xy, xyc, theta)
```

```
pymad8.Visualisation.transformedRect (xyc, dx, dy, theta)
```


VISUALISATION

When working interactively with BDSIM, the Geant4 visualisers are used. When BDSIM is started interactively (without the `--batch` command), Geant4 will print a list of all available visualisers that Geant4 has been compiled with on your system. Should you wish to extend this list, you must follow the Geant4 instructions, which typically involves installing the necessary dependencies (such as qt4), reconfiguring and recompiling Geant4. BDSIM should then be subsequently reconfigured (rerun `cmake`) and recompiled.

As an example an excerpt from the terminal output shows the following list of available visualisers on an example system.



```
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 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, OGLS_FALLBACK, OGLSQt_FALLBACK)
  RayTracer (RayTracer)
  RayTracerX (RayTracerX)
  VRML1FILE (VRML1FILE)
  VRML2FILE (VRML2FILE)
  gMocrenFile (gMocrenFile)

Registering model factories...

You have successfully registered the following model factories.
```

By default, BDSIM uses the **OpenGL Qt** visualiser - we highly recommend this as it's very easy to work with and has a rich feature set.

12.1 Default and Custom Visualisation

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, add a few useful buttons and menus to the user interface. To use these, the user need only **not** specify a visualisation macro.:

```
bdsim --file=mylatttice.gmad
```

- Note, 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=mylatttice.gmad --vis_mac=othervis.mac
```

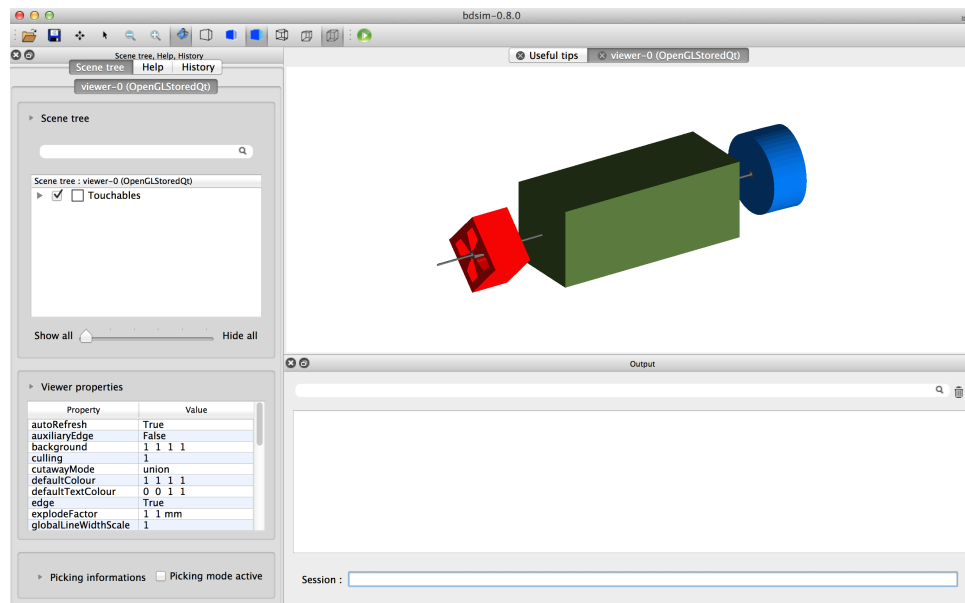
where `othervis.mac` is your visualisation macro.

The BDSIM visualisation macros can be found in the `bdsim` source directory as follows:

```
bdsim/vis/*.mac
```

12.2 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.

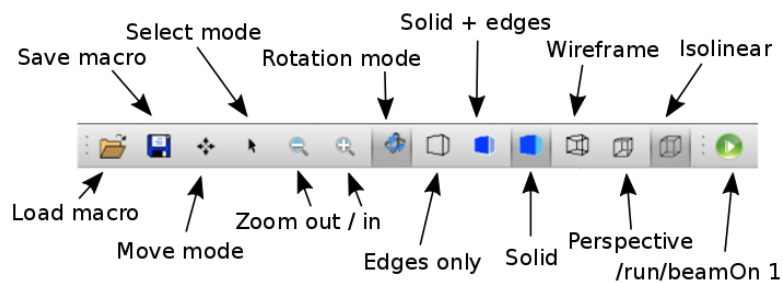
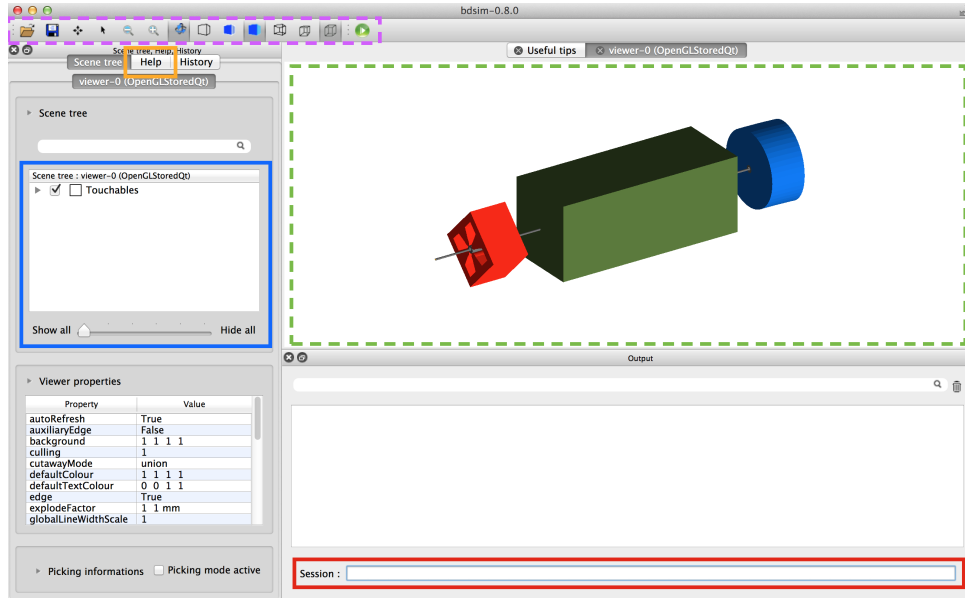
Note: You must use “exit” in the session box at the bottom to exit BDSIM properly.

12.2.1 Control Buttons

12.2.2 Common Useful Commands

A list of our most commonly used commands

- `/vis/viewer/set/viewpointThetaPhi 0 90` - set the view point angle
- `/vis/scene/add/axes 0 0 0` - add a set of unit vector axes at position (0,0,0)
- `/run/beamOn 3` - run 3 primary events
- `exit` - exit the visualiser and BDSIM



EXAMPLES & TESTS

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 the each example.

13.1 Features

13.1.1 Geometry

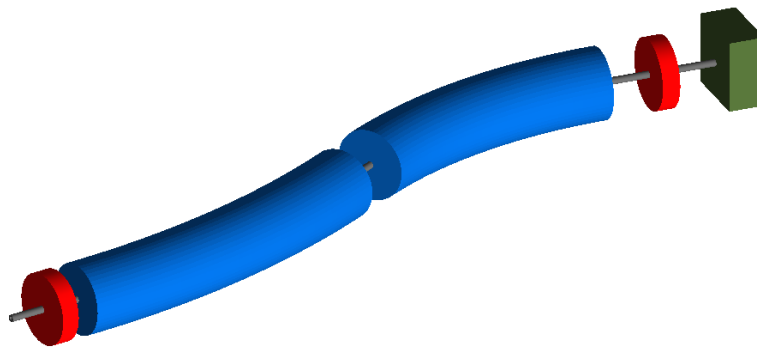
Layout

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
```

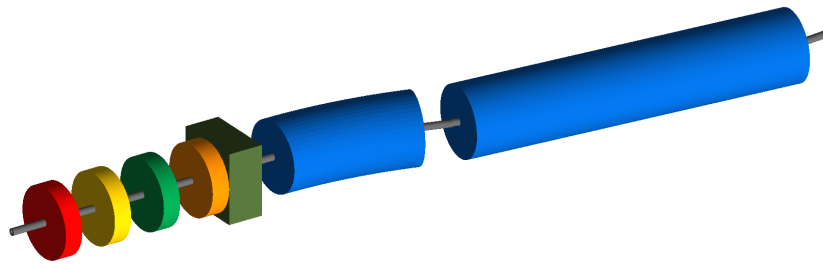


2_one_of_each.gmad

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:

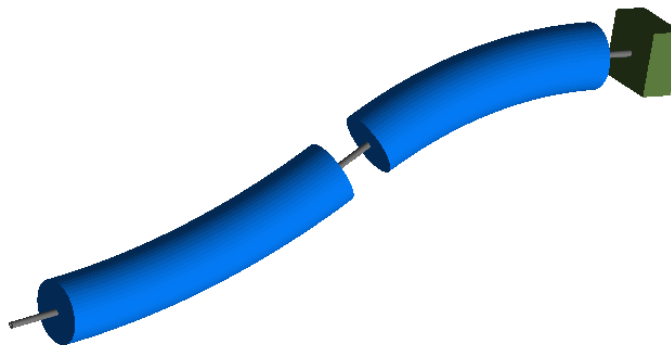
```
bdsim --file=1_rotation.gmad
```

2_rotation_with_bends.gmad

A pair of sector bends but rotated to bend vertically, followed by a 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
```



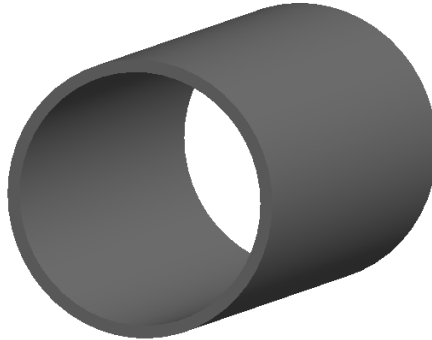
Beam Pipes

1_circular.gmad

A 0.2m section of circular beam pipe - nothing particularly interesting.

How to run:

```
bdsim --file=1_circular.gmad
```

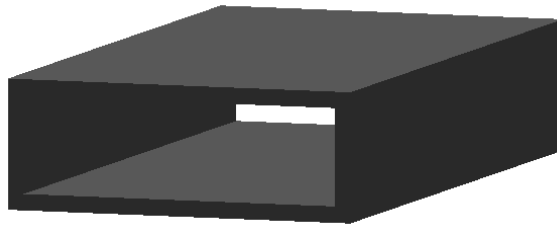


2_rectangular.gmad

A 0.2m section of rectangular beam pipe.

How to run:

```
bdsim --file=2_rectangular.gmad
```

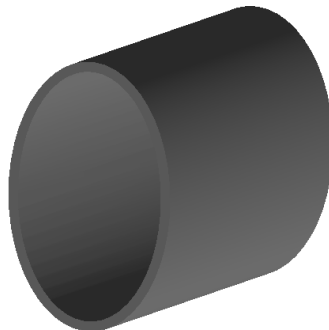


3_elliptical.gmad

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
```



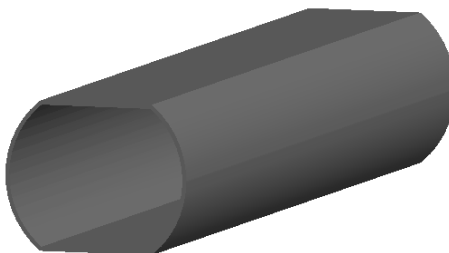
4_lhc.gmad

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 (de-

generate) beampipeRadius option in options.gmad.

How to run:

```
bdsim --file=4_lhc.gmad
```

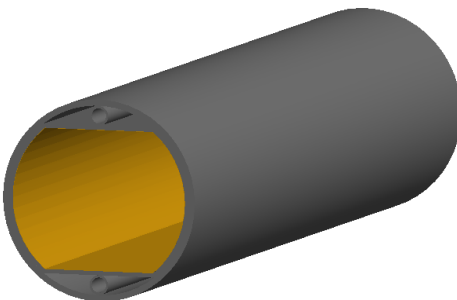


5_lhcdetailed.gmad

Similar to 4), a 0.2m section of lhc-style beam pipe but with the more detailed lhc aperture model.

How to run:

```
bdsim --file=5_lhcdetailed.gmad
```

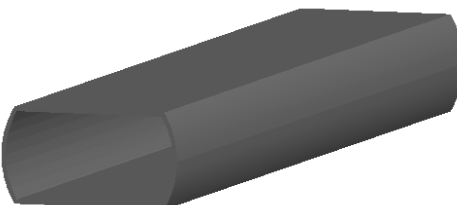


6_rectellipse.gmad

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
```

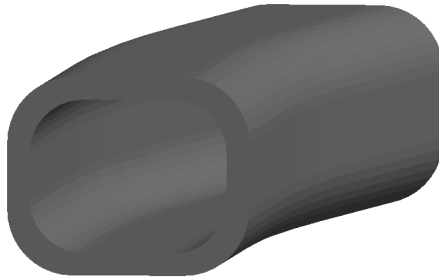


7_racetrack.gmad

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
```

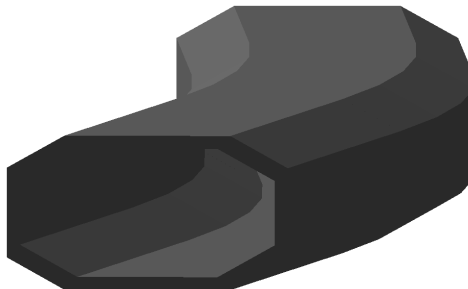


8_octagonal.gmad

A small section of beam pipe with an octagonal aperture style. This is a rectangle with flat cut corners.

How to run:

```
bdsim --file=8_octagonal.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:

```
magnetGeometryType="cylindrical";
```

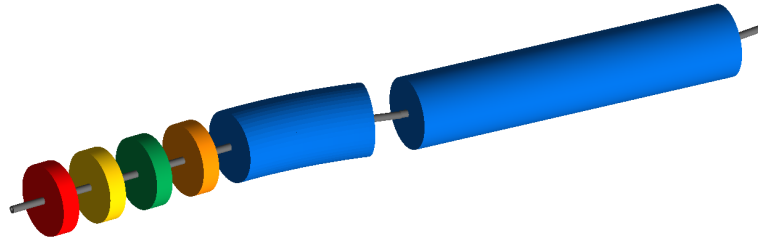
This example uses the sequence of magnets defined in `one_of_each_base.gmad` and ensures that they have cylindrical geometry. This is a little redundant as the default is cylindrical.

How to run:

```
bdsim --file=1_cylindrical.gmad
```

2_poles_circular.gmad

Key code:

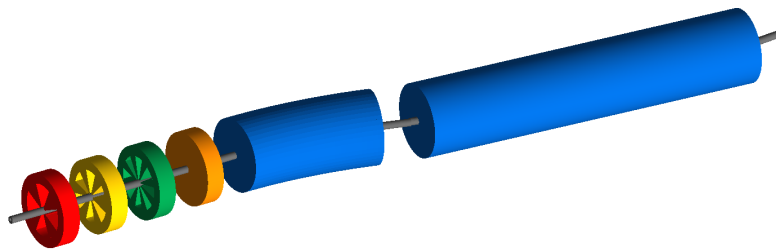


```
magnetGeometryType="polescircular";
```

As above but with poled geometry and a circular yoke.

How to run:

```
bdsim --file=2_poles_circular.gmad
```



3_poles_square.gmad

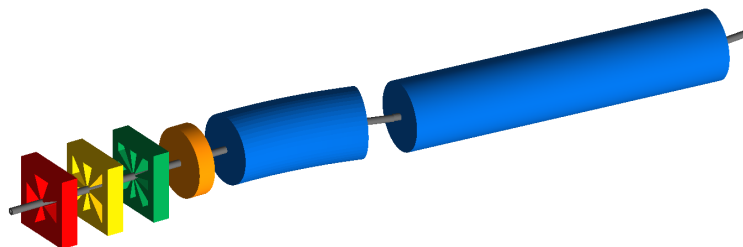
Key code:

```
magnetGeometryType="polessquare";
```

As above but with poled geometry and a square yoke.

How to run:

```
bdsim --file=3_poles_square.gmad
```



4_poles_face.gmad

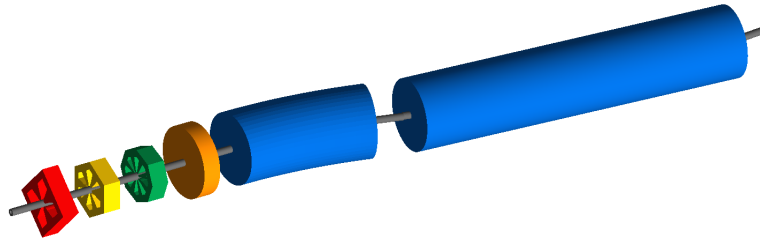
Key code:

```
magnetGeometryType="polesfacet";
```

As above but with poled geometry and a square yoke rotated by 45 degrees.

How to run:

```
bdsim --file=4_poles_facet.gmad
```



5_poles_face_crop.gmad

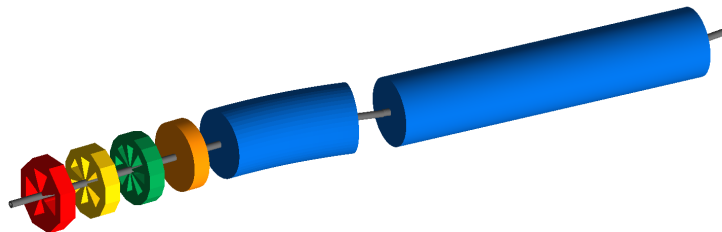
Key code:

```
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, $nx2$ edges, where n is the number of poles the magnet has.

How to run:

```
bdsim --file=5_poles_facet_crop.gmad
```



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.

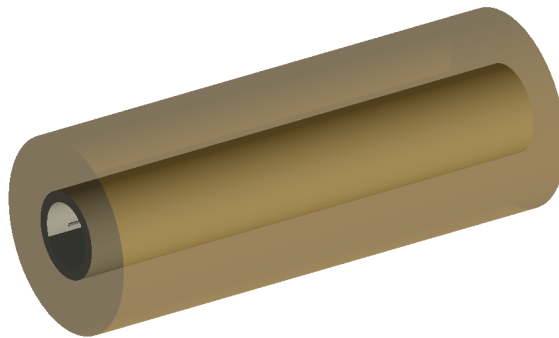
This test builds one straight section of tunnel from start to finish irrespective of the beam line as controlled by `buildTunnelStraigh=1`.

Key code:

```
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:

```
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

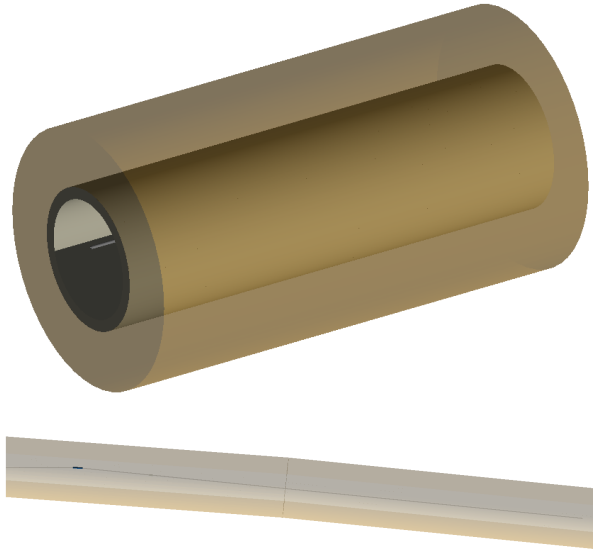
This lattice has a relatively strong bend at the beginning of the lattice, followed by a long straight section. This tests the tunnel building algorithm's ability to follow the beam line after a bend.

How to run:

```
bdsim --file=3_initial_bend.gmad
```

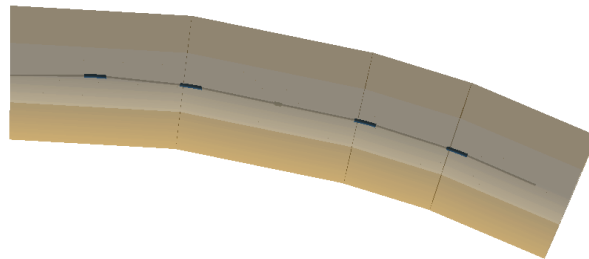
4_several_bends.gmad

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 examples is much like [4_several_bends.gmad](#) but also bends the otherway (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.

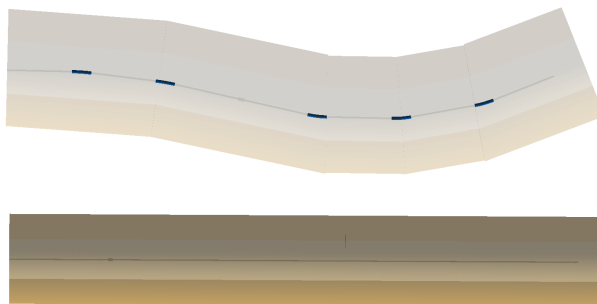
How to run:

```
bdsim --file=6_very_long_following.gmad
```

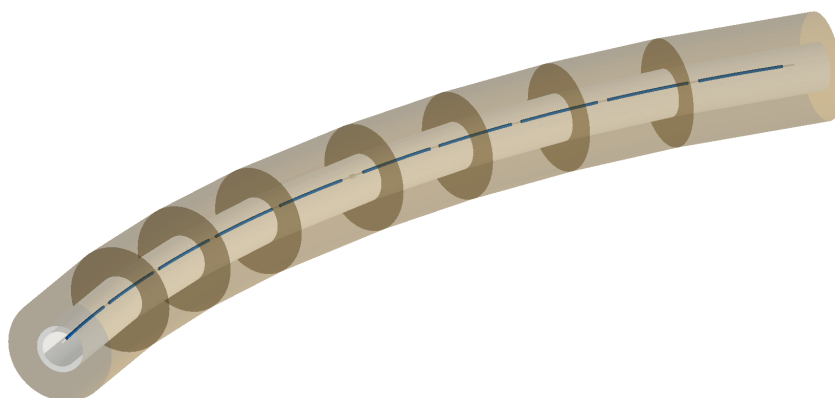
7_long_arc.gmad

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 examples 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 $1\ \mu\text{m}$ gap to avoid geometrical overlaps.

How to run:

```
bdsim --file=8_samplers.gmad
```

13.1.2 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
```

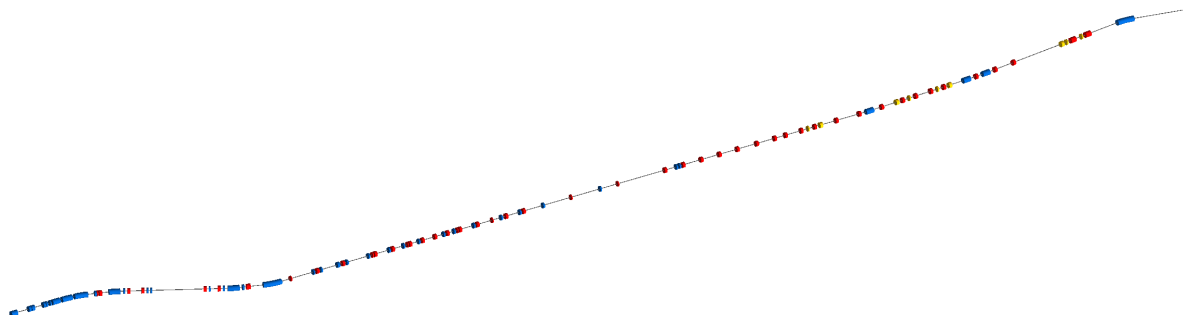


13.2 Accelerator Test Facility 2 - KEK, Japan

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.

How to run:

```
bdsim --file=atf2.gmad --output=root --outfile=test1
```



13.3 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 (2015).

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).

13.4 Large Hadron Collider

The Large Hadron Collider is a approximately 27km in circumference proton proton collider at CERN, Gevena, 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).

13.4.1 LHC Ring

This is the 2012 4TeV lattice for beam 1 fully converted using pybdsim.

13.4.2 LHC Injection

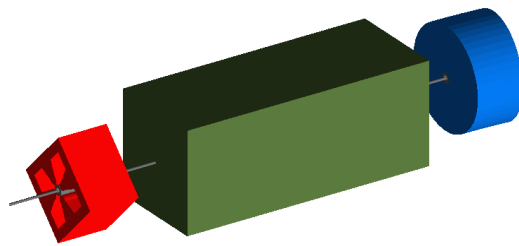
Old model kept for archival purposes. Unsupported.

13.5 Simple Machine

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=simpleMachine.gmad
```

SUPPORT

All support issues can be submitted to our [issue tracker](#)

14.1 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.

14.2 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.

BDSIM VERSION HISTORY

15.1 V0.92 - 2016 / 03 / 29

15.1.1 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.

15.1.2 Bug fixes

Geometry

- Tunnel geometry fixes (issues #88 and #89).

Parser

- Fix 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.

15.1.3 Utilities

- pymadx v0.4
- pybdsim v0.5
- pymad8 v0.3
- robdsim v0.5

15.2 V0.91 - 2015 / 12 / 17

15.2.1 New Features

- New tests for file IO, coordinate transforms, aperture models, extra optical lattice patterns and general ring example.

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, ie collimators.

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.

15.2.2 Bug fixes

Geometry

- Fix for LHC detailed geometry when beam shield is rotated.
- Consolidation and improvement of aperture parameter validity testing.
- Fix for femtometre occasional overlaps in magnet outer geometry.
- Fixed placement overlaps in rbend.
- Fixed seg-fault 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 mid point.
- Particles no longer stepped by teleporter at beginning of 1st turn if starting behind starting mid point.
- Fix 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)

15.2.3 Utilities

- pymadx v0.3
- pybdsim v0.4
- pymad8 v0.2
- robdsim v0.4

15.3 V0.9 - 2015 / 11 / 10

15.3.1 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
- 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

15.3.2 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)
- Increase tolerance for sector bends (issue #73)
- Protect 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 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)
- Improve memory and cpu for output writing (issue #86)

15.3.3 Utilities

- pymadx v0.2
- pybdsim v0.3
- pymad8 v0.2
- robdsim v0.3

15.4 V0.8 - 2015 / 08 / 10

15.4.1 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)
- *stopSecondaries* option
- Remove dependency on boost (issue #57)
- Restructured examples directory - top level contains only full machines and subdirectories contain features.
- Example documentation in manual and in place beside each example with example screenshots
- Updated python utilities *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
- Introduce 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

15.4.2 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
- Fix 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
- Fix for 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 and is also controllable with *printModuloFraction* option
- Protect against bad user specified values of *lengthSafety* to avoid geometry overlaps
- Improved parser speed

Physics

- SR radiation fix in dipole (issue #53)
- Removed continuous synchrotron radiation as 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

- Change all transverse output units to **metres** - manual updated accordingly
- Change *z* in ASCII output to **global Z** instead of local *z*.
- Recorded energy in output is now unweighted but energy recorded in convenience energy loss histogram is. Could have lead to double weighting previously
- Fix for 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

- Fix for wrong print out warning due to logic error (issue #51)
- Fix for boundary effects of energy deposition (issue #52)
- Fix large memory leak for events with large number of particles - was due to accumulation of BDSTrajectory objects

15.5 V0.702 2015 / 07 / 28 - Hotfix

- Fix for physics production range cuts were not obeyed in simulation

15.6 V0.701 2015 / 07 / 02 - Hotfix

- Fix for global X coordinate not written to output for energy deposition

15.7 V0.7 - 2015 / 06 / 30

15.7.1 New Features

- Ability to write no output
- New magnet geometry factories introduced with 7 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).

15.7.2 Bug fixes

- Fixes to overlapping volumes and tracking errors in beam pipes.
- Fix for wrong transverse coordinates for geometry other than cylindrical magnets (issue #30).
- Histograms now written to disk in case of crash or kill signal (issue #38).
- Fix for uncontrolled memory consumption for synchrotron radiation (issue #36).
- Fix 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.

15.8 V0.65 - 2015 / 04 / 10

- New base class for any geometrical object BDSGeometryComponent.
- New interchangeable beam pipes with 6 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.

15.9 V0.64 - 2015 / 02 / 16

- New histogram manager and factorisation of histograms from outputs.
- Extra per element histograms.
- Basic implementation of valid solenoid.

15.10 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

15.11 V0.62 - 2014 / 08 / 07

15.12 V0.61 - 2014 / 08 / 05

- Geant4 version 10 support

15.13 v0.6 - 2013 / 12 / 02

15.14 v0.5 - 2008 / 11 / 08

15.15 v0.4 - 2008 / 02 / 26

15.16 v0.3 - 2007 / 01 / 26

15.17 v0.2 - 2006 / 05 / 18

15.18 v0.1 - 2006 / 02 / 22

15.19 beta - 2005 / 05 / 01

DEVELOPER DOCUMENTATION

16.1 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.

16.2 Style Guide

This section describes the agreed style and conventions used when editing and writing source code in BDSIM.

16.2.1 C++

General

- All variables shall be namespaced and not global
- The asterix is attached to the object not the variable name

```
BDSClclassName* anInstance;
```

Naming

- Underscores are to be avoided as 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

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

```
if (a > 4)
{G4cout << a << G4endl;}
```

- Braces should be on a new line - makes scope easier to determine

```
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

```
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 (ie outside the function) in the source code - document the header

16.2.2 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 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

```
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

16.3 Release Checklist

Things to update immediately before a release, ie from a release candidate branch.

1. README - update at the top and the version history
2. CMakeLists.txt - change major, minor & patch version at very top
3. bdsim.cc - version at top and G4cout statement at beginning of main
4. Update the committed pdf of the manual in the manual directory

16.4 Program Layout

BDSIM progresses in the following general steps:

1. Parser reads and parses input text files preparing 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.

16.4.1 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](#).

16.4.2 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](#).

16.4.3 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](#).

16.4.4 Analysis

Run time analysis is also discussed in the aforementioned [Sensitivity, Output & Analysis](#).

Post run time (outside BDSIM) analysis is discussed in [Analysis Suite](#).

16.4.5 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 psuedo-random number generator engine

16.5 Build System & Testing

16.5.1 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.

16.5.2 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.

Comparison of output is being added currently.

16.5.3 Package System

The package system is based on CPack, which works in combination with CMake. One can package the 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.

16.6 Parser

The main parser interface can be found in `parser/parser.h`. The parser is currently a singleton (only one instance with global scope).

16.6.1 Options

All options for a BDSIM run are contained in an instance of `parser/options.h:GMAD::Options` class. This is passed to BDSIM.

16.7 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 is described in the following sections.

16.7.1 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

16.7.2 G4UserEventAction

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

16.7.3 G4UserStackingAction

This allows the developer to fiddle the priority of particles (primary and secondary) based on their properties.

BDSStackingAction::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.
- Photons are counted

16.7.4 G4UserTrackingAction

The tracking action happens after each step is completed

BDSUserTrackingAction::PreUserTrackingAction

- the tracking manager is always told to store the trajectory

16.7.5 G4UserSteppingAction

Only `UserSteppingAction` is implemented and only to provide verbose output if required.

16.8 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.

16.8.1 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 it 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 to a degree simplified 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 α 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 - ie 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 arclength 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.

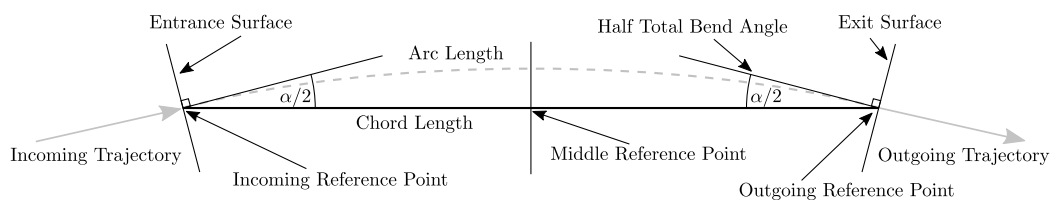


Fig. 16.1: Schematic of chord and arc length as well as reference points and planes for a BDSAcceleratorComponent that bends by finite angle α .

16.8.2 Component Factory

16.8.3 Beam Pipe / Aperture Factories

16.8.4 Magnet Factories

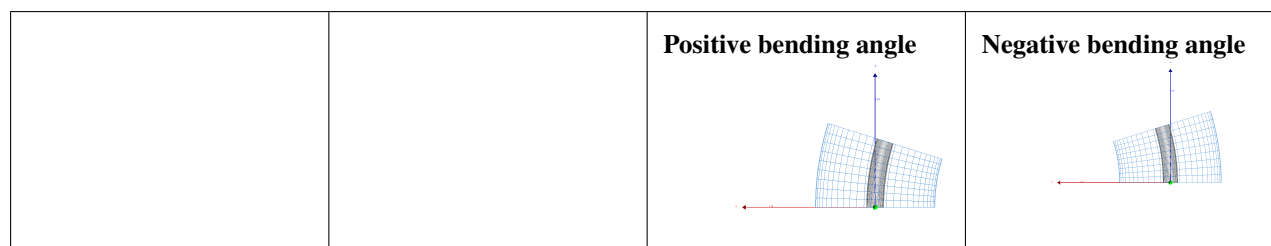
The magnet geometry is built in factories with virtual base class `BDSMagnetOuterFactoryBase`. Many factories inherit this implementing the virtual methods (one for each magnet type) and provide various styles of magnet geometry. In this way, a new magnet style can be added easily or a factory made that 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.

16.8.5 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 co-ordinates).
- A negative angle will bend toward positive X in the $x - z$ plane (local co-ordinates).
- 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 poleface 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.



Fig. 16.2: An example sequence of rbend magnets (without pole face angles).

To split an rbend into multiple segments to create a straight final magnet 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 by the user into smaller consecutive sbends if needed. If multiple sbends are defined as such, the input pole face angle ($e1$) for an sbend must be -1 times the output pole face angle ($e2$) 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 rotation 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 segment 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 $e1$) 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 users specified $e2$.

There are multiple reasons for this implementation. Without the change in face angle for each segment, if a large $e1$ is specified when the length of each segment is short, the projected length of the first segment would overlap

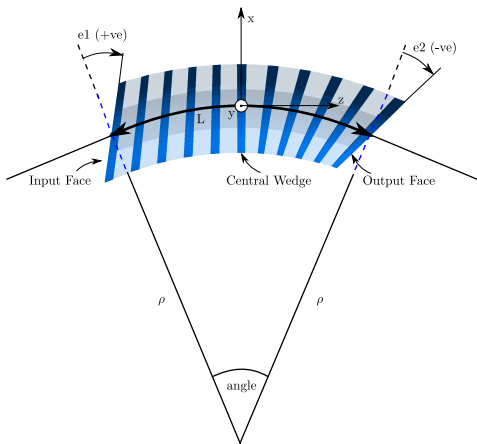


Fig. 16.3: Reference system for an sbend with pole face rotation, with screenshot partially showing an example sbend and the change in inputface and outputface angles along the magnet.

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.

		Element Overlap 		Element Overlap Check
--	--	----------------------------	--	----------------------------------

In certain circumstances, the situation may occur where the angles of both input and output face angles are such so 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 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.

16.8.6 Specific Element Details

Rectangular Bend

16.9 Fields

fields

16.10 Tracking Algorithms

tracking

16.11 Sensitivity, Output & Analysis

16.11.1 Information From Geant4

16.11.2 Output

16.11.3 Run-time Analysis

16.12 Analysis Suite

16.13 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

p

- pybdsim, 79
- pybdsim.Beam, 80
- pybdsim.Constants, 85
- pybdsim.Convert, 83
- pybdsim.Convert._MadxTfs2Gmad, 83
- pybdsim.Data, 85
- pybdsim.Gmad, 85
- pybdsim.Joinhistograms, 87
- pybdsim.ModelProcessing, 87
- pybdsim.Options, 87
- pybdsim.Plot, 89
- pybdsim.Root, 89
- pybdsim.Visualisation, 89
- pymad8, 94
- pymad8.Converter, 94
- pymad8.Mad8, 94
- pymad8.Mad8Converter, 96
- pymad8.Plot, 96
- pymad8.Saveline, 96
- pymad8.Sim, 97
- pymad8.Visualisation, 97
- pymadx, 90
- pymadx.Beam, 93
- pymadx.Plot, 93
- pymadx.Ptc, 93
- pymadx.PtcAnalysis, 94
- pymadx.PtcPlot, 94

A

AddBeam() (pybdsim.Builder.Machine method), 80
 AddBias() (pybdsim.Builder.Machine method), 81
 AddDipole() (pybdsim.Builder.Machine method), 81
 AddDrift() (pybdsim.Builder.Machine method), 81
 AddECol() (pybdsim.Builder.Machine method), 81
 addElement() (pymad8.Mad8.General method), 95
 AddFodoCell() (pybdsim.Builder.Machine method), 81
 AddFodoCellMultiple() (pybdsim.Builder.Machine method), 81
 AddFodoCellSplitDrift() (pybdsim.Builder.Machine method), 81
 AddFodoCellSplitDriftMultiple() (pybdsim.Builder.Machine method), 81
 AddHKicker() (pybdsim.Builder.Machine method), 81
 AddMachineLatticeFromSurveyToFigure() (in module pybdsim.Plot), 89
 AddMachineLatticeToFigure() (in module pybdsim.Plot), 89
 AddMachineLatticeToFigure() (in module pymadx.Plot), 93
 AddMarker() (pybdsim.Builder.Machine method), 81
 AddMultipole() (pybdsim.Builder.Machine method), 81
 AddOctupole() (pybdsim.Builder.Machine method), 81
 AddOptions() (pybdsim.Builder.Machine method), 81
 AddParticle() (pymadx.Ptc.Inrays method), 93
 AddQuadrupole() (pybdsim.Builder.Machine method), 81
 AddRCol() (pybdsim.Builder.Machine method), 81
 AddRFCavity() (pybdsim.Builder.Machine method), 81
 AddSampler() (pybdsim.Builder.Machine method), 81
 AddSextupole() (pybdsim.Builder.Machine method), 81
 AddSolenoid() (pybdsim.Builder.Machine method), 81
 AddTransform3D() (pybdsim.Builder.Machine method), 81
 AddVKicker() (pybdsim.Builder.Machine method), 81
 apertures() (in module pymad8.Plot), 96
 Append() (pybdsim.Builder.Machine method), 81

B

BDSAsciiData (class in pybdsim.Data), 85
 Beam (class in pybdsim.Beam), 80
 Beam (class in pymadx), 92
 Beam (class in pymadx.Beam), 93

C

CalculateOpticalFunctions() (pymadx.PtcAnalysis method), 92
 CalculateOpticalFunctions() (pymadx.PtcAnalysis.PtcAnalysis method), 94
 change() (pybdsim.Gmad.GmadFileComponents method), 86
 Chrom (class in pymad8.Mad8), 94
 Clear() (pymadx.Ptc.Inrays method), 93
 Clear() (pymadx.Tfs method), 90
 Clear() (pymadx.TfsArray method), 91
 ColumnIndex() (pymadx.Tfs method), 90
 ColumnIndex() (pymadx.TfsArray method), 92
 Common (class in pymad8.Mad8), 94
 CompareBDSIMSurveyWithMadXTfs() (in module pybdsim.Plot), 89
 CompareMadX() (pybdsim.Gmad.Survey method), 87
 containsEnergyVariation() (pymad8.Mad8.Common method), 94
 convert() (pymad8.Mad8Converter.Mad8Converter method), 96

D

DefineConstituentElements() (pybdsim.Builder.Line method), 82
 draw() (pybdsim.Visualisation.Helper method), 89
 drawBend() (pymad8.Visualisation.OneDim method), 97
 drawBend() (pymad8.Visualisation.TwoDim method), 97
 drawElement() (pymad8.Visualisation.OneDim method), 97
 drawElement() (pymad8.Visualisation.TwoDim method), 97
 drawElements() (pymad8.Visualisation.OneDim method), 97
 drawElements() (pymad8.Visualisation.TwoDim method), 97
 drawHkic() (pymad8.Visualisation.OneDim method), 97
 drawInst() (pymad8.Visualisation.OneDim method), 97
 drawMachineLattice() (in module pymad8.Plot), 96
 drawMark() (pymad8.Visualisation.OneDim method), 97
 drawMark() (pymad8.Visualisation.TwoDim method), 97

97
 drawMoni() (pymad8.Visualisation.OneDim method), 97
 drawMoni() (pymad8.Visualisation.TwoDim method), 97
 drawMult() (pymad8.Visualisation.OneDim method), 97
 drawProf() (pymad8.Visualisation.OneDim method), 97
 drawQuad() (pymad8.Visualisation.OneDim method), 97
 drawQuad() (pymad8.Visualisation.TwoDim method), 97
 drawSext() (pymad8.Visualisation.OneDim method), 97
 drawVkc() (pymad8.Visualisation.OneDim method), 97
 drawWire() (pymad8.Visualisation.OneDim method), 97

E

Editor (class in pybdsim.Options), 87
 ElectronColliderOptions() (in module pybdsim.Options), 87
 Element (class in pybdsim.Builder), 82
 elementNames() (pybdsim.Gmad.GmadFileComponents method), 86
 energy() (in module pymad8.Plot), 96
 Envelope (class in pymad8.Mad8), 95
 expand_file() (pymad8.Saveline.Loader method), 96

F

fileAnalysis() (pymad8.Saveline.Loader method), 96
 Filter() (pybdsim.Data.BDSAsciiData method), 85
 FinalDiff() (pybdsim.Gmad.Survey method), 87
 findByName() (pymad8.Mad8.General method), 95
 findByType() (pymad8.Mad8.General method), 95
 FindClosestElement() (pybdsim.Gmad.Survey method), 87
 findComponentCoords() (pybdsim.Visualisation.Helper method), 89
 findElement() (pybdsim.Gmad.GmadFileComponents method), 86
 FlatGenerator (class in pymadx.Ptc), 93
 flatten_elements() (pymad8.Saveline.Loader method), 96

G

GaussGenerator (class in pymadx.Ptc), 93
 General (class in pymad8.Mad8), 95
 generate() (pymad8.Sim.Track method), 97
 Generate() (pymadx.Ptc.FlatGenerator method), 93
 Generate() (pymadx.Ptc.GaussGenerator method), 93
 GenerateFullListOfSamplers() (in module pybdsim.ModelProcessing), 87
 GetAllNames() (pybdsim.Gmad.Lattice method), 86
 GetAngle() (pybdsim.Gmad.Lattice method), 86

GetAper() (pybdsim.Gmad.Lattice method), 86
 getApertures() (pymad8.Mad8.Common method), 94
 GetAperX() (pybdsim.Gmad.Lattice method), 86
 GetAperY() (pybdsim.Gmad.Lattice method), 86
 getColumn() (pymad8.Mad8.Common method), 94
 getColumn() (pymad8.Mad8.General method), 95
 GetColumn() (pymadx.Tfs method), 90
 GetColumn() (pymadx.TfsArray method), 92
 GetColumnDict() (pymadx.Tfs method), 90
 getData() (pymad8.Mad8.Chrom method), 94
 getData() (pymad8.Mad8.Common method), 94
 getData() (pymad8.Mad8.Envelope method), 95
 getData() (pymad8.Mad8.Rmat method), 95
 GetDataFromROOTHist() (in module pybdsim.Root), 89
 GetElement() (pybdsim.Gmad.Lattice method), 86
 GetElementNamesOfType() (pymadx.Tfs method), 91
 GetElementsOfType() (pymadx.Tfs method), 91
 GetElementsOfType() (pymadx.TfsArray method), 92
 getIndex() (pymad8.Mad8.General method), 95
 GetIndexOfElementNamed() (pybdsim.Gmad.Lattice method), 86
 GetIntegratedAngle() (pybdsim.Builder.Machine method), 81
 GetIntegratedLength() (pybdsim.Builder.Machine method), 82
 getKeys() (pymad8.Converter.Mad8 method), 94
 GetKs() (pybdsim.Gmad.Lattice method), 86
 GetLength() (pybdsim.Gmad.Lattice method), 86
 GetMetaDataFromROOTHist() (in module pybdsim.Root), 89
 GetName() (pybdsim.Gmad.Lattice method), 86
 getNames() (pymad8.Mad8.General method), 95
 getNElements() (pymad8.Mad8.General method), 95
 getParameter() (pybdsim.Gmad.GmadFileComponents method), 86
 GetPDGInd() (in module pybdsim.Constants), 85
 GetPDGName() (in module pybdsim.Constants), 85
 GetRow() (pymadx.Tfs method), 91
 GetRow() (pymadx.TfsArray method), 92
 getRowByIndex() (pymad8.Mad8.Common method), 95
 getRowByIndex() (pymad8.Mad8.General method), 95
 getRowByName() (pymad8.Mad8.Common method), 95
 getRowByName() (pymad8.Mad8.General method), 95
 GetRowDict() (pymadx.Tfs method), 91
 GetSegment() (pymadx.Tfs method), 91
 GetSegment() (pymadx.TfsArray method), 92
 getType() (pybdsim.Gmad.GmadFileComponents method), 86
 GetType() (pybdsim.Gmad.Lattice method), 86
 getValueByName() (in module pymad8.Mad8), 96
 getWorldCentre() (pybdsim.Visualisation.Helper method), 90
 GmadFile (class in pybdsim.Gmad), 85
 GmadFileBeam (class in pybdsim.Gmad), 85
 GmadFileComponents (class in pybdsim.Gmad), 85

GmadFileOptions (class in pybdsim.Gmad), 86

H

Helper (class in pybdsim.Visualisation), 89

I

IndexFromName() (pymadx.Tfs method), 91

IndexFromNearestS() (pybdsim.Data.BDSAsciiData method), 85

IndexFromNearestS() (pymadx.Tfs method), 91

Inray (class in pymadx.Ptc), 93

Inrays (class in pymadx.Ptc), 93

InterrogateItem() (pymadx.Tfs method), 91

InterrogateItem() (pymadx.TfsArray method), 92

J

JoinRootHistograms() (in module pybdsim.Joinhistograms), 87

K

keys (pymad8.Mad8.Common attribute), 95

keys (pymad8.Mad8.Envelope attribute), 95

keys (pymad8.Mad8.Rmat attribute), 95

keys (pymad8.Mad8.Survey attribute), 96

keys (pymad8.Mad8.Twiss attribute), 96

L

Lattice (class in pybdsim.Gmad), 86

Line (class in pybdsim.Builder), 82

linearOptics() (in module pymad8.Plot), 96

Load() (in module pybdsim.Data), 85

Load() (pybdsim.Gmad.Lattice method), 86

Load() (pybdsim.Gmad.Survey method), 87

Load() (pymadx.Tfs method), 91

Load() (pymadx.TfsArray method), 92

Loader (class in pymad8.Saveline), 96

loadFile() (pymad8.Saveline.Loader method), 96

LoadInrays() (in module pymadx.Ptc), 94

M

Machine (class in pybdsim.Builder), 80

Mad8 (class in pymad8.Converter), 94

Mad8Converter (class in pymad8.Mad8Converter), 96

MadxGmadComparison() (in module pybdsim.Plot), 89

MadxTfs2Gmad() (in module pybdsim.Convert._MadxTfs2Gmad), 83

MadxTfs2Ptc() (in module pymadx), 92

MadxTfsBeta() (in module pybdsim.Plot), 89

MadxTfsBetaSimple() (in module pybdsim.Plot), 89

makeArray() (pymad8.Mad8.General method), 95

makeLocationList() (pymad8.Mad8.Common method), 95

MatchValue() (pybdsim.Data.BDSAsciiData method), 85

MinimumStandard() (in module pybdsim.Options), 87

N

NameFromIndex() (pymadx.Tfs method), 91

NameFromIndex() (pymadx.TfsArray method), 92

NameFromNearestS() (pybdsim.Data.BDSAsciiData method), 85

NameFromNearestS() (pymadx.Tfs method), 91

next() (pybdsim.Builder.Machine method), 82

next() (pybdsim.Gmad.Lattice method), 87

next() (pymadx.Tfs method), 91

next() (pymadx.TfsArray method), 92

O

OneDim (class in pymad8.Visualisation), 97

Options (class in pybdsim.Options), 87

OutputReader (class in pymad8.Mad8), 95

P

parseElement() (pybdsim.Gmad.GmadFileComponents method), 86

parseFile() (pymad8.Converter.Mad8 method), 94

ParseLattice() (pybdsim.Gmad.Lattice method), 86

parseLine() (pymad8.Converter.Mad8 method), 94

phaseAdvance() (in module pymad8.Plot), 96

Plot() (pybdsim.Gmad.Survey method), 87

plot() (pymad8.Visualisation.OneDim method), 97

plot() (pymad8.Visualisation.TwoDim method), 97

Plot() (pymadx.Ptc.Inrays method), 93

Plot() (pymadx.Tfs method), 91

plotAlf() (pymad8.Mad8.Twiss method), 96

plotBeta() (pymad8.Mad8.Twiss method), 96

plotEta() (pymad8.Mad8.Twiss method), 96

plotEtaPrime() (pymad8.Mad8.Twiss method), 96

PlotInrays() (in module pymadx.Ptc), 94

plotMu() (pymad8.Mad8.Twiss method), 96

PlotSimple() (pymadx.Tfs method), 91

PlotTfsBeta() (in module pymadx.Plot), 93

PlotTfsBetaSimple() (in module pymadx.Plot), 93

PlotTH1Bar() (in module pybdsim.Root), 89

PlotTH1Hist() (in module pybdsim.Root), 89

plotUpdate() (pymad8.Visualisation.TwoDim method), 97

plotXY() (pymad8.Mad8.General method), 95

Print() (pybdsim.Gmad.Lattice method), 86

PrintZeroLength() (pybdsim.Gmad.Lattice method), 87

ProtonColliderOptions() (in module pybdsim.Options), 89

PtcAnalysis (class in pymadx), 92

PtcAnalysis (class in pymadx.PtcAnalysis), 94

PtcPlot (class in pymadx), 92

PtcPlot (class in pymadx.PtcPlot), 94

pybdsim (module), 79

pybdsim.Beam (module), 80

pybdsim.Constants (module), 85

pybdsim.Convert (module), 83

pybdsim.Convert._MadxTfs2Gmad (module), 83

pybdsim.Data (module), 85

pybdsim.Gmad (module), 85

pybdsim.Joinhistograms (module), 87

pybdsim.ModelProcessing (module), 87

pybdsim.Options (module), 87

pybdsim.Plot (module), 89
 pybdsim.Root (module), 89
 pybdsim.Visualisation (module), 89
 pymad8 (module), 94
 pymad8.Converter (module), 94
 pymad8.Mad8 (module), 94
 pymad8.Mad8Converter (module), 96
 pymad8.Plot (module), 96
 pymad8.Saveline (module), 96
 pymad8.Sim (module), 97
 pymad8.Visualisation (module), 97
 pymadx (module), 90
 pymadx.Beam (module), 93
 pymadx.Plot (module), 93
 pymadx.Ptc (module), 93
 pymadx.PtcAnalysis (module), 94
 pymadx.PtcPlot (module), 94

R

readChromFile() (pymad8.Mad8.OutputReader method), 95
 readEnvelopeFile() (pymad8.Mad8.OutputReader method), 95
 readFile() (pymad8.Converter.Mad8 method), 94
 readFile() (pymad8.Mad8.OutputReader method), 95
 readRmatFile() (pymad8.Mad8.OutputReader method), 95
 readSurveyFile() (pymad8.Mad8.OutputReader method), 95
 readTwissFile() (pymad8.Mad8.OutputReader method), 95
 ReportPopulations() (pymadx.Tfs method), 91
 ReportPopulations() (pymadx.TfsArray method), 92
 ReturnBeamString() (pybdsim.Beam.Beam method), 80
 ReturnBeamString() (pymadx.Beam method), 92
 ReturnBeamString() (pymadx.Beam.Beam method), 93
 ReturnOptionsString() (pybdsim.Options.Options method), 87
 ReturnPtcString() (pymadx.Beam method), 92
 ReturnPtcString() (pymadx.Beam.Beam method), 93
 Rmat (class in pymad8.Mad8), 95

S

SamplerLoop() (pymadx.PtcAnalysis method), 92
 SamplerLoop() (pymadx.PtcAnalysis.PtcAnalysis method), 94
 SavelineTest() (in module pymad8.Saveline), 96
 SetBeamPipeRadius() (pybdsim.Options.Options method), 87
 SetBeamPipeThickness() (pybdsim.Options.Options method), 87
 SetBLMLength() (pybdsim.Options.Options method), 87
 SetBLMRadius() (pybdsim.Options.Options method), 87
 SetBuildTunnel() (pybdsim.Options.Options method), 87

SetBuildTunnelFloor() (pybdsim.Options.Options method), 88
 setCallbacks() (in module pymad8.Plot), 96
 SetCherenkovOn() (pybdsim.Options.Options method), 88
 SetChordStepMinimum() (pybdsim.Options.Options method), 88
 SetDefaultRangeCut() (pybdsim.Options.Options method), 88
 SetDeltaChord() (pybdsim.Options.Options method), 88
 SetDeltaIntersection() (pybdsim.Options.Options method), 88
 SetDeltaOneStep() (pybdsim.Options.Options method), 88
 SetDistributionType() (pybdsim.Beam.Beam method), 80
 SetDistributionType() (pymadx.Beam method), 92
 SetDistributionType() (pymadx.Beam.Beam method), 93
 SetELossHistBinWidth() (pybdsim.Options.Options method), 88
 SetEMLeadParticleBiasing() (pybdsim.Options.Options method), 88
 SetEnergy() (pybdsim.Beam.Beam method), 80
 SetEnergy() (pymadx.Beam method), 92
 SetEnergy() (pymadx.Beam.Beam method), 93
 SetEPAnnihilation2HadronEnhancementFactor() (pybdsim.Options.Options method), 88
 SetEPAnnihilation2MuonEnhancementFactor() (pybdsim.Options.Options method), 88
 SetGamma2MuonEnhancementFactor() (pybdsim.Options.Options method), 88
 SetIncludeIronMagField() (pybdsim.Options.Options method), 88
 SetLengthSafety() (pybdsim.Options.Options method), 88
 SetLPBFraction() (pybdsim.Options.Options method), 88
 SetMaximumEpsilonStep() (pybdsim.Options.Options method), 88
 SetMinimumEpsilonStep() (pybdsim.Options.Options method), 88
 SetNGenerate() (pybdsim.Options.Options method), 88
 SetNLinesIgnore() (pybdsim.Options.Options method), 88
 SetNPerFile() (pybdsim.Options.Options method), 88
 SetOuterDiameter() (pybdsim.Options.Options method), 88
 SetParticleType() (pybdsim.Beam.Beam method), 80
 SetParticleType() (pymadx.Beam method), 92
 SetParticleType() (pymadx.Beam.Beam method), 93
 SetPhysicsList() (pybdsim.Options.Options method), 88
 SetPipeMaterial() (pybdsim.Options.Options method), 88
 SetProductionCutElectrons() (pybdsim.Options.Options method), 88

- SetProductionCutPhotons() (pybdsim.Options.Options method), 88
- SetProductionCutPositrons() (pybdsim.Options.Options method), 88
- SetRandomSeed() (pybdsim.Options.Options method), 88
- SetSamplerDiameter() (pybdsim.Options.Options method), 88
- SetSensitiveBeamlineComponents() (pybdsim.Options.Options method), 88
- SetSensitiveBeamPipe() (pybdsim.Options.Options method), 88
- SetSenssitiveBLMs() (pybdsim.Options.Options method), 88
- SetSoilMaterial() (pybdsim.Options.Options method), 88
- SetSoilThickness() (pybdsim.Options.Options method), 88
- SetSRLowX() (pybdsim.Options.Options method), 88
- SetSRMultiplicity() (pybdsim.Options.Options method), 88
- SetStopTracks() (pybdsim.Options.Options method), 88
- SetStoreMuonTrajectory() (pybdsim.Options.Options method), 88
- SetStoreNeutronTrajectory() (pybdsim.Options.Options method), 88
- SetStoreTrajectory() (pybdsim.Options.Options method), 88
- SetSynchRadiationOn() (pybdsim.Options.Options method), 89
- SetT0() (pybdsim.Beam.Beam method), 80
- SetT0() (pymadx.Beam method), 92
- SetT0() (pymadx.Beam.Beam method), 93
- SetThresholdCutCharged() (pybdsim.Options.Options method), 89
- SetThresholdCutPhotons() (pybdsim.Options.Options method), 89
- SetTrackSRPhotons() (pybdsim.Options.Options method), 89
- SetTrajectoryCutGTZ() (pybdsim.Options.Options method), 89
- SetTrajectoryCutLTR() (pybdsim.Options.Options method), 89
- SetTunnelFloorOffset() (pybdsim.Options.Options method), 89
- SetTunnelMaterial() (pybdsim.Options.Options method), 89
- SetTunnelOffsetX() (pybdsim.Options.Options method), 89
- SetTunnelOffsetY() (pybdsim.Options.Options method), 89
- SetTunnelRadius() (pybdsim.Options.Options method), 89
- SetTunnelThickness() (pybdsim.Options.Options method), 89
- SetVacuumMaterial() (pybdsim.Options.Options method), 89
- SetVacuumPressure() (pybdsim.Options.Options method), 89
- SetX0() (pybdsim.Beam.Beam method), 80
- SetXP0() (pybdsim.Beam.Beam method), 80
- SetY0() (pybdsim.Beam.Beam method), 80
- SetYP0() (pybdsim.Beam.Beam method), 80
- SetZ0() (pybdsim.Beam.Beam method), 80
- SetZP0() (pybdsim.Beam.Beam method), 80
- Statistics() (pymadx.Ptc.Inrays method), 93
- Step() (pybdsim.Gmad.Survey method), 87
- subline() (pymad8.Mad8.General method), 95
- Survey (class in pybdsim.Gmad), 87
- Survey (class in pymad8.Mad8), 96
- survey() (in module pymad8.Plot), 96
- ## T
- testOneDim() (in module pymad8.Visualisation), 97
- testTrack() (in module pymad8.Sim), 97
- testTwoDim() (in module pymad8.Visualisation), 97
- Tfs (class in pymadx), 90
- TfsArray (class in pymadx), 91
- TfsHasRequiredColumns() (in module pybdsim.Convert._MadxTfs2Gmad), 85
- Track (class in pymad8.Sim), 97
- trackParticle() (pymad8.Sim.Track method), 97
- trackParticles() (pymad8.Sim.Track method), 97
- transformedPoly() (in module pymad8.Visualisation), 97
- transformedRect() (in module pymad8.Visualisation), 97
- Twiss (class in pymad8.Mad8), 96
- TwoDim (class in pymad8.Visualisation), 97
- ## W
- WrapLatticeAboutItem() (in module pybdsim.ModelProcessing), 87
- Write() (pybdsim.Builder.Machine method), 82
- write() (pybdsim.Gmad.GmadFileComponents method), 86
- Write() (pymadx.Ptc.Inrays method), 94
- WriteInrays() (in module pymadx.Ptc), 94