Geomagnetism Library Software Manual

A library of subroutines is provided for both object-time and run-time linking and as source code for building Geomagnetic Model software in 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

Wrapper Subroutines

(color codes : input, output, update)

```
1. int MAG_Geomag(MAGtype_Ellipsoid Ellip,
MAGtype_CoordSpherical CoordSpherical,
MAGtype_CoordGeodetic CoordGeodetic, MAGtype_MagneticModel
*TimedMagneticModel, MAGtype_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.
Action: 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 spherical harmonic 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).

```
2. int MAG_Grid(MAGtype_CoordGeodetic minimum,
MAGtype_CoordGeodetic maximum, double step_size, double
altitude_step_size, double time_step, MAGtype_MagneticModel
*MagneticModel, MAGtype_Geoid
*geoid, MAGtype_Ellipsoid Ellip, MAGtype_Date StartDate,
MAGtype_Date EndDate, int ElementOption, int PrintOption,
char *OutputFile)
```

Input: The minimum, maximum, and intended step sizes of the geodetic coordinates, altitude, and date. This function also takes the MagneticModel and Geoid as inputs. Element Option, Print Option, and OutputFile are inputs that determine the location as well as the content of the output.
Output: Output is either printed to the screen or generated into OutputFile.
Action: 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.

```
3. int MAG_robustReadMagneticModel_Large( char *filename,
char* filenameSV, MAGtype_MagneticModel *MagneticModel[],
int array_size)
```

Input: The filenames of the static and secular variation coefficient files in filename, and filenameSV respectively. Array size is the number of magnetic models.

Output: The function initializes array_size Magnetic Models with coefficients from filename and filenameSV.
Action: This function goes through the files with the given filenames to figure out how many degrees there are in each, allocate memory for the models, then call MAG_readMagneticModel_Large to read the coefficients into the models.

```
4. int MAG_robustReadMagModels(char* filename,
MAGtype_MagneticModels  *(*magneticmodels)[], int
array_size)
```

Input: The filename of the coefficient file. The number of magnetic models.
Output: The function initializes array_size Magnetic Models with coefficients from filename.
Action: This function checks to see the format of the file, then if it is SHDF formatted, the function calls MAG_readMagneticModels_SHDF, otherwise it analyzes the file to calculate the number of terms required, allocates memory for the magnetic models then calls MAG_readMagneticModel to load the appropriate coefficients from filename.

```
5. int MAG_SetDefaults(MAGtype_Ellipsoid *Ellip,
MAGtype_Geoid *Geoid);
```

Input: Data types Ellip and Geoid.
Output: None (updates the fields)
Action: Sets the default for WGS ellipsoidal parameters and EGM96 Geoid.

User Interface

```
1. void MAG_Error(int control)
```

Aim: To provide the user with error information when something goes wrong.
Input: The function takes a control integer (see note) to determine what error to print.
Output: The function prints the specified error to the standard output.

```
2. void MAG_GeomagIntroduction_WMM(MAGtype_MagneticModel
*MagneticModel, char *VersionDate)
```

Aim: To provide the user an introduction to the world magnetic model program.
Input: The current epoch is used in the introduction text and is needed as input. The VersionDate of the source file. 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.

```
3. void MAG_GeomagIntroduction_EMM(MAGtype_MagneticModel,
char *VersionDate)
```

Aim: To provide the user an introduction to the enhance magnetic model program.

Input: The current epoch is used in the introduction text and is needed as input. The VersionDate of the source file. 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. void MAG_GetUserGrid(MAGtype_CoordGeodetic *minimum,
MAGtype_CoordGeodetic *maximum, double *step_size, double
*a_step_size, double *step_time, MAGtype_Date *StartDate,
MAGtype_Date *EndDate, int *ElementOption, int
*PrintOption, char *OutputFile, MAGtype_Geoid *Geoid)
```

Aim: To get the grid input of the user, check it for legality and warnings, then send the information on to the main program.
Input: The geoid.
Output: The program outputs a two CordGeo object the minimum and maximum, two Date objects the start date and end date, and the step size for latitude and longitude, step_size, the step size for altitude, a_step_size, and the step size for the date (in decimal years), step_time. Additionally the function returns the intended magnetic element to be printed (ElementOption), as well as the method of printing the data (to the screen or to a file), PrintOption, and lastly the output filename, OutputFile.
Action: The function prompts the user for the input and stores it in the output variables. ElementOption maps to elements using the following table:

1. Declination	9. Ddot
2. Inclination	10. Idot
3. F	11. Fdot
4. H	12. Hdot
5. X	13. Xdot
6. Y	14. Ydot
7. Z	15. Zdot
8. GV	16. Gvdot

```
5. void MAG_GetUserInput(MAGtype_MagneticModel
*MagneticModel, MAGtype_Geoid *Geoid, MAGtype_CoordGeodetic
*CoordGeodetic, MAGtype_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.

6. void MAG_PrintUserData(MAGtype_GeoMagneticElements GeomagElements, MAGtype_CoordGeodetic SpaceInput, MAGtype_Date TimeInput, MAGtype_MagneticModel *MagneticModel, MAGtype_Geoid *Geoid)

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

7. int MAG_ValidateDMSstringlat(String *input, String *Error) / MAG_ValidateDMSstringlong

Aim: To validate a Degree, Minute, Seconds entry by user.
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.
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).

8. int MAG_Warnings(int control, double value, MAGtype_MagneticModel *MagneticModel)

Aim: To provide the user with warnings when the user attempts to use the model in a way that may produce dubious results.
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.
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.

Memory and File Processing

```
1. MAGtype_LegendreFunction*
MAG_AllocateLegendreFunctionMemory(int NumTerms);
```
Input: Number of coefficients, “NumTerms”.
Output: Pointer to the data type MAGtype_LegendreFunction.
Action: Allocates memory for the data type MAGtype_LegendreFunction.

```
2. MAGtype_MagneticModel* MAG_AllocateModelMemory(int NumTerms);
```
Input: Number of coefficients, “NumTerms”.
Output: Pointer to the data type MAGtype_MagneticModel
Action: Allocates memory for the data type MagneticModel.

```
3.MAGtype_SphericalHarmonicVariables*
MAG_AllocateSphVarMemory(MAGtype_SphericalHarmonicVariables
*SphVariables, int nMax);
```
Input: Max degree of model, nMax.
Output: Pointer to the data type MAGtype_SphericalHarmonicVariables with sufficient memory allocated to it.
Action: Allocates memory for the data type MAGtype_SphericalHarmonicVariables.

```
4. void MAG_AssignHeaderValues(MAGtype_MagneticModel
*model, char values[][MAXLINELENGTH])
```
Aim: The function adds these header values to the model structure.
Input: values[][MAXLINELENGTH] – array of strings that are in the header of the magnetic model for the ISO routine.

```
5. void MAG_AssignMagneticModelCoeffs(MAGtype_MagneticModel
*Assignee, MAGtype_MagneticModel *Source, int nMax, int
nMaxSecVar);
```
Input: The magnetic model to be assigned to, Assignee. The source magnetic model, Source. The maximum static degree to be assigned, nMax. The maximum secular variation degree to be assigned, nMaxSecVar.
Output: Assignee.
Action: Assigns the first nMax degrees of source model static coefficients to assignee model, overwriting existing coefficients. The first nMaxSecVar degrees of secular variation coefficents are then assigned. The nMax and nMaxSecVar cannot be greater than either model's maximum degrees.

```
6. int MAG_FreeMemory(MAGtype_MagneticModel *MagneticModel,
MAGtype_MagneticModel *TimedMagneticModel,
MAGtype_LegendreFunction *LegendreFunction)
```
Input: Data structures MagneticModel, TimedMagneticModel and LegendreFunction
Output: Null Pointers.
Action: Frees memory used by these data structures.

```
7. int MAG_FreeLegendreMemory(MAGtype_LegendreFunction
*LegendreFunction)
```
Input: A pointer to the LegendreFunction coefficients.
Output: A null pointer.
Action: Free the Legendre Coefficients memory used by WMM functions.

```
8. int MAG_FreeMagneticModelMemory(MAGtype_MagneticModel
*MagneticModel)
```
Input: A pointer to the MagneticModel.
Output: A null pointer.
Action: Frees the magnetic model memory used by WMM functions.

```
9. int
MAG_FreeSphVarMemory(MAGtype_SphericalHarmonicVariables
*SphVar)
```
Input: A pointer to a MAGtype_SphericalHarmonicVariables.
Output: A null pointer.
Action: Free the memory used by the Spherical Harmonic Variables.

```
10. void MAG_PrintWMMFormat(char *filename,
MAGtype_MagneticModel *MagneticModel)
```
Input: A filename for the WMM formatted output, filename. The magnetic model to be printed, MagneticModel.
Output: A file with name, filename.
Action: Prints a Magnetic Model to a file in the format of the WMM.COF file.

```
11. void MAG_PrintEMMFormat(char *filename,char *filenameSV
MAGtype_MagneticModel *MagneticModel)
```
Input: A filename for the EMM formatted output of static coefficients, filename. A filename for the EMM formatted output of secular variation coefficients, filenameSV. The magnetic model to be printed, MagneticModel.
Output: A file with name, filename.
Action: Prints a Magnetic Model to a file in the format of the EMM.COF and EMMSV.COF files.

```
12. int MAG_readMagneticModel (char *filename,
MAGtype_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.

```
13. int MAG_readMagneticModel_Large (char *filename, char
*filenameSV, MAGtype_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.

```
14. int MAG_readMagneticModel_SHDF(char *filename,
MAGtype_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 pointers to 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.

```
15. char *MAG_Trim(char *str)
```

Input: The string to be trimmed, str.
Output: The return value of the function is its only output.
Action: The function eliminates preceding and trailing spaces from the input string and returns the modified string.

Conversions, Transformations, and other Calculations

```
1. MAG_CalculateGeomagElements(MAGtype_MagneticResults
*MagneticResultsGeo, MAGtype_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.

```
2. int MAG_CalculateSecularVariationElements
(MAGtype_MagneticResults MagneticVariation,
MAGtype_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.

```
3. int MAG_CalculateGridVariation(MAGtype_CoordGeodetic
location, MAGtype_GeoMagneticElements *elements)
```

Note: 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.

```
4. int MAG_DateToYear(MAGtype_Date *Calendar_Date, String
*Error);
```

Aim: To take an input Calendar Date and convert it into a decimal year.
Input: Calendar Date
Output: 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.

```
5. void MAG_DegreeToDMSstring(double DegreesOfArc,int
UnitDepth, char *DMSstring)
```

Aim: To turn a decimal degree into a DMS string.
Input: Decimal Degree.
Output: 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 MAG_PrintUserData function in mind. Changes to the formatting of the output string will change the formatting (but not the content) of the output in PrintUserData.

```
6. void MAG_DMSstringToDegree (String *DMSstring, double
*DegreesOfArc)
```

Aim: To scan a DMS string into a Decimal degree.
Input: DMS string.
Output: DegreesOfArc is the Decimal degree of the input DMS.
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.

```
7. int MAG_GeodeticToSpherical(MAGtype_Ellipsoid Ellip,
MAGtype_CoordGeodetic CoordGeodetic, MAGtype_CoordSpherical
*CoordSpherical)
```

Aim: To convert geodetic coordinates to spherical coordinates.
Input: The reference ellipsoid, Ellip. The geodetic coordinates, CoordGeodetic.
Output: The spherical coordinates, CoordSpherical.
Action: The function uses the set of equations (7, 8) from the WMM technical report to convert from geodetic to geocentric (spherical) coordinates.

```
8. int MAG_GetTransverseMercator(MAGtype_CoordGeodetic
CoordGeodetic, MAGtype_UTMParameters *UTMParameters)
```

Input : The MAGtype_CoordGeodetic structure, containing a latitude and longitude.
Output: Data structure MAGtype_UTMParameters, containing the easting, northing, UTM zone, the Hemisphere, longitude of the central meridian, convergence of the meridians, and the point scale.
Action: The function gets the UTM Parameters for a given latitude and longitude. As well as the easting, northing, convergence of the meridians, and the point scale.

```
9. int MAG_GetUTMParameters(double Latitude, double
Longitude,  int *Zone, char *Hemisphere, double
*CentralMeridian)
```

Input : The geodetic latitude, Latitude, and longitude, Longitude.
Output: The UTM zone, the hemisphere, and the longitude of the CentralMeridian.
Action: The function converts geodetic coordinates to the UTM projection parameters.

```
10. int MAG_isNaN(double d)
```

Input: The decimal number to be checked, d.
Output: Returns 1 if d is NaN, returns 0 otherwise.
Action: Checks if d is NaN.

```
11. int MAG_RotateMagneticVector(MAGtype_CoordSpherical
CoordSpherical,MAGtype_CoordGeodetic CoordGeodetic,
MAGtype_MagneticResults MagneticResultsSph,
MAGtype_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.

12. void MAG_SphericalToCartesian(MAGtype_CoordSpherical CoordSpherical, double *x, double *y, double *z)

Input : The point spherical coordinates (CoordSpherical), where phi is spherical latitude and lambda is longitude.
Output: The point in cartesian coordinates (x, y, z).
Action:

$$x = r\cos(\lambda)\cos(\varphi)$$

$$y = r\cos(\lambda)\cos(\varphi)$$

$$z = r\sin(\varphi)$$

13. void MAG_TMfwd4(double Eps, double Epssq, double K0R4, double K0R4oa, double Acoeff[], double Lam0, double K0, double falseE, double falseN, int XYonly, double Lambda, double Phi, double *X, double *Y, double *pscale, double *CoM)

Input:	Eps	Eccentricity (epsilon) of the ellipsoid
	Epssq	Eccentricity squared
	(R4	Meridional isoperimetric radius)
	(K0	Central scale factor)
	K0R4	K0 times R4
	K0R4oa	K0 times Ratio of R4 over semi-major axis
	Acoeff	Trig series coefficients, omega as a function of chi
	Lam0	Longitude of the central meridian in radians
	K0	Central scale factor, for example, 0.9996 for UTM
	falseE	False easting, for example, 500000 for UTM
	falseN	False northing
	Xyonly	If equal to 1, then only the X and Y output values will be computed.
	Lambda	Longitude (in radians)
	Latitude	Geodetic Latitude (in radians)

Output: Easting, X. Northing, Y. Point-scale, pscale. Convergence-of-meridians, CoM.
Action: Uses an algorithm developed by C. Rollins to calculate Easting, Northing, point scale and convergence of meridians.

14. int MAG_YearToDate(MAGtype_Date *Date)

Input : The MAGtype_Date structure, containing the decimal year.
Output: The day, month, and year elements of the Date structure are output.

Action: The function calculates the integer day, month and year from the input decimal year. Updating the Date structure by assigning the results to the day, month, and year elements.

Spherical Harmonics

```
1. MAG_AssociatedLegendreFunction(MAGtype_CoordSpherical
CoordSpherical, int nMax, MAGtype_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 MAG_PcupLow (nMax <= 16) or MAG_PcupHigh (nMax > 16) to compute the ALF, $\breve{P}_n^m(\sin\varphi')$
Reference: Equation 6, WMM Technical report.

```
2. int MAG_CheckGeographicPole(MAGtype_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 MAG_SummationSpecial.

```
3. int
MAG_ComputeSphericalHarmonicVariables(MAGtype_Ellipsoid
Ellip, MAGtype_CoordSpherical CoordSpherical, int nMax,
MAGtype_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 $\left(\left(\frac{a}{r}\right)^{n+2}, \cos m\lambda, \sin m\lambda\right)$
Reference: Equations 10-12, WMM Technical report.

```
4. int MAG_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 MAG_PcupLow. However, this function also can be used high degree (large nMax) models.

```
5. int MAG_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 up to degree 16. The newer version (MAG_PcupHigh) is optimized for high degree computation.

```
6. int MAG_SecVarSummation(MAGtype_LegendreFunction
*LegendreFuction, MAGtype_MagneticModel *MagneticModel,
MAGtype_SphericalHarmonicVariables SphVariables,
MAGtype_CoordSpherical CoordSpherical,
MAGtype_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.

```
7. int MAG_SecVarSummationSpecial(MAGtype_MagneticModel
*MagneticModel, MAGtype_SphericalHarmonicVariables
SphVariables, MAGtype_CoordSpherical CoordSpherical,
MAGtype_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.

```
8. MAG_Summation(MAGtype_LegendreFunction *LegendreFuction,
MAGtype_MagneticModel *MagneticModel,
MAGtype_SphericalHarmonicVariables SphVariables,
MAGtype_CoordSpherical CoordSpherical,
MAGtype_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.

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

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

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

Reference: Equations 10-12, WMM Technical report.

```
9. int MAG_SummationSpecial(MAGtype_MagneticModel
MagneticModel, MAGtype_SphericalHarmonicVariables
SphVariables, MAGtype_CoordSpherical CoordSpherical,
MAGtype_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.

```
10. int MAG_TimelyModifyMagModel(MAGtype_Date UserDate,
MAGtype_MagneticModel *MagneticModel,MAGtype_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.

Geoid

```
1. MAG_ConvertGeoidToEllipsoidHeight(MAGtype_CoordGeodetic
*CoordGeodetic, MAGtype_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 MAG_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.

```
2. int MAG_GetGeoidHeight (double Latitude, double
Longitude, double *DeltaHeight, MAGtype_Geoid *Geoid)
```

Input: The 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.

```
3. int MAG_InitializeGeoid(MAGtype_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.
Note: This function is no longer used as the EGM9615.bin has been replaced with a header file.

Data types for the WMM

1. MAGtype_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. MAGtype_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. MAGtype_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. MAGtype_CoordSpherical

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

5. MAGtype_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. MAGtype_MapProjectionCode

```
int    MAG_Mercator;
int    MAG_LambertConformalConic;
int    MAG_PolarStereographic;
int    MAG_TransverseMercator;
```

Note: This structure is deprecated and will be removed in the near future.

7. MAGtype_LegendreFunction

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

8. MAGtype_MagneticResults

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

9. MAGtype_SphericalHarmonicVariables

```
double *RelativeRadiusPower;
double *cos_mlambda;
double *sin_mlambda;
```

10. MAGtype_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. MAGtype_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. MAGtype_CordGeodeticStr

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

13. MAGtype_UTM Parameters

double Easting; ((X) in meters)
double Northing; ((Y) in meters)
int Zone; (UTM Zone)

```
char HemiSphere ;
double CentralMeridian;
double ConvergenceOfMeridians;
double PointScale;
```