MASINT API

Mathmatical Vector Library C API

API Name

libMVector.a - Mathematical vector library

Synopsis

C Application programming interface for the Mathematical vector library. The library provides the user with the ability to manipulate vectors and performance calculations on then as if they were a data type.

Include(s)

```
#include "MVector.h"
```

The location of the include files can be found by running :

```
MVector-config --cflags.
```

For example :

```
gcc -c file.c `MVector-config --cflags`
```

Library(s)

libMVector.h

The location of the library can be found by running:

```
Mvector-config --libs
```

For example:

```
gcc -o prog code.c ``MVector-config --cflags` `MVector-config --libs`
```

It should be noted that libMVector.a is a C++ library. C stubs have been provided to use the library from within a C programme. However it needs to be linked with -lstdc++.

Structures and Data Types

The principal structure of a Vector is given by :

```
typedef struct {
    double x;
    double y;
    double z;
} SPVector;
```

And

```
typedef SPVector VectorH;
```

VectorH should be used as the main C language handle for a Vector data type. SPVector is provided for backwards compatibility with the mtlibrary.

Overview of Functions

The following functions are provided for the C API:

```
VectorH vectCreate(double a, double b, double c);
int vectEquals(VectorH v1, VectorH v2);
int vectNotEqual(VectorH v1, VectorH v2);
VectorH vectAdd(VectorH v1, VectorH v2);
VectorH vectAddScaler(VectorH v1, double scaler);
VectorH vectSub(VectorH v1, VectorH v2);
VectorH vectSubScaler(VectorH v1, double scaler);
VectorH vectMult(VectorH v1, double scaler);
VectorH vectDiv(VectorH v1, double scaler);
VectorH vectMinus(VectorH v);
double vectMag(VectorH v);
VectorH vectNorm(VectorH v);
double vectDot(VectorH v1, VectorH v2);
VectorH vectCross(VectorH v1, VectorH v2);
VectorH vectRotatex(VectorH v, double angle);
VectorH vectRotatey(VectorH v, double angle);
VectorH vectRotatez(VectorH v, double angle);
VectorH vectRotateAxis(VectorH v, VectorH axis, double angle);
```

Syntax / Prototype

```
VectorH vectCreate(double a, double b, double c);
```

Description

Create a vector from arguments a,b and c

Arguments

a,b and c are doubles and are inserted into structure members *x,y,z* respectively

Return Values

Return the handle to the vector of type `VectorH`

Example

```
a = vectCreate(1,2,3);
```

See also

This is equivalent to the mtlib inline macro

```
VECT_CREATE(a,b,c,vect)
```

Defined as

```
#define VECT_CREATE(a,b,c,vect) {vect.x = a; vect.y = b; vect.z = c;}
```

The inline macro can still be used for efficiency but is deprecated.

Syntax / Prototype

```
int vectEquals(VectorH v1, VectorH v2);
```

Description

Performs a test on two vectors to determine if they are equal

Arguments

v1 and *v2* are vectors to be compared

Return Values

Returns 1 if v1 is equal to v2 and 0 otherwise

Example

```
if(vectEquals(a, b)){
  printf("vectEquals: Test passed\n");
}
```

See also

== operator in C++ API

Syntax / Prototype

```
int vectNotEqual(VectorH v1, VectorH v2);
```

Description

Performs a test on two vectors to determine if they are not equal

Arguments

v1 and *v2* are vectors to be compared

Return Values

Returns 1 if v1 is not equal to v2 and 0 otherwise

Example

```
if(vectNotEqual(a, b)){
  printf("vectNotEqual: Test passed\n");
}
```

See also

!= operator in C++ API

Syntax / Prototype

```
VectorH vectAdd(VectorH v1, VectorH v2);
```

Description

Adds two vectors together

Arguments

v1 and *v2* are vectors to be added

Return Values

Returns a VectorH whose constituent members are made up from the sum of the input vectors. Ie

$$\left\{\begin{matrix} a+d \\ b+e \\ c+f \end{matrix}\right\} = \left\{\begin{matrix} a \\ b \\ c \end{matrix}\right\} + \left\{\begin{matrix} d \\ e \\ f \end{matrix}\right\}$$

Example

```
c = vectAdd(a, b);
```

See also

+ operator in C++ API

Syntax / Prototype

```
VectorH vectAddScaler(VectorH v1, double scaler);
```

Description

Adds a scalar to a vector

Arguments

v1 is the vector and *scaler* is the scaler which must be a double.

Return Values

Returns a VectorH whose constituent members are made up by adding the scaler value to each of the members of *v1*

Example

```
c = vectAddScaler(a, 10)
```

See also

+ operator in C++ API (which is overloaded for scaler operations)

Syntax / Prototype

```
VectorH vectSub(VectorH v1, VectorH v2);
```

Description

Vector subtraction. Subtract vector *v2* from vector *v1*

Arguments

v1 and *v2* are both vectors.

Return Values

Returns a VectorH whose constituent members are made by a member by member subtraction

Example

```
c = vectSub(b, a);
```

See also

- operator in C++ API (which is overloaded for scaler operations)

Syntax / Prototype

```
VectorH vectSubScaler(VectorH v1, double scaler);
```

Description

Vector / scaler subtraction. Subtract scaler *scaler* from vector *v1*

Arguments

v1 is a vector. *Scaler* is a double.

Return Values

Returns a VectorH whose constituent members are made by subtracting *scaler* from each member of *v1*

Example

```
c = vectSubScaler(b, 2);
```

See also

- operator in C++ API (which is overloaded for scaler operations)

Syntax / Prototype

```
VectorH vectMult(VectorH v1, double scaler);
```

Description

Multiplication of vector *v1*

Arguments

v1 is a vector. *Scaler* is a double.

Return Values

Returns a VectorH whose constituent members are made by multiplying *scaler* to each member of *v1*

Example

```
c = vectSubScaler(b, 2);
```

See also

* operator in C++ API (which is overloaded)

Syntax / Prototype

```
VectorH vectDiv(VectorH v1, double scaler);
```

Description

Division of vector *v1*

Arguments

v1 is a vector. *Scaler* is a double.

Return Values

Returns a VectorH whose constituent members are made by dividing each member of *v1* by *scaler*

Example

```
ans = vectDiv(ans, 2);
```

See also

* operator in C++ API (which is overloaded)

Syntax / Prototype

```
VectorH vectMinus(VectorH v);
```

Description

Negates vector *v*

Arguments

v is a vector.

Return Values

Returns a VectorH whose constituent members are made by multiplying -1 by each member of v

Example

```
c = vectMinus(a);
```

See also

- operator in C++ API (which is overloaded)

Syntax / Prototype

```
double vectMag(VectorH v);
```

Description

Magnitude of vector *v*

Arguments

v is a vector.

Return Values

Returns a double calculated by:

$$d = \sqrt{v.x^2 + v.y^2 + v.z^2}$$

Example

```
dans = sqrt(4*4+5*5+6*6);
if ( vectMag(b) == dans){
  printf("vectMag: Test passed\n");
}
```

See also

mag() operator in C++ API

Syntax / Prototype

```
VectorH vectNorm(VectorH v);
```

Description

Normalise a vector

Arguments

v is a vector.

Return Values

Returns a vector whose magnitude is 1

Example

```
c = vectNorm(b);
```

See also

norm() operator in C++ API

Syntax / Prototype

```
double vectDot(VectorH v1, VectorH v2);
```

Description

Performs a dot product of two vectors *v1* and *v2*

Arguments

v1 and *v2* are both vectors

Return Values

Returns a double which is the dot product

Example

```
dans = b.x*a.x + a.y*b.y + a.z*b.z;
if(vectDot(a, b) == dans){
  printf("vectDot: Test passed\n");
}
```

See also

dot() operator in C++ API

Syntax / Prototype

```
VectorH vectCross(VectorH v1, VectorH v2);
```

Description

Performs a cross product of two vectors *v1* and *v2*

Arguments

v1 and *v2* are both vectors

Return Values

Returns a vector which is the cross product

Example

```
a=vectCreate(1, 0, 0);
b=vectCreate(0, 1, 0);
ans = vectCreate(0, 0, 1);
c = vectCross(a, b);
if(vectEquals(c, ans)){
  printf("vectCross: Test passed\n");
}
```

See also

cross() or ^ operator in C++ API

Syntax / Prototype

```
VectorH vectRotatex(VectorH v, double angle);
VectorH vectRotatey(VectorH v, double angle);
VectorH vectRotatez(VectorH v, double angle);
VectorH vectRotateAxis(VectorH v, VectorH axis, double angle);
```

Description

Performs a rotation of vector *v* around the specified axis

Arguments

v is a vector. *angle* is a double and specifies the amount of rotation. *axis* is a vector defining an arbitrary axis of rotation

Return Values

Returns a vector which is the vector *v* rotated around the axis by angle *angle*

Example

```
ans = vectCreate(0, 0, 1);
c = vectRotatex(ans, -1*PI/2);
if(vectEquals(c, b)){
  printf("vectRotatex: Test passed\n");
}
```

See also

rotatex(), rotatey(), rotatez(), rotateaxis() member functions in C++ API