BDSIM User's Manual v0.1

Ilya Agapov
last updated October 12, 2005

Table of Contents

BDSIM beta User's Manual

This file is updated automatically from '`manual.texi`', which was last updated on October 12, 2005.

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, procedure of "on the fly" geometry construction, and interface to ROOT analysis.

2 Obtaining, Installing and Running

`BDSIM` can be downloaded at http://flc.pp.rhul.ac.uk/bdsim.html. Alternatively, a development version is accessible under 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. Then start the compilation by typing

```
./make
```

If the compilation is successful `bdsim` executable should be created in the current directory of in the `$G4WORKDIR` directory in case this variable is defined. Next, set up the 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>    : file with the 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
```

`BDSIM` can be also invoked by the `bdsimrun` shell script which handles batch job support and opther features.

To run bdsim one has to define the beamline geometry first in a file which is then passes to bdsim via the `--file` command line option. The next section describes how to do it.

3 Lattice description

The beamline, beam properties and physics processes are specified in the input file written in the GMAD language.[1] This language is a variation of the MAD language and is described in this section.

3.1 Program structure

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

```
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);
use,period=fodo,range=#s/#e;
beam, particle=electron,energy=1;
```

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 Physical elements and Entities

GMAD implements almost all standard MAD elements, but also allows to define arbitrary geometric entities and magnetic field configurations. The syntax of a physical element declaration is

`element : element_type, attributes;`

for example

`qd : quadrupole, l = 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

An already defined element can be used as a new element type. Its attributes are then inherited.

3.2.1 Coordinate system

[1] Note: In the next releases an xml-based description will be also available

3.2.2 Units

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`	[eV/c**2]
`particle momentum`	[eV/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
`KeV`	10^3 (in [eV] units)
`MeV`	10^6 (in [eV] units)
`GeV`	10^9 (in [eV] units)
`TeV`	10^12

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

3.2.3 marker

`marker` has no effect but allows one to identify a position in the beam line. It has no attributes.

Example:

```
m1 : marker;
```

3.2.4 drift

`drift` defines a straight drift space. Attributes:

- `l` - length [m] (default 0)

Example :

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

3.2.5 rbend

`rbend` defines a rectangulat bending magnet. Attributes:

- `l` - length [m] (default 0)
- `angle` - bending angle [rad] (default 0)
- `B` - magnetic field [T]

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, angle = 0.01;
```

3.2.6 sbend

`sbend` defines a sector bending magnet. Attributes:

- `l` - length [m] (default 0)
- `angle` - bending angle [rad] (default 0)
- `B` - magnetic field [T]

Example :

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

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

3.2.7 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 horisontal 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.

  Example :

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

3.2.8 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] Positive `k1` means horisontal focusing of positively charged particles. (default 0)
- ks1 - 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.

  Example :

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

3.2.9 octupole

3.2.10 multipole

3.2.11 collimator

`collimator` defines a collimator

Attributes:

- `l` - length [m] (default 0)
- `aperture` - aperture , defined by the `aperture` element
- `material` - material , defined by `material`

  Example :

  `coll : collimator,l=1, aperture=<aperture>, material=<material>`

3.2.12 solenoid

3.2.13 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:

  `<name> : transform3d, psi=pi/2`

3.2.14 element

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

- `geometry` = `<geometry_description>`
- `bmap` = `<bmap_description>`
- `emap` = `<emap_description>`

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 11.

Example :

`qq : element, geometry = plain:qq.geom, bmap = plain:qq.bmap;`

`<bmap>` and `<emap>` are definitions of E and B field maps according to (field maps).

3.2.15 line

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

`line_name` `: line=(`*`element1`*`,`*`element2`*`,...);`

element`n` can be any element or another line.

Example :

A sequence of three 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);
beamline : line{fodo,fodo,fodo};
```

3.2.16 aperture

3.2.17 material

`<material> : material,Z=,A=,density=,temperature=`

Attributes

- `Z` - atomic number
- `A` - mass number
- `density` - [kg/m]
- `temperature` [K]

3.2.18 pipe

the beam pipe parameters are used for particle tracking inside elements when the use geometry is not defined. The beam pipe radius is assigned by the `pipe` command

`pipe, range=<range>, range=, r=, thickness=, material=<material>;`

Attributes

- `range` - element range to assign the radius for
- `r` - radius [m]
- `thickness` - thickness [m]
- `material` - beam pipe material

Example :

Supposing we want to define a copper beam pipe for ...

```
iron : material, Z=1,A=1,density=100, temperature=;
copper : material,Z=,A=,density=,temperature=;;

fodo : line=(qf,d,qd,d);
pipe, range=qf/qd, r=0.2, thickness = 0.1,material=copper;
pipe, range=d[2], r=0.1, thickness = 0.05,material=iron;
```

3.2.19 laser

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

`<laser_name>: laser, position = {<x>,<y>,<z>},direction={ <dx>, <dy>, <dz> } wavelen=<val>, spotsize=<val>, intensity=<val>;`

Attributes

- `l` - length of the drift section
- `position` - position of an arbitrary point on the beam axis relative to the center of the drift section
- `direction` - vector pointing in the beam direction
- `wavelen` - laser wave length [m]
- `spotsize` - spot size (sigma)[m]
- `intensity` -[W]

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

3.2.20 gas

`gas` command is used to introduce gas into the beam pipe.[2]

`gas, period=, components={c1,c2,...},parts={p1,p2,...}` ;

where

- `c1,c2,...` - gas components names
- `p1,p2,...` - parts (100%=1). They need not sum up to 1.

the gas componens are defined by

`c1 : gas, name=<name>, A=<A>, Z=<Z>, profile=<profile_name>`;

where

[2] in realistic situations the gas profile can vary in transverse dimensions. This is not taken into account for a)technical reasons b)one does not know the profile anyway

- `<Z>` - atomic number
- `<A>` - mass number
- `<profile_name>` - name of gas profile definition

the gas profile is defined as

`<profile_name> : gas_profile = (<element>:<pressure>, <element>:<pressure>)`
;

where

- `<element>` - name of the beamline component
- `<pressure>` - gas pressure (bar)

The gas pressure is then interpolated between the points where it is defined. Issuing multiple `gas` commands acts additively.

Example :

To introduce the gas into a fodo cell

```
...element definitions...

fodo : line=(qf,d,qd,d);

co2 : gas, name="c02", Z=22,A=44,profile=co2profile;
h20 : gas, name="h2", Z=1,A=1,profile=co2profile;

c02profile : gas_profile = (qd:0.01, qf:0.02*nbar,d:0.03*nbar);
h20profile : gas_profile = (qd:0.04, qf:0.01*nbar,d:0.03*nbar);

gas, period=fodo,components= { c02,h20},parts={0.7,0.8};
```

3.3 Run control and output

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

3.3.1 `option`

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

the available options are options are

```
nperfile
beampipeRadius
boxSize
tunnelRadius
beampipeThickness
```

```
    deltaChord
    deltaIntersection
    chordStepMinimum
    lengthSafety
    turnInteractions
    thresholdCutCharged
    thresholdCutPhotons
    useEMHadronic
    storeTrajectory
    stopTracks
```

For a more detailed description of how the option influence the tracking see Chapter 5 [Physics], page 10

Example :

3.3.2 beam

```
beam, particle=electron,energy=100, momentum=1,1,1;
```

3.3.3 sample

To record the tracking results one uses the `sample` command:

```
sample, range=<range>,particle=<particle>,values={value1,value2,...}
```

the parameters are

- `element`
- `range`
- `particle` - particle to record. One `sample` command activates sampling only for one particle type
- `value`
  - `x` - horizontal
  - `px` - horizontal momentum
  - `xx` -
  - `y`
  - `py`
  - `E` - energy
  - `id` - track id. This enables later trajectory analysis.

Example :

```
    sample, element=qd,particle=electron, values={x,px,y,py,e,id };
    sample, range=qd/qf:0.1*m,particle=photon, values={x,px,y,py,e,id };
```

3.3.4 use

`use` command selects the beam line for study

```
use, period=,range=
```

3.3.5 visualization control

when bdsim is invoked in interactive mode, the run is controlled by the Geant4 shell. Some examples

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

To display help menu

`/help`;

For more details see [Geant], page 20.

4 Visualization

Visualization system description

5 Physics

5.1 Transportation

5.1.1 Tracking of charged particles in EM fields of low multipole order

5.1.2 Tracking of charged particles in arbitrary EM fileds

5.1.3 Neutron transport

not implemented yet

5.2 Shower parametrization

5.3 Synchrotron Radiation

5.4 Bremsstrahlung

5.5 Compton

5.6 Gas scattering

5.7 Tuning the tracking procedure

6 Implementation Notes

BDSIM uses Geant4 libraries and execution control mechanism.

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.

- `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 . Two magnet types are available - “QUAD” and “SEXT”. The default is set to no magnet type. Note that if `MAGTYPE` is set to a value whilst `K1` or `K2` 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 it 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 it set to zero.
- `POSX`

  Variable type: `DOUBLE(10,3)`

  This is a required parameter. This is the X-position in mm used to place the object in the component volume. It is defined with respect to the center of the component volume and with respect to the component volume's rotation.
- `POSY`

  Variable type: `DOUBLE(10,3)`

  This is a required parameter. This is the Y-position in mm used to place the object in the component volume. It is defined with respect to the center of the component volume and with respect to the component volume's rotation.
- `POSZ`

  Variable type: `DOUBLE(10,3)`

  This is a required parameter. This is the Z-position in mm used to place the object in the component volume. It is defined with respect to the start of the component volume and with respect to the component volume's rotation.
- `ROTPSI`

  Variable type: `DOUBLE(10,3)`

  This is an optional parameter. This is the psi Euler angle in radians used to rotate the obejct before it is placed. The default is set to zero.
- `ROTTHETA`

  Variable type: `DOUBLE(10,3)`

  This is an optional parameter. This is the theta Euler angle in radians used to rotate the obejct before it is placed. The default is set to zero.
- `ROTPHI`

  Variable type: `DOUBLE(10,3)`

  This is an optional parameter. This is the phi Euler angle in radians used to rotate the obejct before it is placed. The default is set to zero.
- `RED`

  Variable type: `DOUBLE(10,3)`

  This is an optional parameter. This is the red component of the RGB colour assigned to the object and should be a value between 0 and 1. The default is set to zero.

- `BLUE`

  Variable type: `DOUBLE(10,3)`

  This is an optional parameter. This is the blue component of the RGB colour assigned to the object and should be a value between 0 and 1. The default is set to zero.
- `GREEN`

  Variable type: `DOUBLE(10,3)`

  This is an optional parameter. This is the green component of the RGB colour assigned to the object and should be a value between 0 and 1. The default is set to zero.
- `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 wireframe.

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`

  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.
- `LENGTHY`

  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.
- `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.2.1.3 '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.2.1.4 '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.2.1.5 'Polycone' Solid Types

Append `_POLYCONE` to the table name in order to make use of the G4Cons 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.2.2 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

`GUINEAPIG_BUNCH`

7 Authors

Name	Email
Ilya Agapov	`agapov@pp.rhul.ac.uk`
Grahame Blair	`blair@pp.rhul.ac.uk`
John Carter	`carter@pp.rhul.ac.uk`

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://wwwasd.web.cern.ch/wwwasd/geant4/G4UsersDocuments/Overview/html`
4. MAD-X User's Guide, `http://mad.home.cern.ch/mad/uguide.html`
5. NLC Zero-order design report