GCalc 2.1

GDA94 Datum Transformation DLL

Technical Specification

12/08/2016

This publication has been compiled by Darren Burns of Cadastral and Geodetic Services, Department of Natural Resources and Mines.

Table of contents

1. INTRODUCTION 1

1.1 Overview 1

1.2 GCalc DLL Exports 1

1.3 A Note on Bi-Linear Interpolation and Distortion Grids 3

1.4 Supported Coordinate Files 3

2. Structure Type Declarations 5

2.1 typedef struct coordinate 5

2.2 typedef struct pointsfile 6

3. Function Declarations 9

3.1 ANSI C and C++ 9

3.2 Visual Basic 9

4. Calling the Exported Functions 10

4.1 Coordinate Conversion Functions 10

4.1.1 GCALC_DmstoDeg 10

4.1.2 GCALC_DmintoDeg 10

4.1.3 GCALC_DegtoDms 10

4.1.4 GCALC_DegtoDmin 11

4.1.5 GCALC_GeotoGrid 11

4.1.6 GCALC_GridtoGeo 12

4.1.7 GCALC_CarttoGeo 13

4.1.8 GCALC_GeotoCart 14

4.2 Datum Transformation Functions 15

4.2.1 GCALC_CreateGridIndex 15

4.2.2 GCALC_BiLinearTransformation (High Accuracy) 16

4.2.3 GCALC_MapGridTransformation (High Accuracy) 18

4.2.4 GCALC_PointTran (High Accuracy) 19

4.2.5 GCALC_FileTran (High Accuracy) 20

4.2.6 GCALC_DatumTran (Medium Accuracy) 24

4.3 Data Output Functions 26

4.3.1 GCALC_PrintLogFile 26

4.3.2 GCALC_SetFileTranHeader 27

4.4 Progress Functions 27

4.4.1 GCALC_SetByteOffset 28

4.4.2 GCALC_GetByteOffset 28

5. ERROR RETURN STATUS 29

6. USING GCALC.DLL IN YOUR APPLICATION 32

6.1 Win32 Console (ANSI C) and MFC AppWizard C++ Projects 32
6.2 Visual Basic Projects 32
6.3 Recommended Function Calls 33
APPENDIX A – DAT, TXT, PRN and CSV FILE FORMAT SPECIFICATION 34
A.1 Formatted Text Files (*.dat, *.txt, *.prn) 34
A.2 Comma Separated Files (*.csv) 36
A.3 SCDB (MAN) Files 38
A.4 Queensland Interchange Format (QIF) Files 38
A.5 NEWGAN (RES) Files 39
APPENDIX B – NTv2 GRID SHIFT FILE FORMAT 40
B.1 NTv2 Grid Shift File Concept 40
B.2 NTv2 File Format 42
B.2.1 Grid Shift File Overview 43
B.2.2 Sub Grid Format 44
B.3 Necessary Considerations for NTv2 Usage 45
B.3.1 Difference Between Australian and Canadian Binary Files 46
B.3.2 Implications of Using NTv2 Outside Canada 47
References 49
APPENDIX C – Gcalc SOFTWARE LICENSE AGREEMENT 50

Table of figures

Figure A1 – Using Excel to create a CSV file in Decimal Degrees 37
Figure B1 – Grid Node Order 40
Figure B2 – Grid Shift file with a single sub grid 41
Figure B3 – Grid Shift file with a multiple sub grids 41
Figure B4 – Densified sub grid structure 42
Figure B5 – Sample Grid Shift File Overview 44
Figure B6 – Sample Sub Grid Overview 45
Figure B7 – Sample Sub Grid Node Values 45
Figure B8 – NTv2 File Structure 46
Figure B9 – Data storage in a Header record 47
Figure B10 – Longitude Axis Orientation 48

Table of tables

Table 1a – GCalc Exported Functions 2
Table 2a – Formatting of Separated and HP Notation values 7

Table 4a – GCALC_SetFileTranHeader Argument Options 27
Table 5a – GCALC_CreateGridIndex 29
Table 5b – GCALC_BiLinearTransformation and GCALC_MapGridTransformation 30
Table 5c – GCALC_PointTran 30
Table 5d – GCALC_FileTran 31
Table A1 – DAT File Compulsory Items 35
Table A2 – CSV File Compulsory Items 36
Table B1 – Grid Shift File Data Types 43
Table B2 – Grid Shift File Overview Information 43
Table B3 – Sub Grid Overview Information 44
Table B4 – Sub Grid Node Values 45

1. INTRODUCTION

1.1 Overview

In November 1995, the Intergovernmental Committee on Surveying and Mapping (ICSM) recommended the progressive Australia-wide implementation of a new national geodetic datum to be introduced by January 1, 2000. This datum is known as the Geocentric Datum of Australia (GDA).

In recognition of the vast number of spatial data users across Queensland wishing to transform their coordinates to the new datum, the Department of Natural Resources and Mines (DNRM) developed a datum transformation DLL titled "GCalc" that runs under 32-Bit and 64-Bit versions of Windows. The release of version 2.1 contains fixes to known bugs and additional exported functions.

GCalc can be used to transform coordinates between the Australian Geodetic Datum (AGD66 or AGD84) and the Geocentric Datum of Australia, 1994 (GDA94). GCalc supports both *High Accuracy* datum transformations (Bi-linear interpolation from a grid of coordinate shifts) and *Medium Accuracy* datum transformations (conformal 7-parameter transformation). Since the application of bi-linear interpolation is directly related to the region covered by the distortion grid, GCalc may be used to transform coordinates for any location throughout Australia provided an appropriate grid is selected.

With GCalc, users can perform datum transformations and coordinate conversions using geographic coordinates (i.e. Latitude and Longitude) or projection coordinates (i.e. Easting, Northing and Zone). Coordinate conversions may be performed whether working on one datum or between AGD66 or AGD84 and GDA94.

1.2 GCalc DLL Exports

GCalc is a regular Win32-Bit DLL. It can be called by 32-bit and 64-bit C, C++ and Visual Basic applications, including applications in other languages such as Pascal or FORTRAN. To date, it has been tested only with Visual C++ and Visual Basic compilers. Table 1a describes the exported functions.

Calls to `GCALC_GridtoGeo`, `GCALC_GeotoGrid`, `GCALC_GeotoCart`, `GCALC_CarttoGeo`, `GCALC_DatumTran`, `GCALC_PointTran`, `GCALC_FileTran`, `GCALC_BiLinearTransformation` and `GCALC_MapGridTransformation` require the use of pointers to structures as defined in GCalcext.h. The equivalent Declare statements for use in Visual Basic are given in “GCalcext.bas”. If you are not using pointers, the address-of-operator “&” may be used instead (i.e. the value at the address of the structure to which the operand points).

The components making up each structure are detailed in *2. STRUCTURE TYPE DECLARATIONS*. Examples of the function declarations required for ANSI C and Visual Basic applications are given in GCalcext.h and GCalcext.bas respectively. Examples of calling the exported functions are provided throughout *4. CALLING THE EXPORTED FUNCTIONS* and are given in both ANSI C and Visual Basic. The various error return types are discussed in *5. ERROR RETURN STATUS*. The procedure for linking GCalc to C / C++ and Visual Basic applications are discussed in *6. USING GCALC.DLL IN YOUR APPLICATION*.

This document includes information on the following files:

1. GCalc.dll — The Dynamic Link Library (DLL).
2. GCalcext.h — Header file containing structure and function declarations for ANSI C and C++ applications
3. GCalcext.bas — BAS module containing structure and procedure declarations for Visual Basic applications.
4. GCalc.lib — C Library file for GCalc.dll

Table 1a – GCalc Exported Functions

Function name	Description
`GCALC_DmstoDeg`	Converts degrees, minutes and seconds to decimal degrees
`GCALC_DmintoDeg`	Converts degrees and minutes to decimal degrees
`GCALC_DegtoDms`	Converts decimal degrees to degrees, minutes and seconds
`GCALC_DegtoDmin`	Converts decimal degrees to degrees and minutes
`GCALC_GridtoGeo`	Converts projection to geographic coordinates
`GCALC_GeotoGrid`	Converts geographic to projection coordinates
`GCALC_GeotoCart`	Converts geographic to earth-centred cartesian coordinates
`GCALC_CarttoGeo`	Converts earth-centred cartesian to geographic coordinates
`GCALC_DatumTran`	Transforms cartesian coordinates between AGD and GDA94 using conformal 7-Parameter transformation
`GCALC_PointTran`	Transforms geographic coordinates for a single point between AGD84 and GDA94 using bi-linear interpolation from a distortion grid
`GCALC_FileTran`	Transforms a file of coordinates between AGD84 and GDA94 using bi-linear interpolation from a distortion grid
`GCALC_CreateGridIndex`	Opens the specified distortion grid file and reads in the header information. An array of sub grid header information is built from the entire file.
`GCALC_BiLinearTransformation`	Similar to `GCALC_PointTran`, however `GCALC_CreateGridIndex` must be called first. May be called infinitely to transform geographic coordinates without necessitating the grid file to be re-opened each time
`GCALC_MapGridTransformation`	Similar to `GCALC_BiLinearTransformation`, however is used to transform projection coordinates
`GCALC_PrintLogFile`	A simple log-file-print facility
`GCALC_SetFileTranHeader`	Specifies whether a header is printed to the output file
`GCALC_SetByteOffset`	Initialises the byte offset counter (for files being read) to zero

`GCALC_GetByteOffset`	Returns the byte offset of the file pointer location in the opened file

1.3 A Note on Bi-Linear Interpolation and Distortion Grids

The recommended approach in Australia for transforming coordinates between AGD66/AGD84 and GDA94 is to use **bi-linear interpolation**. This is a *High Accuracy* approach, requiring a standard grid of coordinate differences that model the AGD66/AGD84 to GDA94 transformation and AGD66/AGD84 network distortions. The Canadian National Transformation Version 2 (NTv2) is adopted as the standard file format for the grid of coordinate differences. Appendix B provides a detailed explanation of the NTv2 grid shift file format.

The latest version of the Queensland grid file is installed in the same folder as the GDAy software and is titled "QLD_0900.gsb". This replaces the existing grid “QLD_0400.gsb”. All Australian States and Territories currently have a grid file(s) covering their area. These grid files will transform from either AGD66 or AGD84, depending on which version of AGD was previously adopted by that jurisdiction. For instance, "QLD_0900.gsb" can only be used to transform coordinates between AGD84 and GDA94.

GCalc can be used to transform coordinates using any grid file in the NTv2 format developed for transforming coordinates in Australia between AGD66/AGD84 and GDA94.

For a complete coverage of Australia, you can download the latest national distortion grid files from:

National AGD66 to GDA94:

3.10 MB download file:

http://www.icsm.gov.au/gda/national66.zip

National AGD84 to GDA94:

7.74 MB download file:

http://www.icsm.gov.au/gda/national84.zip

If this method of obtaining the required grid file is inconvenient, contact your state’s Land Administration Agency for assistance. Note that the Queensland grid QLD_0900.gsb produces identical results to the National AGD84 grid for the equivalent coverage.

For further information relating to the technical standard and implementation issues of transforming to GDA94, visit ICSM's Geocentric Datum of Australia website:

http://www.icsm.gov.au/gda/

1.4 Supported Coordinate Files

GCalc supports the transformation of coordinates in Formatted Text files (*.dat, *.prn, *.txt), Comma Separated Values files (*.csv), Survey Control DataBase (MAN) files (*.man), Queensland Interchange Format (QIF) files (*.qif), and NEWGAN Res files (*.res). In an effort to make GCalc compatible with other transformation software produced in Australia, the Formatted Text and Comma

Separated file formats required by GCalc follows the format specified by GDAit. Examples of Formatted Text and Comma Separated file formats are shown in Appendix A. For the technical specifications on the format of MAN and QIF files, contact Cadastral and Geodetic Data, Department of Natural Resources and Mines, QLD.

2. Structure Type Declarations

2.1 typedef struct coordinate

The structure type definition using ANSI C and C++ is as follows:

```
typedef struct {
	double dNum1;
	double dNum2;
	double dNum3;
	double dLatacc;
	double dLongacc;
	int iNum4;
	int IO_Status;
} coordinate;
```

The equivalent Declare statements using Visual Basic are shown in “GCalcext.bas”.

dNum1 - Latitude (must be negative for southern hemisphere), Easting, or X.

dNum2 - Longitude, Northing, or Y.

dNum3 - Zone, ellipsoidal height, or Z. dNum3 is referenced only when calls are made to `GCALC_GridtoGeo`, `GCALC_GeotoGrid`, `GCALC_CarttoGeo`, `GCALC_GeotoCart` and `GCALC_DatumTran`.

dLatacc - Latitude interpolation accuracy. If an accuracy value is not available, this is set to -1000.0.

dLongacc - Longitude interpolation accuracy. If an accuracy value is not available, this is set to -1000.0.

iNum4 - A 2-way flag to indicate the transformation direction or reference ellipsoid. For example, where calls are made to `GCALC_PointTran`, `GCALC_BiLinearTransformation` or `GCALC_MapGridTransformation`, specify `iNum4` = 0 to indicate the direction from AGD to GDA or `iNum4` = 1 to indicate GDA to AGD. Where calls are made to `GCALC_GeotoGrid`, `GCALC_GridtoGeo`, `GCALC_GeotoCart` or `GCALC_CarttoGeo`, specify `iNum4` = 0 to indicate that the coordinates are on the Australian National Spheroid (ANS), the reference system for AGD, or `iNum4` = 1 for the Global Reference System of 1980 (GRS80), the reference system for GDA.

IO_Status - Status type for transformation success / failure. Zero is assigned for success, whilst a number from 1 to 19 is used to indicate the type of failure (see 5. ERROR RETURN STATUS).

The structure type `coordinate` is used to declare the structure arguments for:

```
GCALC_GridtoGeo
GCALC_GeotoGrid
GCALC_GeotoCart
GCALC_CarttoGeo
GCALC_DatumTran
GCALC_PointTran
GCALC_BiLinearTransformation
GCALC_MapGridTransformation
```

2.2 typedef struct pointsfile

The structure type `pointsfile` is used to declare the arguments for `GCALC_FileTran`, which is used to perform a file transformation. The structure type definition using ANSI C and C++ syntax is as follows:

```
typedef struct {
       char chAppName[31];
       char chFilein[601];
       char chFInputType[4];
       char chFileout[601];
       char chFOutputType[4];
       char chLogFile[601];
       char chAccessMode[3];
       int InputHP;
       int InputDMS;
       int InputDatum;
       int InputProj;
       int OutputHP;
       int OutputDMS;
       int OutputDatum;
       int OutputProj;
       long lNumTrans[4];
       double dParameters[8];
       int IO_Status;
} pointsfile;
```

The equivalent Declare statements using Visual Basic is shown in “GCalcext.bas”.

chAppName - The name or brief description of the application calling GCalc. This is printed in the header block of all DAT and CSV output files.

chFilein - The full file-path for the input file of coordinates to be transformed.

chFInputType – Three character input file type. The available types are *.dat, *.prn, *.txt, *.csv, *.man, *.qif and *.res.

chFileout - The full file-path for the output file of coordinates to be transformed.

chFOutputType – Three character output file type. The available types are *.dat, *.prn, *.txt, *.csv, *.man, *.qif, and *.res.

chLogFile - The full file-path for the log file. All information relating to the transformation success (as held by `IO_Status`) of each record in the input file is printed to the log file. Set the first element in the char array to NULL if a log file is not desired. For example:

```
       pointsfile_variable->chLogFile[0] = '\0';
```

chAccessMode - The access mode for the log file. This should be set to either "w" or "a" for creating a new file for writing or appending to an existing file respectively.

InputHP - Two-way flag to indicate whether the input file is in separated degrees minutes and seconds format / degrees and minutes format, or HP notation (i.e. 0 = Separated DMS/DMIN fields, 1 = DMS/DMIN HP Notation). If the file is not in DMS/DMIN format (i.e. where data is in decimal

degrees or as projection coordinates), this flag is ignored. Table 2a shows how a value of –27° 02' 48.32156" and –27° 02.805359' should appear as DMS/DMIN HP Notation from Separated DMS/DMIN fields and HP Notation.

InputDMS - Three-way flag to indicate whether the input file is in degrees minutes and seconds; degrees and minutes; or decimal degrees (i.e. 0 = DMS, 1 = DMIN, 2 = DDegrees).

Table 2a – Formatting of Separated and HP Notation values

Separated DMS/DMIN fields	DMS/DMIN HP Notation
-27 02 48.32156	–27.024832156
-27 02.8053593	–27.028053593

InputDatum - Two-way flag to indicate whether the input file of coordinates is in AGD or GDA (i.e. 0 = AGD, 1 = GDA). For example, specify `iNum4` = 0 to indicate the direction from AGD to GDA and `iNum4` = 1 to indicate GDA to AGD. All points within a file must be on the one datum.

InputProj - Two-way flag to indicate whether the input coordinates are geographic (latitude and longitude) or projection (easting, northing and zone). Specify 0 for Geographic and 1 for Projection.

OutputHP - Two-way flag to indicate whether the output file is in separated degrees minutes and seconds format / degrees and minutes format, or HP notation (i.e. 0 = Separated DMS/DMIN fields, 1 = HP Notation). If the file is not in DMS/DMIN format (i.e. where data is in decimal degrees or as projection coordinates), this flag is ignored. See `InputHP` for an example of how a value should appear in a file.

OutputDMS - Three-way flag to indicate whether the output file is in degrees minutes and seconds; degrees and minutes; or decimal degrees (i.e. 0 = DMS, 1 = DMIN, 2 = DDegrees).

OutputDatum - Two-way flag to indicate whether the output file of coordinates is in AGD or GDA (i.e. 0 = AGD, 1 = GDA). For example, specify `iNum4` = 0 to indicate the direction from AGD to GDA and `iNum4` = 1 to indicate GDA to AGD. All points are printed to the file on one datum.

OutputProj - Numeric flag to indicate whether the output coordinates are geographic (latitude and longitude) or projection (easting, northing and zone). Specify 0 for Geographic, or 1 or 47 < x < 58 for Projection. For example, if 0 is specified, geographic coordinates are produced. If 1 is specified, the zone will be calculated for each point. If a number between 47 and 58 is specified, this zone number will be used to force `GCALC_GeotoGrid` to produce projection coordinates in the specified zone.

lNumTrans – An int array that holds the number of successful transformations, the number of records that hold valid numeric input, and the number of transformations performed using the distortion grid. These values are held as follows: `lNumTrans[0]` = no of success; `lNumTrans[1]` = no valid records read, and `lNumTrans[2]` = no of transformations using distortion grid. If plain text is contained within the input file, `lNumTrans[1]` is not incremented, but printed to the log file.

dParameters – A double array that holds the 7 Parameters to be used when a point lies outside the distortion grid. The indexing of the 7 Parameters for transforming 3-D Cartesian coordinates is as follows:

`dParameters[0]` = Origin translation in X-Axis (metres).
`dParameters[1]` = Origin translation in Y-Axis (metres).
`dParameters[2]` = Origin translation in Z-Axis (metres).
`dParameters[3]` = Rotation in X-Axis (seconds).
`dParameters[4]` = Rotation in Y-Axis (seconds).
`dParameters[5]` = Rotation in Z-Axis (seconds).
`dParameters[6]` = Scale factor (ppm).

For a list of AGD84 and Regional AGD66 transformation parameters for use in Australia, see the GDA Technical Manual:

http://www.icsm.gov.au/gda/gda-v_2.4.pdf

IO_Status - holds status type for transformation success / failure. Zero is assigned for success, whilst a number from 1 to 19 is used to indicate the type of failure (see section "5 ERROR RETURN STATUS").

IMPORTANT All points within an input file must be on the same datum (as held by `InputDatum`) and be of the same coordinate type (as held by `InputProj`). If the coordinates are geographic, all input values must be of the same format (as held by `InputDMS` and `InputHP`).

3. Function Declarations

3.1 ANSI C and C++

In order for ANSI C or C++ applications to call the functions exported by GCalc.dll, the functions must be declared in a standard C or C++ Header file and included in your project or workspace.

See GCalcext.h for function declarations in ANSI C.

3.2 Visual Basic

To call the functions exported by GCalc.dll from Visual Basic applications, the functions must be declared as DLL procedures in a BAS Module of your project or in the Declarations section of the code window. See GCalcext.bas in the GDA SDK for the standard function declarations in Visual Basic.

Note that certain exported functions are declared as a **Function**, whilst others are declared as a **Sub**. This is because the DLL procedures declared as a Function return a value and the procedures declared as a Sub do not.

If you use “GCalcext.bas” as a standard module, the GCalc DLL procedures are declared public by default and can be called from anywhere in your application. GCalc DLL procedures declared in any other type of module are private to that module, and you must identify them as such by preceding the declaration with the **Private** keyword.

Where the DLL filename “GCalc.dll” is specified, the full file path may be required.

4. Calling the Exported Functions

This section details how you may call the functions exported by GCalc. Examples of how to call each function using ANSI C, C++ and Visual Basic are provided.

4.1 Coordinate Conversion Functions

4.1.1 GCALC_DmstoDeg

`GCALC_DmstoDeg` can be called to convert a geographic coordinate value in degrees, minutes and seconds format to decimal degrees. `GCALC_DmstoDeg` takes one argument; a double precision value in degrees, minutes and seconds. The function returns a double precision value.

Below is an example of calling `GCALC_DmstoDeg` to convert the double value `dNum1` using ANSI C and C++.

```
double dNum1 = 25.243248642;
dNum1 = GCALC_DmstoDeg ( dNum1 );
```

Below is an equivalent example of calling `GCALC_DmstoDeg` using Visual Basic.

```
Dim dNum1 As Double
dNum1 = 25.243248642
dNum1 = GCALC_DmstoDeg ( dNum1 )
```

4.1.2 GCALC_DmintoDeg

`GCALC_DmintoDeg` can be called to convert a geographic coordinate value in degrees and minutes format to decimal degrees. `GCALC_DmintoDeg` takes one argument; a double precision value in degrees and minutes. The function returns a double precision value.

Below is an example of calling `GCALC_DmintoDeg` to convert the double value `dNum1` using ANSI C and C++.

```
double dNum1 = 25.245414403;
dNum1 = GCALC_DmintoDeg ( dNum1 );
```

Below is an equivalent example of calling `GCALC_DmintoDeg` using Visual Basic.

```
Dim dNum1 As Double
dNum1 = 25.245414403
dNum1 = GCALC_DmintoDeg ( dNum1 )
```

4.1.3 GCALC_DegtoDms

`GCALC_DegtoDms` can be called to convert a geographic coordinate value in decimal degrees to degrees, minutes and seconds format. `GCALC_DegtoDms` takes one argument; a double precision value in decimal degrees. The function returns a double precision value.

Below is an example of calling GCALC_DegtoDms to convert the double value dNum1 using ANSI C and C++.

```
double dNum1 = 25.409024006;
dNum1 = GCALC_DegtoDms ( dNum1 );
```

Below is an equivalent example of calling GCALC_DegtoDms using Visual Basic.

```
Dim dNum1 As Double
dNum1 = 25.409024006
dNum1 = GCALC_DegtoDms ( dNum1 )
```

4.1.4 GCALC_DegtoDmin

GCALC_DegtoDmin can be called to convert a geographic coordinate value in decimal degrees to degrees and minutes format. GCALC_DegtoDmin takes one argument; a double precision value in decimal degrees. The function returns a double precision value.

Below is an example of calling GCALC_DegtoDmin to convert the double value dNum1 using ANSI C and C++.

```
double dNum1 = 25.409024006;
dNum1 = GCALC_DegtoDmin ( dNum1 );
```

Below is an equivalent example of calling GCALC_DegtoDmin using Visual Basic.

```
Dim dNum1 As Double
dNum1 = 25.409024006
dNum1 = GCALC_DegtoDmin ( dNum1 )
```

4.1.5 GCALC_GeotoGrid

GCALC_GeotoGrid can be called to convert geographic coordinates of latitude and longitude (stored within a structure of type coordinate) to Universal Transverse Mercator (UTM) projection coordinates of easting, northing and zone. GCALC_GeotoGrid takes one argument; a pointer to coordinate structure. GCALC_GeotoGrid is a void function and the input latitude and longitude values must be stored in decimal degrees format.

GCALC_GeotoGrid performs projection calculations using either the AGD or GDA, which is specified using the 2 way flag iNum4 as described in *2.1 typedef struct coordinate*. When GCALC_GeotoGrid is called, this flag is used to assign the appropriate semi-major axis and inverse flattening of the ellipsoid to be used (i.e. ANS or GRS80).

dNum3 is used to determine whether a zone should be computed, or whether the coordinates will be computed on a user specified zone. For example, if a number between 47 and 58 is specified, the easting and northing will be computed relative to that zone. If any other number is specified, the zone will be calculated. dNum3 must be initialised prior to calling GCALC_GeotoGrid.

Below is an example of creating a pointer to a coordinate structure named cVar1 and converting the contained geographic coordinates to projection coordinates on the AGD using ANSI C and C++.

```
coordinate cVar1;
coordinate *pVar1;
pVar1 = &cVar1;

pVar1->dNum1 = -25.409024006;    /* latitude (neg south) */
pVar1->dNum2 = 148.037323739;    /* longitude */
pVar1->dNum3 = 1;                /* calculate zone */
pVar1->iNum4 = 0;                /* 0 = AGD (calc's on the ANS) */

GCALC_GeotoGrid ( pVar1 );
```

Alternatively, if you are not using pointers you can write:

```
coordinate cVar1;

cVar1.dNum1 = -25.409024006;
cVar1.dNum2 = 148.037323739;
cVar1.dNum3 = 1;
cVar1.iNum4 = 0;

GCALC_GeotoGrid ( &cVar1 );      /* note the 'address-of' operator */
```

Below is an equivalent example of calling `GCALC_GeotoGrid` using Visual Basic.

```
Dim cVar1 As coordinate

cVar1.dNum1 = -25.409024006
cVar1.dNum2 = 148.037323739
cVar1.dNum3 = 1
cVar1.iNum4 = 0

Call GCALC_GeotoGrid ( cVar1 )
```

4.1.6 GCALC_GridtoGeo

`GCALC_GridtoGeo` can be called to convert a Universal Transverse Mercator (UTM) projection of easting, northing and zone (stored within a structure of type `coordinate`) to geographic coordinates of latitude and longitude. `GCALC_GridtoGeo` takes one argument; a pointer to `coordinate` structure. `GCALC_GridtoGeo` is a void function and the calculated latitude and longitude values are in decimal degrees format.

`GCALC_GridtoGeo` performs projection calculations using either the AGD or GDA, which is specified using the 2 way flag `iNum4` as described in *2.1 typedef struct coordinate*. When `GCALC_GridtoGeo` is called, this flag is used to assign the appropriate semi-major axis and inverse flattening of the ellipsoid to be used (i.e. ANS or GRS80).

Below is an example of creating a pointer to a `coordinate` structure named `cVar1` and converting the contained projection coordinates to geographic coordinates on the GDA using ANSI C and C++.

```
coordinate cVar1;
coordinate *pVar1;
pVar1 = &cVar1;
```

```
pVar1->dNum1 = 604330.847;        /* easting    */
pVar1->dNum2 = 7189345.370;       /* northing */
pVar1->dNum3 = 55.0;              /* zone */
pVar1->iNum4 = 1;                 /* 1 = GDA   (calc's on the GRS80) */

GCALC_GridtoGeo ( pVar1 );
```

Alternatively, if you are not using pointers you can write:

```
coordinate cVar1;

cVar1.dNum1 = 604330.847;
cVar1.dNum2 = 7189345.370;
cVar1.dNum3 = 55.0;
cVar1.iNum4 = 1;

GCALC_GridtoGeo ( &cVar1 );       /* note the 'address-of' operator */
```

Below is an equivalent example of calling `GCALC_GridtoGeo` using Visual Basic.

```
Dim cVar1 As coordinate

cVar1.dNum1 = 604330.847
cVar1.dNum2 = 7189345.370
cVar1.dNum3 = 55.0
cVar1.iNum4 = 1

Call GCALC_GridtoGeo ( cVar1 )
```

4.1.7 GCALC_CarttoGeo

`GCALC_CarttoGeo` can be called to convert earth centred Cartesian coordinates of X, Y and Z (stored within a structure of type `coordinate`) to geographic coordinates of Latitude and Longitude. `GCALC_CarttoGeo` takes one argument; a pointer to `coordinate` structure. `GCALC_CarttoGeo` is a void function and the calculated latitude and longitude values are in decimal degrees format.

`GCALC_CarttoGeo` performs reference system calculations using either the AGD or GDA, which is specified using the 2 way flag `iNum4` as described in *2.1 typedef struct coordinate*. When `GCALC_CarttoGeo` is called, this flag is used to assign the appropriate semi-major axis and inverse flattening of the ellipsoid to be used (i.e. ANS or GRS80).

Below is an example of creating a pointer to a `coordinate` structure named `cVar1` and converting earth centred Cartesian coordinates to geographic coordinates on the AGD using ANSI C and C++.

```
coordinate cVar1;
coordinate *pVar1;
pVar1 = &cVar1;

pVar1->dNum1 = -4890765.512;          /* earth-centred X coordinate */
pVar1->dNum2 = 3051661.332;           /* earth-centred Y coordinate */
pVar1->dNum3 = -2720079.994;          /* earth-centred Z coordinate */
pVar1->iNum4 = 0;                     /* 0 = AGD (calc's on the ANS) */
```

```
GCALC_CarttoGeo ( pVar1 );
```

Alternatively, if you are not using pointers you can write:

```
coordinate cVar1;

cVar1.dNum1 = -4890765.512;
cVar1.dNum2 = 3051661.332;
cVar1.dNum3 = -2720079.994;
cVar1.iNum4 = 0;

GCALC_CarttoGeo ( &cVar1 );      /* note the 'address-of' operator */
```

Below is an equivalent example of calling `GCALC_CarttoGeo` using Visual Basic.

```
Dim cVar1 As coordinate

cVar1.dNum1 = -4890765.512
cVar1.dNum2 = 3051661.332
cVar1.dNum3 = -2720079.994
cVar1.iNum4 = 0

Call GCALC_CarttoGeo ( cVar1 )
```

4.1.8 GCALC_GeotoCart

`GCALC_GeotoCart` can be called to convert geographic coordinates of Latitude and Longitude (stored within a structure of type `coordinate`) to earth centred Cartesian coordinates of X, Y and Z. `GCALC_GeotoCart` takes one argument; a pointer to `coordinate` structure. `GCALC_GeotoCart` is a void function and the input latitude and longitude values must be stored in decimal degrees format.

`GCALC_GeotoCart` performs reference system calculations using either the AGD or GDA, which is specified using the 2 way flag `iNum4` as described in *2.1 typedef struct coordinate*. When `GCALC_GeotoCart` is called, this flag is used to assign the appropriate semi-major axis and inverse flattening of the ellipsoid to be used (i.e. ANS or GRS80).

Below is an example of creating a pointer to a `coordinate` structure named `cVar1` and converting the contained geographic coordinates to earth centred Cartesian coordinates on the GDA using ANSI C and C++.

```
coordinate cVar1;
coordinate *pVar1;
pVar1 = &cVar1;

pVar1->dNum1 = -25.409024006;    /* latitude (neg south) */
pVar1->dNum2 = 148.037323739;    /* longitude */
pVar1->dNum3 = 0.0;              /* Zero ellipsoidal height */
pVar1->iNum4 = 1;                /* 1 = GDA (calc's on the GRS80) */

GCALC_GeotoCart ( pVar1 );
```

Alternatively, if you are not using pointers you can write:

```
coordinate cVar1;

cVar1.dNum1 = -25.409024006;
cVar1.dNum2 = 148.037323739;
cVar1.dNum3 = 0.0;
cVar1.iNum4 = 1;

GCALC_GeotoCart ( &cVar1 );      /* note the 'address-of' operator */
```

Below is an equivalent example of calling `GCALC_GeotoCart` using Visual Basic.

```
Dim cVar1 As coordinate

cVar1.dNum1 = -25.409024006
cVar1.dNum2 = 148.037323739
cVar1.dNum3 = 0.0
cVar1.iNum4 = 1

Call GCALC_GeotoCart ( cVar1 )
```

4.2 Datum Transformation Functions

4.2.1 GCALC_CreateGridIndex

Integral to the bi-linear interpolation technique, an array of the overview header blocks for each sub grid must be built up in virtual memory. `GCALC_CreateGridIndex` can be called to open the specified 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 shift file.

In determining which sub grid the point lies within, the transformation algorithm used by `GCALC_BiLinearTransformation` relies on this array to rapidly determine the desired sub grid. This process greatly increases the efficiency of transforming coordinates, particularly where large numbers of points are to be transformed.

`GCALC_CreateGridIndex` 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 an integer success indicator. The function is a void function.

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

```
char filename[401];
char filetype[4];
int success;

strcpy(filename, “D:\GDA\Grids\QLD_0900.gsb”);
strcpy(filetype, “gsb”);
```

```
GCALC_CreateGridIndex ( filename, filetype, &success );

If (success > 0)
{
	/* Report some error... */
	switch (success)
	{
	case 1:
		printf("Could not read from grid file.");
		break;
	case 2:
		...
	}
}
```

Below is an equivalent example of calling `GCALC_CreateGridIndex` using Visual Basic.

```
Dim strfilename As String
Dim strfiletype As String
Dim lsuccess As Long

strfilename = “D:\GDA\Grids\QLD_0900.gsb”
strfiletype = “gsb”

Call GCALC_CreateGridIndex ( strfilename, strfiletype, lsuccess )

If (lSuccess > 0) Then
	Select Case lSuccess
	Case 3
		' Print Message Box
		MsgBox "The grid file is corrupt" & _
		" ", vbInformation, "Grid File Error..."
	End Select
End If
```

4.2.2 GCALC_BiLinearTransformation (High Accuracy)

Once `GCALC_CreateGridIndex` has been called to open the specified grid file, `GCALC_BiLinearTransformation` can be called to convert geographic coordinates of latitude and longitude on one datum (stored within a structure of type `coordinate`) to latitude and longitude on another datum. `GCALC_BiLinearTransformation` takes one argument; a pointer to `coordinate` structure. The input latitude and longitude values must be stored in decimal degrees format and the calculated values are in decimal degrees.

`GCALC_BiLinearTransformation` applies the bi-linear interpolation technique from the specified distortion grid, using the 2 way flag in `iNum4` to indicate the transformation direction (i.e. AGD to GDA is forward and GDA to AGD is reverse). Since the method of bi-linear interpolation is applied to the latitude and longitude only, `dNum3` is not required. `GCALC_BiLinearTransformation` cannot be used if `GCALC_CreateGridIndex` has not been previously called. Once `GCALC_CreateGridIndex` has been called, `GCALC_BiLinearTransformation` may be called at random infinitely.

On success, `IO_Status` will be equal to zero. If any errors occur during the bi-linear interpolation (such as the distortion grid file has not been opened, or the point being outside the limits of the distortion grid), `IO_Status` will equal a non-zero value (see *5. ERROR RETURN STATUS*).

Below is an example of creating a pointer to a `coordinate` structure named `cVar1` and transforming the geographic coordinates on the AGD to coordinates on the GDA using ANSI C and C++.

```
coordinate cVar1;
coordinate *pVar1;
pVar1 = &cVar1;

pVar1->dNum1 = -25.409024006;    /* latitude (neg south, dec deg) */
pVar1->dNum2 = 148.037323739;    /* longitude (dec deg) */
pVar1->iNum4 = 0;                /* convert from AGD to GDA */

/* call GCALC_CreateGridIndex here */

GCALC_BiLinearTransformation ( pVar1 );

If (pVar1->IO_Status == 13)
{
       printf("The point is outside the grid");
}
```

Alternatively, if you are not using pointers you can write:

```
coordinate cVar1;

cVar1.dNum1 = -25.409024006;
cVar1.dNum2 = 148.037323739;
cVar1.iNum4 = 0;

/* call GCALC_CreateGridIndex here */

/* note the 'address-of' operator */
GCALC_BiLinearTransformation ( &cVar1 );
```

Below is an equivalent example of calling `GCALC_BiLinearTransformation` using Visual Basic.

```
Dim cVar1 As coordinate

cVar1.dNum1 = -25.409024006
cVar1.dNum2 = 148.037323739
cVar1.iNum4 = 0

‘ call GCALC_CreateGridIndex here

Call GCALC_BiLinearTransformation ( cVar1 )

If (cVar1.IO_Status = 13) Then
       ' Print Message Box
       MsgBox "The point was outside the limits of the " & Chr(10) & _
```

```
        "Distortion Grid." & _
        " ", vbInformation, "Error..."
End If
```

4.2.3 GCALC_MapGridTransformation (High Accuracy)

Once `GCALC_CreateGridIndex` has been called to open the specified grid file, `GCALC_MapGridTransformation` can be called to transform projection coordinates of easting, northing and zone on one datum (stored within a structure of type `coordinate`) to easting, northing and zone on another datum. `GCALC_MapGridTransformation` takes one argument; a pointer to `coordinate` structure.

The zone specified by `dNum3` is used to specify the zone of the input coordinates. The zone for the output coordinates is computed from the resultant longitude. If you wish to specify an output zone, use `GCALC_BiLinearTransformation` and then convert the output geographic coordinates manually using `GCALC_GeotoGrid`.

`GCALC_MapGridTransformation` applies the bi-linear interpolation technique from the specified distortion grid using the 2 way flag in `iNum4` to indicate the transformation direction (i.e. AGD to GDA as forward and GDA to AGD as reverse). `GCALC_MapGridTransformation` cannot be used if `GCALC_CreateGridIndex` has not been previously called. Once `GCALC_CreateGridIndex` has been called, `GCALC_MapGridTransformation` may be called at random infinitely.

On success, `IO_Status` will be equal to zero. If any errors occur during the bi-linear interpolation (such as the distortion grid file has not been opened, or the point being outside the limits of the distortion grid), `IO_Status` will equal a non-zero value (see *5. ERROR RETURN STATUS*).

Below is an example of creating a pointer to a `coordinate` structure named `cVar1` and transforming the geographic coordinates on the AGD to coordinates on the GDA using ANSI C and C++.

```
coordinate cVar1;
coordinate *pVar1;
pVar1 = &cVar1;

pVar1->dNum1 = 604330.847;       /* easting */
pVar1->dNum2 = 7189345.370;      /* northing */
pVar1->dNum3 = 55.0;             /* input zone */
pVar1->iNum4 = 0;                /* convert from AGD to GDA */

/* call GCALC_CreateGridIndex here */

GCALC_MapGridTransformation ( pVar1 );

...
```

Alternatively, if you are not using pointers you can write:

```
coordinate cVar1;

cVar1.dNum1 = 604330.847
cVar1.dNum2 = 7189345.370
cVar1.dNum3 = 55.0;
```

```
cVar1.iNum4 = 0;

/* call GCALC_CreateGridIndex here */

/* note the 'address-of' operator */
GCALC_MapGridTransformation ( &cVar1 );
```

Below is an equivalent example of calling `GCALC_MapGridTransformation` using Visual Basic.

```
Dim cVar1 As coordinate

cVar1.dNum1 = 604330.847
cVar1.dNum2 = 7189345.370;
cVar1.dNum3 = 55.0;
cVar1.iNum4 = 0

‘ call GCALC_CreateGridIndex here

Call GCALC_MapGridTransformation ( cVar1 )
...
```

4.2.4 GCALC_PointTran (High Accuracy)

`GCALC_PointTran` can be called to convert geographic coordinates of latitude and longitude on one datum (stored within a structure of type `coordinate`) to latitude and longitude on another datum. `GCALC_PointTran` takes three arguments; a pointer to `coordinate` structure; the distortion grid's full file-path and; the distortion grid's file-type. The input latitude and longitude values must be stored in decimal degrees format and the calculated values are in decimal degrees.

`GCALC_PointTran` applies bi-linear interpolation from the specified distortion grid using the 2 way flag in `iNum4` to indicate the transformation direction (i.e. AGD to GDA as forward and GDA to AGD as reverse). Since the method of bi-linear interpolation is applied to the latitude and longitude only, the height / zone variable `dNum3` is not required. `GCALC_PointTran` is not “datum-specific” and therefore may be applied to either AGD66 or AGD84 coordinates depending on the grid file you have chosen.

On success, `IO_Status` equals zero. If any errors occur during the bi-linear interpolation (such as the point being outside the limits of the distortion grid, or the distortion grid file does not exist), `IO_Status` equals a non-zero value (see *5. ERROR RETURN STATUS*).

Below is an example of creating a pointer to a `coordinate` structure named `cVar1` and transforming the geographic coordinates on the AGD to coordinates on the GDA using bi-linear interpolation using ANSI C and C++.

```
char filename[401];
char filetype[4];

strcpy(filename, “D:\MyFiles\QLD_0900.gsb”);
strcpy(filetype, “gsb”);

coordinate cVar1;
coordinate *pVar1;
```

```
pVar1 = &cVar1;

pVar1->dNum1 = -27.409024006;
pVar1->dNum2 = 153.037323742;
pVar1->iNum4 = 0;

GCALC_PointTran(filename, filetype, pVar1);

if (pVar1->IO_Status == 13)
{
       printf("The point was outside the limits of the grid");
}
```

Alternatively, if you are not using pointers you could write:

```
...
coordinate cVar1;

cVar1.dNum1 = -27.409024006;
cVar1.dNum2 = 153.037323742;
cVar1.iNum4 = 0;

GCALC_PointTran(filename, filetype, &cVar1);

...
```

Below is an equivalent example of calling `GCALC_PointTran` using Visual Basic.

```
Dim cVar1 As coordinate
Dim strfilename As String
Dim strfiletype As String

strfilename = “D:\MyFiles\QLD_0900.gsb”
strfiletype = “gsb”

cVar1.dNum1 = -27.409024006
cVar1.dNum2 = 153.037323742
cVar1.iNum4 = 0

Call GCALC_PointTran(strfilename, strfiletype, cVar1)

If (cVar1.IO_Status = 13) Then
       ' Print Message Box
       MsgBox "The point was outside the limits of the " & Chr(10) &
       _ "Distortion Grid." & _
       " ", vbInformation, "Error..." End If
```

4.2.5 GCALC_FileTran (High Accuracy)

`GCALC_FileTran` can be called to transform a file of coordinates on one datum to another datum using bi-linear interpolation, or for converting coordinates on the same datum. `GCALC_FileTran` takes three arguments: a pointer to `pointsfile` structure; the distortion grid’s full file-path, and; the

distortion grid's file-type. GCALC_FileTran is an int function (returning 1 for success and 0 for failure). 1 is returned if the entire input file has been read and no file-open errors were encountered. If the input file, output file or distortion grid file could not be opened, the function call is aborted and 0 is returned.

GCALC_FileTran uses chFInputType to determine what type of file is being read and what type of coordinates are required from each record in the file. GCALC_FileTran uses chFOutputType to determine what type of file is to be printed to and what type of coordinates are to be printed to each record in the file. See *Appendix A* for details on how GCALC_FileTran handles Formatted Text, Comma Separated Values, SCDB (MAN), Queensland Interchange Format (QIF), and NEWGAN (RES) files.

If any errors occur during the bi-linear interpolation (such as the point being outside the limits of the distortion grid, or the distortion grid file does not exist), IO_Status equals a non-zero value (see *5. ERROR RETURN STATUS*). GCALC_FileTran then uses IO_Status (when non-zero) to determine what error comments are printed to the log file. If chLogFile contains a valid file-path, the record that could not be transformed is appended to the log file. To correctly use this facility, chAccessMode should contain "a" for append mode. Therefore, each time a transformation error occurs, the log file can be appended to and not overwritten. If a log file is not desired, set the first element in the log filename char array to NULL.

For example:

```
cPfile1.chLogFile [0] = '\0';
```

GCALC_FileTran applies bi-linear interpolation from the specified distortion grid using InputDatum and OutputDatum to indicate the transformation direction (i.e. if InputDatum = 0 and OutputDatum = 1, then a forward transformation is performed). These flags are also used to determine which ellipsoid parameters (i.e. ANS or GRS80) should be used to convert between geographic and projection coordinates. Since the method of bi-linear interpolation is applied to the latitude and longitude only, an ellipsoidal height is not required. However, if a point lies outside the distortion grid and a 7 parameter transformation is to be performed, then the ellipsoidal height is set to zero.

When transforming QIF files, GCALC_FileTran uses OutputProj to determine whether a zone should be calculated for each set of output coordinates, or if a specified zone should be used to force the computations to produce coordinates in the specified zone. If OutputProj is 0 (zero), geographic coordinates are produced. If OutputProj is between 47 and 58, it is used to force GCALC_GeotoGrid to compute eastings and northings in the specified zone number. If OutputProj is 1, a zone is calculated for each point.

When Transforming Formatted Text or Comma Separated files (*.dat, *.txt, *.prn or *.csv) and a point within the file lies outside the distortion grid, GCALC_FileTran uses dParameters to transform the coordinates using a 7 parameter transformation. This transformation is performed through GCALC_DatumTran. Where this occurs, a message is printed to the log file.

Below is an example of creating a pointer to a `pointsfile` structure named `cVar1` and transforming the projection coordinates on the GDA to geographic (Separated DMS fields) coordinates on the AGD using bi-linear interpolation.

```
char filename[401];
char filetype[4];

strcpy(filename, “D:\MyFiles\QLD_0900.gsb”);
strcpy(filetype, “gsb”);

pointsfile cPfile1;
pointsfile *pPfile1;
pPfile1 = &cPfile1;

strcpy(pPfile1->chFilein, “D:\MyFiles\AGD DMS sep.dat”);
strcpy(pPfile1->chFInputType, “dat”);
strcpy(pPfile1->chFileout, “D:\MyFiles\MGA.csv”);
strcpy(pPfile1->chFOutputType, “csv”);

strcpy(pPfile1->chLogFile, “D:\MyFiles\AGD to AMG.log”);
strcpy(pPfile1->chAccessMode, “a”);

pPfile1->InputDatum = 1;          /* Input: GDA */
pPfile1->OutputDatum = 0;         /* Output: AGD */
pPfile1->InputProj = 1;           /* Input: projection */
pPfile1->OutputProj = 0;          /* Output: geographic */
pPfile1->OutputDMS = 0;           /* Output: degrees, minutes and seconds */
pPfile1->OutputHP = 0;            /* Output: Separated DMS fields */

/* Use National AGD84-GDA94 parameters for points outside the grid */
pPfile1->dParameters[0] = -117.763;     /* DX: X-Axis Translation */
pPfile1->dParameters[1] = -51.510;      /* DY: Y-Axis Translation */
pPfile1->dParameters[2] = 139.061;      /* DZ: Z-Axis Translation */
pPfile1->dParameters[3] = -0.292;       /* RX: X-Axis Rotation */
pPfile1->dParameters[4] = -0.443;       /* RY: Y-Axis Rotation */
pPfile1->dParameters[5] = -0.277;       /* RZ: Z-Axis Rotation */
pPfile1->dParameters[6] = -0.191;       /* SC: Scale Factor */

if ( !( GCALC_FileTran(filename, filetype, pPfile1)) )
{
       /* Report some error... */
       switch (pPFile1->IO_Status)
       {
       case 1:
              printf("Could not read from grid file.");
              break;
       case 2:
       ...
       }
}
printf("Number of successfully transformed coordinates: %d",
pPfile1->lNumTrans[0]);
```

Alternatively, if you are not using pointers you can write:

```
...
pointsfile cPfile1;

strcpy(cPfile1.chFilein, “D:\MyFiles\AGD DMS sep.dat”);
strcpy(cPfile1.chFInputType, “dat”);
strcpy(cPfile1.chFileout, “D:\MyFiles\MGA.csv”);
strcpy(cPfile1.chFOutputType, “csv”);

strcpy(cPfile1.chLogFile, “D:\MyFiles\AGD to AMG.log”);
strcpy(cPfile1.chAccessMode, “a”);

cPfile1.InputDatum = 1;          /* Input: GDA */
cPfile1.OutputDatum = 0;         /* Output: AGD */
cPfile1.InputProj = 1;           /* Input: projection */
cPfile1.OutputProj = 0;          /* Output: geographic */
cPfile1.OutputDMS = 0;           /* Output: degrees, minutes and seconds */
cPfile1.OutputHP = 0;            /* Output: Separated DMS fields */

/* Use National AGD84-GDA94 parameters for points outside the grid */
pPfile1.dParameters[0] = -117.763;      /* DX: X-Axis Translation */
pPfile1.dParameters[1] = -51.510;       /* DY: Y-Axis Translation */
pPfile1.dParameters[2] = 139.061;       /* DZ: Z-Axis Translation */
pPfile1.dParameters[3] = -0.292;        /* RX: X-Axis Rotation */
pPfile1.dParameters[4] = -0.443;        /* RY: Y-Axis Rotation */
pPfile1.dParameters[5] = -0.277;        /* RZ: Z-Axis Rotation */
pPfile1.dParameters[6] = -0.191;        /* SC: Scale Factor */

if ( !( GCALC_FileTran(filename, filetype, &cPfile1)) )
{
      ...
}
```

Below is an equivalent example of calling `GCALC_FileTran` using Visual Basic.

```
Dim cPfile1 As coordinate
Dim lSuccess As Long
Dim strfilename As String
Dim strfiletype As String

strfilename = “D:\MyFiles\QLD_0900.gsb”
strfiletype = “gsb”

cPfile1.chFilein = “D:\MyFiles\AGD DMS sep.dat”
cPfile1.chFInputType = “dat”
cPfile1.chFileout = “D:\MyFiles\MGA.csv”
cPfile1.chFOutputType = “csv”
cPfile1.chLogFile = “D:\MyFiles\AGD to AMG.log”
cPfile1.chAccessMode = “a”
cPfile1.InputDatum = 1           ' Input: GDA
cPfile1.OutputDatum = 0          ' Output: AGD
cPfile1.InputProj = 1            ' Input: projection
cPfile1.OutputProj = 0           ' Output: geographic
cPfile1.OutputDMS = 0            ' Output: degrees, minutes and seconds
cPfile1.OutputHP = 0             ' Output: Separated DMS fields
```

```
‘ Use National AGD84-GDA94 parameters for points outside the grid
pPfile1.dParameters(0) = -117.763       ‘ DX: X-Axis Translation
pPfile1.dParameters(1) = -51.510        ‘ DY: Y-Axis Translation
pPfile1.dParameters(2) = 139.061        ‘ DZ: Z-Axis Translation
pPfile1.dParameters(3) = -0.292         ‘ RX: X-Axis Rotation
pPfile1.dParameters(4) = -0.443         ‘ RY: Y-Axis Rotation
pPfile1.dParameters(5) = -0.277         ‘ RZ: Z-Axis Rotation
pPfile1.dParameters(6) = -0.191         ‘ SC: Scale Factor

lSuccess = GCALC_FileTran(strfilename, strfiletype, cPfile1)

If (lSuccess = 0) Then
       Select Case cPfile1.IO_Status
       Case 3
              ' Print Message Box
              MsgBox "The grid file is corrupt" & _
              " ", vbInformation, "Grid File Error..." End Select
End If

'      Number of successfully transformed coordinates held
'      by cPfile1.lNumTrans(0).
'      Total number of valid records read held by cPfile1.lNumTrans(1)
```

4.2.6 GCALC_DatumTran (Medium Accuracy)

`GCALC_DatumTran` can be called to transform earth centred Cartesian coordinates of X, Y and Z on one datum (stored within a structure of type `coordinate`) to X, Y, Z coordinates on another. `GCALC_DatumTran` takes two arguments; a pointer to `coordinate` structure and a double array holding the specified 7 parameters. `GCALC_DatumTran` is a void function.

The transformation direction (i.e. forward or reverse) is specified using the 2 way flag `iNum4` as described in *2.1 typedef struct coordinate*. When `GCALC_DatumTran` is called, this flag is used to determine whether the sign of the 7 parameters should change to reflect the reverse transformation direction (i.e. where `iNum4` = 1). For example, when `iNum4` = 0, the specified 7 parameters will be applied “as-is”. `GCALC_DatumTran` is not “datum-specific”, and therefore may be used to transform Cartesian coordinates from and to any datum, provided the correct parameters are passed.

The transformation parameters for use in Queensland are the National AGD84 to GDA94 transformation parameters (current as at May 2000). These are:

X Translation	=	-117.763
Y Translation	=	-51.510
Z Translation	=	139.061
X Rotation	=	-0.292
Y Rotation	=	-0.443
Z Rotation	=	-0.277
Scale change	=	-0.191

See the GDA Technical Manual for more details and Australian Regional AGD66 parameters:

http://www.icsm.gov.au/gda/gda-v_2.4.pdf

`GCALC_DatumTran` is labelled *Medium Accuracy* because it does not cater for the distortions in the AGD84 network by applying the distortion grid.

Below is an example of creating a pointer to a `coordinate` structure named `cVar1` and transforming X, Y, Z coordinates on the AGD to X, Y, Z coordinates on the GDA using ANSI C and C++. The national AGD84 are used for the examples.

```
coordinate cVar1;
coordinate *pVar1;
pVar1 = &cVar1;

double dParameters[8];

pVar1->dNum1 = -4890765.512;           /* earth-centred X coordinate */
pVar1->dNum2 = 3051661.332;            /* earth-centred Y coordinate */
pVar1->dNum3 = -2720079.994;           /* earth-centred Z coordinate */
pVar1->iNum4 = 0;                      /* 0 = AGD to GDA */

/* Use National AGD84-GDA94 parameters */
dParameters[0] = -117.763;
dParameters[1] = -51.510;
dParameters[2] = 139.061;
dParameters[3] = -0.292;
dParameters[4] = -0.443;
dParameters[5] = -0.277;
dParameters[6] = -0.191;

GCALC_DatumTran ( pVar1, dParameters );
```

Alternatively, if you are not using pointers you can write:

```
coordinate cVar1;

cVar1.dNum1 = -4890765.512;
cVar1.dNum2 = 3051661.332;
cVar1.dNum3 = -2720079.994;
cVar1.iNum4 = 0;

/* note the 'address-of' operator */
GCALC_DatumTran ( &cVar1, dParameters );
```

Below is an equivalent example of calling `GCALC_DatumTran` using Visual Basic.

```
Dim cVar1 As coordinate
Dim dParameters(7) As Double

cVar1.dNum1 = -4890765.512
cVar1.dNum2 = 3051661.332
cVar1.dNum3 = -2720079.994
```

```
cVar1.iNum4 = 0

‘ Use National AGD84-GDA94 parameters
dParameters(0) = -117.763
dParameters(1) = -51.510
dParameters(2) = 139.061
dParameters(3) = -0.292
dParameters(4) = -0.443
dParameters(5) = -0.277
dParameters(6) = -0.191

‘  Note: when passing Parameters, append (0)
Call GCALC_DatumTran ( cVar1, dParameters(0) )
```

4.3 Data Output Functions

4.3.1 GCALC_PrintLogFile

`GCALC_PrintLogFile` can be called to print a string of information to a specified ASCII file. `GCALC_PrintLogFile` takes three arguments; the log file’s full file-path; a one-character string literal indicating the file-access mode, and; a character array to be printed. The only access modes that can be used are `"a"` to append to an existing ASCII file, and `"w"` to create a new ASCII file for writing. `GCALC_PrintLogFile` is an `int` function (returning 1 for success and 0 for failure). 1 is returned if the log file has been opened and the data has been successfully printed. If the log file could not be opened, the function call is aborted and 0 is returned.

If the character array contains the string `<< Date >>` it is replaced with the full system date and time. For example, executing the following code in ANSI C:

```
char cHeader[201];
char cFilename[601];
int success;

strcpy(cHeader, "LogFileCreated: << Date >> \n\n");
strcpy(cFilename, “D:\MyFiles\MyLogFile.log”);

success =GCALC_PrintLogFile(cFilename,"w",cHeader);
```

will create a new ASCII file containing:

```
Log FileCreated:    Monday, 29 May 2000, 02:35:08 PM.
```

Below is an equivalent example of calling `GCALC_PrintLogFile` using Visual Basic.

```
Dim strHeaderAsString
Dim strFilenameAsString
Dim lngSuccessAsLong

strHeader ="LogFileCreated: << Date >> "
```

```
strFilename =“D:\MyFiles\MyLogFile.log”

lngSuccess =GCALC_PrintLogFile(strFilename,"w",strHeader);
```

Output:

```
Log FileCreated:    Monday, 29 May 2000, 02:35:08 PM.
```

4.3.2 GCALC_SetFileTranHeader

`GCALC_SetFileTranHeader` can be called to specify whether a standard header is printed to the transformed output files when using `GCALC_FileTran`. `GCALC_SetFileTranHeader` is a void function that takes one argument; an integer value in the range 0 ≤ n ≤ 2. Table 4a lists the actions performed based on the argument passed.

Table 4a – GCALC_SetFileTranHeader Argument Options

Argument	Action performed for all DAT, TXT, PRN and CSV output files
0	Standard header is printed
1	No header is printed
2	A Carriage return is printed at the start of the file

Calling `GCALC_SetFileTranHeader` initialises a global variable and as such, needs to be called only once throughout the life of an application. If `GCALC_SetFileTranHeader` is not called, for every file transformed by `GCALC_FileTran`, GCalc will by default print a standard header to all output files.

Below is an example of calling `GCALC_SetFileTranHeader` to specify “No-Header” mode using ANSI C and C++.

```
/* do not print header block to output files */
GCALC_SetFileTranHeader ( 1 );
```

Below is an equivalent example of calling `GCALC_DmstoDeg` using Visual Basic.

```
‘ do not print header block to output files
Call GCALC_SetFileTranHeader ( 1 )
```

4.4 Progress Functions

For every file transformed by `GCALC_FileTran`, the byte offset of the input file pointer is stored each time a new record is read from the input file. The byte offset is held by a global variable and therefore can be used to track the progress of a file transformation.

4.4.1 GCALC_SetByteOffset

`GCALC_SetByteOffset` is used to initialise the global byte offset variable to zero. If the byte offset is required to track the progress of more than one file transformation, `GCALC_SetByteOffset` must be called prior to every call made to `GCALC_FileTran`.

Below is an example of calling `GCALC_SetByteOffset` using ANSI C and C++.

```
GCALC_SetByteOffset();
```

Below is an equivalent example of calling `GCALC_SetByteOffset` using Visual Basic.

```
Call GCALC_SetByteOffset()
```

4.4.2 GCALC_GetByteOffset

`GCALC_GetByteOffset` can be called to retrieve the value held by the global byte offset variable. `GCALC_GetByteOffset` returns a long value, and can be called at any time while a file is being transformed, or after a call made to `GCALC_FileTran` has returned.

Below is an example of calling `GCALC_GetByteOffset` using ANSI C and C++.

```
long lByteOffset;
lByteOffset = GCALC_GetByteOffset ();
```

Below is an equivalent example of calling `GCALC_GetByteOffset` using Visual Basic.

```
Dim lByteOffset As Long
lByteOffset = GCALC_GetByteOffset ()
```

5. ERROR RETURN STATUS

When `GCALC_PointTran`, `GCALC_FileTran`, `GCALC_BiLinearTransformation` or `GCALC_MapGridTransformation` are called, the int variable `IO_Status` is assigned zero to initialise the transformation status. If an error occurs anywhere throughout the transformation process, `IO_Status` is assigned a non-zero number to indicate the type of failure. Similarly, if an error occurs during a call made to `GCALC_CreateGridIndex`, *success* is used to hold the error status.

`GCALC_PointTran` will store an error status number from 0 to 3, or from 13 to 16 in `IO_Status`, whilst `GCALC_FileTran` can store any number from 0 through to 22. `GCALC_CreateGridIndex` will store an error status value of –2 or 0 through to 3 in *success* and `GCALC_BiLinearTransformation` and `GCALC_MapGridTransformation` will store a value of –1, 0, or 13 through to 16 in `IO_Status`.

For all files, if the type of error is assigned a number from 1 to 7, or 14 to 16, `GCALC_FileTran` returns `false`. For QIF files only, if any non-zero number is assigned, `GCALC_FileTran` returns `false`.

For Formatted Text, Comma Separated, SCDB MAN and NEWGAN Res files, `GCALC_FileTran` returns `true` regardless of the number of successful transformations. Therefore, `IO_Status` should not be used to determine a further action or output comment since `IO_Status` will hold the transformation status for the last record in the input file. After calling `GCALC_FileTran` for Formatted Text, Comma Separated, SCDB MAN or NEWGAN Res files, `lNumTrans` should be used to determine the number of successfully transformed coordinates.

On return from `GCALC_PointTran`, `GCALC_BiLinearTransformation` or `GCALC_MapGridTransformation` `IO_Status` can be used to determine a further action or output comment. Similarly, after calling `GCALC_CreateGridIndex`, *success* can be used to determine whether another grid file should be sourced.

The following tables indicate the type of failure that may occur for each function call.

Table 5a – GCALC_CreateGridIndex

Success	Description
-2	An Australian NTv2 Binary file has been read. See Appendix B for more information
0	Success. The Grid file has been successfully opened and all sub grid header information is stored in virtual memory
1	Could not read from the grid file
2	Unable to read this type of grid file
3	The grid file is corrupt

Table 5b – GCALC_BiLinearTransformation and GCALC_MapGridTransformation

IO_Status	Description
-1	GCALC_CreateGridIndex has not been previously called
0	Successfully transformed the point
13	Point is outside limits of the grid
14	There is an error in the grid
15	Could not retrieve shifts from the ASCII grid file
16	Could not retrieve shifts from the Binary grid file

Table 5c – GCALC_PointTran

IO_Status	Description
0	Success. The Grid file has been successfully opened and all sub grid header information is stored in virtual memory
1	Could not read from the grid file
2	Unable to read this type of grid file
3	The grid file is corrupt

Table 5d – GCALC_FileTran

IO_Status	Description
0	The last point read was successfully transformed
1	Could not read from the grid file
2	Unable to read this type of grid file
3	The grid file is corrupt
4	Could not read from input file
5	Could not write to output file
6	Unsupported input file
7	Unsupported output file
8	No data contained within line just read
9	Line is too short to contain valid coordinates
10	Line does not contain valid numeric input
11	Could not read required data...possibly wrong format
12	Invalid Zone...cannot convert projection coordinates to geographic
13	Point is outside limits of the grid
14	There is an error in the grid
15	Could not retrieve shifts from the ASCII grid file
16	Could not retrieve shifts from the Binary grid file
17	Insufficient fields for a valid csv record
18	Input coordinate system parameters do not match the specified input QIF file
19	Unable to determine coordinate parameters from the specified input QIF file
20	No data was found in the QIF file
21	The coordinate system parameters do not match the RES file coordinates
22	RES file coordinates of this type are not supported

6. USING GCALC.DLL IN YOUR APPLICATION

This section describes the procedure for using GCalc.dll in ANSI C, Visual C++ and Visual Basic applications.

6.1 Win32 Console (ANSI C) and MFC AppWizard C++ Projects

To link GCalc.dll to your C / C++ project, follow these steps:

1. Copy "GCalcext.h" and "GCalc.lib" to the working folder of your project.
2. Include these files into your project.
3. With the #include statements in the C / C++ file of your project, add:

   ```
   #include "GCalcext.h"
   ```
4. Add the code for the desired function calls in the `main()` function (or equivalent) of your project as described in *4. CALLING THE EXPORTED FUNCTIONS*. For example, a function call to `GCALC_DmstoDeg` in a standard ANSI C program might look like:

   ```
   #include <stdio.h>
   #include "GCalcext.h"

   main()
   {
          double dDMS;

          printf("Enter in a value in degrees, minutes and seconds: ");
          scanf("%lf", &dDMS);

          dDMS = GCALC_DmstoDeg(dDMS);

          printf("\n\nDecimal Degrees:  %.12f", dDMS);
   }
   ```
5. Copy "GCalc.dll" to the folder where the *.exe will be created.
6. Compile and run the project.

6.2 Visual Basic Projects

To link GCalc.dll to your Visual Basic project, follow these steps:

1. Copy "GCalc.dll" to the working folder of your project.
2. In the *.bas file of your project, add the necessary declarations for the functions that you want to call as described in *3. VISUAL BASIC FUNCTION DECLARATIONS*.
3. Add the code for the desired function calls as described in *4. CALLING THE EXPORTED FUNCTIONS*. For example, a function call to `GCALC_DmstoDeg` in a Visual Basic program (i.e. An event for a Command button on a User Form) might look like:

   ```
   Private Sub cmdDmstoDeg_Click()
   ```

```
        ‘ Convert the edit box text to double first
        ‘ Then update the edit box with the converted value

        txtNum1 = GCALC_DmstoDeg(CDbl(txtNum1))

End Sub
```

6.3 Recommended Function Calls

GCalc.dll supports four function calls that can be used to transform coordinates from one datum to another. These are `GCALC_PointTran`, `GCALC_FileTran`, `GCALC_BiLinearTransformation` and `GCALC_MapGridTransformation`. The following points may be of assistance when determining the most appropriate function call(s) to make.

Single Point Transformations

Where only a single point needs to be transformed, such as to update a simple dialog control, the recommended function is `GCALC_PointTran`. Each time `GCALC_PointTran` is called, the grid file is opened; the point is transformed, and; the grid file is closed. Whilst being as equally efficient for transforming a single point as `GCALC_BiLinearTransformation`, calling `GCALC_PointTran` for large numbers of points requires additional time.

File Transformations of Recognised Formats

Where a file of points formatted to the specifications described within this documentation need to be transformed, the recommended function is `GCALC_FileTran`. Each time `GCALC_FileTran` is called, the grid file and input and output files are opened; the coordinates are extracted from each record in the input file, transformed and printed to the output file, and; the grid file and input and output files are closed. `GCALC_FileTran` is extremely efficient for transforming supported file types, however cannot be used to support custom or other proprietary file types.

Multiple Transformations of Points or Files of Custom Format Types

If large numbers of points need to be transformed, whether for updating a screen image or for transforming custom and proprietary file formats, the recommended function calls are `GCALC_CreateGridIndex` and then either `GCALC_BiLinearTransformation` or `GCALC_MapGridTransformation`.

Once `GCALC_CreateGridIndex` has been called to open the grid file, `GCALC_BiLinearTransformation` can be called infinitely to transform single points in geographic coordinates. The efficiency of multiple transformations will be dependent on how you call these functions from your own application.

If your input coordinates are projected and you require the transformed coordinates to be projected also, `GCALC_MapGridTransformation` may be used. `GCALC_MapGridTransformation` does not let you specify what output zone the projected coordinates should lie within.

APPENDIX A – DAT, TXT, PRN and CSV FILE FORMAT SPECIFICATION

Formatted Text files have either "dat", "prn" or "txt" extension. Comma Separated files have "csv" extension.

The input coordinates can be stored as projection or geographic coordinates as defined by `InputProj`. The geographic coordinates may be in decimal degrees format; separated degrees, minutes and seconds format; separated degrees and minutes format; or HP notation. This is determined by `InputDMS` and `InputHP`. Similarly, `OutputProj`, `OutputDMS` and `OutputHP` are used to determine if the output coordinates will be converted and how they will be printed. `InputDatum` and `OutputDatum` may be the same if computations are required on the one datum. See section *2.2 typedef struct pointsfile* for selecting the desired flag indicators.

If valid coordinates could not be read (possibly due to an incorrect coordinate type or file record format), or lie outside the extents of the distortion grid, the record is printed to the log file and the next record is read. `lNumTrans[1]` is incremented only if the current record contains valid numeric input in the required fields. `lNumTrans[0]` is incremented only if the coordinates from the current record were successfully transformed. `lNumTrans[2]` is incremented only if the coordinates were transformed using the distortion grid.

If the coordinates from any record could not be transformed (due to the wrong file format, or a point was outside the distortion grid), `IO_Status` is assigned a value to indicate the type of error. In this instance, an error message and the record are printed to the log file, and `IO_Status` is assigned zero ready for the next record in the file. After the last point has been printed, `FileTran` returns 1 (regardless of the number of failed transformations).

When printing to Formatted Text and Comma Separated files only, a header is printed to the file if `GCALC_SetFileTranHeader` has not been called previously, or if `GCALC_SetFileTranHeader` was called with a parameter of 0 (zero). The contents of `chAppName` are printed in the first row of the header. If `chAppName` is NULL, the string "`GCalc.dll File Output`" is printed. The system date; input file and coordinate system; and output file and coordinate system are printed also.

A.1 Formatted Text Files (*.dat, *.txt, *.prn)

Every record (or line) in a formatted text file must contain data fields in particular file positions (or columns). You may however, omit certain fields depending on what they are. Table A1 lists the compulsory and non-compulsory fields in the required order:

Table A1 – DAT File Compulsory Items

Field	Columns	Characters	Compulsory?
Point ID	1 – 11	11	No
Latitude / Easting	12 – 27	16	Yes
Longitude / Northing	28 – 43	16	Yes
Orthometric Height (AHD)	44 – 52	9	No
UTM Zone	53 - 55	3	Compulsory when output coordinates are projected. Optional for geographic coordinates.
Datum (AGD or G94)	56 – 59	4	No
Description	60 – 85	26	No

Note that the UTM zone is only required when you are transforming a file of projection coordinates.

File position (column no's):

```
        1         2         3         4         5         6         7         8

123456789 123456789 123456789 123456789 123456789 123456789 123456789 123456789 12345
```

Decimal Degrees:

```
 69540299   -27.876576516   153.132465498    765.010    AGD PSM 6954 Feb 1999
           -26.926733461   149.654651687           55     1st Order Control
```

DMS HP Notation:

```
69540299   –27.523567546    153.075687579
STATION B  -26.553624046    149.391674607                  1st Order Control
```

Separated DMS Fields:

```
69540299   –27  52 35.67546 153 07  56.87579  765.010   AGD
STATION B  -26  55 36.24046 149 39  16.74607               1st Order Control
```

DMIN HP Notation:

```
 69540299  –27.525945910    153.079479298
           -26.556040077    149.392791012                  1st Order Control
```

Separated DMIN Fields:

```
69540299   –27  52.5945910 153  07.9479298  765.010   AGD
STATION B  -26  55.6040077 149  39.2791012                 1st Order Control
```

Projection Coordinates

```
69540299      513039.205      6916452.137    765.010 56 AGD PSM 6954 Feb 1999
STATION B     763609.224      7018902.575            55     1st Order Control
```

A.2 Comma Separated Files (*.csv)

Every record (or line) in a comma separated file must contain data fields separated by commas. You may however, omit certain fields depending on what they are. If you choose not to enter in the non-compulsory fields, you must still enter in the commas to delineate the presence of the particular field. Below is a description of the compulsory and non-compulsory fields in the required order:

Table A2 – CSV File Compulsory Items

Field	Compulsory?
Point ID	No
Latitude / Easting	Yes
Longitude / Northing	Yes
Orthometric Height (AHD)	No
UTM Zone	Compulsory when output coordinates are projected. Optional for geographic coordinates.
Datum (AGD or G94)	No
Description	No

Note that the UTM zone is only required when you are transforming a file of projection coordinates.

Decimal Degrees:

```
69540299, -27.876576516, 153.132465498, 765.010,,AGD,PSM 6954 Feb 1999,
-26.926733461,149.654651687
```

DMS HP Notation:

```
69540299,–27.523567546,153.075687579,,,,
STATION B,-26.553624046,149.391674607,,,,1st Order Control
```

Separated DMS Fields:

```
 ,   –27,       52,    35.67546,   153,    07,     56.87579, 765.010,,AGD,
STATION B,-26,55,36.24046,149,39,16.74607
```

Note that when you are entering in geographic coordinates in separated DMS fields, you must include a comma to separate the degrees, minutes and seconds values.

DMIN HP Notation:

```
69540299,–27.525945910,153.079479298,,,,
STATION B,-26.556040077,149.392791012,,,,1st Order Control
```

Separated DMIN Fields:

```
,– 27,52.5945910,153,079479298,765.010,,AGD,
STATION B,-26,55.6040077,149,39.2791012,,,,1st Order Control
```

Note that when you are entering in geographic coordinates in separated DMIN fields, you must include a comma to separate the degrees and minutes values.

Projection Coordinates

```
69540299,513039.205,6916452.137,765.010,56,AGD,PSM 6954 Feb 1999
STATION B,763609.224,7018902.575,,55,,1st Order Control
```

Note that when a certain data field is not required, you must still enter in the commas to delineate the presence of the field.

The simplest way to create a comma separated file is to use Microsoft Excel. Alternatively, open one of the sample files included in the installation and save it as a new file. Using a text editor, such as Notepad or UltraEdit, add your data to this file using the same format.

To create a CSV file in Excel

1 Create a new spreadsheet

2 Enter in a unique identifier for each new point in Column A.

3 Enter in values for latitude and longitude in Columns B and C, or Easting Northing and Zone in Columns B, C and E.

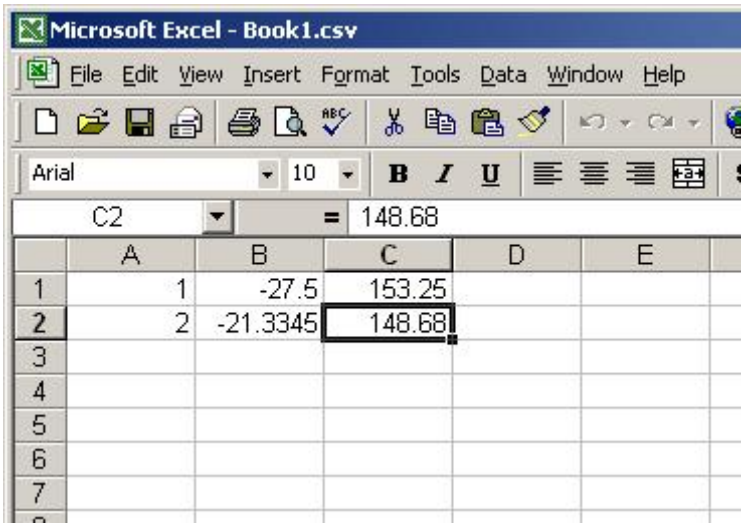
4 Click **Save As** from the **File** Menu

5 In the **Save As type** box, click **CSV (Comma delimited) (*.csv)**.

6 Click **Save**.

As an example, Figure A1 shows two records for a CSV file (to be created) in decimal degrees.

Figure A1 – Using Excel to create a CSV file in Decimal Degrees



A.3 SCDB (MAN) Files

SCDB files have "man" extension. There is no version description associated with MAN files.

Since the coordinates within a SCDB file are always stored in geographic coordinates in Separated DMS fields, `InputProj`, `InputDMS` and `InputHP` are not required. Similarly, `OutputProj`, `OutputDMS` and `OutputHP` are not required since all output coordinates are printed in the standard format. Therefore, only `InputDatum` and `OutputDatum` are required to transform a SCDB file. `InputDatum` and `OutputDatum` cannot be the same.

If valid coordinates could not be read (possibly due to an incorrect coordinate type or file record format), or lie outside the extents of the distortion grid, the record is printed to the log file and the next record is read. `lNumTrans[1]` is incremented only if the current record being read contains valid numeric input in the required fields. `lNumTrans[0]` is incremented only if the coordinates from the current record were successfully transformed.

If the coordinates from any record could not be transformed (due to the wrong file format, or a point was outside the distortion grid), `IO_Status` is assigned a value to indicate the type of error. In this instance, an error message and the record are printed to the log file, and `IO_Status` is assigned zero ready for the next record in the file. After the last point has been printed, *FileTran* returns 1 (regardless of the number of failed transformations).

A.4 Queensland Interchange Format (QIF) Files

Queensland Interchange Format (QIF) files have "qif" extension. The supported QIF file Version is Version 3.0.

Within all QIF files, the input coordinates can be stored as projection or geographic coordinates in decimal degrees. The type of coordinates stored within the file is defined within the QIF file. When the Qif file is opened, the type of coordinates stored within the file are compared with `InputProj`. Therefore, when calling *FileTran*, `InputProj` must be defined correctly to match the type of coordinates within the file for successful read / write operations. Since the QIF standards require a single zone be used for all projection coordinates within an extract, `OutputProj` must be defined as a number between 47 and 58 for a valid file of projection coordinates to be produced. `InputDatum` and `OutputDatum` may be the same if computations are required on the one datum. For example, if a file contains geographic coordinates on the AGD, you may transform it to projection coordinates also on the AGD, provided that a zone number is specified for the entire file.

If at least one valid coordinate could not be read (possibly due to an incorrect coordinate type or file record format), or lies outside the extents of the distortion grid, the transformation is aborted and *FileTran* returns 0. Use `IO_Status` to determine the type of error.

A.5 NEWGAN (RES) Files

NEWGAN Res files have "res" extension.

With NEWGAN Res files, you may transform files in either projection or geographic coordinates. When a *.res file is opened, the type of coordinates stored within the file are compared with `InputProj`. Therefore, when calling *FileTran*, `InputProj` must be defined correctly to match the type of coordinates within the file for successful read / write operations. However, when transforming coordinates in geographic coordinates, you do not need to specify the parameters for `InputDMS` or `InputHP`, since geographic coordinates are always stored in Separated DMS fields. Similarly, if you want your output coordinates in geographic coordinates, `OutputDMS` and `OutputHP` are not required. Therefore, only `InputDatum`, `InputProj`, `OutputDatum` and `OutputProj` are required to transform a NEWGAN Res file. `InputDatum` and `OutputDatum` may be the same if you want to convert coordinates between geographic and projection format.

If valid coordinates could not be read (possibly due to an incorrect coordinate type or file record format), or lie outside the extents of the distortion grid, the record is printed to the log file and the next record is read. `lNumTrans[1]` is incremented only if the current record being read contains valid numeric input in the required fields. `lNumTrans[0]` is incremented only if the coordinates from the current record were successfully transformed.

If the coordinates from any record could not be transformed (due to the wrong file format, or a point was outside the distortion grid), `IO_Status` is assigned a value to indicate the type of error. In this instance, an error message and the record are printed to the log file, and `IO_Status` is assigned zero ready for the next record in the file. After the last point has been printed, *FileTran* returns 1 (regardless of the number of failed transformations).

APPENDIX B – NTv2 GRID SHIFT FILE FORMAT

The Grid Shift file used by GDAy contains coordinate shift values at nominated grid nodes in a format known as NTv2 (National Transformation Version 2). This format was developed by the Geodetic Survey Division of Geomatics Canada to implement the transformation of coordinates between NAD27 and NAD83 in Canada. Australia recognised the advantages of this format and decided to adopt NTv2 as a standard format.

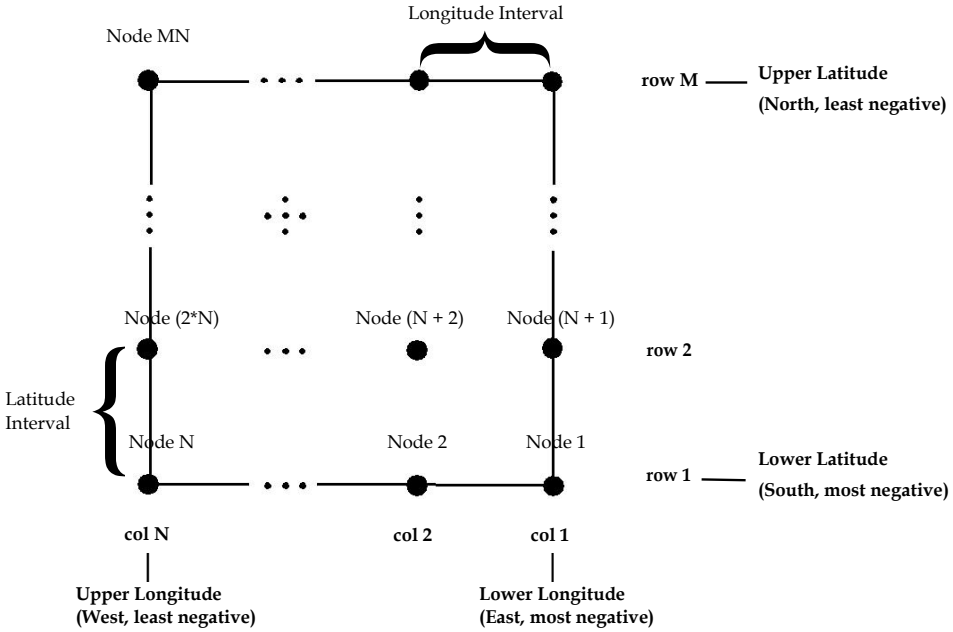
The following sections describe the Specifications of the NTv2 file format and the considerations for its usage in Australia.

B.1 NTv2 Grid Shift File Concept

The NTv2 grid format is a structured rectangular region of evenly spaced rows of latitude and columns of longitude. The intersection of a row and column is referred to as a grid node. Each node consists of a shift value for both latitude and longitude, as well as a value for the accuracy of each shift as determined in the least squares modelling process.

To provide for a structured means of extracting the values at each node from a digital file, the nodes are consecutively indexed in rows and columns starting at the south east (lower right) corner, finishing at the north west (upper left) corner. The fundamental concept of the grid and node index structure is shown in figure B1.

Figure B1 – Grid Node Order



In its simplest form, as shown by Figure B2, the grid shift file contains a single “sub grid” bound by unique values for each of the following: lower latitude, upper latitude, lower longitude, and upper longitude. The size of the file is directly related to the area it covers and is therefore governed by the number of nodes (file records) contained within the file. The order of these nodes within the file is critical and must be adhered to for accurate grid interpolation.

The ordered grid-node structure within the file allows for direct computation of the location of the file records containing the interpolated shifts. Since, the size of the file is independent of the retrieval method the data retrieval is almost instantaneous. All information required to compute the position of each node within the file is contained within a block of header information at the beginning of the file. The structure and format of NTv2 grid shift file header information is explained in section B2.

Figure B2 – Grid Shift file with a single sub grid

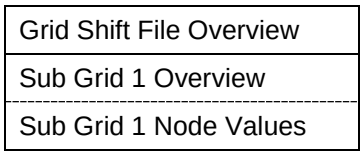
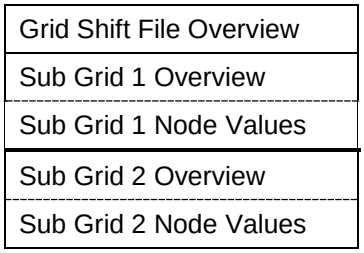
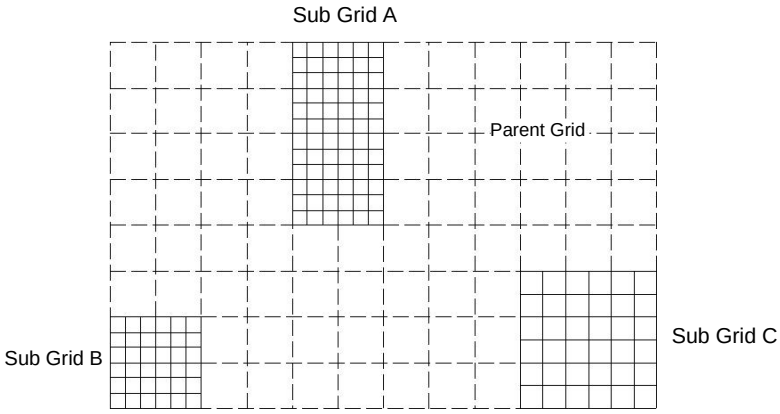


Figure B3 – Grid Shift file with a multiple sub grids



The grid shift file may also contain many sub-grids (Figure B3) of greater densities that cover designated areas within the parent grid area. Figure B4 illustrates the concept of densified sub-grids within a parent grid. In these instances, the grid shift file contains an extra block of header information for each densified sub grid, which are each followed by the interpolated shifts for the nodes within the sub grid.

Figure B4 – Densified sub grid structure



The order of these sub grids with in the file is of no importance, since the method of retrieval of data from the file follows a logical method based on the concept of a simple hierarchical tree. For example, each time the grid shift file is opened an index (or array) of sub-grids and their parent grid is built and stored in memory.

To conform to the NTv2 format, a set of rules must be followed when combining two or more grid areas within the one file. These rules ensure that a unique sub grid is selected for subsequent interpolation. For more information on the rules regarding the NTv2 format, see “NTv2 Developer’s Guide” (Junkins and Farley, 1995).

B.2 NTv2 File Format

An NTv2 Grid Shift file can be written as an ASCII or binary file. The binary format described in the *NTv2 Developer's Guide* (Junkins and Farley, 1995) gives the impression of being FORTRAN specific but it is actually a pure byte dump. The *NTv2 Developer's Guide* (Junkins and Farley, 1995) does not specify whether the format of a binary is Little or Big Endian. In Australia all NTv2 binary grid shift files will be distributed as Little Endian binary files.

The following sections describe the format of each component of a Grid Shift file. In describing the format, the data types used are given in Table B1 along with the byte size required for each type. This table relates to a binary Grid Shift file, the format of fields in an ASCII version will vary so they are specified in the Format columns of Tables B.2 to B.4. If an identifier is specified, both the *Identifier* and *Value* are written to file otherwise only the *Value* is written. An *Identifier* is written to the Grid Shift file as a *string*. The important thing to note here concerns how integer values are dealt with in a binary file. Each header record must be 16 bytes long so to make a record containing an integer value be 16 bytes, 4 bytes of padding are inserted. The padding used is 4 NULL characters (ASCII character 0).

Table B1 – Grid Shift File Data Types

Type	Bytes	Comment
Integer	4	Followed by 4 NULL characters of padding in binary version of file, i.e. 8 bytes in total.
Float	4	
Double	8	
String	8	8 characters

B.2.1 Grid Shift File Overview

Table B2 specifies the format of the Overview section of a Grid Shift file and a sample is given in Figure B5. This section contains eleven records which give general information about the sub grids within the file. The `NUM_FILE` record is the most useful as it specifies how many sub grids the Grid Shift file contains.

Table B2 – Grid Shift File Overview Information

Record	Identifier	Value	Description	ASCII format
1	NUM_OREC	Integer	# header records in overview	%8s%3d
2	NUM_SREC	Integer	# header records in sub grid	%8s%3d
3	NUM_FILE	Integer	# of sub grids	%8s%3d
4	GS_TYPE	String	Shift type (SECONDS)	%8s%-8s
5	VERSION	String	Distortion model	%8s%-8s
6	SYSTEM_F	String	"From" ellipsoid name	%8s%-8s
7	SYSTEM_T	String	"To" ellipsoid name	%8s%-8s
8	MAJOR_F	Double	"From" semi major axis	%8s%12.3f
9	MINOR_F	Double	"From" semi minor axis	%8s%12.3f
10	MAJOR_T	Double	"To" semi major axis	%8s%12.3f
11	MINOR_T	Double	"To" semi minor axis	%8s%12.3f

Figure B5 – Sample Grid Shift File Overview

```
        NUM_OREC      11
        NUM_SREC      11
        NUM_FILE      1
        GS_TYPE       SECONDS
        VERSION       MAY98V20
        SYSTEM_FANS
        SYSTEM_TGRS80
        MAJOR_F       6378160.000
        MINOR_F       6356774.719
        MAJOR_T       6378137.000
        MINOR_T       6356752.314
```

B.2.2 Sub Grid Format

A sub grid consists of a Sub Grid Overview section followed by the values of the nodes in the sub grid. The Overview section specifies the sub grid extents, the grid spacing, the name of the sub grid and the name of the parent sub grid if there was one. A sub grid will have a parent if its extents fall within another sub grid. The format of the Sub Grid Overview is given in Table B3 and a sample is given in Figure B6.

Table B3 – Sub Grid Overview Information

Record	Identifier	Value	Description	ASCII format
1	SUB_NAME	String	Sub grid name	%8s%-8s
2	PARENT	String	Parent sub grid name	%8s%-8s
3	CREATED	String	Date	%8s%-8s
4	UPDATED	String	Date	%8s%-8s
5	S_LAT	Double	Lower latitude	%8s%15.6f
6	N_LAT	Double	Upper latitude	%8s%15.6f
7	E_LONG	Double	Lower longitude	%8s%15.6f
8	W_LONG	Double	Upper longitude	%8s%15.6f
9	LAT_INC	Double	Latitude interval	%8s%15.6f
10	LONG_INC	Double	Longitude interval	%8s%15.6f
11	GS_COUNT	Integer	Grid node count	%8s%6d

Figure B6 – Sample Sub Grid Overview

```
SUB_NAMEMELB
PARENT  NONE
CREATED 7/1998
UPDATED 7/1998
S_LAT    -138780.000000
N_LAT    -134406.000000
E_LONG   -526104.000000
W_LONG   -519354.000000
LAT_INC       54.000000
LONG_INC      54.000000
GS_COUNT 10332
```

Table B4 specifies the values at each grid node and a sample is given in Figure B7. The number of nodes in this section of the file is specified in the Sub Grid Overview value `GS_COUNT`. At each node, only the four values specified are stored, i.e. no identifier or coordinate information is stored.

(**Note:** The shift values at each node consist of a conformal transformation component and a distortion component. If the distortion component at the node cannot be modelled, e.g. because of insufficient data, then the shift value at the node contains the conformal component only. When this occurs, both of the accuracy values are set to -1 to denote this. These nodes can still be used in the interpolation of shift values; however it is not possible to interpolate the accuracy of the shifts).

Table B4 – Sub Grid Node Values

Record	Identifier	Value	Description	ASCII format
1		Float	Latitude shift value	%10.6f
2		Float	Longitude shift value	%10.6f
3		Float	Latitude shift accuracy	%10.6f
4		Float	Longitude shift accuracy	%10.6f

Figure B7 – Sample Sub Grid Node Values

```
       5.414650 -4.727520  0.002171  0.000617
       5.413610 -4.728820  0.001615  0.000235
       5.413050 -4.729720  0.001563  0.000233
       ……..
```

B.3 Necessary Considerations for NTv2 Usage

As previously discussed, the NTv2 file format was adopted for use as a standard format for transforming coordinates from AGD to GDA94 within Australia. Since the initial implementation of the NTv2 file format for Australian use, there has been a revised NTv2 file format (Canadian Binary file) produced which will supersede the original Australian Binary file. The following sections clarify the

differences between the revised Canadian Binary file and the original Australian Binary file, and note the necessary considerations for the usage of the NTv2 format outside Canada.

B.3.1 Difference Between Australian and Canadian Binary Files

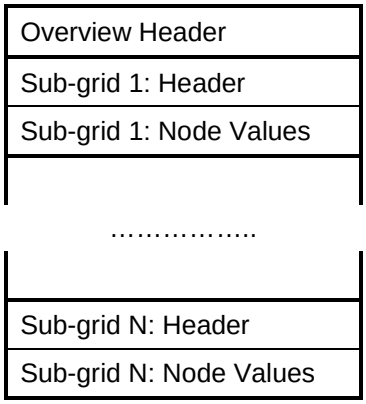
NTv2 files can exist in two forms, binary and ASCII. In Australia the only version of a grid shift file that strictly adheres to the NTv2 file format is the ASCII version (States only distribute the binary form, but the ASCII equivalent is easily obtained from utilities such as GDAit). The differences in the binary form are minor but the consequence is that there are two binary forms of an NTv2 file that aren't compatible: an Australian version and a Canadian version. Only the Canadian version is a true NTv2 file. The following explains the difference between the two binary forms and outlines the reason why it occurred.

What is the difference?

The simple answer is that any integer value in a Canadian NTv2 binary file must be read as a 4 byte number followed by 4 bytes of padding. Australian binary files DO NOT contain this padding, Canadian binary files do. This affects three records in the Overview Header (`NUM_OREC`, `NUM_SREC` and `NUM_FILE`), and one record in each of the Sub-grid Headers (`GS_COUNT`).

Considering this in more detail, an NTv2 file is comprised of an Overview Header and is followed by one or more Sub-grids. This is represented in Figure B8.

Figure B8 – NTv2 File Structure



The Overview Header and each Sub -grid Header consist of 11 records of 16 bytes, i.e. each header is 176 bytes long. The first 8 bytes of each record is a string identifier, the last 8 bytes contain the value of the identifier. The value can be one of three data types: an integer, a double or a string. A representation of how each data type is stored in an NTv2 header record is shown in Figure B9 with the number of bytes required being given in brackets.

Figure B9 – Data storage in a Header record

Double:	Identifier (8)	Value (8)	
String:	Identifier (8)	Value (8)	
Integer:	Identifier (8)	Value (4)	Padding (4)

The padding used for integer values is 4 NULL characters (ASCII character 0). The only purpose of the padding is to make a record with an integer value 16 bytes long, all other data types will automatically have records 16 bytes long. The records affected in the Overview Header (see Table B2) are `NUM_OREC`, `NUM_SREC` and `NUM_FILE`. The record affected in the Sub-grid Header (see Table B3) is `GS_COUNT`.

The format of the "Node Values" in a Sub-grid is identical for both Australian and Canadian binary files, i.e. no padding is used. For more information about the NTv2 file format refer to *Appendix B.2 NTv2 File Format*.

Why did the difference occur?

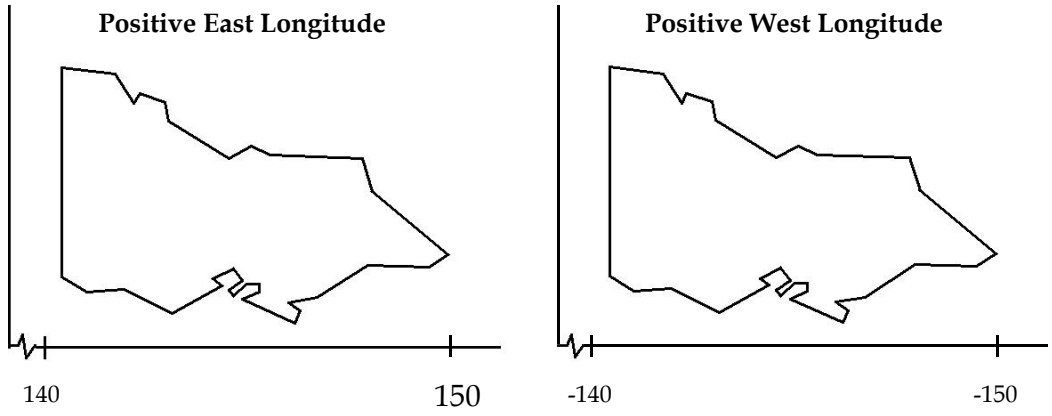
Australian binary came into existence because it was believed that NTv2 binary was compiler dependent. The Canadians had implemented it with FORTRAN and information at the time suggested that the use of records within a FORTRAN binary file would create a file that other development environments would have difficulty reading and writing. Given that most software development today is done in a language other than FORTRAN, the decision was made to format the file as specified in *Appendix B* of the *NTv2 Developer's Guide* (Junkins and Farley, 1995), but to ignore any auxiliary record identifiers that FORTRAN used. At the time this was not considered to be a major issue. A FORTRAN version of the grid file could easily be created for those who required it by using a FORTRAN utility to convert an ASCII version of the grid file to the binary form.

However it was apparent that NTv2 binary was not compiler dependent. The justification for having an Australian version of the binary is no longer valid and retaining it will lead to further confusion in the future. Ultimately, the best option is to abandon this version and adopt the Canadian implementation.

B.3.2 Implications of Using NTv2 Outside Canada

NTv2 was developed for use in Canada and therefore regards longitude as increasing *Positive West*, which is opposite to the convention used in Australia of *Positive East* longitudes. The implication of this is that longitude values that are east of Greenwich (all Australian longitudes), must be made negative to be compatible with a *Positive West* system. (Latitude values do not need to be modified as they are considered as *Positive North* in NTv2 which is the standard convention).

Figure B10 – Longitude Axis Orientation



The sign reversal of the longitude axis explains why the sub grid parameters Upper and Lower Longitude in Figure B10 appear in the order they do. The Lower Longitude value is the smallest or most negative longitude, Upper Longitude is the largest, or least negative value.

References

Collier P. and Mitchell, D., 2000, *GDAit (GDA94 InTerpolation) Users Guide*, Department of Geomatics, University of Melbourne.

Collier P. and Mitchell, D., 2000, *GDAit (GDA94 InTerpolation) Software Documentation*, Department of Geomatics, University of Melbourne.

Junkins, D.R. and Farley, S.A., 1995, *NTv2 Developer's Guide*. Geodetic Survey Division, Geomatics Canada, 27pp.

APPENDIX C – Gcalc SOFTWARE LICENSE AGREEMENT

This GCalc Software License Agreement is a legal agreement between THE STATE OF QUEENSLAND represented by the Department of Natural Resources and Mines (the **Licensor**) and you, the end user of GCalc (the **Licensee**), whereas:

- The **Licensor** has developed and is the owner of the intellectual property rights in the GCalc Software.
- The **Licensor** agrees to grant a non-exclusive, non-transferable Licence to the **Licensee**.

IT IS AGREED:

1. DEFINITIONS

- 'Agreement' means this Deed of Agreement between the parties;
- 'Software' means the GCalc Software provided by the Licensor to the Licensee;
- 'Intellectual Property Rights' means all rights in copyright, patents, registered and unregistered trademarks, registered designs, trade secrets, knowhow, rights in relation to circuit layouts and all other rights of intellectual property as defined in Article 2 of the Convention establishing the World Intellectual Property Organisation of July 1967;
- 'Licence' means the non-exclusive, non-transferable licence granted by the Licensor to the Licensee pursuant to this agreement.

2. ENTIRE AGREEMENT

2.1 This Agreement constitutes the entire agreement between the parties. Any prior arrangements, agreements, representations or undertakings are superseded. No modification or alteration of any clause of this Agreement will be valid unless it is in writing signed by each party.

3. LICENCE

3.1 The Licensor grants a licence to the Licensee to use the Software on and from the date of this agreement.

3.2 The use of the software shall be limited to personal use or for use in the ordinary course of business. The licensee must not sell the Software (or any part of it) or distribute it (for reward) to any other third party nor produce hardcopy products incorporating the Software except with the prior written consent of the Licensor.

3.3 The Licensee shall acknowledge on any electronic or hard paper copy documents produced that the Software was made available by the State of Queensland through the Department of Natural Resources and Mines.

4. DURATION OF LICENCE

4.1 The Licensor and the Licensee agree that the Licence shall remain in force from the date of receiving this Software Agreement until the termination of this Agreement.

5. STATUS OF LICENSEE

5.1 The Licensee is not partner or agent of the Licensor and does not have the power or authority, directly or indirectly or through its servants or agents, to bind the Licensor to any agreement with a Licensee or other third party or otherwise to contract, negotiate or enter into a binding relationship for or on behalf of the Licensor.

6. CONFIDENTIALITY

6.1 The Licensee agrees that the Software is valuable commercial information of the Licensor.

6.2 The Licensee agrees to disclose the Software only to such of its employees, servants and contractors who need to know it for the purpose of the Licensee exercising its rights under this Agreement.

6.3 The Licensee shall take all reasonable steps to maintain and safeguard the confidentiality of the Software and to ensure that its employees, servants, contractors, consultants, clients and business associates, maintain the confidentiality of the Software and use the Software solely for the purposes permitted under the Agreement.

7. INTELLECTUAL PROPERTY

7.1 Both Parties agree and acknowledge that the Licensor is the owner of, the Intellectual Property Rights of and related to the Software.

7.2 Except where expressly permitted in this Agreement, the Licensee must not make any use of the Software which is not referred to in this agreement.

7.3 These conditions do not confer on the Licensee any rights of ownership in the Software.

8. WARRANTY

8.1 The Licensor warrants that it has the full right, power and authority:

- To enter into this agreement; and
- To grant to the Licensee all rights which are conferred upon the Licensee under this Agreement.

8.2 The Licensee acknowledges that the Licensor does not guarantee the accuracy or completeness of the Software or that it is free from error, and other than the warranty contained in clause 8.1 does not make any other warranty about the Software.

9. LIMITATION OF LIABILITY

9.1 Except as expressly provided to the contrary in this Agreement, the Licensor shall not be under any liability to the Licensee in respect of any loss or damage (including consequential loss or damage) however caused, which may be suffered or incurred or which may arise directly or indirectly in respect of the supply of the Software pursuant to this Agreement or the failure or omission on the part of the Licensor to comply with its obligations under this Agreement.

9.2 Except as expressly provided to the contrary in this Agreement, all warranties, whether express, implied, statutory or otherwise, relating in any way to the subject matter of this Agreement or to this Agreement generally, are excluded. Where any Act of Parliament implies in this Agreement any term, and that Act avoids or prohibits provisions in a contract excluding or modifying the application of or exercise of or liability under such term, such term shall be deemed to be included in this Agreement. However, the liability of the Licensor for any breach of such term shall, if permitted by that Act be limited, at the option of the Licensor, to the replacement of the Software.

9.3 The Licensee warrants that it has not relied on any representation made by the Licensor which has not been stated expressly in this Agreement, nor has it relied upon any descriptions or illustrations or specifications contained in any document produced by the Licensor.

9.4 The Confidentiality clause 6, Warranty clause 8, Limitation of Liability clause 9, and Indemnity clause 10, shall continue to have full force and effect following termination of this agreement.

9.5 The Licensor shall not be under any liability to the Licensee in respect of any loss or damage (including consequential loss or damage) howsoever caused which may be suffered or incurred or which may arise directly or indirectly in respect of the use of the Software.

10. INDEMNITY

10.1 The Licensee shall indemnify and hold the Licensor harmless from and against any and all action, losses, damage, judgments, awards, settlements and expenses, directly or indirectly arising from or connected with any claim, demand, law suits or legal proceedings arising out of the accuracy of the Software or any work produced from the Software, the use made by any third party of the Software or the unauthorised use by the Licensee of the Software.

10.2 The Licensee shall be solely responsible for the use, supervision, management and control of the Software.

10.3 The Licensee shall ensure that the Software is protected at all times from access, use or misuse, damage or destruction by any person not authorised by the Licensor or Licensee for that purpose.

11. OBLIGATIONS OF THE LICENSEE

11.1 The Licensee shall notify the Licensor immediately if the Licensee becomes aware of any unauthorised use of the whole or part of the Software by any person.

12. TERMINATION OF THE LICENCE

12.1 In the event that the Licensee fails to perform any of its obligations or responsibilities under this Agreement the Licensor may terminate this Licence forthwith upon written notice to the Licensee.

12.2 The parties agree that upon termination of this Agreement all rights granted by the Licensor to the Licensee shall cease immediately.

12.3 The parties agree that termination of this Agreement by the Licensor shall not derogate from or prejudice any other rights of the Licensor against the Licensee in respect of anything done or omitted to be done by the Licensee under this Agreement.

12.4 The Licensee agrees that upon termination of this Agreement the Software shall be returned to the Licensor and all copies shall be erased from all forms of storage. The Licensee shall provide a written certificate to the Licensor specifying either that the Software has been destroyed, returned or otherwise, as directed by the Licensor.

13. APPLICABLE LAW

13.1 The parties hereby agree that this Agreement shall be governed by and construed in accordance with the law of the State of Queensland.

14. ASSIGNMENT

14.1 Neither this Agreement nor any rights granted hereunder may be assigned or sub-licensed in any manner whatsoever by the Licensee.

15. WAIVER

15.1 None of the conditions of this Agreement shall be waived, varied, discharged or released whether at law or in equity, unless both parties agree in writing.