BDSIM Documentation

Release 0.6.develop

L. Nevay S. Boogert L. Deacon
H. Garcia-Morales J. Snuverink

May 22, 2015

Contents:

CHAPTER

ONE

INTRODUCTION

1.1 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.

1.1.1 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

1.1.2 What BDSIM is not intended for

- Long term tracking studies
- Simulating collective effects
- Lattice optical design and optimisation
- A replacment for SixTrack

1.1.3 Example Applications

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

1.2 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 either as a single instance or on a farm.
5. Analyse output data as desired.

1.3 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 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 claculate the interaction with the beampipe and 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 generate Geant4 geometry automatically and uses custom 'steppers' for specific magnetic fields in place of normal Runge-Kutta integrators used by Geant4. Equations of motion descirbing particle motion in magnetic fields such as that of a quadruople 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.

Although BDSIM uses fairly generic geometry, the user may specify the style of geometry to be used, which 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.

CHAPTER

TWO

INSTALLATION

2.1 Obtaining BDSIM

BDSIM may be obtained either from the BDSIM webiste (http://www.pp.rhul.ac.uk/twiki/bin/view/JAI/BdSim) or from the GIT repository (https://www.bitbucket.org/stewartboogert/bdsim). The user must compile it on their system and must have Geant4 already present (or access to AFS).

We **recommend obtaining BDSIM from the git repository** as this provides an easy method to obtain updates in the future.

2.1.1 From the Website

The source can be downloaded from: http://twiki.ph.rhul.ac.uk/twiki/bin/view/PP/JAI/BDsimDownload in compressed tar format. To decompress use the following command:

```
tar -xzf BDSIM-v064.tar.gz
```

2.1.2 From the GIT Repository

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

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

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

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

2.2 Requirements

1. Geant4 installed or access to **AFS** [1]. Version 4.9.6 or higher.
2. CMake 2.6.4 or higher
3. Flex 2.5.37 or higher
4. Bison 2.3 or higher
5. CLHEP 2.1.3.1 or higher
6. Boost libraries (**to be removed**)

Optional dependencies

[1] Note, **AFS** is not supported with Mac OSX as there is no Mac compatible version of Geant4 available on AFS.

7. ROOT framework for binary data output

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
$G4LEDATA            $G4NEUTRONXSDATA    $G4REALSURFACEDATA
$G4LEVELGAMMADATA    $G4PIIDATA          $G4SAIDXSDATA
```

If these do not exists, please source the Geant4 environmental script before installing BDISM 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*.

2.3 Setting Up

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

2.3.1 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.

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

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

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 0.6.develop
-- Build Type RelWithDebInfo
-- Compiler supports C++11
-- Looking for CLHEP... - found
-- Looking for ROOT...
-- Found ROOT 5.34/26 in /opt/local/libexec/root5
-- ROOT support ON
-- Boost version: 1.57.0
-- Looking for XercesC... - found
-- 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: 10.0.2
-- Found Doxygen: /opt/local/bin/doxygen (found version "1.8.9.1")
-- Found BISON: /opt/local/bin/bison (found version "2.7.12-4996")
-- Found FLEX: /opt/local/bin/flex (found version "2.5.37")
-- Configuring done
-- Generating done
-- Build files have been written to: /Users/nevay/physics/reps/bdsim-test2
```

CMake will search your system for the required dependencies. In the above example, this proceeded without any errors. In the case where a required dependecy 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 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. From the build directory you can verify your installation using a series of tests included with BDSIM.

```
> ctest -E LONG
```

2.3.2 Scientific Linux

For SL5 you will have to use Geant 4.9.6 as Geant 4.10 onwards is not compatable. Older version of Geant4 can be downloaded from their archive . For SL6, we recommend the latest version of Geant4, currently 4.10.1. Once ready, make a directory **outisde** the BDSIM source directory to build BDSIM in:

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

From this directory use the following CMake command to configure the BDSIM installation:

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

You can then compile BDSIM with:

```
> make
```

BDSIM can then be installed 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. From the build directory you can verify your installation using a serious of tests included with BDSIM.:

```
> ctest -E LONG
```

2.3.3 Scientific Linux with AFS Access

When the machine has AFS connection, then the dependent packages like Geant4 can be taken from there and don't need to be installed. The same compiler version needs to be used for BDSIM as the one for Geant4. The following scripts must be sourced before using CMake.

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
```

For the tags 0.61-0.63 and the develop branch:

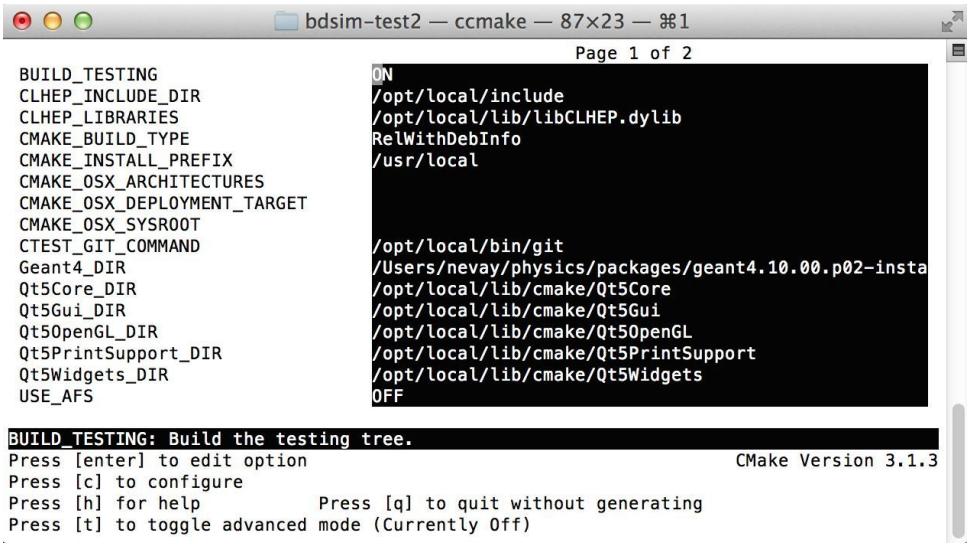
```
source /afs/cern.ch/user/j/jsnuveri/public/gcc47-setup.sh
source /afs/cern.ch/user/j/jsnuveri/public/geant4.10-setup.sh
```

After this, the installation procedure for *Scientific Linux* should be followed.

2.3.4 Configuring the BDSIM Build with CMake

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

```
> ccmake .
```



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

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

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

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

2.3.5 Making Doxygen Documentation

From the build directory, after running `make`, run the following command:

```
> make doc
```

to make the Doxygen documentation in a folder called `Doxygen`.

2.3.6 Geant4 Installation Guide

BDSIM now builds with the most recent versions of Geant4, but only the BDSIM develop branch. A new official version will be released soon (January 2015). If not built with **MacPorts** then download the 4.10.01 version or an older version from the Geant archive. Move and unpack to a suitable place

```
> tar -xzf geant4.10.04.tar.gz
> ls
geant4.10.04
```

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

```
> mkdir geant4.10.04-build
> mkdir geant4.10.04-install
```

Configure Geant4 using CMake

```
> cd geant4.10.04-build
> cmake ../geant4.10.04
```

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

```
> ccmake .
```

Once the installation directory is set, press `c` to run the configuration process, and when complete, press `g` to generate the build. Geant4 can then be compiled

```
> make
```

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

```
> make -jN
```

where `N` is the number of cores on your computer [2]. Geant4 should then be installed

```
> make install
```

Note, if you've specified the directory to install, you will not need the `sudo` command, however, if you've left the settings as default, it'll be installed in a colder 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/geant4.10.04-install/bin/geant4.sh
```

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

2.4 Troubleshooting

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

[2] If your computer supports hyper-threading, you can use twice the number of cores with the `make -jN` command. Ie a computer has 4 cores and supports hyper-threading, can suuport up to `make -j8`. Exceeding this number will result in slower than normal compilation.

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:
G4HepRepFile (HepRepFile)
OpenGLImmediateQt (OGLIQt)
OpenGLImmediateX (OGLIX)
OpenGLImmediateXm (OGLIXm)
OpenGLStoredQt (OGLSQt)
OpenGLStoredX (OGLSX)
OpenGLStoredXm (OGLSXm)
RayTracerX (RayTracerX)
```

If your favourite is not there check that Geant4 is correctly compiled with those graphics system.

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.

CHAPTER

THREE

RUNNING BDSIM

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

–file=<file>	specify the input gmad file
–output=<fmt>	output format “root” or “ascii” (default)
–outfile=<file>	output file name. Will be appended with _N where N = 0, 1, 2, 3...
–vis_mac=<file>	visualization macro script, default vis.mac
–gflash=<N>	whether or not to turn on gFlash fast shower parameterisation.
–gflashemax=<N>	maximum energy for gflash shower parameterisation in GeV.
–gflashemin=<N>	minimum energy for gflash shower parameterisation in GeV.
–help	display this message.
–verbose	display general parameters before run
–verbose_event	display information for every event
–verbose_step	display tracking information after each step
–verbose_event_num=<N>	display tracking information for event N
–batch	batch mode - no graphics
–outline=<file>	print geometry / optics info to <file>
–outline_type=<fmt>	type of outline format where fmt is one of “optics” or “survey”
–materials	list materials included in BDSIM by default
–circular	assume circular machine - turn control
–seed=<N>	seed for the random number generator
–seedstate=<file>	file containing CLHEP::Random seed state NB - this overrides other seed value

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.

3.1 Interactively

Features:

- default option
- interactive visualisation of 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` commmand. The user must 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 exit. Once executed a window such as this will appear (depending on the visualiser you use).

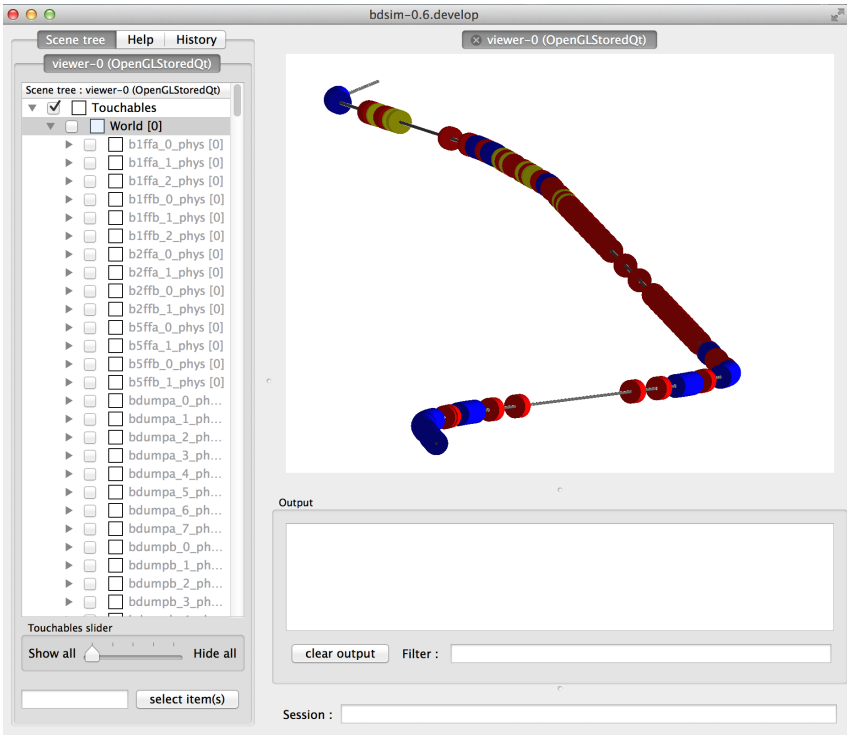


Fig. 3.1: BDSIM running interactively with OpenGL Qt visualiser from Geant4.

Note, the visualiser is part of Geant4 so if the desired visualiser isn't available, you must recopmile Geant4 with the correct visualiser (and subsequently BDSIM afterards). 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.

Example vis.mac files can be found in the supplied BDSIM examples.

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.

3.2 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.

3.3 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.

CHAPTER

FOUR

MODEL DESCRIPTION - INPUT SYNTAX

4.1 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 name can begin with a number

The following functions are provided

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

4.1.1 Examples

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

4.2 Coordinates & Units

In Geant4, global euclidean coordinates are used for tracking purposes, however, in describing a lattice with BDISM, curvlinear 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)
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}
clight	2.99792458×10^{8}
m	1
cm	10^{-2}
mm	10^{-3}
um	10^{-6}
nm	10^{-9}
s	1
ms	10^{-3}
us	10^{-6}
ns	10^{-9}

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

4.3 Lattice Description

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

1. Component definition
2. Sequence defention (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

4.4 Lattice Elements

Any element in BDSIM is descrbied with the following pattern:

```
type: name, paramter=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*
- *muspoiler*
- *solenoid*
- *laser*
- *transform3d*
- *element*
- *marker*

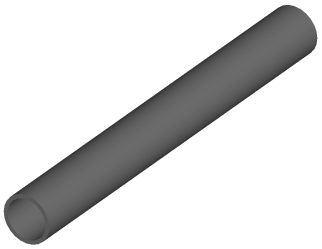
These are detailed in the following sections.

4.4.1 drift

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

parameter	description	default	required
l	length [m]	0.1	yes
vacuumMaterial	the vacuum material to use, can be user defined	vacuum	no

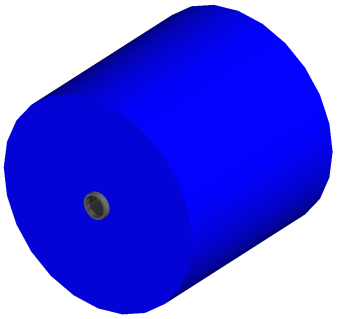
- The *aperture parameters* may also be specified.



Examples

```
l203b: drift, l=1*m;
l204c: drift, l=3*cm, beampipeRadius=10*cm;
```

4.4.2 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. Furthermore, an additional very small drift section is added on either side and the magnetic field up-scaled for the shorter field length to ensure that the magnet body fits inside the start and end faces of the element volume and doesn't protrude into the previous and next elements.

parameter	description	default	required
l	length [m]	0.1	yes
angle	angle [rad]	0	yes, or *B*
B	magnetic field [T]	0	yes
material	magnet outer material	Iron	no

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

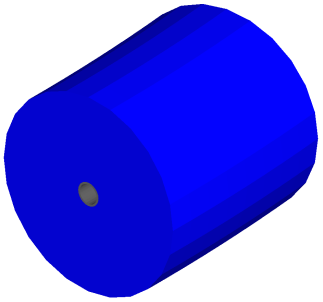
Note: For large angles (> 100 mrad) particles may hit the aperture as the beam pipe is is represented by a straight (chord) section and even nominal energy particles may hit the aperture depending on the degree of tracking accuracy specified. In this case, consider splitting the *rbend* into multiple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

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

Examples

```
MRB20: rbend, l=3*m, angle=0.003;
r1: rbend, l=5.43m, beampipeRadius=10*cm, B=2*Tesla;
```

4.4.3 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 curvlinear 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. For an LHC for example with a bending angle of ~0.005rad and l = 14m, the magnet is typically split into 5 cojoined *sbend* magnets.

parameter	description	default	required
l	length [m]	0.1	yes
angle	angle [rad]	0	yes, or *B*
B	magnetic field [T]	0	yes
material	magnet outer material	Iron	no

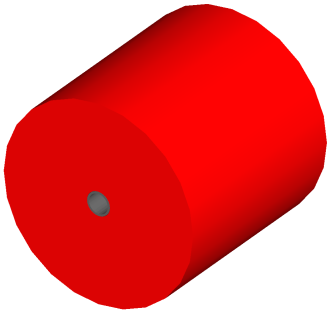
- The *aperture parameters* may also be specified.
- The *magnet geometry parameters* may also be specified.

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

Examples

```
s1: sbend, l=14.5*m, angle=0.005, magnetGeometryType="lhcright";
mb201x: sbend, l=304.2*cm, b=1.5*Tesla;
```

4.4.4 quadrupole



quadrupole defines a quadrupole magnet. The strength parameter *k1* is defined as $k1 = 1/(B\rho)\ dB_y\ /\ dx\ [m^{-2}]$. *ks1* specifies a skew quadrupole component as with *k1* but rotated by 45 degrees.

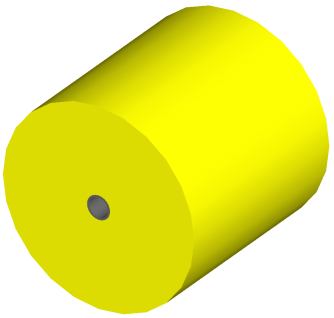
parameter	description	default	required
l	length [m]	0.1	yes
k1	quadrupole coefficient	0	yes
ks1	skew quadrupole coefficient	0	no
material	magnet outer material	Iron	no

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

Examples

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

4.4.5 sextupole



sextupole defines a sextupole magnet. The strength parameter *k2* is defined as $k2 = 1/(B\rho)\ dB_y^2\ /\ dx^2\ [m^{-3}]$. *ks2* specifies a skew sextupole component as with *k2* but rotated by 30 degrees.

parameter	description	default	required
l	length [m]	0.1	yes
k2	sextupole coefficient	0	yes
ks2	skew sextupole coefficient	0	no
material	magnet outer material	Iron	no

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

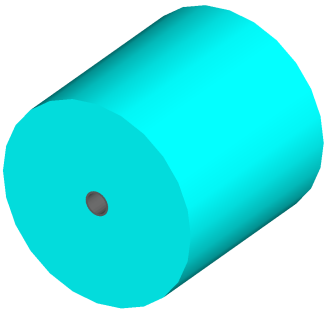
Examples

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

4.4.6 octupole

octupole defines an octupole magnet. The strength parameter *k3* is defined as $k3 = 1/(B\rho)\ dB_y^3\ /\ dx^3\ [m^{-4}]$. *ks3* specifies a skew octupole component as with *k3* but rotated by 15 degrees.

parameter	description	default	required
l	length [m]	0.1	yes
k3	octupole coefficient	0	yes
ks3	skew octupole coefficient	0	no
material	magnet outer material	Iron	no



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

Examples

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

4.4.7 decapole

Warning: To be completed - not yet implemented

decapole defines a decapole magnet. The strength parameter *k4* is defined as $k4 = 1/(B\rho)\ dB_y^4\ /\ dx^4\ [m^{-5}]$. *k43* specifies a skew decapole component as with *k4* but rotated by 7.5 degrees.

parameter	description	default	required
l	length [m]	0.1	yes
k4	decapole coefficient	0	yes
ks4	skew decapole coefficient	0	no
material	magnet outer material	Iron	no

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

Examples

```
MXDEC3: decapole, l=0.3*m, k3=32.9;
```

4.4.8 multipole

multipole defines a general multipole magnet.

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

Examples

```
**To be copmleted**
```

4.4.9 vkick

vkick 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;
```

4.4.10 hkick

hkick 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;
```

4.4.11 rf

rf defines an rf cavity

parameter	description	default	required
l	length [m]	0.1	yes
gradient	field gradien [MV/m]	0	yes
material	outer material	Iron	no

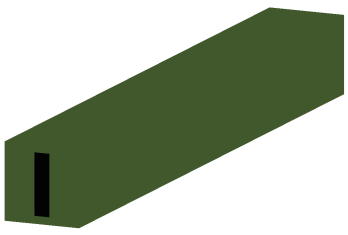
- The *aperture parameters* may also be specified.

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;
```

4.4.12 rcol



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

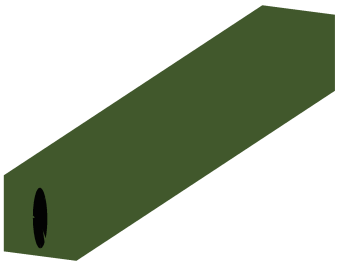
parameter	description	default	required
l	length [m]	0.1	yes
xsize	horizontal half aperture [m]	0	yes
ysize	veritcal half aperture [m]	0	yes
material	outer material	Iron	no
outerDiameter	outer full width [m]	global	no

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

Examples

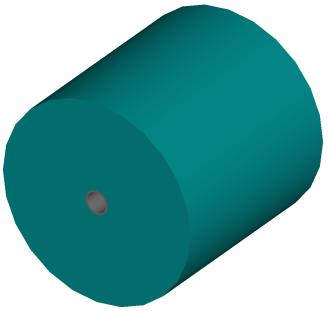
```
TCP15: rcol, l=1.22*m, material="graphite", xsize=104*um, ysize=5*cm;
```

4.4.13 ecol



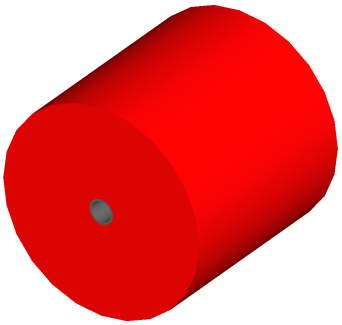
ecol defines an elliptical collimator. This is exactly the same as *rcol* except that the aperture is elliptical and the *xsize* and *ysize* define the horizontal and vertical half axes repsectively.

4.4.14 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
l	length [m]	0.1	yes
B	magnetic field [T]	1	yes
material	outer material	Iron	no
outerDiameter	outer full width [m]	global	no



4.4.15 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
l	length [m]	0.1	yes
ks	solenoid strength [ ]	0	yes
material	outer material	Iron	no
outerDiameter	outer full width [m]	global	no

Examples

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

4.4.16 laser

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

parameter	description	default	required
l	length of drift section [m]	0.1	yes
x, *y*, *z*	components of laser direction vector (normalised)	(1,0,0)	yes
waveLength	laser wavelength [m]	532*nm	yes

Examples

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

4.4.17 transform3d

transform3d defines an arbitrary 3-dimensional transformation of the the curvlinear 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
x	x offset	0	no
y	y offset	0	no
z	z offset	0	no
phi	phi Euler angle	0	no
theta	theta Euler angle	0	no
psi	psi Euler angle	0	no

Note: this permanently changes the coordinate frame, so care must be taken to undo any rotation if it intedded

for only one component.

Examples

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

4.4.18 element

element defines an arbitrary element that's defined by external geometry and magnetic field maps. Several geometry formats are supported. The user must also supply the outer radius of the object for tunnel geometry compatability.

parameter	description	default	required
geometry	filename of geometry	NA	yes
outR	outermost radius [m]	NA	yes
bmap	filename of magnetif 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* | *mokka*) and filename is the filename of the geometry file.

Examples

```
detector: element, geometry="gdml:atlasreduced.gmdl", outR=20*m;
detec: element, geometry="mokka:qq.sql", bmap ="mokka:qq.bmap";
```

4.4.19 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;
```

4.5 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 specifing them with the element definition. The aperture is controlled throught 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	*aper1*	*aper2*	*aper3*	*aper4*
circular	1	radius	NA	NA	NA
rectangular	2	x half width	y half width	NA	NA
elliptical	2	x semi-axis	y semi-axis	NA	NA
lhc	3	x half width of rectangle	y half width of rectangle	radius of circle	NA
lhcdetailed	3	x half width of rectangle	y half width of rectangle	radius of circle	NA
rectellipse	4	x half width of rectangle	y half width of rectangle	x semi-axis of ellipse	y semi-axis of ellipse

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.

MADX *racetrack* and *octagon* are currently unavailable but will be completed shortly.

4.6 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 magnet geometry is controlled by two parameters:

parameter	description	default	required
outerDiameter	full width of magnet in metres	1 m	no
outerMaterial	material of magnet	"iron"	no

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

Note: 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 variousmagnet styles be required for your simulation, please contact us (see *Feature Request* and this can be added - it is a relatively simple processes.

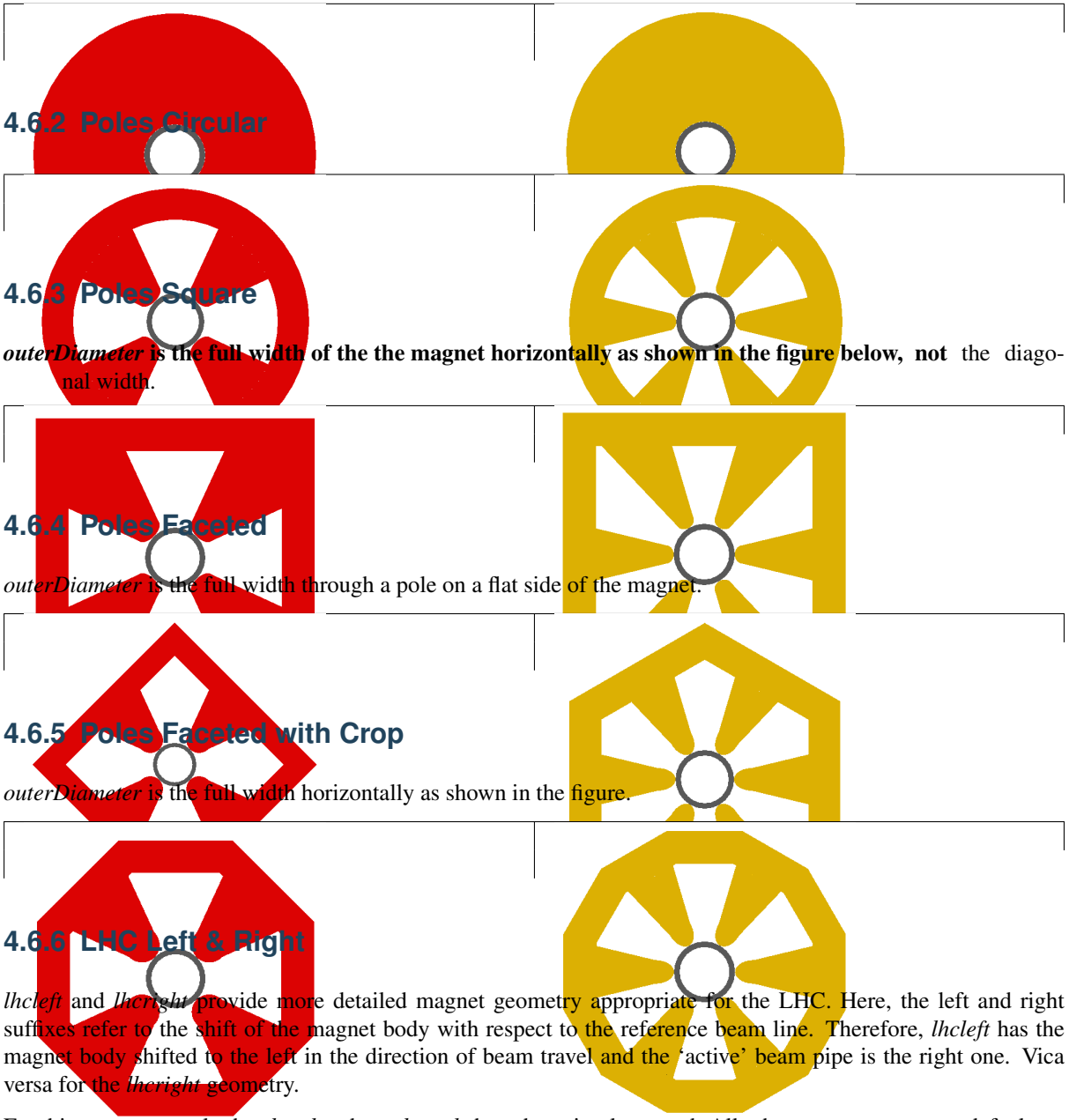
Examples of the different styles of magnet geometry are shown below.

4.6.1 Cylindrical (Default)

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.



4.6.2 Poles Circular

4.6.3 Poles Square

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

4.6.4 Poles Faceted

outerDiameter is the full width through a pole on a flat side of the magnet.

4.6.5 Poles Faceted with Crop

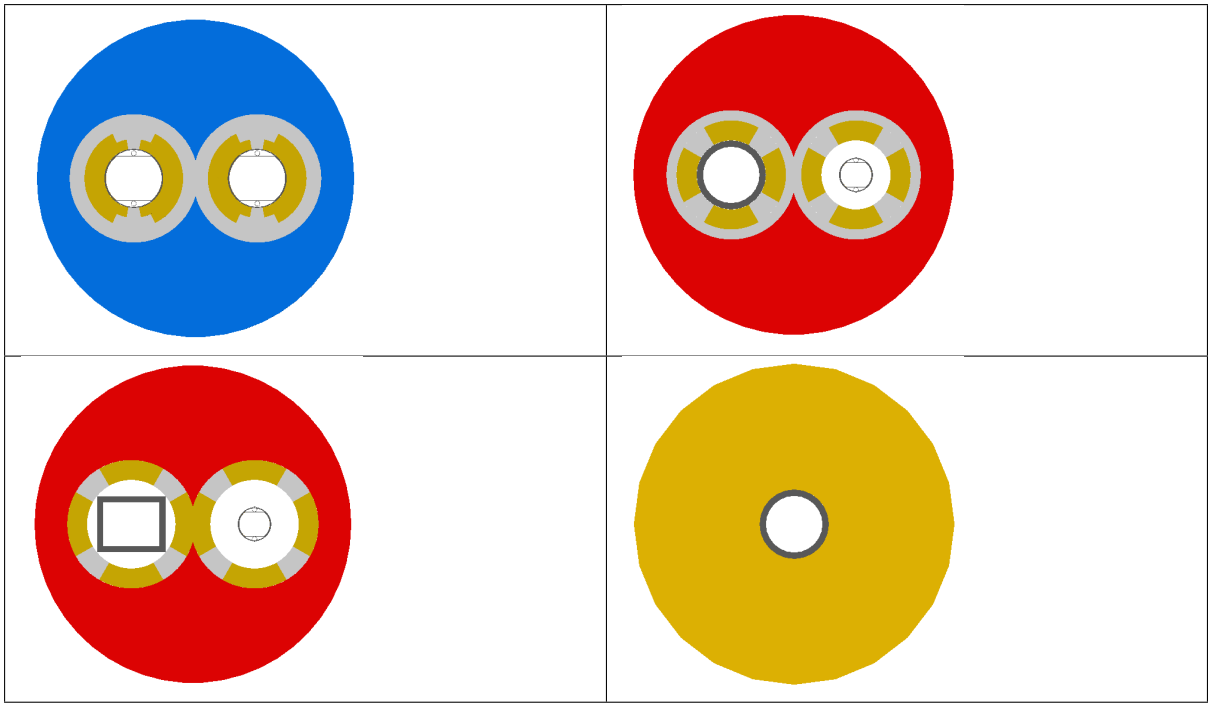
outerDiameter is the full width horizontally as shown in the figure.

4.6.6 LHC Left & Right

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

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

This geometry is parameterised to a degree regarding the beam pipe chosen. Of course, parameters similar to the LHC make most sense as does use of the *lhcdetailed* aperture type. Examples are shown with various beam pipes and both *sbend* and *quadrupole* geometries.



4.7 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 [1]. A sequence of elements is defined by a *line*. Lines of lines can be made to describe the accelerator sequence programatically 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);
```

4.7.1 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.

4.7.2 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:

[1] Note, if a sampler is attached to a beam line element and that element is use more than once in a *line*, then output will only be from the first occurence of that element in the sequence. This will be addressed in future releases.

```
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;
```

4.8 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 entrance 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 after an item, attach it to the next item. If however, you wish to record the coordinates at the end of the line or with another name, you must define a marker, place it in the sequence and then define a sampler that uses that marker:

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

sample, range=endoftheline;
```

Note: Samplers **can only** be defined **after** the main sequences 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.

4.9 Physics Lists

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 application. BDSIM follows the Geant4 ethos in this regard.

The physics list can be selected with the following syntax:

```
option, physicsList="physicslistname";
```

Note: Some physics lists allow biasing and re-weighting for some processes to further improve simulation efficiency. (See *options* for more details).

4.9.1 Physics Lists In BDSIM

standard	transportation of primary particles only - no scattering in material
em_standard	transportation of primary particles, ionization, bremsstrahlung, Cerenkov, multiple scattering/
em_low	the same as *em_standard* but using low energy electromagnetic models.
em_single_scatter	**TBC**.
em_muon	*em_standard* plus muon production processes with biased muon cross-sections.
lw	list for laser wire simulation - *em_standard* and "laserwire" physics, which is Compton Scattering with total cross-section renormalized to 1.
merlin	transportation of primary particles, and the following processes for electrons: multiple scattering, ionisation, and bremsstrahlung.
hadronic_standard	*em_standard* plus fission, neutron capture, neutron and proton elastic and inelastic scattering.
hadronic_muon	*hadronic_standard* plus muon production processes with biased muon cross-sections.
hadronic_QGSP_BERT	*em_standard* plus hadronic physics using the quark gluon string plasma (QGSP) model and the Bertini cascade model (BERT).
hadronic_QGSP_BERT_muon	*hadronic_QGSP_BERT* plus muon production processes with biased muon cross-sections.
hadronic_FTFP_BERT	*em_standard* plus hadronic physics using the Fritiof model followed by Reggion cascade and Precompound and evaporation models for the nucleus de-excitation (FTFP) model and the Bertini cascade model (BERT).
hadronic_FTFP_BERT_muon	*hadronic_FTFP_BERT* plus muon production processes with biased muon cross-sections.
hadronic_QGSP_BERT_HP	*hadronic_QGSP_BERT_muon* plus high precision low energy neutron scattering models

4.10 Options

Various simulation details can be controlled throught 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>;
```

4.10.1 options in BDSIM

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

Option	Function
Common Paramters	
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
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 wi
thresholdCutPhotons	the minimum energy above which to simulate photons - any below this energy will be killed

Continued on

Table 4.1 – continued from previous page

Option	Function
stopTracks	whether to track secondaires or not (defalt = 1)
circular	whether the accelerator is circular or not
Geometry Parameters	
samplerDiamater	diameter of samplers (default 8 m) [m]
includeIronMagFields	whether to include magnetic fields in the magnet poles
sensitiveBeamlineComponents	whether all beam line componetns record energy loss
sensitiveBeamPipe	wheterh the beam pipe records energy loss
vacuumMaterial	the material to use for the beam pipe vacuum
vacuumPressure	the pressure of the vacuum gas
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 electons
prodCutPositrons	standard overall production cuts for positrons
prodCutPositronsP	precision production cuts for positrons
turnOnCerenkov	whether to produce cerenkov radiation
defaultRangeCut	the default predicted range at which a particle is cut (default 0.7 mm) [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	wheter to use electromagnetic lead particle biasing (default = 0)
LPBFraction	the fraction of electromagnetic process in which lead particle biasing is used (0 < LPBFraactio
trajCutGTZ	global z position cut (minimum) for storing trajectories
trajCutLTR	radius cut for storing trajectories (maximum)
Output Parameters	Function
storeTrajectory	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
nperfile	number of evens to record per output file
nlinesIgnore	number of lines to ignore when reading user bunch input files
Tunnel Parameters	**Currently Not Working**
buildTunnel	whether to build a tunnel (default = 0)
builTunnelFloor	whether to add a floor to the tunnel
tunnelRadius	tunnel inner radius [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

Continued on

Table 4.1 – continued from previous page

Option	Function
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

4.11 Beam Parameters

To specify the input particle distribution to the accelerator model, the *beam* command is used. 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 has been specified.

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

The user must specify at least *energy*, *particle* and *distrType* (the distribution type). Additional parameters can be specified to detail in the input distribution. The beam is defined using the following syntax:

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

Enegy is in *GeV* by default. The partilce 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.

4.11.1 Beam Distributions

- gauss
- gaussTwiss
- reference

4.12 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:

CHAPTER

FIVE

MODEL PREPARATION

CHAPTER

SIX

OUTPUT

Output from BDSIM can be in various formats. 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

6.1 Output Format Differences

6.1.1 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

6.1.2 ASCII Output

With ASCII output, a folder is created with the given output name. Inside this histograms and sampler output is 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

6.2 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 losse 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.

6.3 Samplers

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

Coordinate Name	Units	Meaning
E0	GeV	Initial energy
x0	μm	Initial x position
y0	μm	Initial y position
z0	μm	Initial z position
xp0	rad	Initial x' position
yp0	rad	Initial y' position
zp0	rad	Initial z' position
t0	s	Initial time
E	GeV	Energy
x	μm	x position
y	μm	y position
z	μm	z position
xp	rad	x' position
yp	rad	y' position
zp	rad	z' position
t	s	Time
X	μm	Global x position
Y	μm	Global y position
Z	μm	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
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

6.4 Primary Coordinates

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

CHAPTER

SEVEN

OUTPUT ANALYSIS

7.1 ROOT Output

7.2 ASCII Output

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 Options - methods to generate bdsim options Plot - some nice plots for data

Classes: Analysis - encapsulates functions & plots for a single file Beam - a beam options dictionary with methods Builder

Build generic machines for bdsim. You can create a lattice using one of the predefined simple lattices or by adding many pieces together of your own design. Finally, output the gmad files required.

Classes: Element - beam line element that always has name,type and length Machine - a list of elements

pybdsim.Builder.**CreateDipoleFodoRing**(*filename*, *ncells=60*, *circumference=200.0*, *samplers='first'*)

Create a ring composed of fodo cells with 2 dipoles per fodo cell.

filename ncells - number of fodo+dipole cells to create circumference - circumference of machine in metres samplers - 'first','last' or 'all'

Hard coded to produce the following cell fractions: 50% dipoles 20% quadrupoles 30% beam pipe / drift

pybdsim.Builder.**CreateDipoleRing**(*filename*, *ncells=60*, *circumference=100.0*, *dfraction=0.1*, *samplers='first'*)

Create a ring composed solely of dipoles filename ncells - number of cells, each containing 1 dipole and a drift circumference - in metres dfraction - the fraction of dipoles in each cell (0.0<dfraction<1.0) samplers - 'first', 'last' or 'all'

pybdsim.Builder.**CreateFodoLine**(*filename*, *ncells=10*, *driftlength=4.0*, *magnetlength=1.0*, *samplers='all'*, ***kwargs*)

Create a FODO lattice with ncells.

ncells - number of fodo cells driftlength - length of drift segment in between magnets magnetlength - length of quadrupoles samplers - 'all','first' or 'last' **kwargs - kwargs to supply to quadrupole constructor

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

Element - a beam element class - inherits dict

Element(name,type,**kwargs)

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.

Numbers are converted to a python Decimal type to provide higher accuracy in the representation of numbers - 15 decimal places are used.

pybdsim.Builder.**GenerateSamplersFromBDSIMSurvey**(*surveyfile*, *outputfilename*, *excludesamplers=True*)

Create a gmad file with samplers for all the elements in a beamline as described by the survey outline from bdsim

bdsim –file=mylattice.gmad –outline=survey.dat –outline_type=survey

excludesamplers - bool - exclude any existing samplers

pybdsim.Builder.**SuggestFodoK**(*magnetlength*, *driftlength*)

returns k1 (float) value for matching into next quad in a FODO cell. f = 1/(k1 * magnetlength) = driftlength -> solve for k1

Note the convention in pybdsim.Builder is that the quadrupoles in the fodo cell are split in two. So this is in fact half the integrated k you need. This matches with the other functions in Builder.

pybdsim.Builder.**WriteLattice**(*machine(machine)*, *filename(string)*, *verbose(bool)*)

Write a lattice to disk. This writes several files to make the machine, namely:

filename_components.gmad - component files (max 10k per file) filename_sequence.gmad - lattice definition filename_samplers.gmad - sampler definitions (max 10k per file) filename_options.gmad - options (TO BE IMPLEMENTED) filename.gmad - suitable main file with all sub

files in correct order

these are prefixed with the specified filename / path

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

Filter(*booleanarray*)

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

Return type is BDSAsciiData

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") 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

CHAPTER

EIGHT

SUPPORT

8.1 Installation Trouble

8.2 Trouble Running BDSIM

8.3 Feature Request

CHAPTER

NINE

APPENDIX 1 - TRACKING ROUTINES

9.1 Quadrupole

CHAPTER

TEN

APPENDIX 2 - GEOMETRY INPUT

10.1 Mokka

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

p

B

C

E

F

G

M

P

S

W