cMag

0.5

Generated by Doxygen 1.8.18

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

Chapter 2

File Documentation

2.1 /Users/davidheddle/cmag/src/magfield.c File Reference

```
#include "magfield.h"
#include "magfieldutil.h"
#include "munittest.h"
#include "testdata.h"
#include <stdlib.h>
#include <math.h>
```

Functions

- void setAlgorithm (enum Algorithm algorithm)
- bool containsCylindrical (MagneticFieldPtr fieldPtr, double rho, double z)
- bool containsCartesian (MagneticFieldPtr fieldPtr, double x, double y, double z)
- void resetCell3D (Cell3DPtr cell3DPtr, double phi, double rho, double z)
- void resetCell2D (Cell2DPtr cell2DPtr, double rho, double z)
- void getFieldValue (FieldValuePtr fieldValuePtr, double x, double y, double z, MagneticFieldPtr fieldPtr)
- void getCompositeFieldValue (FieldValuePtr fieldValuePtr, double x, double y, double z, MagneticFieldPtr field1, MagneticFieldPtr field2)
- int getCompositeIndex (MagneticFieldPtr fieldPtr, int n1, int n2, int n3)
- void invertCompositeIndex (MagneticFieldPtr fieldPtr, int index, int ∗phiIndex, int ∗rhoIndex, int ∗zIndex)
- void getCoordinateIndices (MagneticFieldPtr fieldPtr, double phi, double rho, double z, int ∗nPhi, int ∗nRho, int ∗nZ)
- char ∗ nearestNeighborUnitTest ()
- char ∗ containsUnitTest ()
- char ∗ compositeIndexUnitTest ()
- FieldValuePtr getFieldAtIndex (MagneticFieldPtr fieldPtr, int compositeIndex)

Variables

- MagneticFieldPtr testFieldPtr
- enum Algorithm _algorithm = INTERPOLATION

2.1.1 Function Documentation

2.1.1.1 compositeIndexUnitTest()

```
char* compositeIndexUnitTest ( )
```

A unit test for the composite indexing

Returns

an error message if the test fails, or NULL if it passes.

2.1.1.2 containsCartesian()

```
bool containsCartesian (
            MagneticFieldPtr fieldPtr,
            double x,
            double y,
            double z )
```

This checks whether the given point is within the boundary of the field. This is so the methods that retrieve a field value can short-circuit to zero. NOTE: this assumes, as is the case at the time of writing, that the CLAS12 fields have grids in cylindrical coordinates and length units of cm.

Parameters

fieldPtr	
x	the x coordinate in cm.
y	the y coordinate in cm.
z	the z coordinate in cm.

Returns

true if the point is within the boundary of the field.

2.1.1.3 containsCylindrical()

```
bool containsCylindrical (
            MagneticFieldPtr fieldPtr,
            double rho,
            double z )
```

This checks whether the given point is within the boundary of the field. This is so the methods that retrieve a field value can short-circuit to zero. Note ther is no phi parameter, because all values of phi are "contained." NOTE: this assumes, as is the case at the time of writing, that the CLAS12 fields have grids in cylindrical coordinates and length units of cm.

Parameters

fieldPtr	
rho	the rho coordinate in cm.
z	the z coordinate in cm.

Returns

true if the point is within the boundary of the field.

2.1.1.4 containsUnitTest()

```
char* containsUnitTest ( )
```

A unit test for checking the boundary contains check.

Returns

an error message if the test fails, or NULL if it passes.

2.1.1.5 getCompositeFieldValue()

```
void getCompositeFieldValue (
            FieldValuePtr fieldValuePtr,
            double x,
            double y,
            double z,
            MagneticFieldPtr field1,
            MagneticFieldPtr field2 )
```

Obtain the combined value of two fields. The field is obtained by tri-linear interpolation or nearest neighbor, depending on settings.

Parameters

fieldValuePtr	should be a valid pointer to a FieldValue. Upon return it will hold the value of the field, in kG, in Cartesian components Bx, By, BZ, regardless of the field coordinate system of the maps, obtained from all the field maps that it is given in the variable length argument list. For example, if torus and solenoid point to fields, then one can obtain the combined field at (x, y, z) by calling getCompositeFieldValue(fieldVal, x, y, x, torus, solenoid).
x	the x coordinate in cm.
y	the y coordinate in cm.
z	the z coordinate in cm.
field1	the first field.
field2	the second field.

2.1.1.6 getCompositeIndex()

```
int getCompositeIndex (
            MagneticFieldPtr fieldPtr,
            int n1,
            int n2,
            int n3 )
```

Get the composite index into the 1D data array holding the field data from the coordinate indices.

Parameters

fieldPtr	the pointer to the field map
n1	the index for the first coordinate.
n2	the index for the second coordinate.
n3	the index for the third coordinate.

Returns

the composite index into the 1D data array.

2.1.1.7 getCoordinateIndices()

```
void getCoordinateIndices (
            MagneticFieldPtr fieldPtr,
            double phi,
            double rho,
            double z,
            int * nPhi,
            int * nRho,
            int * nZ )
```

Get the coordinate indices from coordinates.

Parameters

fieldPtr	the field ptr.
phi	the value of the phi coordinate.
rho	the value of the rho coordinate.
z	the value of the z coordinate.
nPhi	upon return, the phi index.
nRho	upon return, the rho index.
nZ	upon return, the z index.

2.1.1.8 getFieldAtIndex()

```
FieldValuePtr getFieldAtIndex (
            MagneticFieldPtr fieldPtr,
            int compositeIndex )
```

Get the field at a given composite index.

Parameters

fieldPtr	a pointer to the field.
compositeIndex	the composite index.

Returns

a pointer to the field value, or NULL if out of range.

2.1.1.9 getFieldValue()

```
void getFieldValue (
            FieldValuePtr fieldValuePtr,
            double x,
            double y,
            double z,
            MagneticFieldPtr fieldPtr )
```

Obtain the value of the field by tri-linear interpolation or nearest neighbor, depending on settings.

Parameters

fieldValuePtr	should be a valid pointer to a FieldValue. Upon return it will hold the value of the field in kG, in Cartesian components Bx, By, BZ, regardless of the field coordinate system of the map.
x	the x coordinate in cm.
y	the y coordinate in cm.
z	the z coordinate in cm.
fieldPtr	a pointer to the field map.

2.1.1.10 invertCompositeIndex()

```
void invertCompositeIndex (
            MagneticFieldPtr fieldPtr,
            int index,
            int * phiIndex,
            int * rhoIndex,
            int * zIndex )
```

This inverts the "composite" index of the 1D data array holding the field data into an index for each coordinate. This can be used, for example, to find the grid coordinate values and field components.

Parameters

fieldPtr	the pointer to the field map
index	the composite index into the 1D data array. Upon return, coordinate indices of -1 indicate error.
phiIndex	will hold the index for the first coordinate.
rhoIndex	will hold the index for the second coordinate.
zIndex	will hold the index for the third coordinate.

2.1.1.11 nearestNeighborUnitTest()

```
char* nearestNeighborUnitTest ( )
```

A unit test for the test field.

Returns

an error message if the test fails, or NULL if it passes.

2.1.1.12 resetCell2D()

```
void resetCell2D (
            Cell2DPtr cell2DPtr,
            double rho,
            double z )
```

Reset the cell based on a new location. If the location is contained by the cell, then we can use some cached values, such as the neighbors. If it isn't, we have to recalculate all.

Parameters

cell2DPtr	a pointer to the 2D cell.
rho	the transverse coordinate, in cm.
z	the z coordinate, in cm.

2.1.1.13 resetCell3D()

```
void resetCell3D (
            Cell3DPtr cell3DPtr,
            double phi,
            double rho,
            double z )
```

Reset the cell based on a new location. If the location is contained by the cell, then we can use some cached values, such as the neighbors. If it isn't, we have to recalculate all.

Parameters

cell3DPtr	a pointer to the 3D cell.
phi	the azimuthal angle, in degrees
rho	the transverse coordinate, in cm.
z	the z coordinate, in cm.

2.1.1.14 setAlgorithm()

```
void setAlgorithm (
            enum Algorithm algorithm )
```

Set the global option for the algorithm used to extract field values.

Parameters

algorithm	it can either be

2.1.2 Variable Documentation

2.1.2.1 _algorithm

```
enum Algorithm _algorithm = INTERPOLATION
```

2.1.2.2 testFieldPtr

```
MagneticFieldPtr testFieldPtr
```

2.2 /Users/davidheddle/cmag/src/magfielddraw.c File Reference

```
#include "magfielddraw.h"
#include "svg.h"
#include "mapcolor.h"
#include "magfieldutil.h"
```

Functions

- void createSVGImageFixedZ (char ∗path, double z, MagneticFieldPtr torus, MagneticFieldPtr solenoid)
- void createSVGImageFixedPhi (char ∗path, double phi, MagneticFieldPtr torus, MagneticFieldPtr solenoid)

2.2.1 Function Documentation

2.2.1.1 createSVGImageFixedPhi()

```
void createSVGImageFixedPhi (
            char * path,
            double phi,
            MagneticFieldPtr torus,
            MagneticFieldPtr solenoid )
```

Create an SVG image of the fields at a fixed value of phi.

Parameters

path	the path to the svg file.
phi	the fixed value of phi in degrees. For the canonical sector 1 midplane, use phi = 0;
fieldPtr	torus the torus field (can be NULL).
fieldPtr	torus the solenoid field (can be NULL).

2.2.1.2 createSVGImageFixedZ()

```
void createSVGImageFixedZ (
            char * path,
            double z,
            MagneticFieldPtr torus,
            MagneticFieldPtr solenoid )
```

Create an SVG image of the fields at a fixed value of z.

Parameters

path	the path to the svg file.
z	the fixed value of z in cm.
fieldPtr	torus the torus field (can be NULL).
fieldPtr	torus the solenoid field (can be NULL).

2.3 /Users/davidheddle/cmag/src/magfieldio.c File Reference

```
#include "magfieldio.h"
#include "magfieldutil.h"
#include <stdlib.h>
#include <time.h>
#include <math.h>
```

Functions

- MagneticFieldPtr initializeTorus (const char ∗torusPath)
- MagneticFieldPtr initializeSolenoid (const char ∗solenoidPath)
- void createCell3D (MagneticFieldPtr fieldPtr)
- void createCell2D (MagneticFieldPtr fieldPtr)
- void freeCell3D (Cell3DPtr cell3DPtr)
- void freeCell2D (Cell2DPtr cell2DPtr)

2.3.1 Function Documentation

2.3.1.1 createCell2D()

```
void createCell2D (
            MagneticFieldPtr fieldPtr )
```

Create a 2D cell, which is used by the solenoid, since the lack of phi dependence renders the solenoidal field effectively 2D. Note that nothing is returned, the field's 2D cell pointer is made to point at the cell, and the cell is given a reference to the field.

Parameters

fieldPtr	a pointer to the solenoid field.

2.3.1.2 createCell3D()

```
void createCell3D (
            MagneticFieldPtr fieldPtr )
```

Create a 3D cell, which is used by the torus. Note that nothing is returned, the field's 2D cell pointer is made to point at the cell, and the cell is given a reference to the field.

Parameters

fieldPtr	a pointer to the solenoid field.

2.3.1.3 freeCell2D()

```
void freeCell2D (
            Cell2DPtr cell2DPtr )
```

Free the memory associated with a 2D cell.

Parameters

cell2DPtr	a pointer to the cell.

2.3.1.4 freeCell3D()

```
void freeCell3D (
            Cell3DPtr cell3DPtr )
```

Free the memory associated with a 3D cell.

Parameters

cell3DPtr	a pointer to the cell.

2.3.1.5 initializeSolenoid()

```
MagneticFieldPtr initializeSolenoid (
            const char * solenoidPath )
```

Initialize the solenoid field.

Parameters

solenoidPath	a path to a solenoid field map. If you want to use environment variables, pass NULL in this parameter, in which case the code make two attempts two attempts at finding the field. The first will be to try the a path specified by the COAT_MAGFIELD_SOLENOIDMAP environment variable. If that fails, it will then check SOLENOIDMAP.

Returns

a valid field pointer on success, NULL on failure.

2.3.1.6 initializeTorus()

```
MagneticFieldPtr initializeTorus (
            const char * torusPath )
```

Initialize the torus field.

Parameters

torusPath	a path to a torus field map. If you want to use environment variables, pass NULL in this parameter, in which case the code make two attempts two attempts at finding the field. The first will be to try the a path specified by the COAT_MAGFIELD_TORUSMAP environment variable. If that fails, it will then check TORUSMAP.

Returns

a valid field pointer on success, NULL on failure.

2.4 /Users/davidheddle/cmag/src/magfieldutil.c File Reference

```
#include "magfield.h"
#include "magfieldio.h"
#include "magfieldutil.h"
#include "munittest.h"
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
```

Functions

- double toDegrees (double angRad)
- double toRadians (double angDeg)
- bool sameNumber (double v1, double v2)
- void cartesianToCylindrical (const double x, const double y, double *phi, double *rho)
- void cylindricalToCartesian (double *x, double *y, const double phi, const double rho)
- void normalizeAngle (double *angDeg)
- double fieldMagnitude (FieldValue *fvPtr)
- double relativePhi (double absolutePhi)
- int getSector (double phi)
- void printFieldSummary (MagneticFieldPtr fieldPtr, FILE *stream)
- const char * fieldUnits (MagneticFieldPtr fieldPtr)
- const char * lengthUnits (MagneticFieldPtr fieldPtr)
- void printFieldValue (FieldValuePtr fvPtr, FILE *stream)
- MagneticFieldPtr createFieldMap ()
- void freeFieldMap (MagneticFieldPtr fieldPtr)
- void stringCopy (char **dest, const char *src)
- int randomInt (int minVal, int maxVal)
- int sign (double x)
- double randomDouble (double minVal, double maxVal)

- char * conversionUnitTest ()
- char * randomUnitTest ()
- int descBinarySearch (double *array, int lower, int upper, double x)
- int binarySearch (double *array, int lower, int upper, double x)
- int cmpfunc (const void *a, const void *b)
- void sortArray (double *array, int length)
- char * binarySearchUnitTest ()

Variables

- double const TINY = 1.0e-8
- int mtests_run = 0
- const double PIOVER180 = M_PI/180.
- const char * csLabels [ ] = { "cylindrical", "Cartesian" }
- const char * lengthUnitLabels [ ] = { "cm", "m" }
- const char * angleUnitLabels [ ] = { "degrees", "radians" }
- const char * fieldUnitLabels [ ] = { "kG", "G", "T" }

2.4.1 Function Documentation

2.4.1.1 binarySearch()

```
int binarySearch (
            double * array,
            int lower,
            int upper,
            double x )
```

A binary search through a sorted array.

Parameters

array	an array sorted (ascending).
lower	pass 0 to this, it is here for recursion
upper	pass the length of the array - 1.
x	pass the value to search for.

Returns

-1 if the value is out of range. othewise return index [0..length-2] such that array[index] $<$ value $<$ array[index+1];

2.4.1.2 binarySearchUnitTest()

```
char* binarySearchUnitTest ( )
```

A unit test for the binary search

Returns

an error message if the test fails, or NULL if it passes.

2.4.1.3 cartesianToCylindrical()

```
void cartesianToCylindrical (
            const double x,
            const double y,
            double * phi,
            double * rho )
```

Converts 2D Cartesian coordinates to polar. This is used because the two coordinate systems we use are Cartesian and cylindrical, whose 3D transformations are equivalent to 2D Cartesian to polar. Note the azimuthal angle output is in degrees, not radians.

Parameters

x	the x component.
y	the y component.
phi	will hold the angle, in degrees, in the range [0, 360).
rho	the longitudinal component.

2.4.1.4 cmpfunc()

```
int cmpfunc (
            const void * a,
            const void * b )
```

A comparator for qsort

Parameters

a	one value
b	another value

Returns

2.4.1.5 conversionUnitTest()

```
char* conversionUnitTest ( )
```

A unit test for the conversions

Returns

an error message if the test fails, or NULL if it passes.

2.4.1.6 createFieldMap()

```
MagneticFieldPtr createFieldMap ( )
```

Allocate a field map with no content.

Returns

a pointer to an empty field map structure.

2.4.1.7 cylindricalToCartesian()

```
void cylindricalToCartesian (
            double * x,
            double * y,
            const double phi,
            const double rho )
```

Converts polar coordinates to 2D Cartesian. This is used because the two coordinate systems we use are Cartesian and cylindrical, whose 3D transformations are equivalent to 2D polar to Cartesian. Note the azimuthal angle input is in degrees, not radians.

Parameters

x	will hold the x component.
y	will hold the y component.
phi	the azimuthal angle, in degrees.
rho	the longitudinal component.

2.4.1.8 descBinarySearch()

```
int descBinarySearch (
            double * array,
            int lower,
            int upper,
            double x )
```

A binary search through a sorted array.

Parameters

array	an array sorted (descending).
lower	pass 0 to this, it is here for recursion
upper	pass the length of the array - 1.
x	pass the value to search for.

Returns

-1 if the value is out of range. othewise return index [0..length-2] such that array[index] $<$ value $<$ array[index+1];

2.4.1.9 fieldMagnitude()

```
double fieldMagnitude (
            FieldValue * fvPtr )
```

Get the magnitude of a field value.

Parameters

fvPtr	a pointer to a field value

Returns

return: the magnitude of a field value.

2.4.1.10 fieldUnits()

```
const char* fieldUnits (
            MagneticFieldPtr fieldPtr )
```

Get the field units of the magnetic field

Parameters

fieldPtr	a pointer to the field.

Returns

a string representing the field units, e.g. "kG".

2.4.1.11 freeFieldMap()

```
void freeFieldMap (
            MagneticFieldPtr fieldPtr )
```

Free the memory associated with a field map.

Parameters

fieldPtr	a pointer to the field

2.4.1.12 getSector()

```
int getSector (
            double phi )
```

Obtain the CLAS12 sector from the phi value

Parameters

phi	the azimuthal angle in degrees

Returns

the sector [1..6].

2.4.1.13 lengthUnits()

```
const char* lengthUnits (
            MagneticFieldPtr fieldPtr )
```

Get the length units of the magnetic field

Parameters

fieldPtr	a pointer to the field.

Returns

a string representing the length units, e.g. "cm".

2.4.1.14 normalizeAngle()

```
void normalizeAngle (
            double * angDeg )
```

This will normalize an angle in degrees. We use for normaliztion that the angle should be in the range [0, 360).

Parameters

angDeg	the angle in degrees. It will be normalized.

2.4.1.15 printFieldSummary()

```
void printFieldSummary (
            MagneticFieldPtr fieldPtr,
            FILE * stream )
```

Print a summary of the map for diagnostics and debugging.

Parameters

fieldPtr	the pointer to the map.
stream	a file stream, e.g. stdout.

2.4.1.16 printFieldValue()

```
void printFieldValue (
            FieldValuePtr fvPtr,
            FILE * stream )
```

Print the components and magnitude of field value.

Parameters

fvPtr	a pointer to the field value
stream	a file stream, e.g. stdout.

2.4.1.17 randomDouble()

```
double randomDouble (
            double minVal,
            double maxVal )
```

Obtain a random double in the range[minVal, maxVal]. Used for testing.

Parameters

minVal	the minimum value
maxVal	the maximum Value;

Returns

2.4.1.18 randomInt()

```
int randomInt (
            int minVal,
            int maxVal )
```

Obtain a random int in an inclusive range[minVal, maxVal]. Used for testing.

Parameters

minVal	the minimum value
maxVal	the maximum Value;

Returns

2.4.1.19 randomUnitTest()

```
char* randomUnitTest ( )
```

A unit test for the random number generator

Returns

an error message if the test fails, or NULL if it passes.

2.4.1.20 relativePhi()

```
double relativePhi (
            double absolutePhi )
```

Must deal with the fact that for a symmetric torus we only have the field between 0 and 30 degrees.

Parameters

absolutePhi	the absolut value of phi in degrees.

Returns

a phi relative to the midplabe, [-30, 30]

2.4.1.21 sameNumber()

```
bool sameNumber (
            double v1,
            double v2 )
```

The usual test to see if two floating point numbers are close enough to be considered equal. Test accuracy depends on the global const TINY, set to 1.0e-10.

Parameters

v1	one value.
v2	another value.

Returns

true if the values are close enough to be considered equal.

2.4.1.22 sign()

```
int sign (
            double x )
```

Sign function

Parameters

x	the value to check

Returns

-1, 0 or 1

2.4.1.23 sortArray()

```
void sortArray (
            double * array,
            int length )
```

Use built in quick sort to sort a double array in ascending order

Parameters

array	the array to sort
length	the length of the array

2.4.1.24 stringCopy()

```
void stringCopy (
            char ** dest,
            const char * src )
```

Copy a string and create the pointer

Parameters

dest	on input a pointer to an unallocated string. On output the string will be allocated and contain a copy of src.
src	the string to be copied.

2.4.1.25 toDegrees()

```
double toDegrees (
            double angRad )
```

Convert an angle from radians to degrees.

Parameters

angRad	the angle in radians.

Returns

the angle in degrees.

2.4.1.26 toRadians()

```
double toRadians (
            double angDeg )
```

Convert an angle from degrees to radians.

Parameters

angDeg	the angle in degrees.

Returns

the angle in radians.

2.4.2 Variable Documentation

2.4.2.1 angleUnitLabels

```
const char* angleUnitLabels[] = { "degrees", "radians" }
```

2.4.2.2 csLabels

```
const char* csLabels[] = { "cylindrical", "Cartesian" }
```

2.4.2.3 fieldUnitLabels

```
const char* fieldUnitLabels[] = { "kG", "G", "T" }
```

2.4.2.4 lengthUnitLabels

```
const char* lengthUnitLabels[] = { "cm", "m" }
```

2.4.2.5 mtests_run

```
int mtests_run = 0
```

2.4.2.6 PIOVER180

```
const double PIOVER180 = M_PI/180.
```

2.4.2.7 TINY

```
double const TINY = 1.0e-8
```

2.5 /Users/davidheddle/cmag/src/maggrid.c File Reference

```
#include "maggrid.h"
#include "magfieldutil.h"
#include "munittest.h"
#include <stdlib.h>
#include <math.h>
#include <time.h>
```

Functions

- GridPtr createGrid (const char *name, const double minVal, const double maxVal, const unsigned int num)
- char * gridStr (GridPtr gridPtr)
- int getIndex (const GridPtr gridPtr, const double val)
- double valueAtIndex (GridPtr gridPtr, int index)
- char * gridUnitTest ()

2.5.1 Function Documentation

2.5.1.1 createGrid()

```
GridPtr createGrid (
            const char * name,
            const double minVal,
            const double maxVal,
            const unsigned int num )
```

Create a uniform (equally spaced) coordinate coordinate grid.

Parameters

the	name of the coordinate, e.g. "phi".
minVal	the minimum value of the grid.
maxVal	the maximum value of the grid.
num	the number of points on the grid, including the ends.

Returns

a pointer to the coordinate grid.

2.5.1.2 getIndex()

```
int getIndex (
            const GridPtr gridPtr,
            const double val )
```

Get the index of a value.

Parameters

gridPtr	the pointer to the coordinate grid.
val	the value to index.

Returns

the index, [0..N-2] where, or -1 if out of bounds. The value should be bounded by values[index] and values[index+1].

2.5.1.3 gridStr()

```
char* gridStr (
            GridPtr gridPtr )
```

Get a string representation of the grid.

Parameters

gridPtr	the pointer to the coordinate grid.

Returns

a string representation of the grid.

2.5.1.4 gridUnitTest()

```
char* gridUnitTest ( )
```

A unit test for the coordinate grid code.

Returns

an error message if the test fails, or NULL if it passes.

2.5.1.5 valueAtIndex()

```
double valueAtIndex (
            GridPtr gridPtr,
            int index )
```

Get the value of the grid at a given index

Parameters

gridPtr	the pointer to the grid
index	the index

Returns

the value of the grid at the given index, or NAN if the index is out of range

2.6 /Users/davidheddle/cmag/src/main.c File Reference

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "magfield.h"
#include "magfieldio.h"
#include "munittest.h"
#include "magfieldutil.h"
#include "magfielddraw.h"
```

Functions

- int main (int argc, const char ∗argv[ ])

2.6.1 Function Documentation

2.6.1.1 main()

```
int main (
            int argc,
            const char * argv[] )
```

The main method of the test application.

Parameters

argc	the number of arguments
argv	the command line argument. Only one is processed, the path to the directory containing the magnetic fields. If that argument is missing, it will look in /Users/davidheddle/magfield.

Returns

0 on successful completion, 1 if any error occurred.

2.7 /Users/davidheddle/cmag/src/mapcolor.c File Reference

```
#include "mapcolor.h"
#include "magfieldutil.h"
#include <stdlib.h>
```

Functions

- char ∗ getColor (ColorMapPtr cmapPtr, double value)
- ColorMapPtr defaultColorMap ()
- void colorToHex (char ∗colorStr, int r, int g, int b)

2.7.1 Function Documentation

2.7.1.1 colorToHex()

```
void colorToHex (
            char * colorStr,
            int r,
            int g,
            int b )
```

Get the hex color string from color components

Parameters

colorStr	must be at last 8 characters
r	the red component [0..255]
g	the green component [0..255]
b	the blue component [0..255]

2.7.1.2 defaultColorMap()

```
ColorMapPtr defaultColorMap ( )
```

Get the default color map optimized for displaying torus and solenoid

Returns

he default color map.

2.7.1.3 getColor()

```
char* getColor (
            ColorMapPtr cmapPtr,
            double value )
```

Get a color from a color map.

Parameters

cmapPtr	a valid pointer to a color map.
value	the value to convert into a color.

Returns

the color in "#rrggbb" format.

2.8 /Users/davidheddle/cmag/src/svg.c File Reference

```
#include "svg.h"
```

Functions

- svg ∗ svgStart (char ∗path, int width, int height)
- void svgEnd (svg ∗psvg)
- void svgCircle (svg ∗psvg, char ∗stroke, int strokewidth, char ∗fill, int r, int cx, int cy)
- void svgLine (svg ∗psvg, char ∗stroke, int strokewidth, int x1, int y1, int x2, int y2)
- void svgRectangle (svg ∗psvg, int width, int height, int x, int y, char ∗fill, char ∗stroke, int strokewidth, int radiusx, int radiusy)
- void svgFill (svg ∗psvg, char ∗fill)
- void svgText (svg ∗psvg, int x, int y, char ∗fontfamily, int fontsize, char ∗fill, char ∗stroke, char ∗text)
- void svgRotatedText (svg ∗psvg, int x, int y, char ∗fontfamily, int fontsize, char ∗fill, char ∗stroke, int angle, char ∗text)
- void svgEllipse (svg ∗psvg, int cx, int cy, int rx, int ry, char ∗fill, char ∗stroke, int strokewidth)

2.8.1 Function Documentation

2.8.1.1 svgCircle()

```
void svgCircle (
            svg * psvg,
            char * stroke,
            int strokewidth,
            char * fill,
            int r,
            int cx,
            int cy )
```

Draw a circle. All units are pixels.

Parameters

psvg	pointer to the svg information.
stroke	the outline color, usually in "#rrggbb" format.
strokewidth	border line width.
fill	the fill color, usually in "#rrggbb" format.
r	the radius.
cx	the x center
cy	the y center.

2.8.1.2 svgEllipse()

```
void svgEllipse (
            svg * psvg,
            int cx,
            int cy,
            int rx,
            int ry,
            char * fill,
            char * stroke,
            int strokewidth )
```

Draw an ellipse. All units are pixels.

Parameters

psvg	pointer to the svg information.
cx	the horizontal center.
cy	the vertical center.
rx	the horizontal radius.
ry	the vertical radius.
fill	the fill color, usually in "#rrggbb" format.
stroke	the outline color, usually in "#rrggbb" format.
strokewidth	the width of the outline.

2.8.1.3 svgEnd()

```
void svgEnd (
            svg * psvg )
```

Finalize the svg file and free all space.

Parameters

psvg	pointer to the svg information.

2.8.1.4 svgFill()

```
void svgFill (
            svg * psvg,
            char * fill )
```

Fill the whole image area.

Parameters

psvg	pointer to the svg information.
fill	the fill color, usually in "#rrggbb" format.

2.8.1.5 svgLine()

```
void svgLine (
            svg * psvg,
            char * stroke,
            int strokewidth,
            int x1,
            int y1,
            int x2,
            int y2 )
```

Draw a line. All units are pixels.

Parameters

psvg	pointer to the svg information.
stroke	the line color, usually in "#rrggbb" format.
strokewidth	the width of the line.
x1	x coordinate of start.
y1	y coordinate of start.
x2	x coordinate of end.
y2	y coordinate of end.

2.8.1.6 svgRectangle()

```
void svgRectangle (
            svg * psvg,
            int width,
            int height,
            int x,
            int y,
            char * fill,
            char * stroke,
            int strokewidth,
            int radiusx,
            int radiusy )
```

Draw a rectangle. All units are pixels.

Parameters

psvg	pointer to the svg information.
width	the width of the rectangle.
height	the height of the rectangle.
x	the left of the rectangle.
y	th top of the rectangle.
fill	the fill color, usually in "#rrggbb" format.
stroke	the outline color, usually in "#rrggbb" format.
strokewidth	the width of the outline.
radiusx	for rounding the corners.
radiusy	for rounding the corners.

2.8.1.7 svgRotatedText()

```
void svgRotatedText (
            svg * psvg,
            int x,
            int y,
            char * fontfamily,
            int fontsize,
            char * fill,
            char * stroke,
            int angle,
            char * text )
```

Draw some text. All units are pixels.

Parameters

psvg	pointer to the svg information.

Parameters

x	the baseline horizontal start
y	the baseline vertical start
fontfamily	the font family.
fontsize	the font size.
fill	the fill color, usually in "#rrggbb" format.
stroke	the outline color, usually in "#rrggbb" format.
angle	the rotation angle in degrees.
text	the text to draw.

2.8.1.8 svgStart()

```
svg* svgStart (
            char * path,
            int width,
            int height )
```

Initialize the svg file creation process.

Parameters

path	a path to the ouyput file.
width	the width of the image in pixels.
height	the height of the image in pixels.

Returns

a pointer to the svg object.

2.8.1.9 svgText()

```
void svgText (
            svg * psvg,
            int x,
            int y,
            char * fontfamily,
            int fontsize,
            char * fill,
            char * stroke,
            char * text )
```

Draw some text. All units are pixels.

Parameters

psvg	pointer to the svg information.
x	the baseline horizontal start
y	the baseline vertical start
fontfamily	the font family.
fontsize	the font size.
fill	the fill color, usually in "#rrggbb" format.
stroke	the outline color, usually in "#rrggbb" format.
text	the text to draw.

2.9 /Users/davidheddle/cmag/src/testdata.c File Reference

Variables

- double torusNN [33][6]
- double solenoidNN [34][6]

2.9.1 Variable Documentation

2.9.1.1 solenoidNN

```
double solenoidNN[34][6]
```

2.9.1.2 torusNN

```
double torusNN[33][6]
```