Mice Module Documentation

This document documents the main functionality of the Mice Modules that control geometry and fields of G4MICE. The document is divided into three parts. Part 1 gives an overview of the basic usage of Mice Modules. Part 2 gives details on how to implement different geometry models in G4MICE. Part 3 gives details on how to implement different electromagnetic field models in G4MICE.

How to Use this document

This document is best used in conjunction with the various examples than can be found in any installation of G4MICE. If you are familiar with Mice Modules already, the first part can be omitted and the latter two parts used as reference materials.

Any problems or omissions should be emailed to the document author, chris.rogers@stfc.ac.uk. I really like to get feedback on what is annoying in the documentation, what doesn't make sense and what is just wrong – so I encourage you to hassle me!

Table of Contents

Overview

Mice Modules are the objects that control the geometry and fields that are simulated in G4MICE. They are used in conjunction with a datacard file, which provides global run control parameters. Mice Modules are created by reading in a series of text files when G4MICE applications are run. All Mice Module files must be in a subdirectory of the directory defined by the environment variable *$ {MICEFILES}*.

This geometry information is used primarily by the Simulation application for tracking of particles through magnetic fields. A few commands are specific to detector Reconstruction and accelerator beam Optics applications.

The Mice Modules are created in a tree structure. Each module is a parent of any number of child modules. Typically the parent module will describe a physical volume, and child modules will describe physical volumes that sit inside the parent module. Modules cannot be used to describe volumes that do not sit at least partially inside the volume if the parent module.

Each Mice Module file consists of a series of lines of text. Firstly the Module name is defined. This is followed by an opening curly bracket, then the description of the module and the placement of any child modules, and finally a closing curly bracket. Commands, curly brackets etc must be separated by an end of line character.

Comments are indicated using either two slashes or an exclamation mark. Characters placed after a comment on a line are ignored.

G4MICE operates in a right handed coordinate system *(x,y,z)*. In the absence of any rotation, lengths are considered to be extent along the *z*-axis, widths to be extent along the *x*-axis and heights to be extent along the *y*-axis. Rotations (θ_x, θ_y, θ_z) are defined as a rotation about the z-axis through θ_z, followed by a rotation about the y-axis through θ_y, followed by a rotation about the x-axis through θ_x.

Configuration File

The Configuration file places the top level objects in MICE. The location of the file is controlled by the datacard *MiceModel*. G4MICE looks for the configuration file in the first instance in the directory

$*{MICEFILES}/Models/Configuration/<MiceModel>*

where *${MICEFILES}* is a user-defined environment variable. If G4MICE fails to find the file it searches the local directory.

The world volume is defined in the Configuration file and any children of the world volume are referenced by the Configuration file. The Configuration file looks like:

```
Configuration <Configuration Name>
{
      Dimensions  <x> <y> <z> <Units>
      <Properties>
      <Child Modules>
}
```

<Configuration Name> is the name of the configuration. Typically the Configuration file name is

given by <*Configuration Name*>.dat. The world volume is always a rectangular box centred on (0,0,0) with x, y, and z extent set by the Dimensions. Properties and Child Modules are described below and added as in any Module.

Substitutions

It is possible to make keyword substitutions that substitutes all instances of <name> with <value> in all Modules. The syntax is

```
Substitution <name> <value>
```

<name> must start with a single $ sign. Substitutions must be defined in the Configuration file. Note this is a direct text substitution in the MiceModules before the modules are parsed properly. So for example,

```
Substitution $Sub SomeText
PropertyString FieldType $Sub
PropertyDouble $SubValue 10
```

would be parsed as G4MICE like

```
PropertyString FieldType     SomeText
PropertyDouble SomeTextValue 10
```

Expressions

The use of equations in properties of type double and Hep3Vector is also allowed in place of a single value. So, for example,

```
PropertyDouble FieldStrength 0.5*2 T
```

would result in a FieldStrength property of 1 Tesla.

Expression Substitutions

Some additional variables can be defined in specific cases by G4MICE itself for substitution into experssions, in which case they will start with @ symbol. For these variable substitutions, it is only possible to make the substitution into expressions. So for example,

```
PropertyDouble ScaleFactor 2*@RepeatNumber
```

Would substitute @RepeatNumber into the expression. @RepeatNumber is defined by G4MICE when repeating modules are used (see RepeatModule2, below). Note the following code is not valid

```
PropertyString FileName File@RepeatNumber //NOT VALID
```

This is because Expression Substitutions can only be used in an expression (i.e. an equation).

Module Files

Children of the top level Mice Module are defined by Modules. G4MICE will attempt to find a module in an external file. The location of the file is controlled by the parent Module. Initially G4MICE looks in the directory

$*{MICEFILES}/Models/Modules/<Module>*

If the Mice Module cannot be found, G4MICE searches the local directory. If the module file still cannot be found, G4MICE will issue a warning and proceed.

The Module description is similar in structure to the Configuration file:

```
Module <Module Name>
{
      Volume      <Volume Type>
      Dimensions  <Dimension1> <Dimension2> <Dimension3> <Units>
      <Properties>
      <Child Modules>
}
```

<*Module Name*> is the name of the module. Typically the Module file name is given by <*Module Name*>.dat.

The definition of Volume, Dimensions, Properties and Child Modules are described below.

Volume and Dimensions

The volume described by the MiceModule can be one of several types. Replace <*Volume Type*> with the appropriate volume below. Cylinder, Box and Tube define cylindrical and cuboidal volumes. Polycone defines an arbitrary volume of rotation and is described in detail below. Wedge describes a wedge with a triangular projection in the y-z plane and rectangular projections in x-z and x-y planes. Quadrupole defines an aperture with four cylindrical pole tips.

In general, the physical volumes that scrape the beam are defined independently of the field maps. This is the more versatile way to do things, but there are some pitfalls which such an implementation introduces. For example, in hard-edged fields like pillboxes, tracking errors can be introduced when G4MICE steps into the field region. This can be avoided by adding windows (probably made of vacuum material) to force GEANT4 to stop tracking, make a small step over the field boundary, and then restart tracking inside the field. However, such details are left for the user to implement.

Volume	**Dimension1**	**Dimension2**	**Dimension3**
None	No dimensions required. Note cannot define daughter Modules for this volume type.		
Cylinder	Radius	Length in z	Not used (leave blank)
Box	Width in x	Height in y	Length along z
Tube	Inner Radius	Outer Radius	Length in z
Polycone	No dimensions required. Volume defined from external file.		
Wedge	See documentation below.		
Quadrupole	No dimensions required. Dimensions defined from module properties.		
Multipole	No dimensions required. Dimensions defined from module properties.		
Boolean	No dimensions required. Dimensions defined from module properties.		

Properties

Many of the features of G4MICE that can be enabled in a module are described using properties. For example, properties enable the user to define detectors and fields. Properties can be either of several types: PropertyDouble, PropertyString, PropertyBool, PropertyHep3Vector or PropertyInt. A property is declared via

```
<Property Type> <Property Name> <Value> <Units if appropriate>
```

Different properties that can be enabled for Mice Modules are described elsewhere in this document. Properties of type PropertyDouble and PropertyHep3Vector can have units. Units are defined in the CLHEP library. Units are case sensitive; G4MICE will return an error message if it fails to parse units. Combinations of units such as T/m or N*m can be defined using '*' and '/' as appropriate. Properties cannot be defined more than once within the same module.

Child Modules

Child Modules are defined with a position, rotation and scale factor. This places, and rotates, the child volume and any fields present relative to the parent volume. Scale factor scales fields defined in the child module and any of its children. Scale factors are recursively multiplicative; that is the field generated by a child module will be scaled by the product of the scale factor defined in the parent module and all of its parents.

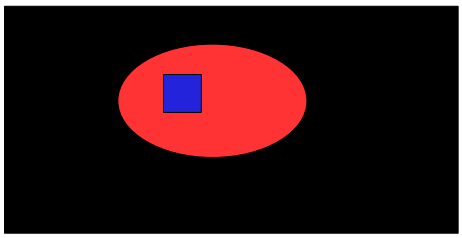
The child module definition looks like:

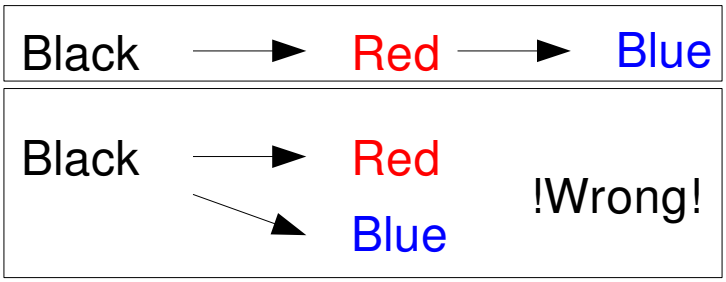
```
Module <Module File Name>
{
      Position    <x position> <y position> <z position> <Units>
      Rotation    <x rotation> <y rotation> <z rotation> <Units>
      ScaleFactor <Value>
      <Define volume, dimensions and properties for this instance only>
}
```

<Module File Name> is defined relative to the folder *${MICEFILES}/Models/Modules/*. The position and rotation default to 0 0 0 and the scale factor defaults to 1.

- Volume, Dimension and Properties of the child module can be defined at the level of the parent; in this case these values will be used only for this instance of the module.
- If no file can be found, G4MICE will press on regardless, attempting to build a geometry using the information defined in the parent volume.

Module Hierarchy and GEANT4 Physical Volumes





G4MICE enables users to place arbitrary physical volumes in a GEANT4 geometry. The formatting of G4MICE is such that users are encouraged to use the GEANT4 tree structure for placing physical volumes. This is a double-edged sword, in that it provides users with a convenient interface for building geometries, but it is not a terribly safe interface.

Consider the cartoon of physical volumes shown above. Here there is a blue volume sitting inside a red volume sitting inside the black world volume. For the volumes to be represented properly, the module that represents the blue volume MUST be a child of the module that represents the red volume. The module that represents the red volume MUST, in turn, be a child of the module that represents the black volume, in this case the Configuration file.

What would happen if we placed the blue volume directly into the Black volume, i.e. the Configuration file? GEANT4 would silently ignore the blue volume, or the red volume, depending on the order in which they are added into the GEANT4 geometry. What would happen if we placed the blue volume overlapping the red and black volumes? The behaviour of GEANT4 is not clear in this case.

- Never allow a volume to overlap any part of another volume that is not it's direct parent.

It is possible to check for overlaps by setting the datacard *CheckVolumeOverlaps* to 1.

A Sample Configuration File

Below is listed a sample configuration file, which is likely to be included in the file *ExampleConfiguration.dat;* the actual name is specified by the datacard MiceModel.

```
Configuration ExampleConfiguration
{
      Dimensions 1500.0 1000.0 5000.0 cm
      PropertyString Material AIR
      Substitution   $MyRedColour 0.75
      Module BeamLine/SolMag.dat
      {
            Position 140.0 0.0 -2175.0 cm
            Rotation 0.0 30.0 0.0 degree
            ScaleFactor 1.
      }
      Module BeamLine/BendMag.dat
      {
            Position 0.0 0.0 -1935.0 cm
            Rotation 0.0 15.0 0.0 degree
            ScaleFactor 1.
      }
      Module NoFile_Box1
      {
            Volume    Box
            Dimension 1.0 1.0 1.0
            Position 0.0 0.0  200.0 cm
            Rotation 0.0 15.0 0.0 degree
            PropertyString Material Galactic
            PropertyDouble RedColour $MyRedColour
            Module NoFile_Box2
            {
                  Volume    Box
                  Dimension 0.5 0.5 0.5*3 m //z length = 0.5*3 = 1.5 m
                  Rotation  0.0 15.0 0.0 degree //Rotation relative to parent
                  PropertyString Material Galactic
                  PropertyDouble RedColour $MyRedColour
            }
      }
}
```

A Sample Child Module File

Below is listed a sample module file, which is likely to be included in the file *SolMag.dat*; the actual location is specified by the module or configuration that calls FCoil. The module contains a number of properties that define the field.

```
Module SolMag
{
      Volume Tube
      Dimensions 263.0 347.0 210.0 mm
      PropertyString Material Al
      PropertyDouble BlueColour 0.75
      PropertyDouble GreenColour 0.75
      //field
      PropertyString FieldType       Solenoid
      PropertyString FileName        focus.dat
      PropertyDouble CurrentDensity      1.
      PropertyDouble Length            210. mm
      PropertyDouble Thickness          84. mm
      PropertyDouble InnerRadius       263. mm
}
```

Geometry and Tracking Properties

Properties for various aspects of the physical and engineering model of the simulation are described below. This includes properties for sensitive detectors.

General Properties

There are a number of properties that are applicable to any MiceModule.

Property	**Type**	**Description**
Material	string	The material that the volume is made up from
Invisible	bool	Set to 1 to make the object invisible in visualisation, or 0 to make the object visible.
RedColour	double	Alter the colour of objects as they are visualised
GreenColour	double	
BlueColour	double	
G4StepMax	double	The maximum step length that Geant4 can make in the volume. Inherits values from the parent volumes.
G4TrackMax	double	The maximum track length and particle time of a track. Tracks outside this bound are killed. Inherits values from the parent volumes.
G4TimeMax	double	
G4KinMin	double	The minimum kinetic energy of a track. Tracks outside this bound are killed. Inherits values from the parent volumes.
SensitiveDetector	string	Set to the type of sensitive detector required. Possible sensitive detectors are *TOF*, *SciFi*, *CKOV*, *SpecialVirtual, Virtual, Envelope* or *EMCAL*.

Sensitive Detectors

A sensitive detector (one in which hits are recorded) can be defined by including the SensitiveDetector property. When a volume is set to be a sensitive detector G4MICE will automatically record tracks entering, exiting and crossing the volume. Details such as the energy deposited by the track are sometimes also recorded in order to enable subsequent modelling of the detector response.

Some sensitive detectors use extra properties.

Scintillating Fibre Detector (SciFi)

Cerenkov Detector (CKOV)

Time Of Flight Counter (TOF)

Special Virtual Detectors

Special virtual detectors are used to monitor tracking through a particular physical volume. Normally particle tracks are written in the global coordinate system, although an alternate coordinate system can be defined. Additional properties can be used to parameterise special virtual detectors.

Property	**Type**	**Description**
ZSegmentation	int	Set the number of segments in the detector in Z, R or ϕ. Defaults to 1.
PhiSegmentation	int	
RSegmentation	int	
SteppingThrough	bool	Set to true to record tracks stepping through, into, out of or across the volume. Defaults to true.
SteppingInto	bool	
SteppingOutOf	bool	
SteppingAcross	bool	
Station	int	Define an integer that is written to the output file to identify the station. Defaults to a unique integer identifier chosen by G4MICE, which will be different each time the same Special Virtual is placed.
LocalRefRotation	Hep3 Vector	If set, record hits relative to a reference rotation in the coordinate system of the SpecialVirtual detector.
GlobalRefRotation	Hep3 Vector	If set, record hits relative to a reference rotation in the coordinate system of the Configuration.
LocalRefPosition	Hep3 Vector	If set, record hits relative to a reference position in the coordinate system of the SpecialVirtual detector.
GlobalRefPosition	Hep3 Vector	If set, record hits relative to a reference position in the coordinate system of the Configuration.

Virtual Detectors

Virtual detectors are used to extract all particle data at a particular plane, irrespective of geometry. Virtual detectors do not need to have a physical volume. The *plane* can be a plane in z, time, proper time, or a physical plane with some arbitrary rotation and translation.

Property	**Type**	**Description**
IndependentVariable	String	• If set to *t*, particle data will be written for particles at the time defined by the *PlaneTime* property. • If set to *tau*, particle data will be written for particles at the

Property	Type	Description
		proper time defined by the PlaneTime property. • If set to *z*, particle data will be written for particles crossing the module's z-position. • If set to *u*, particle data will be written for particles crossing a plane extending in *x* and *y*.
PlaneTime	Double	If *IndependentVariable* is *t* or *tau,* particle data will be written out at this time. Mandatory if *IndependentVariable* is *t* or *tau*.
RadialExtent	Double	If set, particles outside this radius in the plane of the detector will not be recorded by the Virtual detector.
GlobalCoordinates	Bool	If set to 0, particle data is written in the coordinate system of the module. Otherwise particle data is written in global coordinates.
MultiplePasses	String	Set how the VirtualPlane handles particles that pass through more than once. If set to Ignore, particles will be ignored on second and subsequent passes. If set to SameStation, particles will be registered with the same station number. If set to NewStation, particles will be registered with a NewStation number given by the *(total number of stations) + (this plane's station number)*, i.e. a new station number appropriate for a ring geometry.
AllowBackwards	Bool	Set to true to record backwards-going particles. Default is false.

Envelope Detectors

Envelope detectors are a type of Virtual detector that take all of the properties listed under virtual detectors, above. In addition, in the optics application they can be used to interact with the beam envelope in a special way. The following properties can be defined for Envelope Detectors *in addition to* the properties specified above for virtual detectors.

The The EnvelopeOut properties are used to make output from the envelope for use in the Optics optimiser.

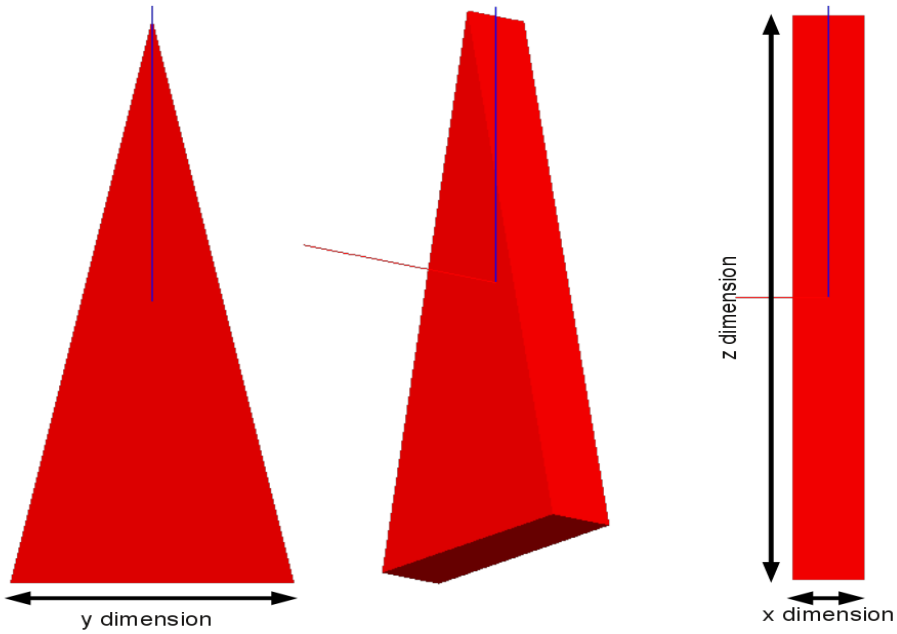
Property	Type	Description
EnvelopeOut1_Name	String	Defines the variable name that can be used as an expression substitution at the end of each iteration, typically substituted into the Score parameters in the optimiser (see optimiser, below).
EnvelopeOut1_Type	String	Defines the type of variable that will be calculated for the substitution. Options are •Mean •Covariance •Standard_Deviation •Correlation •Bunch_Parameter
EnvelopeOut1_Variable	String	Defines the variable that will be calculated for the substitution. Options are for Bunch_Parameter

Property	Type	Description
		◦*emit_6d* : 6d emittance ◦*emit_4d:* 4d emittance (in x-y space) ◦*emit_t:* 2d emittance (in time space) ◦*emit_x:* 2d emittance (in x space) ◦*emit_y:* 2d emittance (in y space) ◦*beta_4d:* 4d transverse beta function ◦*beta_t:* 2d longitudinal beta function ◦*beta_x:* 2d beta function (in(x space) ◦*beta_y:* 2d beta function (in y space) ◦*alpha_4d:* 4d transverse alpha function ◦*alpha_t:* 2d longitudinal alpha function ◦*alpha_x:* 2d alpha function (in(x space) ◦*alpha_y:* 2d alpha function (in y space) ◦*gamma_4d:* 4d transverse gamma function ◦*gamma_t:* 2d longitudinal gamma function ◦*gamma_x:* 2d gamma function (in(x space) ◦*gamma_y:* 2d gamma function (in y space) ◦*disp_x:* x-dispersion ◦*disp_y:* y-dispersion ◦*ltwiddle:* normalised angular momentum ◦*lkin:* standard angular momentum For Mean, Standard_Deviation, Covariance and Correlation, variables should be selected from the options •*x: x*-position •*y:y-position* •*t:* time •*px:* x-momentum •*py:* y-momentum •*E:* energy For Mean, a single variable should be selected and value corresponding to the reference trajectory will be returned. For Standard_Deviation, a single variable should be selected and the 1 sigma beam size will be returned. For Covariance and Correlation, two variables should be selected separated by a comma.

Unconventional Volumes

It is possible to define a number of volumes that use properties rather than the Dimensions keyword to define the volume size.

Volume Wedge



A wedge is a triangular prism as shown in the diagram. Here the blue line extends along the positive z-axis and the red line extends along the x-axis.

Property	Type	Description
Dimensions	Hep3 Vector	1.Width of the prism in x 2.Open end height of the prism in y 3.Length of the prism in z

Volume Polycone

A polycone is a volume of rotation, defined by a number of points in r and z. The volume is found by a linear interpolation of the points.

Property	Type	Description
PolyconeType	string	Set to Fill to define a solid volume of rotation. Set to Cone to define a shell volume of rotation with an inner and outer surface.
FieldMapMode	string	The name of the file that contains the polycone data.

Volume Quadrupole

Quadrupoles are defined by an empty cylinder with four further cylinders that are approximations to pole tips.

Property	Type	Description
PhysicalLength	double	The length of the quadrupole container.
QuadRadius	double	The distance from the quad centre to the outside of the quad.
PoleTipRadius	double	The distance from the quad centre to the pole tip.

Property	Type	Description
CoilRadius	double	
CoilHalfWidth	double	
BeamlineMaterial	string	The material from which the beamline volume is made.
QuadMaterial	string	The material from which the quadrupole volume is made.

Volume Multipole

Multipoles are defined by an empty box with an arbitrary number of cylinders that are approximations to pole tips. Poles are placed around the centre of the box with n-fold symmetry. Multipoles can be curved, in which case poles cannot be defined – only a curved rectangular aperture will be present.

Property	Type	Description
ApertureCurvature	double	Radius of curvature of the multipole aperture. For now curved apertures cannot have poles. Set to 0 for a straight aperture.
ApertureLength	double	Length of the multipole aperture.
NumberOfPoles	int	Number of poles.
PoleCentreRadius	double	The distance from the centre of the aperture to the centre of the cylindrical pole.
PoleTipRadius	double	The distance from the centre of the aperture to the tip of the cylindrical pole.
ApertureInnerHeight	double	The inner full height of the aperture.
ApertureInnerWidth	double	The inner full width of the aperture.
AppertureOuterHeigh t	double	The outer full height of the aperture.
ApertureOuterWidth	double	The outer full width of the aperture.

Volume Boolean

Boolean volumes enable several volumes to be combined to make very sophisticated shapes from a number of elements. Elements can be combined either by union, intersection or subtraction operations. A union creates a volume that is the sum of two elements; an intersection creates a volume that covers the region where two volumes intersect each other; and a subtraction creates a volume that contains all of one volume except the region that another volume sits in.

Boolean volumes combine volumes modelled by other MiceModules (submodules), controlled using the properties listed below. Only the volume shape is used; position, rotation and field models etc are ignored. Materials, colours and other relevant properties are all taken only from the Boolean Volume's properties.

Note that unlike in other parts of G4MICE, submodules for use in Booleans (BaseModule, BooleanModule1, BooleanModule2 ...) must be defined in a separate file, either defined in $MICEFILES/Models/Modules or in the working directory.

Also note that visualisation of boolean volumes is rather unreliable. Unfortunately this is a feature of GEANT4. An alternative technique is to use special virtual detectors to examine hits in boolean volumes.

Property	Type	Description
BaseModule	string	Name of the physical volume that the BooleanVolume is based on. This volume will be placed at (0,0,0) with no rotation, and all subsequent volumes will be added, subtracted or intersected with this one.
BooleanModule1	string	The first module to add. G4MICE will search for the MiceModule with path $MICEFILES/Models/Modules/<BooleanModule1>.
BooleanModule1Type	string	The type of boolean operation to apply, either “Union”, “Intersection” or “Subtraction”.
BooleanModule1Pos	Hep3 Vector	The position of the new volume with respect to the Base volume.
BooleanModule1Rot	Hep3 Vector	The rotation of the new volume with respect to the Base volume.
BooleanModuleN	string	Add extra modules as required. Replace “N” with the module number. N must be a continuous series incrementing by 1 for each new module. Note that the order in which modules are added is important – (A-B) ∪ C is different to A-(B ∪ C).
BooleanModuleNType	string	
BooleanModuleNPos	Hep3 Vector	
BooleanModuleNRot	Hep3 Vector	

Repeating Modules

It is possible to set up a repeating structure for e.g. a repeating magnet lattice. The RepeatModule property enables the user to specify that a particular module will be repeated a number of times, with all properties passed onto the child module, but with a new position, orientation and scale factor. Each successive repetition will be given a translation and a rotation relative to the coordinate system of the previous repetition, enabling the construction of circular and straight accelerator lattices. Additionally, successive repetitions can have fields scaled relative to previous repetitions, enabling for example alternating lattices.

Property	Type	Description
RepeatModule	bool	Set to 1 to enable repeats in this module.
NumberOfRepeats	int	Number of times the module will be repeated in addition to the initial placement.
RepeatTranslation	Hep3 Vector	Translation applied to successive repeats, applied in the coordinate system of the previous repetition.
RepeatRotation	Hep3 Vector	Rotation applied to successive repeats, applied in the coordinate system of the previous repetition.

Property	Type	Description
RepeatScaleFactor	double	ScaleFactor applied to successive repeats, applied relative to previous repetition's scale factor.

The RepeatModule2 property also enables the user to specify that a particular module will be repeated a number of times. In this case, G4MICE will set a substitution variable @RepeatNumber that holds an index between 0 and NumberOfRepeats. This can be used in an expression in to provide a versatile interface between user and accelerator lattice.

Property	Type	Description
RepeatModule2	bool	Set to 1 to enable repeats in this module.
NumberOfRepeats	int	Number of times the module will be repeated in addition to the initial placement.

Beam Definition and Beam Envelopes

The Optics application can be used to track a trajectory and associated beam envelope through the accelerator structure. Optics works by finding the Jacobian around some arbitrary trajectory using a numerical differentiation. This is used to define a linear mapping about this trajectory, which can then be used to transport the beam envelope.

The Simulation application can be used to generate a random particle beam with a Gaussian distribution and RMS parameters defined in the same way as the Optics beam envelope. Alternatively, pencil beams and beams from some input file can also be defined here.

A beam envelope is defined by a reference trajectory and a beam ellipse. The reference trajectory takes its position and direction from the position and rotation of the module. No rotation builds a reference trajectory along the z-axis. The magnitude of the momentum and the initial time of the reference trajectory is defined by properties. RF cavities are phased using the reference trajectory defined here.

The beam ellipse is represented by a matrix, which can either be set using

- Twiss-style parameters in (x,px),(y,py) and (t,E) spaces.
- Twiss-style parameters in (t,E) space and Penn-style parameters in a cylindrically symmetric (x,px,y,py) space.
- A 6x6 beam ellipse matrix where the ellipse equation is given by **X**.**T**() **M X** = 1.

The Penn ellipse matrix is given by

$$M=\begin{pmatrix} \epsilon_L mc\frac{\beta_L}{p} & -\epsilon_L mc\alpha_L & 0 & 0 & 0 & 0 \\ & \epsilon_L mc\gamma_L p & \frac{D_x}{E}V(E) & \frac{D_x^{'}}{E}V(E) & \frac{D_y}{E}V(E) & \frac{D_y^{'}}{E}V(E) \\ & & \epsilon_T mc\frac{\beta_T}{p} & -\epsilon_T mc\alpha_T & 0 & -\epsilon_T mc(\frac{q}{2}\beta_T\frac{B_z}{P}-L) \\ & & & \epsilon_T mc\gamma_T p & \epsilon_T mc(\frac{q}{2}\beta_T\frac{B_z}{P}-L) & 0 \\ & & & & \epsilon_T mc\frac{\beta_T}{p} & -\epsilon_T mc\alpha_T \\ & & & & & \epsilon_L mc\gamma_T p \end{pmatrix}$$

Here L is a normalised canonical angular momentum, q is the reference particle charge, B_z is the nominal on-axis magnetic field, p is the reference momentum, m is the reference mass, ε_T is the transverse emittance, β_T and α_T are the transverse Twiss-like functions, ε_L is the longitudinal emittance and β_L and α_L are the longitudinal Twiss-like functions. Additionally D_x, D_y, D'_x, D'_y are the dispersions and their derivatives with respect to z and V(E) is the variance of energy (given by the (2,2) term in the matrix above).

The Twiss ellipse matrix is given by

$$M = \begin{pmatrix} \epsilon_L mc\frac{\beta_L}{p} & -\epsilon_L mc\alpha_L & 0 & 0 & 0 & 0 \\ & \epsilon_L mc\gamma_L p & \frac{D_x}{E}V(E) & \frac{D_x'}{E}V(E) & \frac{D_y}{E}V(E) & \frac{D_y'}{E}V(E) \\ & & \epsilon_x mc\frac{\beta_x}{p} & -\epsilon_x mc\alpha_x & 0 & 0 \\ & & & \epsilon_x mc\gamma_x p & 0 & 0 \\ & & & & \epsilon_y mc\frac{\beta_y}{p} & -\epsilon_y mc\alpha_y \\ & & & & & \epsilon_y mc\gamma_y p \end{pmatrix}$$

Here p is the reference momentum, m is the reference mass, ε_i, β_i and α_i are the emittances and Twiss functions in the (t,E), (x,p_x) and (y,p_y) planes respectively, D_x, D_y, D'_x, D'_y are the dispersions and their derivatives with respect to z and V(E) is the variance of energy (given by the (2,2) term in the matrix above).

Property	Type	Description
EnvelopeType	string	Set to *TrackingDerivative* to evolve a beam envelope in the Optics application.
BeamType	string	Set to *Random* to generate a beam using the parameters below for the Simulation application. Set to *Pencil* to generate a pencil beam (with no random distribution). Set to *ICOOL*, *Turtle, G4MICE_PrimaryGenHit* or *G4BeamLine* to use a beam file.
Pid	int	The particle ID of particles in the envelope or beam.
Longitudinal Variable	string	Set the longitudinal variable used to define the reference trajectory momentum. Options are *Energy*, *KineticEnergy, Momentum* and *ZMomentum.*
Energy	double	Define the value of the longitudinal variable used to calculate the mean momentum and energy. The usual relationship $E^2+p^2c^2=m^2c^4$ applies. Kinetic energy E_k is related to energy E by $E_k+m=E$.
KineticEnergy	double	
Momentum	double	
ZMomentum	double	
EllipseDefinition	string	Define the beam ellipse that will be used in calculating the evolution of the Envelope, or used to generate a beam for BeamType *Random*. Options are *Twiss, Penn* and *Matrix*.
The following properties are only used if EllipseDefinition is set to Twiss		
Emittance_X	double	Emittance in each 2d subspace, (x,px), (y,py) and (t,E).
Emittance_Y	double	
Emittance_L	double	
Beta_X	double	Twiss β function in each 2d subspace, (x,px), (y,py) and (t,E).
Beta_Y	double	
Beta_L	double	
Alpha_X	double	Twiss α function in each 2d subspace, (x,px), (y,py) and (t,E).
Alpha_Y	double	
Alpha_L	double	
The following properties are only used if EllipseDefinition is set to Matrix		
Covariance(t,t)	double	Set the 6x6 matrix that will be used in the to define the beam ellipse. Covariances should be covariances of elements of the matrix (x,Px,y,Py,t,E). This must be a positive definite matrix, i.e. determinant > 0. Note that this means that at least the 6 terms on the diagonal must be defined. Other terms default to 0.
Covariance(t,E)	double	
Covariance(t,x)	double	
...	double	
Covariance(Py,Py)	double	
The following properties are only used if EllipseDefinition is set to Penn		
Emittance_T	double	Transverse emittance for the 4d (x,px,y,py) subspace.

Property	**Type**	**Description**
Emittance_L	double	Longitudinal emittance for the 2d (t,E) subspace.
Beta_T	double	Transverse beta for the 4d (x,px,y,py) subspace.
Beta_L	double	Longitudinal beta for the 2d (t,E) subspace.
Alpha_T	double	Transverse alpha for the 4d (x,px,y,py) subspace.
Alpha_L	double	Longitudinal alpha for the 2d (t,E) subspace.
Normalised AngularMomentu	double	Normalised angular momentum for the transverse phase space.
Bz	double	Nominal magnetic field on the reference particle.
The following properties are used if EllipseDefinition is set to Penn or Twiss		
Dispersion_X	double	Dispersion in x (x-energy correlation).
Dispersion_Y	double	Dispersion in y (y-energy correlation).
DispersionPrime_X	double	D' in x (Px-energy correlation).
DispersionPrime_Y	double	D' in y (Py-energy correlation).
The following properties are only relevant for generating a beam envelope		
RootOutput	string	Output file name for writing output beam envelope in ROOT binary format.
LongTextOutput	string	Output file name for writing output beam envelope in string format.
ShortTextOutput	string	Output file name for writing output beam envelope in string format. This abbreviated output omits some of the fields that are present in LongTextOutput files.
BeamOutput	string	If a BeamType is defined, this property controls the file name to which beam data is written.
Delta_t	double	Offset in time used for calculating numerical derivatives. Default is 0.1 ns.
Delta_E	double	Offset in energy used for calculating numerical derivatives. Default is 1 MeV.
Delta_x	double	Offset in x position used for calculating numerical derivatives. Default is 1 mm.
Delta_Px	double	Offset in x momentum used for calculating numerical derivatives. Default is 1 MeV/c.
Delta_y	double	Offset in y position used for calculating numerical derivatives. Default is 1 mm.
Delta_Py	double	Offset in y momentum used for calculating numerical derivatives. Default is 1 MeV/c.

Property	Type	Description
Max_Delta_t	double	Maximum offsets when polyfit algorithm is used. In some cases the offset can keep increasing without limit unless these maximum offsets are defined. Default is no limit.
Max_Delta_E	double	
Max_Delta_x	double	
Max_Delta_Px	double	
Max_Delta_y	double	
Max_Delta_Py	double	
The following properties are only relevant for generating a particle beam		
UseAsReference	Bool	If set to true and the datacard *FirstParticleIsReference* is set to 0, the first event in the Module will be used as the reference particle that sets cavity phases. This particle will then have the mean trajectory (i.e. no gaussian distribution).
BeamFile	string	If the BeamType is *ICOOL*, *Turtle, G4MICE_PrimaryGenHit* or *G4BeamLine*, this property defines the name of the file containing tracks for G4MICE.
NumberOfEvents	int	Set the maximum number of events to take from this module. If other modules are defined, G4MICE will iterate over the modules until it the datacard *numEvts* is reached or all modules have been run to *NumberOfEvents*. Default is for G4MICE to keep tracking from the first module it finds until *numEvts* is reached.

Optimiser

It is possible to define an optimiser for use in the Optics application. The optimiser enables the user to vary parameters in the MiceModule file and try to find some optimum setting. For each value of the parameters, G4MICE Optics will calculate a score; the optimiser attempts to find a minimum value for this score.

Property	Type	Description
Optimiser	string	Controls the function used for optimising. For now Minuit is the only available option.
Algorithm	string	For *Minuit* optimiser, controls the *Minuit* algorithm used. In general Simplex is a good option to use here. An alternative is Migrad. See Minuit documentation (for example at http://root.cern.ch/root/html/TMinuit.html) for further information. Minuit attempts to minimise the score function defined by the Score properties.
NumberOfTries	int	Maximum number of iterations G4MICE will make in order to find the optimum value.
StartError	double	Guess at the initial error in the score.
EndError	double	Required final error in the score for the optimisation to converge

Property	Type	Description
		successfully.
RebuildSimulation	bool	Set to False to tell G4MICE not to rebuild the simulation on each iteration. This should be used to speed up the optimiser when a parameter is used that does not change the field maps. Default is true.
Parameter1_Start	double	Seed value for the parameter, that is used in the first iteration.
Parameter1_Name	string	Name of the parameter. This name is used as an expression substitution variable elsewhere in the code and should start with @. See Expression Substitutions above for details on usage of expression substitutions.
Parameter1_Delta	double	Estimated initial error on the parameter. Default is 1.
Parameter1_Fixed	bool	Set to true to fix the parameter (so that it is excluded from the optimisation). Default is false.
Parameter1_Min	double	If required, set to the minimum value that the parameter can hold.
Parameter1_Max	double	If required, set to the maximum value that the parameter can hold.
Parameter2_Start	...	Define an arbitrary number of parameters. Parameters must be numbered consecutively, and each parameter must have at least the start value and name defined.
...	...	
Parameter2_Max	...	
Score1	double	The optimiser will attempt to optimise against a score that is calculated by summing the Score1, Score2,... parameters on each iteration.
Score2	...	
...	...	

Field Properties

Invoke a field using PropertyString FieldType <fieldtype>. The field will be placed, normally centred on the MiceModule Position and with the appropriate Rotation. Further options for each field type are specified below, and relevant datacards are also given. If a fieldtype is specified some other properties must also be specified, while others may be optional, usually taking their value from defaults specified in the datacards. Only one fieldtype can be specified per module. However, fields from multiple modules are superimposed, each transformed according to the MiceModule specification. This enables many different field configurations to be simulated using G4MICE.

To use BeamTools fields, datacard FieldMode Full must be set. This is the default.

Property	**Type**	**Description**
FieldType	string	Set the field type of the MiceModule.

FieldType CylindricalField

Sets a constant magnetic field in a cylindrical region symmetric about the z-axis of the module.

Property	**Type**	**Description**
ConstantField	Hep3 Vector	The magnetic field that will be placed in the region.
Length	double	The physical extent of the region.
Radius	double	

FieldType RectangularField

Sets a constant magnetic field in a rectangular region.

Property	**Type**	**Description**
ConstantField	Hep3 Vector	The magnetic field that will be placed in the region.
Length	double	The physical extent of the region.
Width	double	
Height	double	

FieldType Solenoid

G4MICE simulates solenoids using a series of current sheets. The field for each solenoid is written to a field map on a rectangular grid and can then be reused. The field from each current sheet is calculated using the formula for current sheets detailed in MUCOOL Note 281, *Modeling solenoids using coil, sheet and block conductors*.

Property	**Type**	**Description**
FileName	string	Read or write solenoid data to the filename. If different modules have

Property	Type	Description
		the same filename, G4MICE assumes they are the same.
FieldMapMode	string	If set to Read, G4MICE will attempt to read existing data from the FileName. If set to Write, G4MICE will generate new data and write it to the FileName. If set to Analytic, G4MICE will calculate fields directly without interpolating. If set to WriteDynamic acts as in Write except the grid extent and grid spacing at each point is chosen dynamically to some tolerance defined in the FieldTolerance property. Takes default from datacard SolDataFiles (Write).
Length	double	Coil physical parameters. Only used in Write/Analytic mode where they are mandatory.
Thickness	double	
InnerRadius	double	
CurrentDensity	double	
ZExtentFactor	double	Field map extends to length + ZExtentFactor*innerRadius in Write mode. Takes default from datacard SolzMapExtendFactor (10.). Map size is chosen dynamically in WriteDynamic mode.
RExtentFactor	double	Field map extends to radius RExtentFactor*innerRadius in Write mode. Takes default from datacard SolrMapExtendFactor (2.018...). Avoid allowing grid nodes to fall on sheets.
NumberOfZCoords	int	Number of coordinates in *z* in field map grid in Write mode. Takes default from datacard NumberNodesZGrid (500).
NumberOfRCoords	int	Number of coordintes in *r* in field map grid in Write mode. Takes default from datacard NumberNodesRGrid (100).
NumberOfSheets	int	Number of sheets used to calculate the field. Takes default from datacard DefaultNumberOfSheets (10).
FieldTolerance	double	Mandatory when FieldMapMode is WriteDynamic. If field map mode is write dynamic, this datacard controls the tolerance on errors in the field with which the field grid and the grid extent will be chosen.
Interpolation Algorithm	string	Choose the interpolation algorithm. Options are BiLinear for a linear interpolation in *r* and *z*, or LinearCubic for a linear interpolation in *r* and a cubic spline in *z*. Default is LinearCubic.
IsAmalgamated	bool	Set to 1 to add the coil to CoilAmalgamtion parent field (see below).

FieldType FieldAmalgamation

During tracking, G4MICE stores a list of fields and for each one G4MICE checks to see if tracking is performed through a particular field map's bounding box. This can be slow if a large number of fields are present. One way to speed this up is to store contributions from many coils in a single CoilAmalgamation. A CoilAmalgamation searches through child modules for solenoids with PropertyBool IsAmalgamated set to true. If it finds such a coil, it will add the field generated by the solenoid to its own field map and disable the coil.

Property	**Type**	**Description**
Length	double	The Length of the field map generated by the CoilAmalgamation.
RMax	double	The maximum radius of the field map generated by the CoilAmalgamation.
Interpolation Algorithm	string	Choose the interpolation algorithm. Options are BiLinear for a linear interpolation in *r* and *z*, or LinearCubic for a linear interpolation in *r* and a cubic spline in *z*. Default is LinearCubic.
ZStep	double	Step size of the field map generated by the CoilAmalgamation.
RStep	double	

FieldType FastSolenoid

This is an alternative field model for solenoids that uses a power law expansion of the on-axis magnetic field and its derivatives, and an exponential fall-off for the fringe field (tanh).

Property	**Type**	**Description**
PeakField	double	Nominal peak field of the solenoid.
EFoldLength	double	The fall-off length for the fringe field.
CentreLength	double	Nominal length for the peak field region.
Order	int	Order to which the field will be calculated.
FieldTolerance	double	If positive, G4MICE will abort tracking if a particle crosses through a field with estimated error > FieldTolerance. G4MICE estimates the error as the field value calculated at highest order. Default is -1 (i.e. inactive).

Phasing Models

G4MICE has a number of sophisticated models for phasing RF cavities. These powerful models can make setting up RF cavities with realistic fields both quick and easy.

When CavityMode is Unphased, G4MICE attempts to phase the cavity itself. When using CavityMode Unphased G4MICE needs to know when particles enter, cross the middle, and leave cavities. This means that:

- The cavity must sit in a rectangular or cylindrical physical volume.
- No other physical volumes may overlap or sit within the physical volume of the cavity.

If these conditions are not met the phasing algorithm is likely to fail.

To phase a cavity, G4MICE builds a volume in the centre of the cavity that is used for phasing and then fires a reference particle through the system. Stochastic processes are always disabled during this process, while mean energy loss can be disabled using the datacard ReferenceEnergyLossModel. If a reference particle crosses a plane through the centre of a cavity, it sets the phase of the cavity to the time at which the particle crosses.

The field of the cavity during phasing is controlled by the property FieldDuringPhasing. There are four modes:

- *None*: Cavity fields are disabled during phasing
- *Electrostatic*: An electrostatic field with no positional dependence given by PeakEField*sin(ReferenceParticlePhase) is enabled during phasing.
- *TimeVarying*: A standard time varying field is applied during phasing, initially with arbitrary phase relative to the reference particle. G4MICE uses a Newton-Raphson method to find the appropriate reference phase iteratively, with tolerance set by the datacard PhaseTolerance.
- *EnergyGainOptimised*: A standard time varying field is applied during phasing, initially with arbitrary phase and peak field relative to the reference particle. G4MICE uses a 2D Newton-Raphson method to find the appropriate reference phase and peak field iteratively, so that the

reference particle crosses the cavity centre with phase given by property ReferenceParticlePhase and gains energy over the whole cavity given by property EnergyGain with tolerances set by the datacards PhaseTolerance and RFDeltaEnergyTolerance.

Tracking Stability Around RF Cavities

Usually RF cavities have little or no fringe field, and this can lead to some instability in the tracking algorithms. When G4MICE makes a step into an RF cavity volume, the tracking algorithms tend to smooth out a field in a non-physical way. This can be prevented by either (i) making the step size rather small in the RF cavity or (ii) forcing G4MICE to stop tracking by adding a physical volume at the entrance of the RF cavity (a window, typically made of vacuum). Doing this should improve stability of tracking.

FieldType PillBox

Sets a PillBox field in a particular region. G4MICE represents pillboxes using a sinusoidally varying TM010 pill box field, with non-zero field vector elements given by

$$B_\phi = J_1(k_r r)\cos(\omega t)$$
$$E_z = J_0(k_r r)\cos(\omega t)$$

Here J_n are Bessel functions and k_r is a constant. See, for example, SY Lee VI.1. All other fields are 0.

Property	**Type**	**Description**
Length	double	Length of the region in which the field is present.
CavityMode	string	Phasing mode of the cavity - options are Phased, Unphased and Electrostatic.
FieldDuringPhasing	string	Controls the field during cavity phasing – options are None, Electrostatic, TimeVarying and EnergyGainOptimised.
EnergyGain	double	WhenFieldDuringPhasing is set to EnergyGainOptimised, controls the peak electric field.
Frequency	double	The cavity frequency.
PeakEField	double	The peak field of the cavity. Not used when the FieldDuringPhasing is EnergyGainOptimised.
TimeDelay	double	In Phased mode the time delay (absolute time) of the cavity.
PhasingVolume	string	Set to SpecialVirtual to make the central volume a special virtual.
ReferenceParticle Energy	double	In Electrostatic mode, G4MICE calculates the peak field and the field the reference particle sees using a combination of the reference particle energy, charge and phase. Take defaults from datacards NominalKineticEnergy and MuonCharge
ReferenceParticle Charge	double	
ReferenceParticle Phase	double	G4MICE tries to phase the field so that the reference particle crosses the cavity at ReferenceParticlePhase (units are angular). 0^o corresponds to no energy gain, 90^o corresponds to operation on-crest. Default from datacard rfAcclerationPhase.

FieldType RFFieldMap

Sets a cavity with an RF field map in a particular region. RFFieldMap uses the same phasing algorithm as described above.

Property	**Type**	**Description**
Length	double	Length of the region in which the field is present.
CavityMode	string	Phasing mode of the cavity - options are Phased and Unphased. RFFieldMaps cannot operated in Electrostatic mode.
FieldDuringPhasing	string	Controls the field during cavity phasing – options are None, Electrostatic, TimeVarying and EnergyGainOptimised.
EnergyGain	double	WhenFieldDuringPhasing is set to EnergyGainOptimised, controls the peak electric field.
Frequency	double	The cavity frequency.
PeakEField	double	The peak field of the cavity. Not used when the FieldDuringPhasing is EnergyGainOptimised.
TimeDelay	double	In Phased mode the time delay (absolute time) of the cavity.
PhasingVolume	string	Set to SpecialVirtual to make the central volume a special virtual.
ReferenceParticle Energy	double	In Electrostatic mode, G4MICE calculates the peak. field and the field the reference particle sees using a combination of the reference particle energy, charge and phase. Take defaults from datacards NominalKineticEnergy and MuonCharge
ReferenceParticle Charge	double	
ReferenceParticle Phase	double	G4MICE tries to phase the field so that the reference particle crosses the cavity at ReferenceParticlePhase (units are angular). 0^o corresponds to no energy gain, 90^o corresponds to operation on-crest. Default from datacard rfAcclerationPhase.
FileName	string	The file name of the field map file.
FileType	string	The file type of the field map. Only supported option is SuperFishSF7.

FieldType Multipole

Creates a multipole of arbitrary order. Fields are generated using either a hard edged model, with no fringe fields at all; or an Enge model similar to ZGoubi and COSY. In the former case fields are calculated using a simple polynomial expansion. In the latter case fields are calculated using the polynomial expansion with an additional exponential drop off. Fields can be superimposed onto a bent coordinate system to generate a sector multipole with arbitrary fixed radius of curvature.

Unlike most other field models in G4MICE, the zero position corresponds to the center of the entrance of the multipole; and the multipole extends in the +z direction.

The method to define end fields is described in the section EndFieldTypes below

Property	**Type**	**Description**
Pole	int	The reference pole of the magnet. 1=dipole, 2=quadrupole, 3=sextupole etc.
Magnitude	double	The field at the position (x,y,z) = (Width/2, 0, Length/2) in the multipole coordinate system. This corresponds to the position at the edge of the field map at the centre of the magnet in z. Note that this is mandatory in all cases except where CurvatureModel is MomentumBased, when the BendingAngle is used to calculate the peak (dipole) field instead.
Height	double	Height of the field region.
Width	double	Width or delta radius of the field region.
Length	double	Length of the field along the bent trajectory.
EndFieldType	string	Set to HardEdged to disable fringe fields. Set to Enge or Tanh to use those models, as described elsewhere. Default is HardEdged.
CurvatureModel	string	Choose the model for curvature. Straight forces no curvature. Constant gives a constant radius of curvature; StraightEnds gives a constant radius of curvature along the length of the multipole with straight end fields beyond this length. MomentumBased gives radius of curvature determined by a momentum and a total bending angle.
ReferenceCurvature	double	Radius of curvature of the magnet in Constant or StraightEnds mode. Set to 0 for a straight magnet. Default is 0.
ReferenceMomentum	double	Reference momentum used to calculate the radius of curvature of a dipole in MomentumBased mode. Default is 0.
BendingAngle	double	The angle used to calculate the radius of curvature of a dipole in MomentumBased mode. Note that this is mandatory in MomentumBased mode.

FieldType CombinedFunction

This creates superimposed dipole, quadrupole and sextupole fields with a common radius of curvature. The field is intended to mimic the first few terms in a multipole expansion of a field like

$$B(y=0)=B_0\left(\frac{r}{r_0}\right)^k$$

The field index is a user defined parameter, while the dipole field and radius of curvature can either be defined directly by the user or defined in terms of a reference momentum and total bending angle. Fields are calculated as in the multipole field type defined above.

Property	**Type**	**Description**
Pole	int	The reference pole of the magnet. 1=dipole, 2=quadrupole, 3=sextupole etc.
BendingField	double	The nominal dipole field B_0. Note that this is mandatory in all cases except where CurvatureModel is MomentumBased, when the BendingAngle and ReferenceMomentum is used to calculate the dipole field instead.
FieldIndex	double	The field index *k*.
Height	double	Height of the field region.
Width	double	Width or delta radius of the field region.
Length	double	Length of the field along the bent trajectory.
EndFieldType	string	Set to HardEdged to disable fringe fields. Set to Enge or Tanh to use those models, as described elsewhere. Default is HardEdged.
CurvatureModel	string	Choose the model for curvature. Straight forces no curvature. Constant gives a constant radius of curvature; StraightEnds gives a constant radius of curvature along the length of the multipole with straight end fields beyond this length. MomentumBased gives radius of curvature determined by a momentum and a total bending angle.
ReferenceCurvature	double	Radius of curvature of the magnet in Constant or StraightEnds mode. Set to 0 for a straight magnet. Default is 0.
ReferenceMomentum	double	Reference momentum used to calculate the radius of curvature of a dipole in MomentumBased mode. Default is 0.
BendingAngle	double	The angle used to calculate the radius of curvature of a dipole in MomentumBased mode. Note that this is mandatory in MomentumBased mode.

EndFieldTypes

In the absence of current sources, the magnetic field can be calculated from a scalar potential using the standard relation

$$\vec{B} = \nabla V_n$$

The scalar magnetic potential of the n[th]-order multipole field is given by

$$V_n = \sum_{q=0}^{q_m} \sum_{m=0}^{n} n!^2 \frac{G^{(2q)}(s)(r^2+y^2)^q \sin(\frac{m\pi}{2}) r^{n-m} y^m}{4^q q!(n+q)!\, m!(n-m)!}$$

where *G(s)* is either the double Enge function,

$$G(s) = E[(x-x_0)/\lambda] + E[(-x-x_0)/\lambda] - 1$$

with

$$E(s) = \frac{B_0}{R_0^n} \frac{1}{1+\exp(C_1 + C_2 s + C_3 s^2 + ...)}$$

or G(s) is the double tanh function,

$$G(s) = \tanh[(x+x_0)/\lambda]/2 + \tanh[(x-x_0)/\lambda]/2$$

and *(r, y, s)* is the position vector in the rotating coordinate system. Note that here s is the distance from the nominal end of the field map.

Property	**Type**	**Description**
EndFieldType	string	Set to HardEdged to disable fringe fields. Set to Enge or Tanh to use those models, as described elsewhere. Default is HardEdged.
The following properties are used for EndFieldType Tanh		
EndLength	double	Set the λ parameter that defines the rapidity of the field fall off.
CentreLength	double	Set the x_0 parameter that defines the length of the flat field region.
MaxEndPole	int	Set the maximum pole that will be calculated – should be larger than the multipole pole.
The following properties are used for EndFieldType Enge		
EndLength	double	Set the λ parameter that defines the rapidity of the field fall off.
CentreLength	double	Set the x_0 parameter that defines the length of the flat field region.
MaxEndPole	int	Set the maximum pole that will be calculated – should be larger than the multipole pole.
Enge1	double	Set the parameters C_i as defined in the Enge function above.
Enge2	double	
...	double	
EngeN	double	

FieldType MagneticFieldMap

Reads or writes a magnetic field map in a particular region. Two sorts of field maps are supported; either a 2d field map, in which cylindrical symmetry is assumed, or a 3d field map.

For 2d field maps, G4MICE reads or writes a file that contains information about the radial and longitudinal field components. This is intended for solenoidal field maps where only radial and longitudinal field components are present. Note that in write mode, G4MICE assumes cylindrical symmetry of the fields. In this case, G4MICE writes the x and z components of the magnetic field at points on a grid in x and z. Fields with an electric component are excluded from this summation.

For 3d field maps, G4MICE reads a file that contains the position and field in cartesian coordinates and performs a linear interpolation. This requires quite large field map files; the file size can be slightly reduced by using certain symmetries, as described below. It is currently not possible to write 3d field maps.

Property	**Type**	**Description**
FieldMapMode	string	Set to Read to read a field map; and Write to write a field map.
FileName	string	The file name that is used for reading or writing.
FileType	string	The file format. Supported options in Read mode are g4micetext, g4micebinary, g4beamline, icool, g4bl3dGrid. Only g4micetext is supported in Write mode. Default is g4micetext.
Symmetry	string	Symmetry option for g4bl3dGrid file type. Options are None, Dipole or Quadrupole. None uses the field map as is, while Dipole and Quadrupole reflect the octant between the positive x, y and z axes to give an appropriate field for a dipole or quadrupole.
ZStep	double	Step size in z and r. Mandatory in Write mode but not used in Read mode (where step size comes from the map file).
RStep	double	
ZMin	double	Upper and lower bounds in z and r. Mandatory in Write mode but not used in Read mode (where boundaries come from the map file).
ZMax	double	
RMin	double	
RMax	double	

Some file formats are described below. I am working towards making the file format more generic and hence possibly easier to use, but backwards compatibility will hopefully be maintained.

g4micetext Field Map Format

The native field map format used by g4mice in text mode is described below.

```
# GridType = Uniform N = number_rows
# Z1 = z_start Z2 = z_end dZ = z_step
# R1 = r_start R2 = r_end dR = r_step
Bz_Values   Br_Values
...         ...
<Repeat as necessary>
```

In this mode, field maps are represented by field values on a regular 2d grid that is assumed to have solenoidal symmetry, i.e. cylindrical symmetry with no tangential component.

Name	Type	Description
number_rows	double	Number of rows in the field map file.
z_start	double	Position of the grid start along the *z* axis.
z_end	double	Position of the grid end along the *z* axis.
z_step	double	Step size in *z*.
r_start	double	Position of the grid start along the *r* axis.
r_end	double	Position of the grid end along the *r* axis.
r_step	double	Step size in *r*.
Bz_Values	double	*Bz* field value.
Br_Values	double	*Br* field value.

g4bl3dGrid Field Map Format

The file format for 3d field maps is a slightly massaged version of a file format used by another code, g4beamline. In this mode, field maps are represented by field values on a regular cartesian 3d grid.

```
number_x_points number_y_points number_z_points global_scale
1 X [x_scale]
2 Y [y_scale]
3 Z [z_scale]
4 BX [bx_scale]
5 BY [by_scale]
6 BZ [bz_scale]
0
X_Values    Y_Values    Z_Values    Bx_values   By_values   Bz_values
...         ...         ...         ...         ...         ...
<Repeat as necessary>
```

where text in bold indicates a value described in the following table

Name	Type	Description
number_x_points	double	Number of points along x axis.
number_y_points	double	Number of points along y axis.
number_z_points	double	Number of points along z axis.
global_scale	double	Global scale factor applied to all x, y, z and Bx, By, Bz values.
x_scale	double	Scale factor applied to all x values.
y_scale	double	Scale factor applied to all y values.
z_scale	double	Scale factor applied to all z values.
bx_scale	double	Scale factor applied to all Bx values.
by_scale	double	Scale factor applied to all By values.
bz_scale	double	Scale factor applied to all Bz values.
X_Values	double	List (column) of each x value.
Y_Values	double	List (column) of each y value.
Z_Values	double	List (column) of each z value.
Bx_Values	double	List (column) of each Bx value.
By_Values	double	List (column) of each By value.
Bz_Values	double	List (column) of each Bz value.