BDSIM User's Manual v0.4

I. Agapov, S.Malton
revision 0.4, last updated Jul 5, 2007

Table of Contents

BDSIM v0.4 User's Manual

This file is updated automatically from '`manual.texi`' last updated on Jul 5, 2007.

1 About BDSIM

BDSIM is a Geant4 extension toolkit for simulation of particle transport in accelerator beamlines. It provides a collection of classes representing typical accelerator components, a collection of physics processes for fast tracking, procedures of "on the fly" geometry construction and interfacing to ROOT analysis.

2 Obtaining, Installing and Running

`BDSIM` can be downloaded from `http://ilc.pp.rhul.ac.uk/bdsim.html`. This site also contains some information on planned releases and other issues. Alternatively, a development version is accessible under `http://cvs.pp.rhul.ac.uk`. Download the tarball and extract the source code. Make sure Geant4 is installed and appropriate environment variables defined. Then go through the configuration procedure by running the `./configure` script.

```
./configure
```

It will create a Makefile from template defined in Makefile.in. You may want to edit the Makefile manually to meet your needs (if your CLHEP version is greater than 2.x put -DCLHEP_VERSION=9). Then start the compilation by typing

```
./make
```

If the compilation is successful `bdsim` executable should be created in the current directory or in the `$G4WORKDIR` directory in case this variable is defined. Next, set up the (DY)LD_LIBRARY_PATH variable to point to the ./parser directory and to the directory where libbdsim.so is.

`BDSIM` is invoked by the command `bdsim` '`options`'

where the options are

```
--file=<filename>    : specify the lattice file
--output=<fmt>       : output format (root|ascii), default ascii
--outfile=<file>     : output file name. Will be appended with _N
                       where N = 0, 1, 2, 3... etc.
--vis_mac=<file>     : visualization macro script, default vis.mac
--help               : display this message
--verbose            : display general parameters before run
--verbose_event      : display information for every event
--verbose_step=N     : display tracking information after each step
--verbose_event_num  : display tracking information for event number N
--batch              : batch mode - no graphics
--outline=<file>     : print geometry/optics info to <file>
--outline_type=<fmt> : type of outline format
                       where fmt = optics | survey
```

To run bdsim one first has to define the beamline geometry in a file which is then passes to bdsim via the `--file` command line option, for example

```
bdsim --file=line.gmad --output=root --batch
```

The next section describes how to do it in more detail.

3 Lattice description

The beamline, beam properties and physics processes are specified in the input file written in the GMAD language which is a variation of MAD language extended to handle sophisticated geometry and parameters relevant to radiation transport. GMAD is described in this section. Examples of input files can be found in the `BDSIM` distribution in the `examples` directory.

3.1 Program structure

A GMAD program consists of a sequence of element definitions and control commands. For example, tracking a 1 GeV electron beam through a FODO cell will require a file like this:

for example

```
qf: quadrupole, l=0.5*m, k1=0.1;
qd: quadrupole, l=0.5*m, k1=-0.1;
d: drift, l=0.5*m;
fodo : line=(qf,d,qd,d);
use,period=fodo;
beam, particle="e-",energy=1*GeV;
```

Generally, the user has to define a sequence of elements (with `drift`, `quadrupole`,`line` etc.), then select the beamline with the `use` command and specify beam parameters and other options with `beam` and `option` commands. The `sample` command controls what sort of information will be recorded during the execution.

The parser is case sensitive. However, for convenience of porting lattice descriptions from MAD the keywords can be both lower and upper case. The GMAD language is discussed in more detail in this section.

3.2 Arithmetical expressions

Throughout the program a standard set of arithmetical expressions is available. Every expression is ended with a semicolon. For example

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

The variables then could be used along with numerical constants. The `if-else` clause is also available, for example

```
z=1;
if(z<2)
y=2.5-x
  else
y=15;
```

3.3 Physical elements and Entities

GMAD implements almost all the standard MAD elements, but also allows to define arbitrary geometric entities and magnetic field configurations. The geometry description capabilities are extended by using "drivers" to other geometry description formats which makes interfacing and standardisation easier. The syntax of a physical element declaration is

`element_name : element_type, attributes;`

for example

`qd : quadrupole, l = 0.1*0.1, k1 = 0.01;`

`element_type` can be of basic type or inherited. Allowed basic types are

- marker
- drift
- sbend
- rbend
- quadrupole
- sextupole
- octupole
- multipole
- vkick
- hkick
- rf
- rcol
- ecol
- laser
- transform3d
- element

All elements except `element` are by default modeled by an iron cylinder (given by the `boxSize` option) with the vacuum beampipe (defined by `beampipeRadius` option). An already defined element can be used as a new element type. The child element will have the attributes of the parent.

```
q:quadrupole, l=1*m, k1=0.1;
qq:q,k1=0.2;
```

3.3.1 Coordinate system

3.3.2 Units

The usual accelerator coordinate system is assumed (see ref)

In GMAD the SI units are used.

Length [m] (metres)
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]
beam current [A] (Amperes)
particle charge [e] (elementary charges)
emittances [pi m mrad]

There are some predefined numerical values

pi 3.14159265358979
me electron rest mass
mp proton rest mass
GeV 1
eV 1^-9
KeV 10^-6
MeV 10^-3
TeV 10^3
m 1
mm 1^-3
cm 1^-2
rad 1
mrad 1^-3
clight 2.99792458e+8

for example, one can write either 100 or 0.1 * KeV when energy constants are concerned.

3.3.3 marker

`marker` has no effect but allows one to identify a position in the beam line (say, where a sampler will be placed). It has no attributes.

Example:

```
m1 : marker;
```

3.3.4 drift

`drift` defines a straight drift space. Attributes:

- `l` - length [m] (default 0)
- `aper` - aperture [m] (default same as beampipe radius)

Example :

```
d13 : drift, l=0.5*m;
```

3.3.5 rbend

`rbend` defines a rectangular bending magnet. Attributes:

- `l` - length [m] (default 0)
- `angle` - bending angle [rad] (default 0)
- `B` - magnetic field [T]
- `aper` - aperture [m] (default same as beampipe radius)
- `outR` - external radius [m] of magnet (default set to `aper`+1cm)

when `B` is set, this defines a magnet with appropriate field strength and `angle` is not taken into account. Otherwise, `B` that corresponds to bending angle `angle` for a particle in use (defined by the `beam` command, with appropriate energy and rest mass) is calculated and used in the simulations.

Example :

```
rb1 : rbend, l=0.5*m, angle = 0.01;
```

3.3.6 sbend

`sbend` defines a sector bending magnet. Attributes:

- `l` - length [m] (default 0)
- `angle` - bending angle [rad] (default 0)
- `B` - magnetic field [T]
- `aper` - aperture [m] (default same as beampipe radius)
- `outR` - external radius [m] of magnet (default set to `aper`+1cm)

Example :

The meaning of `B` and `angle` is the same as for `rbend`.

```
rb1 : rbend, l=0.5*m, angle = 0.01;
```

3.3.7 quadrupole

`quadrupole` defines a quadrupole. Attributes:

- `l` - length [m] (default 0)
- `k1` - normal quadrupole coefficient k1 = (1/B rho) (dBy / dx) [m^-2] Positive `k1` means horizontal focusing of positively charged particles. (default 0)
- ks1 - skew quadrupole coefficient ks1 = (1/B rho) (dBy / dx) [m^-2] where (x,y) is now a coordinate system rotated by 45 degrees around s with respect to the normal one.(default 0).
- `tilt` [rad] - roll angle about the longitudinal axis, clockwise.
- `aper` - aperture [m] (default same as beampipe radius)
- `outR` - external radius [m] of magnet (default set to `aper`+1cm)

Example :

```
qf : quadrupole, l=0.5*m , k1 = 0.5 , tilt = 0.01;
```

3.3.8 sextupole

`sextupole` defines a sextupole. Attributes:

- `l` - length [m] (default 0)
- `k2` - normal sextupole coefficient k2 = (1/B rho) (d^2 By / dx^2) [m^-3]
- `ks2` - skew sextupole coefficient ks2 = (1/B rho) (d^2 By / dx^2) [m^-3] where (x,y) is now a coordinate system rotated by 30 degrees around s with respect to the normal one.(default 0).
- `tilt` [rad] - roll angle about the longitudinal axis, clockwise.
- `aper` - aperture [m] (default same as beampipe radius)
- `outR` - external radius [m] of magnet (default set to `aper`+1cm)

Example :

```
sf : sextupole, l=0.5*m , k2 = 0.5 , tilt = 0.01;
```

3.3.9 octupole

`octupole` defines an octupole. Attributes:

- `l` - length [m] (default 0)
- `k2` - normal sextupole coefficient k3 = (1/B rho) (d^3 By / dx^3) [m^-3] Positive `k1` means horisontal focusing of positively charged particles. (default 0)
- ks3 - skew sextupole coefficient ks3 = (1/B rho) (d^3 By / dx^3) [m^-3] where (x,y) is now a coordinate system rotated by 30 degrees around s with respect to the normal one.(default 0).
- `tilt` [rad] - roll angle about the longitudinal axis, clockwise.
- `outR` - external radius [m] of magnet (default set to `aper`+1cm)

Example :

```
sf : octupole, l=0.5*m , k3 = 0.5 , tilt = 0.01;
```

3.3.10 multipole

`octupole` defines an octupole. Attributes:

- `l` - length [m] (default 0)
- `knl` - normal multipole knl_n = (1/B rho) (d^n By / dx^n) [m^-n] Positive `k1` means horisontal focusing of positively charged particles. (default 0)
- ks3 - skew multipole ksl_n = (1/B rho) (d^n By / dx^n) [m^-n] where (x,y) is now a coordinate system rotated by 30 degrees around s with respect to the normal one.(default 0).
- `tilt` [rad] - roll angle about the longitudinal axis, clockwise.
- `outR` - external radius [m] of magnet (default set to `aper`+1cm)

Example :

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

3.3.11 rf

`rf` defines an rf cavity

Attributes:

- `l` - length [m] (default 0)
- `gradient` - field gradient [MV / m]

Example :

```
rf1 : rf,l=5*m, gradient = 10 * MV / m;
```

3.3.12 rcol

`rcol` defines a rectangular collimator

Attributes:

- `l` - length [m] (default 0)
- `xsize` - horizontal aperture [m]
- `ysize` - vertical aperture [m]
- `material` - material
- `outR` - limits external extent [m] of collimator (default set to `aper`+1cm)

Example :

```
col1 : rcol,l=0.4*m, xsize=2*mm, ysize=1*mm, material="W"
```

The longitudinal collimator structure is not taken into account. To do this the user has to describe the collimator with the generic type `element`

3.3.13 ecol

`ecol` defines an elliptical collimator

Attributes:

- `l` - length [m] (default 0)
- `xsize` - horizontal aperture [m]
- `ysize` - vertical aperture [m]
- `material` - material
- `outR` - limits external extent [m] of collimator (default set to `aper`+1cm)

Example :

```
col2 : ecol,l=0.4*m, xsize=2*mm, ysize=1*mm, material="W"
```

Here the longitudinal collimator structure is also not taken into account.

3.3.14 solenoid

Not yet implemented

3.3.15 hkick and vkick

`hkick` and `vkick` are equivalent to a `rbend` and an `rbend` rotated by 90 degrees respectively.

3.3.16 transform3d

An arbitrary 3-dimensional transformation of the coordinate system is done by placing a `transform3d` element in the beamline. The syntax is

- `x` = `<x offset>`
- `y` = `<y offset>`
- `z` = `<z offset>`
- `phi` = `<phi Euler angle>`
- `theta` = `<theta Euler angle>`
- `psi` = `<psi Euler angle>`

Example:

```
rot : transform3d, psi=pi/2
```

3.3.17 element

All the elements are in principle examples of a general type `element` which can represent an arbitrary geometric entity with arbitrary B field maps Its attributes are

- `geometry` = `<geometry_description>`
- `bmap` = `<bmap_description>`
- `outR` - limits external extent component box size (default set to `aper`+1cm)

Descriptions are of the form

```
format:filename
```

where `filename` is the path to the file with the geometry description and `format` defines the geometry description format. The possible formats are given in Appendix A [Geometry], page 15.

Example :

```
qq : element, geometry ="mokka:qq.sql", bmap ="mokka:qq.bmap";
```

3.3.18 line

elements are grouped into sequences by the `line` command.

`line_name` `: line=(`*`element_1`*`,`*`element_2`*`,...);`

where *element_n* can be any element or another line.

Example :

A sequence of FODO cells can be defines as

```
qf: quadrupole, l=0.5, k1=0.1;
qd: quadrupole, l=0.5, k1=-0.1;
d: drift, l=0.5;
fodo : line=(qf,d,qd,d);
section : line=(fodo,fodo,fodo);
beamline : line=(section,section,section);
```

3.3.19 laser

`laser` defines a drift section with a laser beam inside.

```
<laser_name>: laser, l=<val>, x=<xdir>, y=<ydir>, z=<zdir>, waveLength=<val>;
```

Attributes

- `l` - length of the drift section [m]
- `x,y,z` - components of the laser direction vector
- `waveLength` - laser wave length [m]

The laser is considered to be the intersection of the laser beaam with the volume of the drift section.

3.3.20 spec keyword

Starting from v0.3 it is possible to add the `spec` keyword to all element definition.

```
spec="keyword1=value1&keyword2=value2&.."
```

By this means any set of keyword/value pairs can be passed to the accelerator component construction classes.

```
qd : mquad,l=0.5 * m, k1 = qdk1, spec="type=cylinder";
```

3.3.21 Element number

When several elements with the same name are present in the beamline they can be accessed by their number in the sequence. In the next example the sampler is put before the second drift

```
bl:line=(d,d,d);
sample,range=d[2];
```

3.3.22 Element attributes

Element attributes such as length, multipole coefficients etc, can be accessed by putting square brackets after the element name, e.g.

```
x=d[l];
```

3.3.23 Material table

There is a set of predefined materials for use in elements such as collimators, e.g.

“Al” “W” “Iron” “Copper” “Graphite” etc.

Note that each geometry driver such as Mokka has its own set of materials

3.4 Run control and output

The execution control is performed in the GMAD input file through `option` and `sample` commands. How the results are recorded is controlledby the `sample` command. When the visualization is turned on, it is also controlled through Geant4 command prompt

3.4.1 `option`

Most of the options in `bdsim` are set up by the command

`option, <name>=value,...;`

The following options influence the geometry

```
beampipeRadius    - default beampipe radius [m]
beampipeThickness - default beampipe thickness [m]
tunnelRadius      - tunnel Radius [m]
boxSize           - default accelerator component size [m]
```

The following options influence the tracking

```
deltaChord          - chord finder precision
deltaIntersection   - boundary intersection precision
chordStepMinimum    - minimum step size
lengthSafety        - element overlap safety
thresholdCutCharged - charged particle cutoff energy
thresholdCutPhotons - photon cutoff energy
randomSeed          - seed for the random number generator
                          setting to -1 uses the system clock to gen-
erate the seed
stopTracks          - if set, tracks are terminated after interaction with
                          material and energy deposit recorded
physicsList         - determines the set of physics processes used
ngenerate           - number of primary particles fired when in batch mode
nperfile            - number of events recorded per file
nlinesIgnore        - number of lines to skip when reading bunch files
synchRadOn          - turn on Synchrotron Radiation process
srTrackPhotons      - whether to track the SR photons
srLowX              - Sets lowest energy of SR to X*E_critical
srLowGamE           - lowest energy of propagating SR photons
minimumEpsilonStep  - minimum relative error acceptable in stepping
maximumEpsilonStep  - maximum relative error acceptable in stepping
deltaOneStep        - set position error acceptable in an integration steps
prodCutPhotons      - standard overall production cuts for photons
prodCutPhotonsP     - precision production cuts photons in element
prodCutElectrons    - standard overall production cuts for electrons
prodCutElectronsP   - precision production cuts electrons in element
prodCutPositrons    - standard overall production cuts for positrons
```

```
prodCutPositronsP   - precision production cuts positrons in element
```

For a more detailed description of how the option influence the tracking see Chapter 5 [Physics], page 13

3.4.2 beam

The parameters related to the beam are given by the `beam` command

```
beam, <name>=value,...;
```

The available parameters are:

```
particle    - particle name, "e-","e+","gamma","proton", etc
energy      - particle energy
distrType   - type of distribution
distrFile   - input bunch file
```

```
beam, particle="e+",energy=100*MeV, distrType="gauss";
```

3.4.3 sample

To record the tracking results one uses the `sample`

```
sample, range=<element>;
```

The sampling plane is then inserted before `<element>`

Example :

```
sample, range=d;
```

Cylindrical sampler of length `l` is put around element `<element>` at distance `<r>` with the command

```
sample, range=<element>, r=r0, l=l0;
```

3.4.4 use

`use` command selects the beam line for study

```
use, period=l1,range=q1/q2
```

4 Visualization

When `BDSIM` is invoked in interactive mode, the run is controlled by the Geant4 shell. A visualization macro should be then provided. A simple visualization macro is listed below.

```
# Invoke the OGLSX driver
#  Create a scene handler and a viewer for the OGLSX driver
/vis/open OGLIX

# Create an empty scene
/vis/scene/create

# Add detector geometry to the current scene
/vis/scene/add/volume

# Attach the current scene handler
# to the current scene (omittable)
/vis/sceneHandler/attach

# Add trajectories to the current scene
#  Note: This command is not necessary in exampleN03,
#        since the C++ method DrawTrajectory() is
#        described in the event action.

/vis/viewer/set/viewpointThetaPhi 90 90
# /vis/drawVolume
#/vis/scene/add/trajectories
# /tracking/storeTrajectory 0
#/vis/viewer/zoom
/tracking/storeTrajectory 1
#
# for BDS:
#/vis/viewer/zoom 300
#/vis/viewer/set/viewpointThetaPhi 3 45
```

By default the macro is read from the file named `vis.mac`. The name of the file with the macro can also be passed via the `vis_mac` switch.

```
bdsim --file=line.gmad --vis_mac=my_macro.mac
```

In interactive mode all the Geant4 interactive comamnds are available. For instance, to fire 100 particles type

`/run/beamOn 100` runs the simulation with 100 particles

and to end the session type

`exit`

To display help menu

`/help`;

For more details see [Geant], page 25.

5 Physics

BDSIM can exploit all physics processes that come with Geant4. In addition fast tracking inside multipole magnets is provided. More detailed description of the physics is given below.

5.1 physicsList option

Depending on for what sort of problem BDSIM is used, different sorts of physics processes should be turned on. This processes are groupes into so called "physics lists". The physics list is specified by the 'physicsList option in the input file, e.g.

```
option, physicsList="em_standard";
```

Several predefined physics lists are available

```
"standard"          -   transportation of primary particles only
"em_standard"       -   transporation of primary particles, ionization,
                        bremsstrahlung, multiple scattering
"em_low"            -   the same but using low energy electromagnetic models
"em_muon"            -   the same but using biased muon cross-sections
"lw"                -   list for laser wire simulation - standard electromagnetic
                        physics and "laser wire" physics which is Compton Scattering
                        with total cross-section renormalized to 1.
"hadronic_standard" -   standard electromagnetic, fission, neutron capture, neutron
                        and proton elastic and inelastic scattering.
```

By default the standard physics List is used

5.2 Transportation

The transportation follows the scheme: the step length is selected which is defined either by the distance of the particle to the boundary of the "logical volume" it is currently in (which could be, e.g. field boundary, material boundary or boundary between two adjacent elements) or by the mean free path of the activated processes. Then the particle is pushed to the new position and secondaries are generated if necessary. Each volume has an associated transporatation algorithm. For an on-energy particle travelling close to the optical axis of a quadrupole, dipole or a drift, standard matrix transportation algorithms are used [Course], page 25. For multipoles of higher orders and for off-axis/energy particles Runge-Kutta methods are used.

5.3 Tracking accuracy

The following options influence the tracking accuracy

chordStepMinimum minimum chord length for the step

deltaIntersection determines the precision of locating the point of intersection of the particle trajectory with the boundary and hence the error in the path length in each volume. This may influence the results especially in the case when EM fields are present.

`deltaChord`

`lengthSafety` all volumes will have an additional overlap of this length

`thresholdCutCharged` energy below which charged particles are not tracked.

`thresholdCutPhotons` energy below which photons are not tracked.

6 Output Analysis

During the execution the following things are recorded:

```
Energy deposition along the beamline
Sampler hits
```

If the output format is `ASCII` i.e. if `BDSIM` was invoked with the `--output=ascii` option, then the output file "output.txt" containing the hits will be written which has rows like

```
#hits (PDGtype  p[GeV/c],x[micron],y[micron],z[m],x'[microrad],y'[microrad]):
11 250 -4.72907 -5.86656 5.00001e-06 0 0
11 250 -8.17576 -4.99729 796.001 0.320334 -0.126792
```

if `ROOT` output is used then the `root` files `output_0.root`, `output_1.root` etc. will be created with each file containing the number of events given by `nperfile` option.

The file contains the energy loss histogram and a tree for every sampler in the line with self-explanatory branch names.

7 Implementation Notes

7.1 Architecture

In this section the architecture of `BDSIM` is briefly described for someone wishing to use it as a class library.

- BDSMultipole

- gmad

- Physics list - adding own physics processes

7.2 Features to be added in next releases

current development is focused on the beam-gas scattering and implementation of wake-fields.

Appendix A Geometry description formats

The element with user-defined physical geometry is defined by

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

for example,

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

A.1 gmad format

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

```
Box {
x0=x_origin,
y0=y_origin,
z0=z_origin,
x=xsize,
y=ysize,
z=zsize,
material=MaterialName,
temperature=T
}
```

```
Tubs {
x0=x_origin,
y0=y_origin,
z0=z_origin,
x=xsize,
y=ysize,
z=zsize,
material=MaterialName,
temperature=T
}
```

For example

```
Cons {
x0=0,
y0=0,
z0=0,
rmin1=5
rmax1=500
rmin2=5
rmax2=500
z=250
material="Graphite",
phi0=0,
dphi=360,
temperature=1
```

```
}
```

A file can contain several objects which will be placed consequently into the volume, A user has to make sure that there is no overlap between them.

A.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 - although future versions of BDSIM will also support online querying of MySQL databases. 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 to load geometry descriptions from a list

A.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 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 = 10mm, length_y = 150mm, length_z = 50mm, positioned at x=1m, y=0, z=0.5m and with zero rotation.

```
CREATE TABLE mytable_BOX (

NAME                          VARCHAR(32),

MATERIAL                      VARCHAR(32),

LENGTHX                       DOUBLE(10,3),

LENGTHY                       DOUBLE(10,3),

LENGTHZ                       DOUBLE(10,3),

POSX                          DOUBLE(10,3),

POSY                          DOUBLE(10,3),

POSZ                          DOUBLE(10,3),

ROTPSI                        DOUBLE(10,3),

ROTTHETA                      DOUBLE(10,3),

ROTPHI                        DOUBLE(10,3)

);

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/Mokka/General directory. See the common table parameters and solid type sections below for more information on the table parameters available for use.

A.2.1.1 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)`

  This is an 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)`

  This is an 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)`

  This is an 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)`

  This is an 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)`

  This is an 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)`

  This is an 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)`

  This is an 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)`

  This is an 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)`

  This is an optional parameter. If set to a value other than zero, in conjuction 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)`

  This is an optional parameter. If set to a value other than zero, in conjuction 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)`

  This is an optional parameter. If set to a value other than zero, in conjuction 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)`

  These are required parameters. They are form 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 refers to the center of the parent object.

- `ROTPSI, ROTTHETA, ROTPHI`

  Variable type: `DOUBLE(10,3)`

  These are optional parameters. They are the Euler angles in radians used to rotate the obejct before it is placed. The default is set to zero for each angle.

- `RED, BLUE, GREEN`

  Variable type: `DOUBLE(10,3)`

  These are 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)`

  This is an 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)`

  These are 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 enitre `element` then this uniform field will act in addition to the solenoid field.

A.2.1.2 '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)`

  These are required parameters. There values will be used to specify the box's dimensions.

A.2.1.3 'Trapezoid' Solid Types

Append `_TRAP` to the table name in order to make use of the G4Trd solid type - which is deined 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)`

  This is a 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)`

  This is a 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)`

  This is a 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)`

  This is a 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)`

  This is a required parameter. This value will be used to specify the z-extent of the box's dimensions.

A.2.1.4 '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)`

  This is a required parameter. This value will be used to specify the z-extent of the cone's dimensions.

- `RINNERSTART`

  Variable type: `DOUBLE(10,3)`

  This is an 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)`

  This is an 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)`

  This is a required parameter. This value will be used to specify the outer radius of the start of the cone.

- `ROUTEREND`

  Variable type: `DOUBLE(10,3)`

  This is a required parameter. This value will be used to specify the outer radius of the end of the cone.

- `STARTPHI`

  Variable type: `DOUBLE(10,3)`

  This is an 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)`

  This is an optional parameter. If set then this value will be used to specify the delta angle of the cone. The default value is 2*PI.

A.2.1.5 '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)`

  This is an 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)`

This is a required parameter. This value will be used to specify the outer radius of the torus tube.

- `RSWEPT`

  Variable type: `DOUBLE(10,3)`

  This is a 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)`

  This is an 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)`

  This is an optional parameter. If set then this value will be used to specify the delta swept angle of the torus. The default value is 2*PI.

A.2.1.6 '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)`

  This is a 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)`

  These are 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)`

  These are 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)`

  These are 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)`

  This is an 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)`

  This is an optional parameter. If set then this value will be used to specify the delta angle of the polycone. The default value is 2*PI.

A.2.1.7 '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)`

  This is a required parameter. This value will be used to specify the Semiaxis in X.

- `YSEMIAXIS`

  Variable type: `DOUBLE(10,3)`

  This is a required parameter. This value will be used to specify the Semiaxis in Y.

- `LENGTHZ`

  Variable type: `DOUBLE(10,3)`

  This is a required parameter. This value will be used to specify the height of the elliptical cone.

- `ZCUT`

  Variable type: `DOUBLE(10,3)`

  This is a 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.

A.2.2 Creating a geometry list

A geometry list is a simple file consisting of a list of filenames 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
```

A.2.3 Defining a Mokka element in the gmad file

The Mokka element can be defined by the following command:

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

where `format` must be set to `mokka` and `filename` must point to a file that contains a list of files that have the geometry descriptions.

for example,

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

A.3 gdml

GDML is a XML schema for dtector description. GDML will be supported as an external format starting from next release.

Appendix B Field description formats

The element with user-defined physical geometry is defined by command

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

for example,

```
colli : element, geometry=plain:colli.geom
```

Appendix C Bunch description formats

For compatibility with other simulation codes following bunch formats can be read.For example, to use the file `distr.dat` as input the beam definition should look like

```
beam, particle="e-",distrType="guineapig_bunch",distrFile="distr.dat",...
```

The formats currently supported are listed below

`guineapig_bunch`

E[GeV] x[micrometer] y[micrometer] z[micrometer] x'[microrad] y'[microrad]

`guineapig_slac`

E[GeV] x'[rad] y'[rad] z[nanometer] x[nanometer] y[micrometer]

`guineapig_pairs`

E[GeV] x[rad] y[rad] z[rad] x[nanometer] y[nanometer] z[nanometer]

here a particle with E>0 is assumed to be an electron and with E<0 a positron.

`guineapig_pairs`

a custom distribution file format can be specified in the form

```
distrType="field1[unit1]:field1[unit1]:...
```

```
beam,  particle="e-",
       energy=ener * GeV,
       nparticles=1e+3,
       distrType="pt[1]:E[GeV]:xp[rad]:yp[rad]:z[mum]:x[nm]:y[nm]",
       distrFile="bunches/beam.dat";
```

The following distributoion types can be generated

`Gaussian`

```
beam,distrType="gauss",sigmaX=...,sigmaXp=...,sigmaY=...,sigmaYp=...,sigmaE=...,
```

`Elliptic shell`

generated a thin elliptic shell in `x,x'` and `y,y'` with given semiaxes

```
beam,distrType="eshell",x=...,xp=...,y=...,yp=...;
```

8 References

1. G. Blair, Simulation of the CLIC Beam Delivery System Using BDSIM, CLIC Note 509
2. Root User's Guide, `http://root.cern.ch/root/doc/RootDoc.html`
3. Geant4 User's Guide, `http://geant4.cern.ch/support/userdocuments.shtml`
4. MAD-X User's Guide, `http://mad.home.cern.ch/mad/uguide.html`
5. for example 'Basic course on Accelerator optics' by Schmuesser, Rossbach, CERN Accelerator school