WMM subroutine library

A library of subroutines is provided for both object-time and run-time linking and as source code for building the World Magnetic Model software in its various forms:

1. Stand-alone DOS prompt program
    a. Single point calculation
    b. Grid of points calculation
    c. Time series of values at the same point

2. Stand-alone program(s) with Graphical User Interface

3. Web-based on-line calculator(s)
    a. Single point calculation
    b. Grid of points calculation
    c. Time series of values at the same point

4. Integration into other DoD systems

5. Porting to third party applications / programs

The goal is that all the above software developments could use or should use the subroutine library defined below.

Contact information

Software and Model Support
National Geophysical Data Center
NOAA EGC/2
325 Broadway"
Boulder, CO 80303 USA
Attn: Manoj Nair or Stefan Maus
Phone: (303) 497-4642 or -6522
Email: Manoj.C.Nair@noaa.gov or Stefan.Maus@noaa.gov
URL: http://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml

For more details on the subroutines, please consult the WMM Technical Documentations at
http://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml

Subroutines for general use

(color codes : input, output, update)

1. `int WMM_GeodeticToSpherical(WMMtype_Ellipsoid Ellip, WMMtype_CoordGeodetic CoordGeodetic,WMMtype_CoordSpherical *CoordSpherical)`

Input: The ellipsoid parameter, `Ellip`. The Geodetic Coordinates, `CoordGeodetic`.
Output: The spherical coordinates, `CoordSpherical`

ACTION: Takes the CoordGeodetic and uses equations 7 (WMM Technical Report) along with the ellipsoid parameter to calculate CoordSpherical. The function then returns TRUE (1).

2. `int WMM_ValidateDMSstringlat(String *input, String *Error) / WMM_ValidateDMSstringlong`

Input: A degrees, minutes, seconds string “input”
Output: Returns 1 if valid, 0 otherwise. If the return value is zero an error message is copied into the Error string.
Aim: To validate a Degree, Minute, Seconds entry by user.
ACTION: The function first checks to make sure the string consists only of legal characters.
The function then scans the string into 1 or 3 integers and checks to make sure that there are an appropriate number of entries in the string. The function then checks that the degree value is legal, that the minute value is legal, and that the second value is legal. If any of these checks fail the function copies an error message to Error and returns FALSE (0). Otherwise the function returns TRUE (1).

3. `void WMM_DMSstringToDeg (String *DMSstring, double *DegreesOfArc)`
Input: DMS string.
Output: DegreesOfArc is the Decimal degree of the input DMS.
Aim: To scan a DMS string into a Decimal degree.
ACTION: The function check if the string is simply a single degree entry, then the function attempts to scan the string as though it was separated by commas. If that fails, the program scans the string as though it is separated by spaces. The program then checks if the degree value is less than 0 so it knows how to add the degrees, minutes and seconds together properly.

4. `WMM_DegreeToDMSstring(double DegreesOfArc,int UnitDepth, char *DMSstring)`
Input: Decimal Degree.
Output: A DMS string

Aim: To turn a decimal degree into a DMS string.
ACTION: The function stores the decimal degree in a temporary variable. The function then stores the temporary variable in an integer array, truncating after the decimal point. The temporary variable is reset as the difference between itself and the integer multiplied by sixty. This value is truncated and stored in the array, and this process is repeated. Each value in the array is then copied to a temporary string which is then concatenated with what is currently in the DMSstring.

NOTE: This function is intentionally formatted with the WMM_PrintUserData function in mind. Changes to the formatting of the output string will change the character (but not the content) of the output in PrintUserData.

`5. WMM_DateToYear(WMMtype_Date *Calendar_Date, String *Error);`
Input: Calendar Date
Output: A decimal year

Aim: To take an input Calendar Date and convert it into a decimal year.
Action: The function creates an array of the number of days in each month, accounting for the leap year. The function then checks that the month entry and day entry are valid. The function then creates a temporary variable which adds all the previous month's days to the current day in the current month. It then divides by the number of days in the current year and adds the current year to get the decimal year.

1st Level subroutines for the WMM

`1. int WMM_readMagneticModel (char *filename, WMMtype_MagneticModel *MagneticModel)`
Input: The function takes the filename as input.
Output: The function stores all of the file's Magnetic Model information in MagneticModel
Action: The function opens the coefficient file. If fails it returns an error (FALSE or 0). The function then reads the coefficients file line by line and stores the data in MagneticModel.
Note: The function expects the input file to be formatted like WMM.COF file.

`2. int WMM_readMagneticModel_Large ( char *filename, char *filenameSV, WMMtype_MagneticModel *MagneticModel)`
Input: The function takes the filename of the file the model coefficients are stored in, and the filename of the file that the Secular Variation coefficients are stored in.
Output: The function stores the magnetic model coefficients in MagneticModel.
Action: The function opens the coefficient file. If this fails it returns an error. The function then reads both coefficients files line by line and stores the data in MagneticModel.

`3. int WMM_TimelyModifyMagModel(WMMtype_Date UserDate, WMMtype_MagneticModel *MagneticModel,WMMtype_MagneticModel *TimedMagneticModel)`

Input: The function takes the date, UserDate, and the gauss coefficients and the first derivative of the gauss coefficients, MagneticModel, as input.
Output: The function outputs the time dependent gauss coefficients into TimedMagneticModel, as well as copying over all other data.

Action: The function copies over the edition date, epoch, maximum degree of the model and the model name. The function then calculates the time dependent gauss coefficients from MagneticModel using equation (9), and copies them into TimedMagneticModel.

4. WMMtype_MagneticModel *`WMM_AllocateModelMemory(int NumTerms);`

Input: Number of coefficients, “NumTerms”.
Ouput: Pointer to the data type WMMtype_MagneticModel
Action: Allocates memory for the data type MagneticModel.

`5.WMMtype_LegendreFunction WMM_AllocateLegFunMemory(int NumTerms);`

Input: Number of coefficients, “NumTerms”.
Ouput: Pointer to the data type WMMtype_LegendreFunction.
Action: Allocates memory for the data type LegendreFunction.

`6.WMM_SetDefaults(WMMtype_Ellipsoid *Ellip, WMMtype_MagneticModel *MagneticModel, WMMtype_Geoid *Geoid);`

Inputs: Data types Ellip , MagModel and Geoid
Ouput: None (updates the fields)

Action: Sets the default for WGS ellipsoidal parameters, Magnetic Model and EGM96 Geoid.

`7.WMM_InitializeGeoid(WMMtype_Geoid *Geoid);`

Input: Data type Geoid
Ouput: TRUE or FALSE

Action: The subroutine initializes the Geoid by reading the binary file EMG9615.BIN This file contains EGM96 geoid heights in 15x15 min resolution. Returns FALSE (0) on failure.

```
8. int WMM_Geomag(WMMtype_Ellipsoid Ellip,
WMMtype_CoordSpherical CoordSpherical,
WMMtype_CoordGeodetic CoordGeodetic, WMMtype_MagneticModel
*TimedMagneticModel, WMMtype_GeoMagneticElements
*GeoMagneticElements);
```

Input: Data types: Ellipsoid parameter, Ellip, The geodetic coordinates (CordGeo), The Geocentric coordinates (CordSph), Time adjust model coefficients, TimedMagModel
Output: The geomagnetic elements X,Y,Z,F,H (all in nT) D, I (all in degrees) and their secular changes in the data type GeomagElements.

This function will compute the Legendre polynomials, integrate spherical harmonic expansions and check for N/S poles. The subroutine outputs the X, Y and Z components and their secular variations in the spherical coordinate system. This function will use any one of the two Legendre polynomial computation routines proposed as 2nd level subroutines.

Note: This function will be mainly used for command prompt version of WMM. For applications that 1) do not require the secular change or 2) need to compute a time series or grid, the level two WMM subroutines can be grouped in different ways to avoid redundant calculations. This is demonstrated in the examples included for Grid (wmm_grid.c) and Simple point calculations (wmm_point.c).

```
9. int WMM_Grid(WMMtype_CoordGeodetic minimum,
WMMtype_CoordGeodetic maximum, double step_size, double
altitude_step_size, double time_step, WMMtype_MagneticModel
*MagneticModel, WMMtype_Geoid
*geoid, WMMtype_Ellipsoid Ellip, WMMtype_Date StartDate,
WMMtype_Date EndDate, int ElementOption, int PrintOption,
char *OutputFile)
```

This function calls WMM subroutines to generate a grid as defined by the user. The function may be used to generate a grid of magnetic field elements, time series or a profile. The selected geomagnetic element is either printed to the file GridResults.txt, a user selected filename, or to the screen depending on user option.

```
10. int WMM_FreeMagneticModelMemory(WMMtype_MagneticModel
*MagneticModel)
```

Free the magnetic model memory used by WMM functions.

```
11. int WMM_FreeLegendreMemory(WMMtype_LegendreFunction
*LegendreFunction)
```

Free the Legendre Coefficients memory used by WMM functions.

12. int WMM_FreeMemory(WMMtype_MagneticModel *MagneticModel, WMMtype_MagneticModel *TimedMagneticModel, WMMtype_LegendreFunction *LegendreFunction)

Input: Data structures MagneticModel, TimedMagneticModel an LegendreFunction
Output: None
Frees memory used by these data structures.

2nd Level subroutines for the WMM

1.WMM_PcupLow(double *Pcup, double *dPcup, double x, int nMax)

Input: Maximum spherical harmonic degree to compute (nMax), x cos(colatitude) or sin(latitude).
Ouput: Pcup : A vector of all associated Legendgre polynomials evaluated at x up to nMax. The length must by greater or equal to (nMax+1)*(nMax+2)/2.
dPcup: Derivative of Pcup(x) with respect to latitude

Action: This function evaluates all of the Schmidt-semi normalized associated Legendre functions up to degree nMax. The computation of Legendre functions in this subroutine is similar to that followed in the previous version of WMM software. This is currently retained as a legacy function and may be used for computing Legendre functions upto degree 16. The newer version (WMM_PcupHigh) is optimized for high degree computation.

2.WMM_PcupHigh(double *Pcup, double *dPcup, double x, int nMax)

Input: Maximum spherical harmonic degree to compute (nMax), x cos(colatitude) or sin(latitude).
Ouput: Pcup : A vector of all associated Legendgre polynomials evaluated at x up to nMax. The length must by greater or equal to (nMax+1)*(nMax+2)/2.
dPcup: Derivative of Pcup(x) with respect to latitude

Action:This function evaluates all of the Schmidt-semi normalized associated Legendre functions up to degree nMax. The functions are initially scaled in order to minimize the effects of underflow at large m near the geographic poles. Note that this function performs the same operation as WMM_PcupLow. However this function also can be used high degree (large nMax) models.

3. int WMM_CalculateSecularVariation (WMMtype_MagneticResults MagneticVariation, WMMtype_GeoMagneticElements *GeoMagneticElements)
Input: The current geomagnetic elements, the secular variation of the magnetic field.

Output: The function outputs the secular variation of the geomagnetic elements at the same time and place as the input geomagnetic elements.

Aim: To calculate the secular variation of the Geomagnetic elements at a specific location and time.
Action: This function simply copies the secular variation of the magnetic field components into the appropriate northerly, easterly and up elements. It then uses these elements as well as equations 19 (WMM Technical Report) in the report to calculate the other four elements.

```
4.int WMM_CalculateGridVariation(WMMtype_CoordGeodetic
location, WMMtype_GeoMagneticElements *elements)
```

Grivation (or grid variation) is the angle between grid north and magnetic north. This routine calculates the Grivation for the Polar Stereographic projection.

Input: The current declination and the current longitude.
Output: The function outputs the grid variation into the correct element.
Aim: To calculate the Grid Variation at a specified location and time.
Action: The function uses equation 1 (WMM Technical Report) of the write up.

```
5. int WMM_GetGeoidHeight (double Latitude, double
Longitude, double *DeltaHeight, WMMtype_Geoid *Geoid)
```

Input: : Data structure Geoid , Latitude, Longitude
Output: Geoid height with respect to the WGS-84 Ellipsoid in meters (DeltaHeight)

Action: Checks for latitude and longitude limits, performs interpolation of the EGM-96 grid to obtain the Geoid height.

```
6. WMM_ConvertGeoidToEllipsoidHeight(WMMtype_CoordGeodetic
*CoordGeodetic, WMMtype_Geoid *Geoid);
```

Input: Data structure Geoid and CordGeo.
Output: Ellipsoid height “h” in the data structure CoordGeodetic

Action: Converts height above MSL to height above WGS-84. Calls the function WMM_GetGeoidHeight to get the Geoid height (wrt WGS-84) for the location specified in CordGeo. Corrects for the geoid height and copies result to CordGeo->h. Note. If user wants to bypass converting height Geoid from Ellipsoid, the height must be directly copied to CordGeo->h.

```
7. WMM_AssociatedLegendreFunction(WMMtype_CoordSpherical
CoordSpherical, int nMax, WMMtype_LegendreFunction
*LegendreFunction);
```

Input: Data types LegendreFunction, CoordSpherical and maximum degree of the model, nMax.

Output: Schmidt quasi-normalized Associated Legendre Function (ALF) stored in LegFun

Action: calls 2nd level subroutines WMM_PcupLow (nMax <= 16) or WMM_PcupHigh (nMax > 16) to compute the ALF, $P_n^m(\sin\varphi')$

Reference: Equation 6, WMM Technical report.

```
8. WMM_ComputeSphericalHarmonicVariables(WMMtype_Ellipsoid
Ellip, WMMtype_CoordSpherical CoordSpherical, int nMax,
WMMtype_SphericalHarmonicVariables *SphVariables)
```

Input: Data types Ellipsoid (WGS-84 ellipsoidal parameters), CoordSpherical (Spherical coordinates) and nMax (maximum degree of the model).
Output: The relative radius ratios and sin and cosines of the latitude multiplied by order

$m \quad \left(\left(\frac{a}{r}\right)^{n+2}, \cos m\lambda, \sin m\lambda \right)$

Reference: Equations 10-12, WMM Technical report.

```
9. WMM_Summation(WMMtype_LegendreFunction *LegendreFuction,
WMMtype_MagneticModel *MagneticModel,
WMMtype_SphericalHarmonicVariables SphVariables,
WMMtype_CoordSpherical CoordSpherical,
WMMtype_MagneticResults *MagneticResults)
```

Input: Data types LegendreFunction, MagneticModel, SphVariables
Output: Magnetic field elements X, Y, and Z in spherical coordinate system or Xdot, Ydot, and Zdot.

Action: Spherical harmonic integration of the coefficients and variables.

$$X'(\varphi',\lambda,r) = -\frac{1}{r}\frac{\partial V}{\partial \varphi'} = -\sum_{n=1}^{12}\left(\frac{a}{r}\right)^{n+2}\sum_{m=0}^{n}\left(g_n^m(t)\cos m\lambda + h_n^m(t)\sin m\lambda\right)\frac{d\,\mathrm{P}_n^m(\sin\varphi')}{d\varphi'} \qquad ()$$

$$Y'(\varphi',\lambda,r) = -\frac{1}{r\cos\varphi'}\frac{\partial V}{\partial \lambda} = \frac{1}{\cos\varphi'}\sum_{n=1}^{12}\left(\frac{a}{r}\right)^{n+2}\sum_{m=0}^{n} m\left(g_n^m(t)\sin m\lambda - h_n^m(t)\cos m\lambda\right)\breve{\mathrm{P}}_n^m(\sin\varphi') \qquad ()$$

$$Z'(\varphi',\lambda,r) = \frac{\partial V}{\partial r} = -\sum_{n=1}^{12}(n+1)\left(\frac{a}{r}\right)^{n+2}\sum_{m=0}^{n}\left(g_n^m(t)\cos m\lambda + h_n^m(t)\sin m\lambda\right)\breve{\mathrm{P}}_n^m(\sin\varphi')$$

Reference: Equations 10-12, WMM Technical report.

10.int WMM_SummationSpecial(WMMtype_MagneticModel MagneticModel, WMMtype_SphericalHarmonicVariables SphVariables, WMMtype_CoordSpherical CoordSpherical, WMMtype_MagneticResults *MagneticResults)

Input: Data structures MagneticModel, SphVariables, CoordSpherical
Output: Magnetic field element Y in spherical coordinate system

Action: Spherical harmonic integration of the coefficients and variables.
This function takes care of the geographic poles by specially computing the “Y” element.
Reference: Section 1.4 “Singularities at the geographic poles”, WMM Technical Report.

11. int WMM_SecVarSummation(WMMtype_LegendreFunction *LegendreFuction, WMMtype_MagneticModel *MagneticModel, WMMtype_SphericalHarmonicVariables SphVariables, WMMtype_CoordSpherical CoordSpherical, WMMtype_MagneticResults *MagneticResults)

Input: Data types LegendreFun, MagneticModel, SphVariables
Output: Secular changes in the magnetic field elements X, Y, and Z in spherical coordinate system (Xdot, Ydot, and Zdot)

Action: Spherical harmonic integration of the secular variation coefficients and variables.
Reference: Equations 13-15, WMM Technical report.

12. int WMM_SecVarSummationSpecial(WMMtype_MagneticModel *MagneticModel, WMMtype_SphericalHarmonicVariables SphVariables, WMMtype_CoordSpherical CoordSpherical, WMMtype_MagneticResults *MagneticResults)

Input: Data structures MagneticModel, SphVariables, CoordSpherical
Output: Secular change in the magnetic field element Y in spherical coordinate system (Ydot).

Action: Spherical harmonic integration of the secular variation coefficients and variables.
This function takes care of the geographic poles by specially computing the “Y” element.
Reference: Section 1.4 “Singularities at the geographic poles”, WMM Technical Report.

13. WMM_RotateMagneticVector(WMMtype_CoordSpherical CoordSpherical,WMMtype_CoordGeodetic CoordGeodetic, WMMtype_MagneticResults MagneticResultsSph, WMMtype_MagneticResults *MagneticResultsGeo)

Input : Geomagnetic elements in Spherical coordinates (MagResultsGeoSph), Coordinates in spherical (CordSph) and Geodetic (CordGeod) system.
Output: Geomagnetic elements in Geodetic coordinates (MagResultsGeo),

Action:

$$X = -X'\cos\psi + Z'\sin\psi$$

$$Y = Y'$$

$$Z = -X'\sin\psi + Z'\cos\psi$$

Reference: Equation 16, WMM Technical report.

```
14. WMM_CalculateGeomagElements(WMMtype_MagneticResults
*MagneticResultsGeo, WMMtype_GeoMagneticElements
*GeoMagneticElements)
```

Input: Geomagnetic elements X,Y and Z in Geodetic coordinates (MagResultsGeo),
Ouput: Geomagnetic elements X,Y Z, F, H, Z, Decl, Incl in GeomagElements.
Action:

$$H = \sqrt{X^2 + Y^2}, \quad F = \sqrt{H^2 + Z^2}, \quad D = \arctan(Y, X), \quad I = \arctan(Z, H),$$

Reference: Equation 18, WMM Technical report.

```
15. int WMM_CheckGeographicPole(WMMtype_CoordGeodetic
*CoordGeodetic)
```

Input: Location coordinates in Geodetic system
Output: Location coordinates in Geodetic system

Action: Check if the latitude is equal to -90 or 90. If it is, offset it by 1e-5 to avoid division by zero. This is not currently used in the WMM software. This may be used to avoid calling WMM_SummationSpecial.

Subroutines specific to command prompt program

```
1. void WMM_PrintUserData(WMMtype_GeoMagneticElements
GeomagElements, WMMtype_CoordGeodetic SpaceInput,
WMMtype_Date TimeInput, WMMtype_MagneticModel
*MagneticModel, WMMtype_Geoid *Geoid)
```

Input: All of the values to be printed are input as objects.
Output: The function prints its output to the screen.
Aim: To print the results in a formatted, readable, fashion.

Action: The function takes in the geomagnetic elements and secular variation components. The program then checks to see if the user is at one of the geographic poles, if the user is, the program then prints NAN for all of the undefined values, and also prints the other geomagnetic elements. If the user is not at a geographic pole then the program prints all seven geomagnetic elements.

```
2. void WMM_GetUserInput(WMMtype_MagneticModel
*MagneticModel, WMMtype_Geoid *Geoid, WMMtype_CoordGeodetic
*CoordGeodetic, WMMtype_Date *MagneticDate)
```

Aim: To get the input of the user, check it for legality and warnings, then send the information on to the main program.
Input: The current epoch from the magnetic model is taken as input, as well as the user entered input.
Output: The program outputs a CordGeo object, and a Date object.
Action: The function prompts the user for the input then searches the input string for telltale characters (a comma for example) , that indicate the way the user wants to input the data. The program then tries to scan the input string as though it is formatted in the way associated with that character, for example if there is a period in the input the function attempts to scan he. If the function cannot it returns an error message, and prompts the user to re-enter the data.

```
3. Void WMM_GeomagIntroduction(WMMtype_MagneticModel
*MagneticModel)
```

Aim: To provide the user an introduction to the program.
Input: The current epoch is used in the introduction text and is needed as input. The user may also input a help request.
Output: The function prints the introduction to the screen. The function also will print the help information to the screen if that is requested.
Action: The function prints the introduction then prompts the user to either continue or check the help information. The program then either returns, or prints the help information.

```
4. Int WMM_Warnings(int control, double value,
WMMtype_MagneticModel *MagneticModel)
```

Input: The function takes a control integer (see note) to determine what warning to print, it also takes the value that is causing the warning, and the mag model to print the epoch information. The user inputs whether to get new data, continue with the value or exit the program.
Output: The function prints the specified warning to the screen and returns 0, 1, or 2, based on whether the user chooses to exit, get new data, or continue.
Aim: To provide the user with warnings when the user attempts to use the model in a way that may produce dubious results.

Note: 1 = Horizontal field strength low, 2 = Horizontal field strength very low, 3 = Location at Geographic pole, 4 = Elevation outside recommended range, 5 = Date outside recommended range.

Subroutines specific to the ISO reading routine

1. int WMM_readMagneticModel_ISO(char *filename, WMMtype_MagneticModel *magneticmodels[], int array size)

Input: filename - Path to the ISO format model file to be read, array_size - Max No of models to be read from the file
Output: *magneticmodels[] - Array of magnetic models read from the file
Return value: Returns the number of models read from the file. -2 implies that internal or external static degree was not found in the file, hence memory cannot be allocated. -1 implies some error during file processing (I/O). 0 implies no models were read from the file. If ReturnValue > array_size then there were too many models in model file but only <array_size> number were read. If ReturnValue <= array_size then the function execution was successful.
Aim: The function reads the ISO format model file and each model found in order in the magneticmodels array. In the WMM example wrapper file wmm_point_iso.c only one magnetic model is used.

2. void assignheadervalues(WMMtype_MagneticModel *model, char values[][MAXLINELENGTH])

Input: values[][MAXLINELENGTH] – array of strings that are in the header of the magnetic model for the ISO routine.
Aim: The function adds these header values to the model structure.

3. void printMagneticModels(WMMtype_MagneticModel *models[], int num)

Input: The magnetic models to be printed, and the number of magnetic models to be printed.
Output: The function prints the basics (Name, Edition Date, epoch, Number of internal static degrees, number of internal secular variation, and whether or not secular variation is used) about the stored magnetic model and one row of the coefficients to the screen. This function is not used in the example files, but may be useful in checking to see if a new ISO format file is being read correctly.

Data types for the WMM

1. WMMtype_MagneticModel

double EditionDate; (Model Release date)

double epoch; (base time of geomagnetic model (YRS) epoch)
char ModelName[20]; (File name)
double *Main_Field_Coeff_G; (G gauss coefficients)
double *Main_Field_Coeff_H; (H gauss coefficients)
double *Secular_Var_Coeff_G; (G gauss coefficients of secular variation (nT/yr)
double *Secular_Var_Coeff_H; (H gauss coefficients of secular variation (nT/yr)
int nMax; (maximum degree of spherical harmonic model.)
int nMaxSecVar; (maximum degree of spherical harmonic secular model)
int SecularVariationUsed; (A flag for secular variation calculation)

2. WMMtype_Ellipsoid;

double a; (semi-major axis of the ellipsoid WGS-84)
double b; (semi-minor axis of the ellipsoid WGS-84)
double fla; (flattening)
double epssq; (first eccentricity squared)
double eps; (first eccentricity)
double re; (mean radius of ellipsoid)

3. WMMtype_CoordGeodetic;

double lambda; (longitude in degrees)
double phi; (Geodetic latitude in degrees)
double HeightAboveEllipsoid; (Height above the ellipsoid (HaE) in Kilometers)
double HeightAboveGeoid; ((Height above the geoid in Kilometers)

Note: The WMM core functions will always use the height above ellipsoid. The user entered height above MSL will be converted to HAE by referencing it to EGM-96 model.

4. WMMtype_CoordSpherical

double lambda(longitude in degrees)
double phig; (geocentric latitude in degrees)
double r; (distance from the center of the ellipsoid in Kilometers)

5. WMMtype_Date

int Year;
int Month;
int Day;
double DecimalYear; (decimal years)

Note. : The WMM core functions will always use the time in decimal years. If user enters time in Year, Moth and Date format, this should be converted to decimal years before calling WMM core functions.

6. WMMtype_MapProjectionCode

int	WMM_Mercator;
int	WMM_LambertConformalConic;
int	WMM_PolarStereographic;
int	WMM_TransverseMercator;

7. WMMtype_LegendreFunction

double *Pcup; (Legendre Function)
double *dPcup; (Derivative of Legendre Function)

8. WMMtype_MagneticResults

double Bx; (North)
double By; (East)
double Bz; (Down)

9. WMMtype_SphericalHarmonicVariables

double RelativeRadiusPower[WMM_MAX_MODEL_DEGREES+1];
double cos_mlambda[WMM_MAX_MODEL_DEGREES+1];
double sin_mlambda[WMM_MAX_MODEL_DEGREES+1];

10. WMMtype_GeomagElements;

double Decl;
Angle between the magnetic field vector and true north, positive east
double Incl;
Angle between the magnetic field vector and the horizontal plane, positive down
double F;
Magnetic field Strength
double H;
Horizontal Magnetic Field Strength
double X;
Northern component of the magnetic field vector
double Y;
Eastern component of the magnetic field vector
double Z;
Downward component of the magnetic field vector
double GV;

The Grid Variation
double Decldot;
Yearly Rate of change in declination
double Incldot;
Yearly Rate of change in inclination
double Fdot;
Yearly rate of change in Magnetic field strength
double Hdot;
Yearly rate of change in horizontal field strength
double Xdot;
Yearly rate of change in the northern component
double Ydot;
Yearly rate of change in the eastern component
double Zdot;
Yearly rate of change in the downward component
double GVdot;
Yearly rate of change in the grid variation

11. WMMtype_Geoid

int NumbGeoidCols ; (360 degrees of longitude at 15 minute spacing)
int NumbGeoidRows ; (180 degrees of latitude at 15 minute spacing)
int NumbHeaderItems ; (min, max lat, min, max long, lat, long spacing)
int caleFactor; (4 grid cells per degree at 15 minute spacing)
float *GeoidHeightBuffer;
int NumbGeoidElevs;
int Geoid_Initialized ; (indicates successful initialization)
int UseGeoid (indicates the geoid is being used)

12. WMMtype_CordGeodeticStr

char Longitude[40];
char Latitude[40];

13. WMMtype_UTM Parameters

double Easting; ((X) in meters)
double Northing; ((Y) in meters)
int Zone; (UTM Zone)
char HemiSphere ;
double CentralMeridian;
double ConvergenceOfMeridians;
double PointScale;