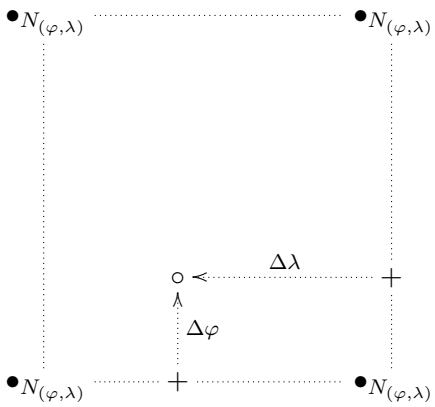
# Geoid Interpolation Software Technical Specification and User's Guide




**Version 1.0.3.0**

**December 2011**

ICSM INTERGOVERNMENTAL COMMITTEE ON SURVEYING AND MAPPING

**Geoid Interpolation Technical Specification**
AusGeoid Grid File (NTv2) Interpolation library and executables
December 2011

---



For queries and/or feedback relating to the AusGeoid grid file, please contact:

Nicholas Brown
Project Officer, National Geospatial Reference Systems
Geospatial & Earth Monitoring Division
Geoscience Australia
Cnr Jerrabomberra Avenue and Hindmarsh Drive, Symonston, Canberra, 2601
nicholas.brown@ga.gov.au

For technical and software related matters, please contact:

Roger Fraser
Manager, Geodetic Survey
Office of Surveyor-General Victoria
Department of Sustainability and Environment
Level 17/570 Bourke St, Melbourne, Victoria, 3000
roger.fraser@dse.vic.gov.au

# Contents

# Preface

In 2009, the Intergovernmental Committee on Surveying and Mapping (ICSM) commissioned the development and implementation of a new ellipsoid–geoid correction surface for Australia. Known as AusGeoid09, this surface will enable users of Global Navigation Satellite Systems (GNSS) to compute Australian Height Datum of 1971 (AHD71) heights directly from ellipsoidal heights.

To facilitate the conversion of ellipsoidal and orthometric heights for randomly located points, AusGeoid09 has been implemented as a file of regularly spaced grid node values. In order to interpolate the N values from the grid file in an efficient and standardised way, the National Transformation version 2.0 (NTv2) format[1] has been adopted for structuring and storing the AusGeoid09 grid node values. Whilst this format was specifically developed for interpolating grid values for two–dimensional coordinates, NTv2 is well suited to the problem of geoid interpolation for the following reasons:

- Instantaneous interpolation of grid node values independent of grid file size
- Support for user–defined grid shift intervals and grid shift units
- Inclusion of multiple sub–grids having various grid shift intervals
- Management of metadata relating to each sub–grid
- Ability to manage and interpolate uncertainty at each grid node
- Widely recongised and supported format amongst the geodetic community

In order to provide a consistent approach to N–value interpolation for the widest possible range of user needs in a platform–independent way, a suite of geoid interpolation applications has been developed. This document describes the use of these applications and provides the technical specifications of the application programming interface (API).

[1] Visit http://www.geod.nrcan.gc.ca/tools-outils/ntv2_e.php for information and technical specifications on the NTv2 file format.

# 1 Introduction

This document describes the user guidelines and API for a suite of applications that can be used for interpolating geoid–ellipsoidal (N) values and deflections of the vertical. The geoid interpolation functionality relies on the use of the NTv2 file format.

Three software products are described:

**dnaGeoid** A standard C++ product that can be compiled as a Windows dynamic link library (DLL) or UNIX/Linux shared object (SO). dnaGeoid provides all the required functionality for obtaining N values and deflections from user–supplied interpolants, and for creating NTv2 grid files from legacy WINTER dat files.

dnaGeoid is primarily intended for software developers wishing to incorporate geoid interpolation functionality within existing applications. dnaGeoid can be called from applications written in C, C++, C#, VB6/VB.Net, FORTRAN or Java using the Java Native Interface (JNI), and may be run on either Windows or UNIX/Linux platforms.

**geoid** A standard C++ product that can be compiled as a Windows or UNIX/Linux executable. geoid provides a simple command–line interface for controlling the use of dnaGeoid to perform geoid interpolation functions.

**GeoidInt** A Microsoft Windows 2000/XP/Vista executable that provides a simple Windows interface for controlling the use of dnaGeoid to perform geoid interpolation functions.

Figure 1.1 illustrates the technical architecture of the suite of software applications. As shown by Figure 1.1, platform–independent C++ source code may be compiled on either UNIX or Windows environments to produce dnaGeoid and geoid binaries. Microsoft C++ and GCC compilers have been used to build and test these products. Where grid file interpolation is required in a Web services environment, either dnaGeoid or geoid may be called. It is anticipated that developers of proprietary software products will prefer to statically link dnaGeoid at compile time, however the option for dynamic linking is also provided.

GeoidInt has been developed for users who require an interactive, Windwos–based approach to geoid interpolation. Unlike dnaGeoid and geoid, GeoidInt has been specifically written for Microsoft Windows environments using Visual C++ and the Microsoft Foundation Class (MFC) library.

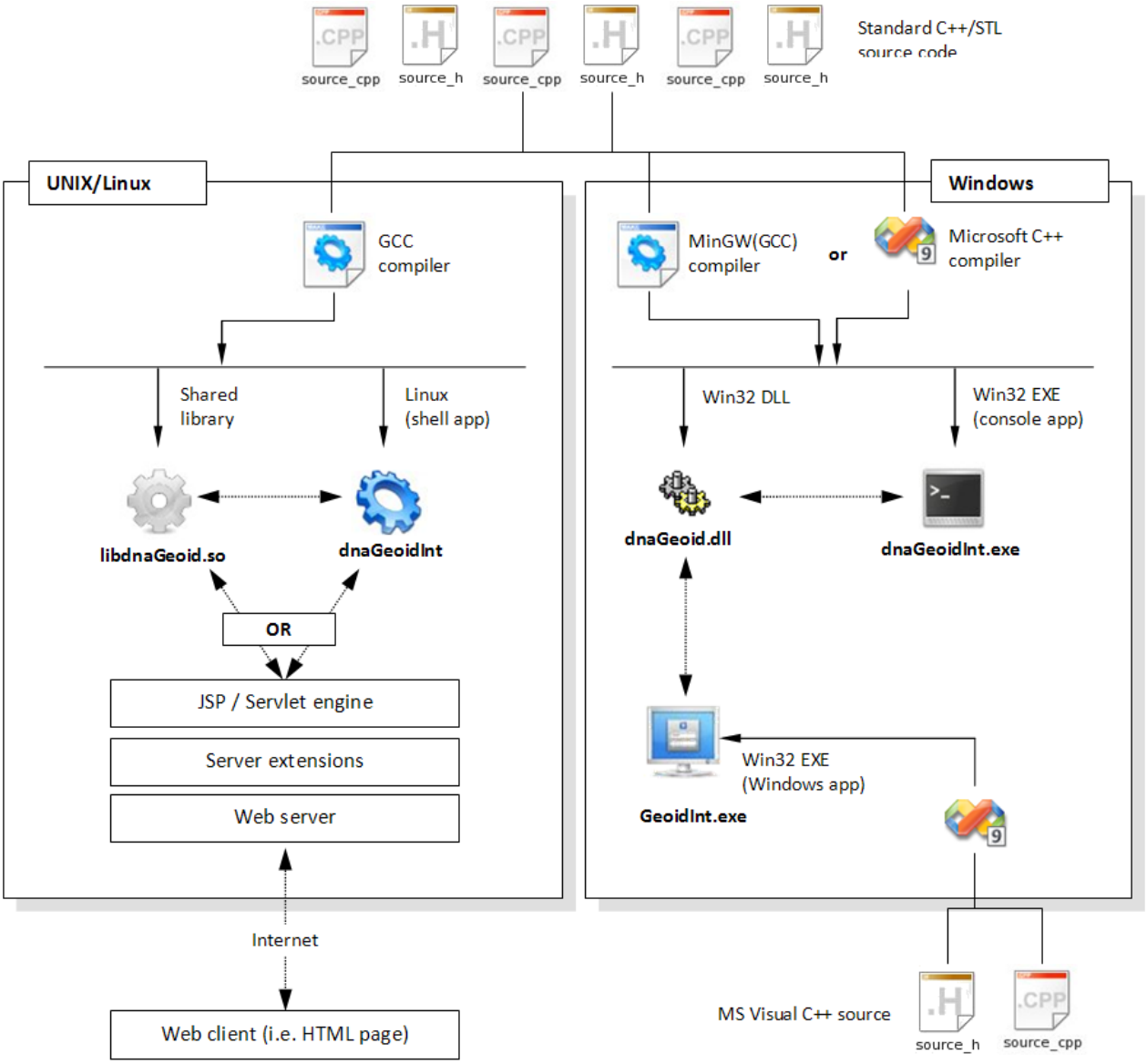



Figure 1.1: Geoid Interpolation Technical Architecture

# 2 dnaGeoid Library

## 2.1 Data struct type declarations

Various data structs have been declared to group common data elements used by dnaGeoid. The following is a description of these data structs.

### typedef struct `interpolant`

`interpolant` is used to hold the coordinates and height for user specified interpolants, transformed heights and the status of geoid interpolations.

```
typedef struct {
    double dLatitude;
    double dLongitude;
    double dHeight;
    int iDatum;
    int iHeightSystem;
    int IO_Status;
} interpolant;
```

`dLatitude` double precision variable to store interpolant latitude in decimal degrees. Must be negative for southern hemisphere.

`dLongitude` double precision variable to store interpolant longitude in decimal degrees.

`dHeight` double precision variable to store interpolant height in metres.

`iDatum` integer variable used to store user–defined datum for the interpolant coordinates.

`iHeightSystem` integer variable used to specify whether height is orthometric or ellipsoidal.

`IO_Status` integer variable used to hold the numeric switch for determining success or failure and type of failure of an interpolation call.

### typedef struct `geoid_values`

`geoid_values` is used to hold the interpolated values from the geoid grid file.

```
typedef struct {
    double dN_value;
    double dDefl_meridian;
    double dDefl_primev;
} geoid_values;
```

`dN_value` double precision variable to store interpolated ellipsoid–geoid separation (N) value.

`dDefl_meridian` double precision variable to store interpolated deflection of the vertical in the prime meridian.

`dDefl_primev` double precision variable to store interpolated deflection of the vertical in the prime vertical.

**typedef struct** `_geoid_point`

`_geoid_point` is defined to group the interpolant and interpolated values in a single variable.

```
typedef struct {
    interpolant cVar;
    geoid_values gVar;
} _geoid_point;
```

`cVar` variable to store interpolant values (see `interpolant`).

`gVar` variable to store interpolated values (see `geoid_values`).

## 2.2 Exported functions

To cater for the various geoid interpolation requirements, dnaGeoid exports the following functions:

| | |
|---|---|
| BiCubicTransformation | Interpolates geoid values using bi–cubic interpolation |
| BiLinearTransformation | Interpolates geoid values using bi–linear interpolation |
| CreateGridIndex | Opens an NTv2 geoid grid file and loads the header information into memory |
| CreateNTv2File | Creates a NTv2 geoid grid file from the legacy WINTER DAT file format |
| FileTransformation | Interpolates geoid values for a file of points using either bi–linear or bi–cubic interpolation |
| GetByteOffset | Gets the current position in (or byte offset from the beginning of) a file that is being processed. |
| ReportGridProperties | Retrieves the geoid grid file header information |
| ReturnFileProgress | Returns the progress of the file conversion/transformation process as a percentage |
| SetByteOffset | Initialises the byte offset to zero |
| Version | Retrieves the version of the library |

Explanation of these functions, including sample ANSI C/C++ code showing how to call each function is provided in §2.4.

## 2.3 Handling exceptions

For the errors that are known to occur during the run–time life of dnaGeoid, an exception handler based on the C++ Standard library has been developed. The exception handler is called `NetGeoidException` and provides an ability to react intelligently to well known errors. Given that logical/run-time errors can originate from a source external to dnaGeoid, it is recommended that a catch statement be provided for other known error types such as `exception`, `runtime_error` and `out_of_range`. Figure 2.1 shows an example of handling `NetGeoidException` and `runtime_error` references thrown by dnaGeoid.

```
try {
    // DynaNetGeoidInt method called here
}
catch (NetGeoidException& e) {
    // handle error
}
catch (runtime_error& e) {
    // handle error
}
```

Figure 2.1: Exception handling example

## 2.4 Calling the exported functions

### 2.4.1 BiCubicTransformation

**Declaration** `void BiCubicTransformation(_geoid_point* apPoint);`

Once `CreateGridIndex` (see §2.4.3) has been called to open a geoid grid file, `BiCubicTransformation` can be called at random to interpolate N values using bi–cubic interpolation, and to apply the interpolated value to the supplied height according to the direction specified by `iHeightSystem` (see `_geoid_point` in §2.1).

`BiCubicTransformation` takes one argument — a pointer to an `_geoid_point` object (see §2.1 for an explanation of the members). The function is a void function and throws `NetGeoidException` upon failure. See §2.3 for a full description on the types of exceptions that can be thrown.

Using ANSI C/C++, below is an example of calling `BiCubicTransformation` to interpolate an N value for a point and to compute an orthometric height.

```
_geoid_point gPt;
gPt.cVar.dLatitude = -17.489296814;
gPt.cVar.dLongitude = 140.833946983;
gPt.cVar.dHeight = 55.960;
gPt.cVar.iHeightSystem = 1; // current system is ellipsoidal
gPt.cVar.IO_Status = ERR_TRANS_SUCCESS;

DynaNetGeoidInt g;
try {
    g.BiCubicTransformation(&gPt, 1);
}
catch (NetGeoidException& e) {
    cout << "Error: " << e.what() << endl;
}

if (gPt.cVar.IO_Status != ERR_TRANS_SUCCESS)
    return false;

cout << "N value = " << gPt.gVar.dN_value << endl;
```

### 2.4.2 BiLinearTransformation

**Declaration** `void BiLinearTransformation(_geoid_point* apPoint);`

Once `CreateGridIndex` (see §2.4.3) has been called to open a geoid grid file, `BiLinearTransformation` can be called at random to interpolate N values using bi–linear interpolation, and to apply the interpolated value to the supplied height according to the direction specified by `iHeightSystem` (see `_geoid_point` in §2.1).

`BiLinearTransformation` takes one argument — a pointer to an `_geoid_point` object (see §2.1 for an explanation of the members). The function is a void function and throws `NetGeoidException` upon failure.

Using ANSI C/C++, below is an example of calling `BiLinearTransformation` to interpolate an N value for a point and to compute an ellipsoidal height.

```
_geoid_point gPt;
gPt.cVar.dLatitude = -17.489296814;
gPt.cVar.dLongitude = 140.833946983;
gPt.cVar.dHeight = 4.254;
gPt.cVar.iHeightSystem = 0; // current system is orthometric
gPt.cVar.IO_Status = ERR_TRANS_SUCCESS;

DynaNetGeoidInt g;
try {
    g.BiLinearTransformation(&gPt, 1);
}
catch (NetGeoidException& e) {
    cout << "Error: " << e.what() << endl;
}

if (gPt.cVar.IO_Status != ERR_TRANS_SUCCESS)
    return false;

cout << "N value = " << gPt.gVar.dN_value << endl;
```

### 2.4.3 CreateGridIndex

**Declaration** `void CreateGridIndex(const char* fileName, const char* fileType);`

Integral to interpolating from an NTv2 geoid grid file, an array of the overview header blocks for each sub grid must be built up in virtual memory. `CreateGridIndex` can be called to open a geoid grid file and build this array. The array is built each time when a new file name or file type is passed, and checks are made to determine the integrity of the entire grid file. Note that the entire grid file is not loaded, rather, only the header blocks. Thus, the performance of `CreateGridIndex` and `BiLinearTransformation/BiCubicTransformation` is completely unaffected by the size of the grid file.

`CreateGridIndex` takes two arguments — the full file path of the grid file as a char array and the grid file type as a char array (maximum of 3 characters). The function is a void function and throws `NetGeoidException` upon failure.

The gridfile header block information is retained in memory until the library or object is unloaded, or the destructor is called when a `DynaNetGeoidInt` object goes out of scope. Hence, this method only needs to be called once for the lifetime of the use of the library.

Below is an example of calling `CreateGridIndex` to open a binary grid file using ANSI C/C++.

```
char gridfilePath[401], gridfileType[4];
strcpy(gridfilePath, "/opt/geoid/ausgeoid09.gsb");
strcpy(gridfileType, "gsb");

DynaNetGeoidInt g;
try {
    g.CreateGridIndex(gridfilePath, gridfileType);
}
catch (NetGeoidException& e) {
    cout << "Error: " << e.what() << endl;
    return false;
}

cout << endl << "Grid file:\n   " << gridfilePath <<
    endl << "successfully opened." << endl;
```

### 2.4.4 CreateNTv2File

**Declaration** `void CreateNTv2File(const char* datFile, const n_file_par* grid, const n_gridfileindex* subgrid);`

`CreateNTv2File` can be called to create a binary geoid grid file in the NTv2 file format from a legacy WINTER dat file. `CreateNTv2File` takes three arguments — the full file path of the WINTER dat file as a char array, a pointer to a `n_file_par` object and a pointer to a `n_gridfileindex` object. The function is a void function and throws `NetGeoidException` upon failure.

`CreateNTv2File` uses the members in `n_file_par` and `n_gridfileindex` (see §2.1) to initialise the general parameters of the new grid file. Upon reading the contents of the WINTER dat file, `CreateNTv2File` sets the geometrical parameters for the grid file. These parameters include:

- The grid's upper, lower, western and eastern limits (see `dSlat`, `dNlat`, `dElong` and `dWlong`)
- The grid's north/south and east/west grid node separation (see `dLatinc` and `dLonginc`)
- The total number of nodes within the grid (see `lGscount`)

Hence, values written to these variables prior to calling `CreateNTv2File` will be overwritten during the creation process.

Below is an example of calling `CreateNTv2File` using ANSI C/C++.

```
char winterfilePath[401];
strcpy(winterfilePath, "/opt/geoid/ausgeoid09.dat");
n_file_par ntv2;
n_gridfileindex subgrid;

// set parameters for new ntv2 grid file
strcpy(ntv2.filename, "/opt/geoid/ausgeoid09.gsb");
strcpy(ntv2.chGs_type, "SECONDS");
strcpy(ntv2.chVersion, "1.0.0.0");
strcpy(ntv2.chSystem_f, "GDA94   ");
strcpy(ntv2.chSystem_t, "AHD_1971");
ntv2.daf = 6378137.;     // semi-major of 'from' system
ntv2.dat = 6378137.;     // semi-major of 'to' system
ntv2.dbf = 6356752.314; // semi-minor of 'from' system
```

```
ntv2.dbt = 6356752.314; // semi-minor of 'to' system

// set parameters for the subgrid
strcpy(subgrid.chSubname, "AUSGEOID");
strcpy(subgrid.chCreated, "18032010");
strcpy(subgrid.chUpdated, "18032010");

DynaNetGeoidInt g;
try {
    g.CreateNTv2File(winterfilePath, &ntv2, &subgrid);
}
catch (NetGeoidException& e) {
    cout << endl << "Error: " << e.what() << endl;
    return false;
}
cout << endl << "Grid file:\n   " << gridfilePath <<
    endl << "successfully created." << endl;
```

Users wishing to track the progress of the NTv2 file creation should consider calling `CreateNTv2File` from a separate process (or thread) and then calling `GetByteOffset` (see §2.4.6) or `ReturnFileProgress` (see §2.4.8).

### 2.4.5 FileTransformation

**Declaration** `void FileTransformation(const char* fileIn, const char* fileOut, const int& method, const int& intEllipsoidtoOrtho, const int& intDmsFlag);`

Users should call `FileTransformation` to transform a file of coordinates on one height system to another using bi–linear or bi–cubic interpolation. `FileTransformation` takes five arguments — the path to the coordinate input file, the path to the coordinate output file, a flag representing the interpolation method, a flag representing the height transformation direction, and a flag representing the format of the input coordinates. The interpolation method used will be bi–linear when `method` equals 0 (zero) and bi–cubic when `method` equals 1 (one). If `intEllipsoidtoOrtho` equals 1 (one), then the input height is assumed to be an ellipsoidal height and an orthometric height is written to the output file. The opposite is assumed when `intEllipsoidtoOrtho` equals 0 (zero). If `intDmsFlag` equals 0 (zero), then the input coordinates are assumed to be in degrees, minutes and seconds format (ddd.mmssss). If `intDmsFlag` equals 1 (one), decimal degrees format (dd.dddd) is assumed.

`FileTransformation` is a void function, and throws `NetGeoidException` upon failure. Typical exception examples include the cases when a point lies outside the grid file extents, and when the input file or output file cannot be opened for reading and writing respectively. See §2.3 for a full description on the types of exceptions that can be thrown.

`FileTransformation` uses the file extension of the input file to determine what type of file is to be read. `FileTransformation` supports two file types — formatted text files and comma separated values files. See §5 for details on the file format specification for these file types.

Once `FileTransformation` has interpolated the N value from the specified grid file, the height supplied on each line is transformed using `intEllipsoidtoOrtho` to indicate the transformation direction. To obtain the N value for each point in the file, leave the height field blank. If available, deflections in prime meridian and prime vertical will be printed after the height

field. If a point lies outside the grid file, then the output height is set to −999.999. Since `FileTransformation` is only concerned with heights, the input latitude and longitude are written directly to the output file without any alteration.

Using ANSI C/C++, below is an example of how to call `FileTransformation` to transform a formatted text file of orthometric heights to ellipsoidal heights using bi–cubic interpolation.

```
char infileName[401], outfileName[401];
strcpy(infileName, "/home/guest/data/orthometric.txt");
strcpy(outfileName, "/home/guest/data/ellipsoidal.txt");

DynaNetGeoidInt g;

try {
    FileTransformation(infileName, outfileName, 0, 0, 1);
}
catch (NetGeoidException& e) {
    cout << "Error: " << e.what() << endl;
}
```

Users wishing to track the progress of the file transformation should consider calling `FileTransformation` from a separate process (or thread) and then calling `GetByteOffset` (see §2.4.6) or `ReturnFileProgress` (see §2.4.8).

### 2.4.6 GetByteOffset

**Declaration** `const inline long GetByteOffset();`

During calls to `FileTransform` or `CreateNTv2File`, the byte offset of the input file pointer from the beginning of the file is updated each time a new record is read from the input file. Hence, the byte offset can be used to track the progress of an individual file transformation or NTv2 file creation. `GetByteOffset` can be called to retrieve the value held by the global byte offset variable.

`GetByteOffset` returns a long value, and can be called at any time while a file is being processed, or after a call made to `FileTransformation` or `CreateNTv2File` has returned. Thus, if an exception is thrown, the last point at which data was read from the file can be quickly accessed. If several files are to be transformed, `GetByteOffset` can be used to increment a total progress counter if the sum of all file sizes is known. See §2.4.9 for notes on initialising the byte offset.

Below is some pseudocode showing an example of calling `GetByteOffset`.

```
DynaNetGeoidInt g;
long lByteOffset(0);

// open a grid file using CreateGridIndex
...

// begin a new worker thread to transform a file
...

// while worker thread is alive {
    // track the position in the file
    lByteOffset = g.GetByteOffset();
```

```
    // wait
    this_thread::sleep(500);        // using boost
// }
```

### 2.4.7 ReportGridProperties

**Declaration** `void ReportGridProperties(const char* fileName, const char* fileType, n_file_par* gridProperties);`

`ReportGridProperties` can be called to retrieve all header block array information for a particular grid file. Hence, the general and geometrical parameters of a particular grid file can be assessed without overriding the currently opened file. `ReportGridProperties` takes three arguments — the full file path of the grid file as a char array, the grid file type as a char array (maximum of 3 characters) and a pointer to a `n_file_par` object (see §2.1 for details). `ReportGridProperties` is a void function and throws `NetGeoidException` upon failure.

Below is an example of calling `ReportGridProperties` using ANSI C/C++.

```
char gridfilePath[401], gridfileType[4];
strcpy(gridfilePath, "/opt/geoid/ausgeoid09.gsb");
strcpy(gridfileType, "gsb");
n_file_par ntv2;

DynaNetGeoidInt g;
try {
    g.ReportGridProperties(gridfilePath, gridfileType, &ntv2);
}
catch (NetGeoidException& e) {
    cout << "Error: " << e.what() << endl;
    return false;
}

cout << endl << "Grid properties for \"" << gridfilePath << "\":" << endl;
cout << "+ GS_TYPE  = " << ntv2.chGs_type << endl;
cout << "+ VERSION  = " << ntv2.chVersion << endl;
cout << "+ NUM_OREC = " << ntv2.iH_info << endl;
cout << "+ NUM_SREC = " << ntv2.iSubH_info << endl;
cout << "+ NUM_FILE = " << ntv2.iNumsubgrids << endl;

// number of subgrids in file (NUM_FILE)
for (int i=0; i<ntv2.iNumsubgrids; ++i)
{
    cout << "  + SUBGRID " << i << ":" << endl;
    cout << "    SUB_NAME = " << ntv2.ptrIndex[i].chSubname << endl;
    cout << "    PARENT   = " << ntv2.ptrIndex[i].chParent << endl;
    cout << "    CREATED  = " << ntv2.ptrIndex[i].chCreated << endl;
    cout << "    UPDATED  = " << ntv2.ptrIndex[i].chUpdated << endl;
    cout << "    S_LAT    = " << ntv2.ptrIndex[i].dSlat << endl;
    cout << "    N_LAT    = " << ntv2.ptrIndex[i].dNlat << endl;
    cout << "    E_LONG   = " << ntv2.ptrIndex[i].dElong << endl;
    cout << "    W_LONG   = " << ntv2.ptrIndex[i].dWlong << endl;
    cout << "    LAT_INC  = " << ntv2.ptrIndex[i].dLatinc << endl;
    cout << "    LONG_INC = " << ntv2.ptrIndex[i].dLonginc << endl;
    cout << "    GS_COUNT = " << ntv2.ptrIndex[i].lGscount << endl;
}
```

### 2.4.8 ReturnFileProgress

**Declaration** `const inline int ReturnFileProgress();`

Like `GetByteOffset`, `ReturnFileProgress` can be called to track the progress of an individual file transformation or NTv2 file creation. The return value of `ReturnFileProgress` is a percentage of the byte offset in relation to the total size of the file being processed.

`ReturnFileProgress` returns an integer value, and can be called at any time while a file is being processed, or after a call made to `FileTransformation` or `CreateNTv2File` has returned. As with `GetByteOffset`, a separate thread can be used to monitor progress.

Using ANSII C/C++, below is an example of calling `ReturnFileProgress`.

```
DynaNetGeoidInt g;
int fileProgress = g.ReturnFileProgress();
```

### 2.4.9 SetByteOffset

**Declaration** `inline void SetByteOffset();`

`SetByteOffset` initialises the global byte offset variable to zero (see §2.4.6 for an explanation of the variable's purpose). Whilst the global byte offset is initialised each time a call is made to `FileTransform`, `SetByteOffset` is provided to enable users to initialise the offset after a file has been transformed. It is not mandatory to call `SetByteOffset` but should be used if other processes are dependent upon the value of `GetByteOffset` prior to calling `FileTransform`.

Using ANSII C/C++, below is an example of calling `SetByteOffset`.

```
DynaNetGeoidInt g;
g.SetByteOffset();
```

### 2.4.10 Version

**Declaration** `static void Version(char* version);`

`Version` can be called to obtain the current version of dnaGeoid. `Version` is a static member function and can be called without instantiating the `DynaNetGeoidInt` class. Below is an example of calling `Version` using ANSI C/C++.

```
char dnageoid_version[401];
DynaNetGeoidInt::Version(dnageoid_version);

cout << "Version: " << dnageoid_version << endl;
```

The following is the output from the current version of dnaGeoid.

```
+-------------------------------------------------------------------------
+ Title:        geoid
+ Description:  Geoid Grid File (NTv2) Interpolation software.
+ Version:      1.0.3.0, Release
+ Build:        Dec  7 2011, 11:30:20 (MSVC++ 9.0)
```

```
+ Copyright:    (C) 2010 ICSM GTSC.
                This is free software released under a restricted license.
+ Contact:      nicholas.brown@ga.gov.au
+               +61 2 6249 9831
+-------------------------------------------------------------------------
```

# 3 Command–line software: geoid

geoid has been developed to simplify the use of the dnaGeoid library. Whether compiled for Windows or UNIX/Linux, geoid is a console application that can be accessed from the DOS prompt or UNIX/Linux shell. geoid may also be called from other applications using the standard syntax for calling system commands. To run geoid, simply type `geoid` on the command line with the required arguments. The command line usage and arguments are as follows. Default options are provided in parentheses.

### Usage

```
geoid [standard options] [NTv2 options] [interpolation|file options]
```

### Standard options

| | |
|---|---|
| `-i [--interpolate]` | Interpolate geoid information from command–line input coordinates. |
| `-f [--file-interpolate]` | Interpolate geoid information from a file of coordinates. |
| `-m [--interpolation-method] arg` | Interpolation method:<br>0 = Bi–linear<br>1 = Bi–cubic (default) |
| `--decimal-degrees` | Specify input coordinates in decimal degrees. Default is degrees, minutes and seconds. |
| `-c [--create-ntv2]` | Create NTv2 grid file from standard DAT file. |
| `-n [--ntv2-filepath] arg` | File path of the NTv2 grid file. |
| `-s [--summary]` | Print a summary of the grid file. |
| `-h [--help]` | Show this help message |

### NTv2 options

| | |
|---|---|
| `-d [--dat-filepath] arg` | File path of the DAT grid file. If `--create-ntv2` is supplied, this argument is mandatory. All following arguments are optional. |
| `--GS_TYPE arg (=SECONDS)` | Grid shift type (seconds or radians). |
| `--VERSION arg (=1.0.0.0)` | Grid file version. |
| `--SYSTEM_F arg (=GDA94)` | 'From' reference system. |
| `--SYSTEM_T arg (=AHD_1971)` | 'To' reference system. |
| `--MAJOR_F arg (=6378137.000)` | Semi major of 'From' system |
| `--MAJOR_T arg (=6378137.000)` | Semi major of 'To' system |
| `--MINOR_F arg (=6356752.314)` | Semi minor of 'From' system |
| `--MINOR_T arg (=6356752.314)` | Semi minor of 'To' system |
| `--SUB_NAME arg (=AUSGEOID)` | name of subgrid. |
| `--CREATED arg (=01012010)` | Date of creation. |
| `--UPDATED arg (=01012010)` | Date of last file update. |

### Interpolation options

Both `-p` and `-l` arguments are mandatory if `--interpolate` is supplied. Specify `--decimal-degrees` if input coordinates are in decimal degrees format.

| | |
|---|---|
| `-p [--latitude] arg` | Latitude of the interpolant. Default in degrees, minutes and seconds. |
| `-l [--longitude] arg` | Longitude of the interpolant. Default in degrees, minutes and seconds. |

### File interpolation options

| | |
|---|---|
| `-t [--input-file] arg` | ASCII text file path of the input coordinates and height. This argument is mandatory if `--file-interpolate` is supplied. `--direction` is optional. Specify `--decimal-degrees` if input coordinates are in decimal degrees format. |
| `-r [--direction] arg` | Conversion of heights (if supplied):<br>0 = Orthometric to ellispoid<br>1 = Ellispoid to orthometric (default) |
| `-s [--bin-stn-file] arg` | Binary station filename. If extant, populates all records within the binary station file with N value and deflections of the vertical. |
| `--convert=stn-hts` | If a user–supplied height in the binary file is orthometric, the height is converted to ellipsoidal. |

### Example Usage

To create a new NTv2 AusGeoid file using the default options, run:

```
geoid -c -n ./ausgeoid09.gsb -d ./ausgeoid09.txt
```

To create a new NTv2 AusGeoid file based upon GDA94, with a creation/updated date of 18 March 2010, using all other default options, run:

```
geoid -c -n ./ausgeoid09.gsb -d ./ausgeoid09.txt --SYSTEM_F GDA94 --CREATED 18032010
--UPDATED 18032010
```

To interpolate a single N value and corresponding deflections of the vertical for a point located at $-38.93875308°$ latitude (phi, $\varphi$) and $145.53129736°$ longitude (lambda, $\lambda$), run:

```
geoid -i -n ./ausgeoid09.gsb -p -38.93875308 -l 145.53129736 --decimal-degrees
```

To interpolate a list of N values and corresponding deflections of the vertical for a list of points (defined by latitudes and longitudes) stored within a text file "data.txt", run:

```
geoid -f -n ./ausgeoid09.gsb -t /home/guest/geoid_data/data.txt
```

With this sequence of arguments, `geoid` will create a text file named "data_out.txt". This file will contain the original list of latitudes and longitudes. If heights were supplied in the input file, the output heights will be either ellipsoidal height or orthometric height, the determination of which is based upon the argument `--direction`. During the process, N values are interpolated for each point and the output height is computed using the following formula:

$$\begin{aligned} H &= h - N \\ h &= H + N \end{aligned}$$

where $H$ is orthometric height and $h$ is ellipsoidal height.

When supplying a text file using `--file-interpolate` and `--input-file` arguments, the user may supply a formatted text file or a comma separated values file. The required format of each line supplied in formatted text files and comma separated values files is shown in §5.

# 4 Graphical User Interface software: GeoidInt

As discussed earlier, GeoidInt provides an interactive approach to geoid grid file interpolation. Figure 4.1 shows the main GeoidInt dialog. At the top-left of the dialog is the main menu. Figure 4.2 shows the items contained in the File menu. This menu can be used to select the default geoid grid file, create a NTv2 geoid grid file from a WINTER dat file, and interpolate geoid grid values for a single point. The About menu displays a simple dialog which shows information about GeoidInt. The remaining control items on the main dialog offer the same functionality provided in the File menu, and some additional controls for interpolating geoid values for a file of points.

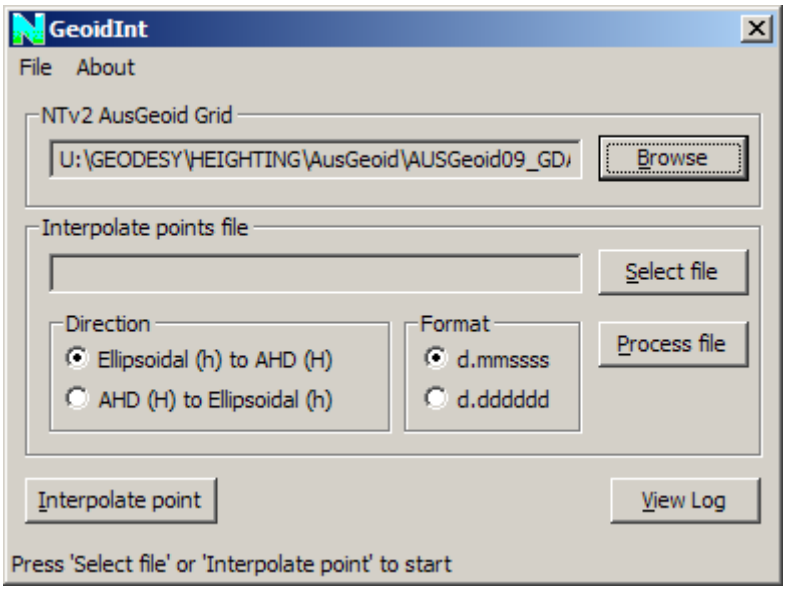



Figure 4.1: Main dialog

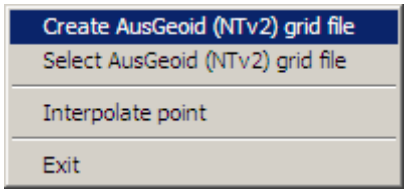



Figure 4.2: File menu

## 4.1 Interpolating geoid values for a single point

To retrieve the N value, deflection in the prime meridian and deflection in the prime vertical for a single point, either click the Interpolate point button from the main dialog or Interpolate

point from the File menu. Figure 4.3 shows the Interpolate point dialog that is displayed. Using this dialog, choose the desired format for the input coordinates. The two options are degrees, minutes and seconds, and decimal degrees. Enter the coordinates into the Latitude and Longitude edit boxes using numerical digits only with no spaces. A minus sign is required for latitudes in the southern hemisphere (as shown in Figure 4.3) or longitudes west of the zero meridian. Upon leaving the input Latitude and Longitude edit boxes, the input values will be formatted according to the chosen coordinate format. Finally, click Interpolate. The interpolated values will appear in the Output values and Astronomical coordinates group boxes on the right.

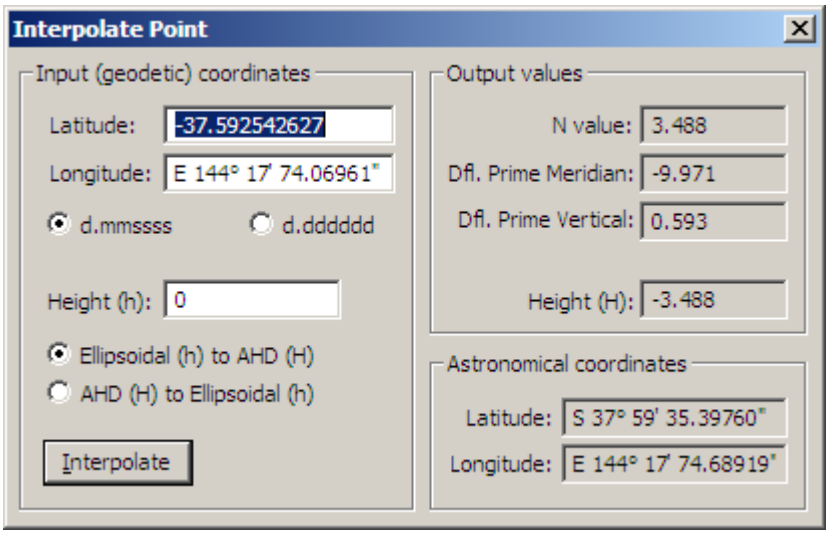



Figure 4.3: Interpolate dialog

To transform a height between the ellipsoidal and orthometric height systems, simply select the desired direction from the Ellipsoid (h) / AHD (H) radio buttons on the left, enter in the appropriate height and then click Interpolate. A height is not compulsory for interpolating from the grid file. If the output deflection values equal zero, then no data is available for that point. Each time Interpolate is clicked, the interpolant coordinates together with the interpolated values are printed to a log file. To view the log file, click View log from the main dialog.

If the interpolant lies outside the limits of the geoid grid file, the error message shown in Figure 4.4 will be displayed.

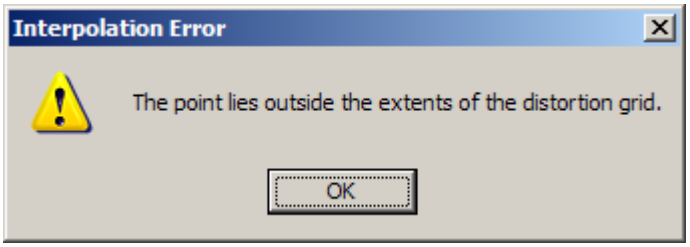



Figure 4.4: Interpolation error message — point outside the grid file limits

## 4.2 Interpolating geoid values for a file of points

To retrieve the N value for a list of points contained within a file:

1. Click the Select file button from the main dialog
2. From the Look in box, select the drive or folder that contains the input file.
3. In the Files of type box, choose the appropriate input file-type[1]
4. In the folder list, double–click folders until the folder that contains the required file is opened.
5. Double–click the required input file.
6. If a file of Ellipsoidal heights is to be transformed to a file of AHD heights, select the Elllipsoid (h) to AHD (H) radio button from the main dialog.
7. Click the Process file button.

Upon transforming a points file, the output file name will be the same as the input file name with a "_out" inserted between the file name and the file extension.

During the transformation process, the height supplied on each line is transformed according to the transformation direction. To obtain the N value for each point in the file, leave the height field blank. If available, deflections in prime meridian and prime vertical will be printed after the height field. If a point lies outside the grid file, then the output height is set to −999.999. Since GeoidInt is only concerned with heights, the input latitude and longitude are written directly to the output file without any alteration. The interpolation method used in the file transformation is bi–cubic by default and cannot be changed.

[1]The available types include Formatted Text (*.dat, *.prn, *.txt) and Comma Separated Files (*.csv) files formatted according to the specifications outlined in Chapter 5.

# 5 File Format Specification

GeoidInt supports Formatted Text files (e.g. *.dat, *.prn, *.txt) and Comma Separated Values files (*.csv). GeoidInt expects all input coordinates in both file formats to be geographic coordinates in degrees, minutes and seconds unless otherwise specified. In this form, the latitude and longitude fields should each contain only one numeric value. When working with formatted text files, the maximum number of significant digits the values can have is 15 significant figures[1]. For latitudes in the southern hemisphere and longitudes west of the zero meridian, the number of significant figures is further reduced by 1 to cater for the minus sign.

## Formatted text files

Every line in a formatted text file must contain data fields in particular file positions (or columns). Certain fields may be omitted depending on what they are. Table 5.1 lists the compulsory and non-compulsory fields in the required order. Figure 5.1 shows an example formatted text file and illustrates the use of the non–compulsory fields. Column numbers shown for reference only.

| Field | Columns | Characters | Compulsory? |
|---|---|---|---|
| Point ID | 1 – 11 | 11 | No |
| Latitude | 12 –27 | 16 | Yes |
| Longitude | 28 – 43 | 16 | Yes |
| Height | 44 – 52 | 9 | No |

Table 5.1: Formatted text file fields

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
----------><--------------><--------------><-------><
Point (11) Latitude (16)   Longitude (16)  Hght (9)
MT HIGH        -27.498408428   153.001072611
               -27.498421786   150.001124192
               -29.086179181   151.966654878
               -29.073486997   151.805272886   4.23
62 / 54        -29.000294436   151.457723186
GBM16          -28.636707072   151.970252700
GBM34          -28.619868617   151.650131492
               -28.235994419   151.990397797  36.281
```

Figure 5.1: Example formatted text file

[1]Since the maximum field width for both latitude and longitude is 16 characters, excluding the decimal point leaves a maximum of 15 characters.

## Comma separated values files

Every line in a CSV file must contain data fields separated by commas. Non-compulsory fields may be empty, however a sufficient number of commas must be present to delineate the presence of compulsory fields. Table 5.2 lists the compulsory and non-compulsory fields in the required order.

| Field | Compulsory? |
|---|---|
| Point ID | No |
| Latitude | Yes |
| Longitude | Yes |
| Height | No |

Table 5.2: Comma separated values file fields

According to Table 5.2, a minimum of two commas is sufficient to delineate Point ID, Latitude and Longitude. Figure 5.2 shows an example CSV file. Note that a header line is not required.

```
MT HIGH,-10.498408428,153.001072611
,-20.498421786,140.001124192
,-40.086179181,121.966654878,
1596-4      ,    -37.593644101,    144.204321339
1596-5      ,    -37.593616320,    144.204318245
62 / 54,-40.000294436,141.457723186, 4.23
GBM16,-30.636707072,151.970252700,
GBM34,-20.619868617,141.650131492
4,-20.235994419,121.99039779,36.281
```

Figure 5.2: Example comma separated values file

The simplest way to create a CSV file is to use a spreadsheet program such as Microsoft Excel or OpenOffice Calc. To create a CSV file using a spreadsheet program, perform the following:

1. Create a new spreadsheet
2. If available, enter in the unique identifier for each point in column A.
3. Enter in values for latitude and longitude in columns B and C.
4. Enter in a height in column D. Leave this column blank if geoid values are required.
5. Click Save As... from the File menu
6. In the Save as type box, click CSV (Comma delimited) (*.csv).
7. Click Save.

As an example, Figure 5.3 shows two records for a CSV file (to be created) in decimal degrees using Microsoft Excel.

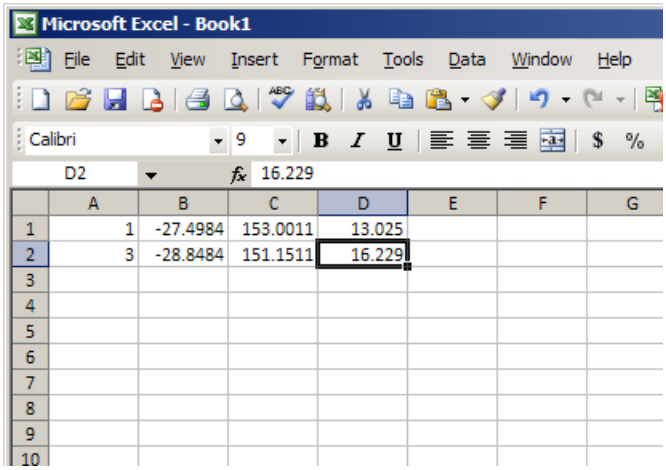



Figure 5.3: Using Excel to create a CSV file in decimal degrees