GDA*it* (GDA94 InTerpolation)

Software documentation

DJ Mitchell
PA Collier

Department of Geomatics
University of Melbourne

Version 2.0 – April 2000

Table of Contents

Acknowledgements

GDAit (GDA94 InTerpolation) has been developed on behalf of Land Victoria, by the Department of Geomatics at the University of Melbourne. The *GDAit* software has been written by David Mitchell and this document was co-authored by David Mitchell and Dr Philip Collier.

1. Introduction

This manual is designed to explain the software architecture of *GDAit*, which is software that uses Bi-Linear interpolation to transform coordinates from AGD66 to GDA94 and vice versa. This manual documents the two versions that *GDAit* has been implemented in:

- a PC based application that runs on both 16 and 32 bits platforms (i.e. Windows 3.1/3.11, and Window95/NT)
- a version that runs on a Unix Web Server.

While the underlying functionality of the two versions is similar, the actual implementation of the two is quite different. The algorithm used by both versions is described in section 2. The implementation of the algorithm in the PC version is described in section 3, while its implementation in the Web version is described in section 4. Reference to *GDAit* in sections 3 or 4 is specific to that version of the program unless otherwise stated.

2. Transformation Algorithm

2.1 Overview

Using the known shift values at the nearest four grid nodes, *GDAit* applies bi-linear interpolation (refer to Appendix D) to compute the shift at the nominated (non-grid) point. The known shift values are extracted from a file of grid shifts and are selected based on their proximity to the interpolation point. To assist in understanding the algorithm described below, the reader should be familiar with the basic structure of a Grid Shift file which is detailed in Appendix C. The process for transforming the coordinates of a point is summarised in Figure 2.1.

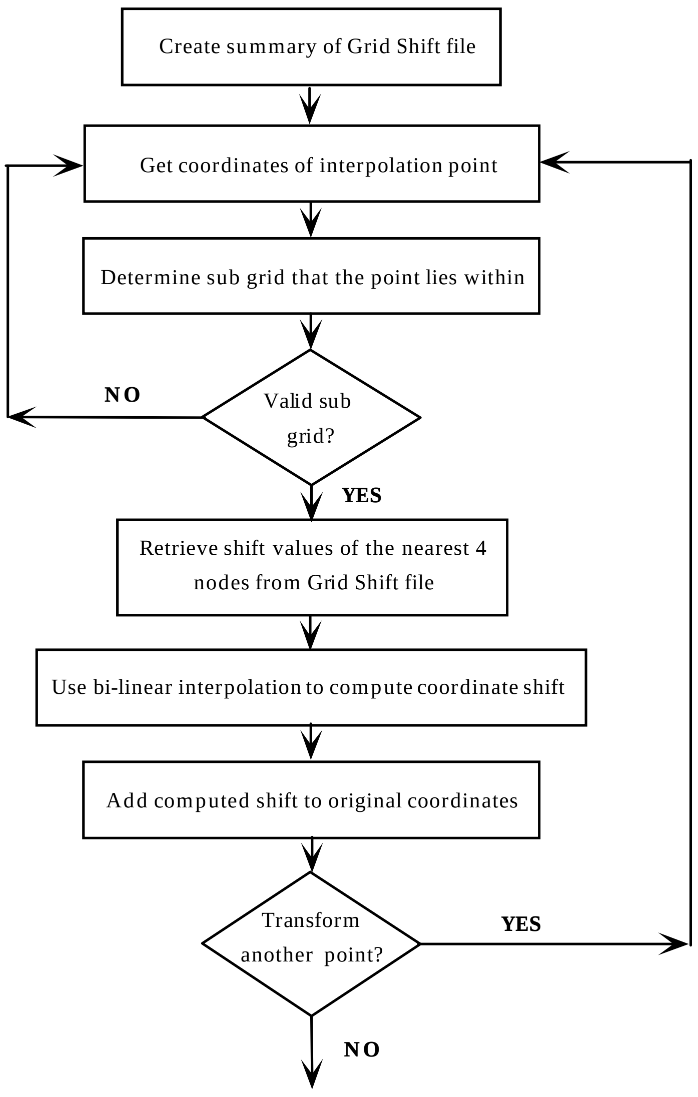

Create summary of Grid Shift file
Get coordinates of interpolation point
Determine sub grid that the point lies within
NO
Valid sub grid?
YES
Retrieve shift values of the nearest 4 nodes from Grid Shift file
Use bi-linear interpolation to compute coordinate shift
Add computed shift to original coordinates
Transform another point?
YES
NO

Figure 2.1 - Coordinate Transformation Process

2.2 Transformation Algorithm

2.2.1 Creation of Grid Shift file Summary

Efficient access to each sub grid in the Grid Shift file is essential. Storing the contents of each sub grid overview and the file location (byte offset) of the first node in each sub grid enhances efficiency. The stored information can be used to readily retrieve grid shift values from any sub grid in the Grid Shift file.

2.2.2 Input of Coordinates and Transformation Direction

The values in a Grid Shift file relate to a specific transformation direction which is referred to as the *forward transformation*. For Victorian Grid Shift files, the *forward transformation* direction is from AGD66 to GDA94. The flowchart in Figure 2.1

describes the *forward transformation* process. The transformation of coordinates in the opposite direction (GDA94 to AGD66 in Victoria) is referred to as a *reverse transformation*.

The *reverse transformation* process is different to the *forward transformation*. In a *forward transformation*, the shifts can be interpolated directly off the grid as the coordinates of the interpolation point and the grid nodes are related to the same datum (AGD66). In a *reverse transformation*, coordinates of the interpolation point are in a different datum (GDA94) to the grid nodes (AGD66), so the required shifts can not be directly interpolated off the grid. Because of this, the *reverse transformation* problem requires an iterative solution.

The *reverse transformation* involves computing the shift components at the approximate AGD66 coordinates of a GDA94 point. Subtracting the shifts computed from the GDA94 coordinates produces a better approximation of the AGD66 coordinates. By updating the initial AGD66 coordinates with the new values and repeating the process three times, the exact AGD66 coordinates of a GDA94 point are produced. (To get started, the initial AGD66 coordinates are set to the GDA94 coordinates of the point being transformed). The *reverse* transformation process is described below and shown as a flowchart in Figure 2.2.

```
1.  Initialise AGD66 coordinates:
    $\phi'_{AGD66} = \phi_{GDA94}$
    $\lambda'_{AGD66} = \lambda_{GDA94}$
2.  Repeat four times:
{
        3.  Interpolate shifts ($\delta\phi$ and $\delta\lambda$) at AGD66 point.
        4.  Compute new AGD66 coordinates:
            $\phi'_{AGD66} = \phi_{GDA94} - \delta\phi$
            $\lambda'_{AGD66} = \lambda_{GDA94} - \delta\lambda$
}
```

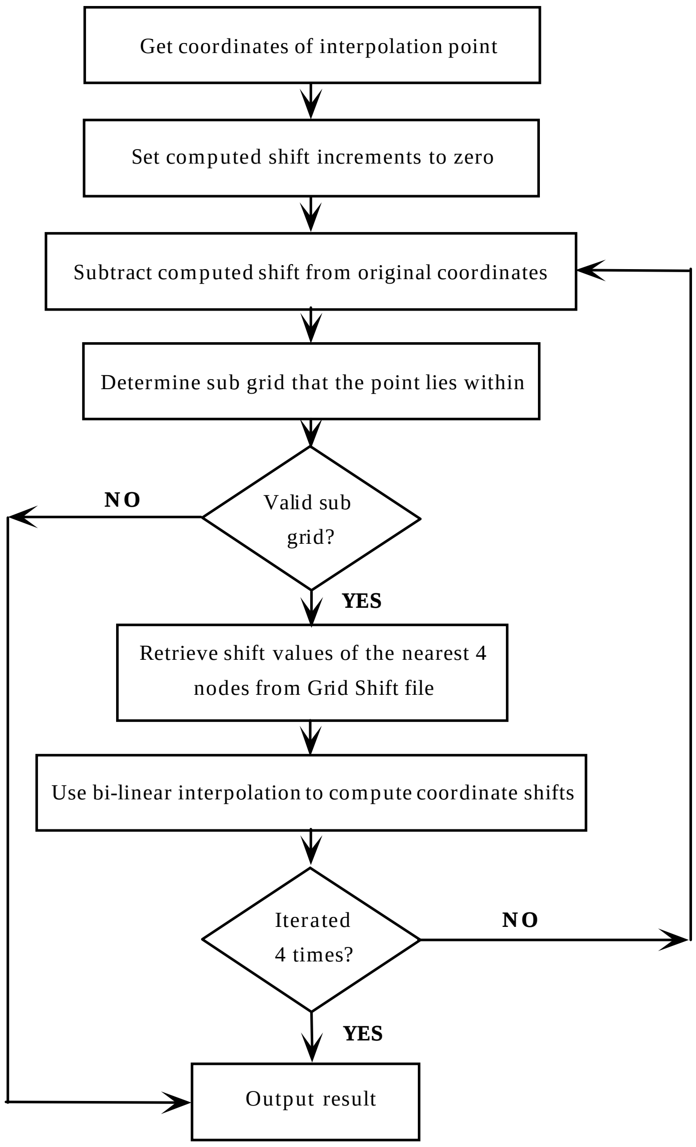

Get coordinates of interpolation point
Set computed shift increments to zero
Subtract computed shift from original coordinates
Determine sub grid that the point lies within
Valid sub grid?
NO
YES
Retrieve shift values of the nearest 4 nodes from Grid Shift file
Use bi-linear interpolation to compute coordinate shifts
Iterated 4 times?
NO
YES
Output result

Figure 2.2 - Reverse Coordinate Transformation Process

2.2.3 Sub grid determination

If the Grid Shift file contains only one sub grid then this step is elementary, however if there is more than one sub grid then a more complicated searching procedure is required to determine which sub grid should be used.

The basic process is to compare the coordinates of the interpolation point with the extents of each sub grid. Rather than comparing the densities of the relevant sub grids, the hierachial structure of the Grid Shift file is used to select the most appropriate (more dense) sub grid. This hierachial structure is implemented via the PARENT sub grid value. Each sub grid has a name and the value of the PARENT variable is the name of the sub grid that the current sub grid is completely contained within. If the sub grid does not have a parent sub grid then the value of PARENT will be "NONE". The algorithm to select a sub grid is:

```
Set PARENT_GRID to "NONE".

Repeat
{
1.  Test if the point falls within any of the child sub grids of the current
    PARENT_GRID and select it if it does.  (Note: If a point falls on either of
    the upper limits of a sub grid then the point is considered outside the sub
    grid).
2.  IF a child sub grid is found set PARENT_GRID to the name of the selected
    grid, ELSE exit the loop.
}
```

2.2.4 Retrieving Shift Values from Sub Grid

After determining that the point lies within one of the sub grids, the four nodes closest to the point must be identified. This calculation is done using:

- the latitude and longitude of the point,
- the lower latitude and longitude extents of the sub grid,
- the latitude and longitude interval between nodes.

The four nodes to retrieve are shown in Figure 2.3. The equations to calculate the row and column of the lower right grid node (Node A) are:

$$\text{row } i = 1 + \text{integer}\frac{\phi_P - \phi_{Lower}}{\Delta\phi} \qquad \ldots(2.1)$$

$$\text{column } j = 1 + \text{integer}\frac{\lambda_P - \lambda_{Lower}}{\Delta\lambda} \qquad \ldots(2.2)$$

Note: The longitude value must be that for a *positive west* coordinate system. This value is the negative of a *positive east* longitude. Refer to section 3 of Appendix C for more information.

The *Node Number* of Node A is its position relative to the first node in the grid shift file (i.e. the node in the lower right corner of the grid). The equation to calculate the node number of the node in row i, column j, is:

$$\text{Node Number of A} = (N * (i-1) + j) \qquad \ldots(2.3)$$

Using the file location (byte offset) of the first node in the sub grid (stored in the summary of the Grid Shift file), the file position of each node in the sub grid can be calculated by adding the offset to the node:

Node Offset = (Node Number - 1) * Node Record Length …(2.4)
Node's File Position = Sub Grid File Offset + Node Offset …(2.5)

(The Node Record Length will be 16 (4 floating point values each of 4 bytes)).

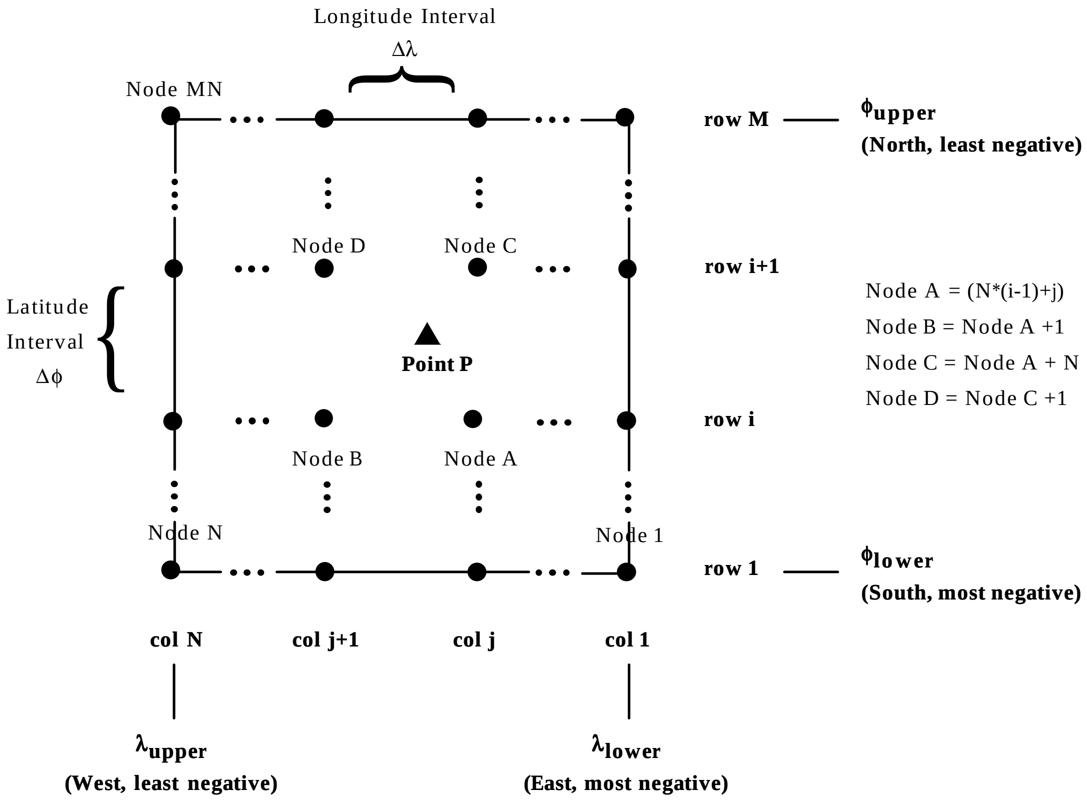


Figure 2.3 - Selection of Shift Nodes

2.2.5 Interpolation of Shift Values

Using the shift and accuracy values at the nearest 4 grid nodes, the required shift and accuracy at the point can be determined using Bi-Linear Interpolation. Refer to Appendix D for a description of Bi-Linear Interpolation.

2.3 Sample Calculations

Appendix E contains sample calculations for both *forward* and *reverse* transformations.

3. *GDAit* : PC Version

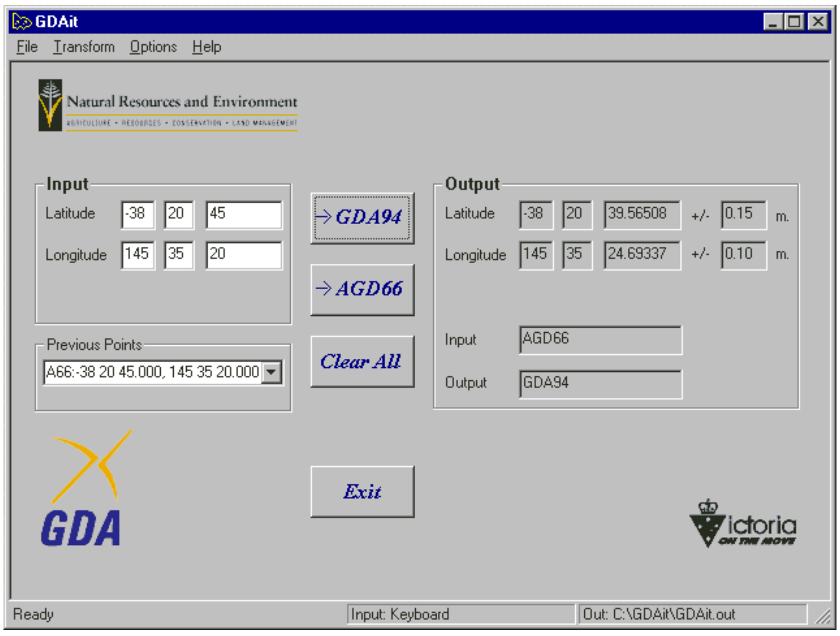
3.1 Overview

The PC version of *GDAit* was written in both a 16 and 32 bit version. The 16 bit version was developed in Microsoft Visual version 1.52c and the 32 bit version was developed with Microsoft Visual C++ version 5. *GDAit* uses the Microsoft Foundation Class (MFC) and is based on Microsoft's Document-View architecture.

Many of the data access functions in *GDAit* are provided by generic C++ functions which are outside the Document-View architecture. The reason for this is that *GDAit* was also to be implemented on a Web Server so it was desirable to reuse as much code as possible.

GDAit is a dialog based application (see Figure 3.1). It accepts input via edit boxes and command buttons, validates the data, processes it, then displays the result. This is referred to as ***Interactive Mode***. *GDAit* can also be used to process data from an input file which is referred to as ***Point File Mode***. In both modes, output to file is generated.

GDAit was originally developed as a 32 bit application and when most of the functionality was implemented, it was ported back to a 16 bit version. The reason for this was the superior development environment that Visual C++ version 5 has over version 1.52. The 32 bit application was considered the more important of the two as it is expected to have a longer lifetime, so more time was spent refining this version.


GDAit
File Transform Options Help
Natural Resources and Environment
→GDA94
→AGD66
Clear All
Exit
GDA
Victoria on the move

Input
Latitude -38 20 45
Longitude 145 35 20

Previous Points
A66:-38 20 45.000, 145 35 20.000

Output
Latitude -38 20 39.56508 +/- 0.15 m.
Longitude 145 35 24.69337 +/- 0.10 m.
Input AGD66
Output GDA94

Ready | Input: Keyboard | Out: C:\GDAit\GDAit.out

Figure 3.1 - *GDAit* Interface

3.2 Program Structure

The flowchart for the PC based interactive version of *GDAit* is quite simple:

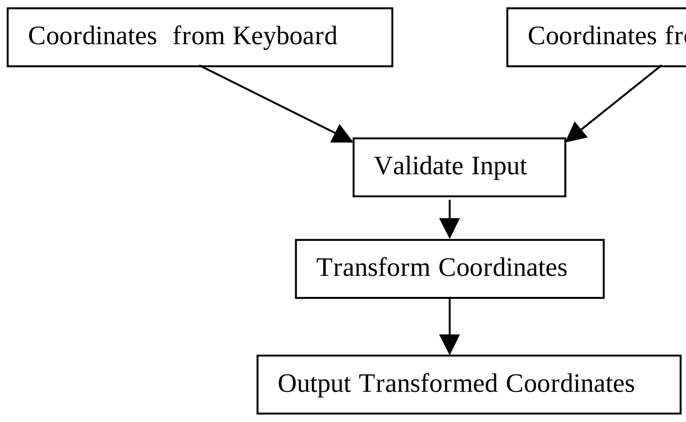


Figure 3.2 - *GDAit* PC Version Flowchart

GDAit's implementation can be broken down into three types of files. The input and output in *GDAit* is mainly contained in the files generated by App Wizard and Class Wizard. The mathematical component of *GDAit* is provided by functions which are external to the MFC architecture. The following section describes the files in each of these three components.

3.3 File Listing

The following lists the names of the source code, executable, library and help files that are common to both PC versions and specific to a particular version of PC *GDAit*. Refer to section 3.4 for a description of the purpose of each file.

3.3.1 Source Code File Names

Common:
The following file names are common to both versions. (**Note:** the contents of the 16 and 32 bit versions of these files may differ, only the name is common).

- StdAfx.h/cpp, MainFrm.h/cpp, InputDlg.h/cpp, OutputDlg.h/cpp, PtNameDg.h/cpp, DispDlg.h/cpp, Angles.h/cpp, CoordTran.h/cpp, GridFile.h/cpp, BMPapi.h/cpp
- Resource.rc, Resource.rc2, Resource.h
- GDAit.rtf, GDAit.hpj

16 Bit Specific:
- GDAit16A.h/cpp, GDAit16View.h/cpp, GDAit16Doc.h/cpp

32 Bit Specific:
- Grid32a.h/cpp, Grid32aView.h/cpp, Grid32aDoc.h/cpp

3.3.2 MFC Library Files

<u>16 Bit Version:</u>
- MFC250.dll

<u>32 Bit Version:</u>
- MFC42.dll, MSVCRT.dll, MSVCIRT.dll

3.3.3 Executable Files Created

<u>16 Bit Version:</u>
GDAit16.exe, GDAit16.hlp

<u>32 Bit Version:</u>
GDAit.exe, GDAit.hlp

3.4 App Wizard Files

GDAit uses a Document-View architecture. The application was originally called *Grid32*, but this was later changed to *GDAit*. The following file names that were generated by App Wizard have the legacy of the original name chosen. (The filenames of the 16 bit version files use *GDAit*). The following files were generated as part of a standard App Wizard generated Document -View application:

3.4.1 StdAfx (.h and .cpp)
This is the standard include file generated. No files were added to the default files included.

3.4.2 MainFrm (.h and .cpp)
The MainFrame class contains the code to set the initial appearance of the frame that surrounds the application window and create the status bars along the bottom of the window.

3.4.3 Grid32a (.h and .cpp)
The application class. This class sets up the initial state of *GDAit* and is used to store information that will be required throughout an entire session using *GDAit*. It includes functions for:

<u>Output Files</u>
GDAit creates at least two output files. A log file (***GDAit*.log**) is used to record general operational events such as names of files opened and for recording any errors. A default output file (***GDAit*.out**) is used for the output of all coordinates transformed while in ***Interactive Mode***. The application class provides functions for other classes to access these files and to write information to them. It also provides a function to display them using Notepad.

Interpolation Data
The underlying basis of *GDAit* is bi-linear interpolation using a grid file of shift values. The application maintains a summarised representation of this grid file so that other classes can access it efficiently.

Coordinate Type Constants
Four constants are defined to represent the four coordinate types that *GDAit* can accept (projection, decimal degrees, HP notation, and separate degrees, minutes and seconds). These constants are used throughout many of the files in the application e.g. for defining radio button values and then subsequently, file input and output.

INI File
The application also reads and writes the *GDAit* INI file which stores and restores the appearance of the Interface and dialog box settings. This INI file also stores the location and format (ASCII or binary) of the grid shift file.

3.4.4 Grid32aView (.h and .cpp)

The view class used is derived from CFormView to provide dialog like behaviour in the main application window. The view class contains the bulk of the code for handling user input:

- Coordinates entered into the text boxes.
- Accepts and validates user input from the text entry boxes.
- Displays Dialog boxes in response to menu item selections.

The view class also controls the appearance of the *GDAit* window:

- Dynamically positions and displays the logos.
- Dynamically positions and displays the button controls.
- Dynamically changes the number of text boxes displayed and their captions to match the coordinate type being entered.
- Dynamically changes the appearance of the window depending on the current mode (Interactive or Point File) and updates the contents of the status bar panes.
- Inserts and retrieves the coordinates of points stored in the Previous Point Drop List Box.

3.4.5 Grid32aDoc (.h and .cpp)

The document class is not used explicitly, but is required to implement the underlying document view architecture.

3.4.6 Resource (.rc, rc2 and .h)

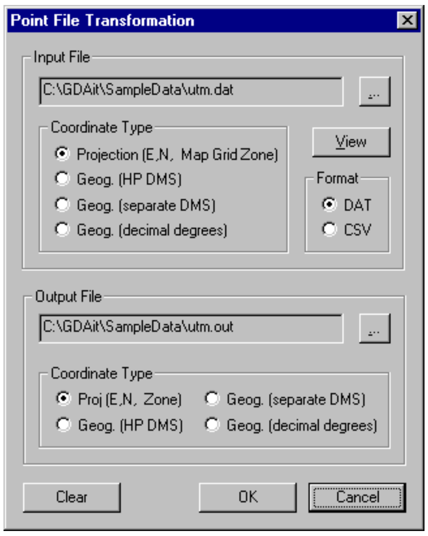
These files store the resources that were created for *GDAit*, i.e. the dialog boxes, menus, logos and icons. The RC2 files is used to include the NRE and GDA logos as they contain different colours to the default 16 colour palette used by the resource editor in the 16 bit version of Visual C++.

3.5 Class Wizard Files

The resources created in App Studio are connected to class files by Class Wizard. The following were created to implement functionality to resources created with App Studio:

3.5.1 InputDlg (.h and .cpp)

These files implement the functionality of the Point File Transformation dialog box (Figure 3.3). This dialog box is used to enter the names of an input and output file. The dialog also accepts the file format and type of coordinates.


Point File Transformation

Input File: C:\GDAit\SampleData\utm.dat

Coordinate Type: [✓ Projection (E,N, Map Grid Zone)] Geog. (HP DMS) Geog. (separate DMS) Geog. (decimal degrees)

Format: [✓ DAT] CSV

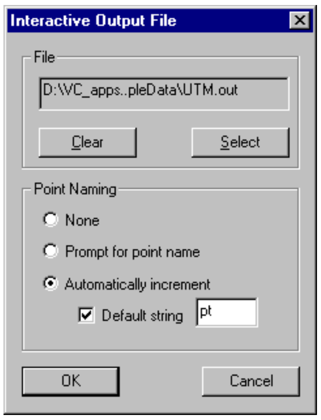
Output File: C:\GDAit\SampleData\utm.out

Coordinate Type: [✓ Proj (E,N, Zone)] Geog. (separate DMS) Geog. (HP DMS) Geog. (decimal degrees)

Figure 3.3 - Point File Transformation dialog box

3.5.2 OutptDlg (.h and .cpp)

These files implement the functionality of the Interactive Output dialog box (Figure 3.4). This dialog is used to enter the name of an output file for points converted in ***Interactive Mode***. The dialog box gets the type of coordinate to output and whether or not a point identifier is required.


Interactive Output File

File

D:\VC_apps..pleData\UTM.out

Point Naming

( ) None
( ) Prompt for point name
(●) Automatically increment
[✓] Default string: pt

Figure 3.4 - Interactive Output File dialog box

3.5.3 PtNameDg (.h and .cpp)

These files implement the functionality of the Point Name dialog box. If *Prompt for point name* was selected in the Interactive Output File dialog box (Figure 3.4), then the dialog box associated with these files is displayed to get the name of the point.

3.5.4 DispDlg (.h and .cpp)

These files implement the functionality of the Coordinate Display dialog box. This is a dialog box of six radio buttons (three for coordinate input type and three for coordinate output type).

3.6 General C++ Files

The following general C++ files were created to perform operations that are common to both the PC and Web versions of *GDAit*. These files are part of a library of C functions used by other applications so they contain some functions that aren't used in *GDAit*.

3.6.1 Angles (.h and .cpp)

The functions defined by these files are used to convert angular values between various formats, e.g. between radians and the various degree formats.

3.6.2 CordTran (.h and .cpp)

The functions defined by these files implement Coordinate Transformation functions which convert coordinates between Universal Transverse Mercator and Geographical coordinates. The header file declares a point structure which is also used by the View class.

3.6.3 GridFile (.h and .cpp)

The functions in these files are the main component of the interpolation routine. A structure is defined (**t_SubGrid**) which is used to represent a summarised version of the grid file to enable fast access to individual grid shift values. The file also implements functions to:

- Read and write the grid file in NTv2 format (ASCII and binary).
- Retrieve individual records from the grid file.
- Interpolate the shifts for a point within the extents of the grid file.

3.6.4 BMPapi (.h and .cpp)

The functions contained in this class are used to display the logos on the *GDAit* interface. These logos contain colours that are different to the default 16 colour palette that the LoadBitmap() API function can display truly. The functions in this file create a palette for the logo bitmaps and display them with the best palette that the video display permits. The three logos were created with a common 16 colour palette to minimise the demands on the video display.

3.7 Online Help

GDAit has online help. The help file was generated using Microsoft's Help Compiler version 3.1 to allow compatibility with the 16 bit version. The help is not context sensitive and the context ID of the contents page is hard coded. Requests to display the Help file are overridden in the application class, where a context ID of 101 is passed to WinHelp().

3.8 Compilation Instructions

Compiling *GDAit* requires the appropriate compiler for the destination platform: Visual C++ 1.52 C for the 16 bit version, or Visual C++ 5 for the 32 bit version. It may also be possible to compile the 32 bit version with version 4 of the compiler.

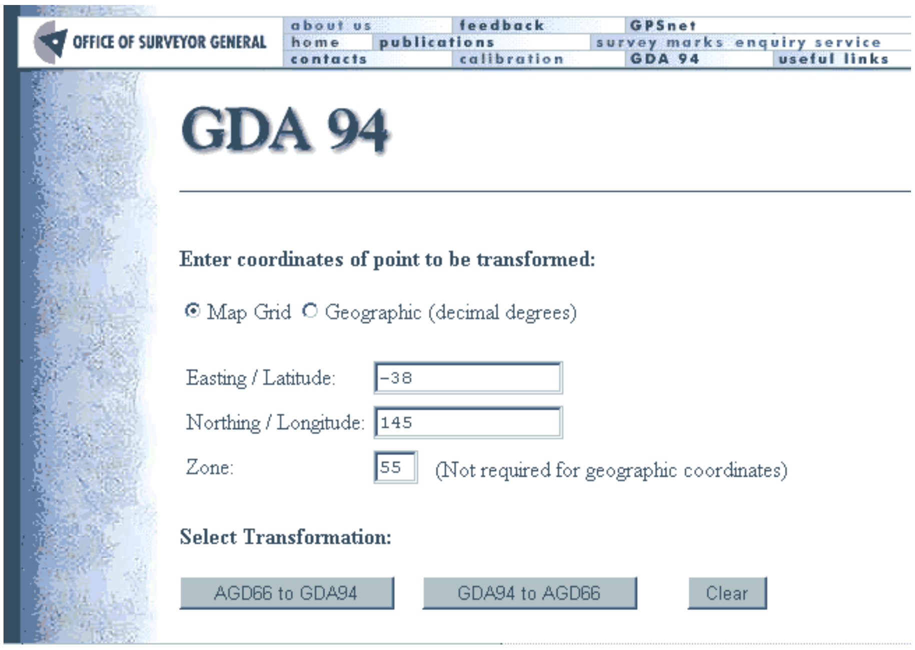
The source code for the 16 and 32 bit versions is different so a separate project is required for each type. It is not possible to use conditional compilation flags to compile the different versions. The two versions can share the General C++ files with the exception of Angles.cpp, which is unsuitable for the 16 bit version due to the 16 bit version's handling of CString objects. (CString objects are used within the 16 bit version but they have to be cast to (const char*)).

The source files, resource files, class wizard file and project workspace files are included on the source code disk. Opening the project workspace should recreate the project.

4. *GDAit:* Web Version

4.1 Overview

The Web version of *GDAit* is written as a cgi-bin application. The program operates by accepting input from a HTML Form that is sent using the "**GET**" submit method. After decoding and validating the input data, the coordinates are transformed and then written back to the user's web browser as a HTML document. The data written back to the browser is contained within a HTML template file which specifies the appearance of the page. The template file is read at run time so it can be easily changed. The interface of the Web version used during testing is shown in Figure 4.1.



Enter coordinates of point to be transformed:

[✓ Map Grid] Geographic (decimal degrees)

Easting / Latitude:	-38
Northing / Longitude:	145
Zone:	55 (Not required for geographic coordinates)

Select Transformation:

Figure 4.1 - *GDAit* Web Interface

4.2 Program Structure

The structure of the Web version of *GDAit* is slightly different to the PC version as there is no file input or output options. The program does use a number of files to read data from though, but these are not specified by the user. The following diagram shows the flowchart for the Web version:

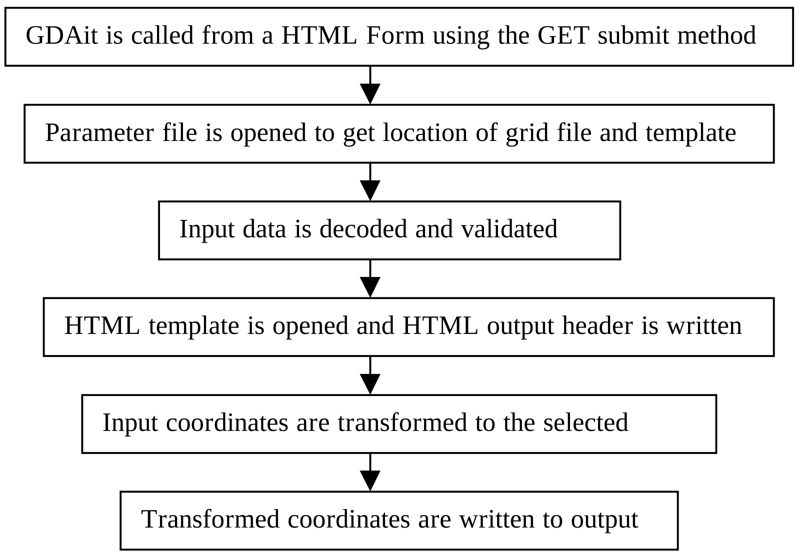

GDAit is called from a HTML Form using the GET submit method
Parameter file is opened to get location of grid file and template
Input data is decoded and validated
HTML template is opened and HTML output header is written
Input coordinates are transformed to the selected
Transformed coordinates are written to output

Figure 4.2 - *GDAit* Web Version Flowchart

4.3 HTML Input Form

The Web version uses a HTML Input Form that sends data from the user's web browser to *GDAit* which is located in the cgi-bin directory of the server. The data is sent to *GDAit* as a URLencoded string using the **get method**. The Input Form contains the five variables that *GDAit* receives (Appendix A contains a section of HTML code that implements these variables):

- **coordtype**: Coordinate type (projection or geographic in decimal degrees),
- **value1**: Latitude or Easting,
- **value2**: Longitude or Northing,
- **value3**: UTM zone (not required for geographic input),
- **direction**: the selected transformation direction.

The actual appearance of the Input Form is insignificant, as long as the above variables are specified within a **<FORM>** </**FORM>** code block, *GDAit* should be able to interpret the values. There are two things to note:

- **coordtype** uses a value of 2 for decimal degrees, and 3 for projection coordinates. (These are the values that are used in the coordinate transformation library code).
- The **action=** parameter must include a forward slash (/) at the end of the application name as *GDAit* uses the **PATH_TRANSLATED** environment variable to determine the location of the **document directory**. *GDAit* looks in the **document directory** for its parameter file **GDA_it.dat**. This file contains the location and name of the grid shift file, the location and name of the HTML template file and the two strings that define the start and end of the insert block in the template file. (**note:** If the parameter file can't be placed in the document directory, then it must be located in a subdirectory of it. In this case, the

additional directories beneath the document directory must be appended to the string used in the action statement so the file can be located).

4.4 Source Files

4.4.1 Web Version Specific Code

gda_it (.c and .h)

These files provide the driver code for *GDAit*. This file only contains two functions, the first is used to read the parameter file to find the location of the data files needed by *GDAit*. The other function is the main routine which sequentially calls functions to process the steps described by the flowchart in Figure 4.2. If any of the steps cause an error, program execution stops and an error message is returned to the web browser.

cgi_funcs (.c and .h)

These files provide the functions to decode the URLencoded string and extract values for the variables passed to *GDAit*. The string of variables that are passed to *GDAit* as input are stored as a linked list of individual strings. When the value for a variable is requested, the list is searched for the variable name and the values is returned. If the variable names doesn't exist an error code is returned.

The other main functionality of this file is for output, with functions to:

- Print an error message based on the value of an error code returned by a function.
- Print a sub section of a file where the section is defined by a start and end string. (This function is used to print the required sections of the HTML template file).
- Print the coordinates of a point into a HTML table so that the values are aligned.

4.4.2 General Functions

angles_w (.c and .h), gridfile_w (.c and .h),cordtran_w (.c and .h).

The functions here very similar to those for the PC version of *GDAit*. The only difference is that some modifications were required to make the files ANSI C compatible. This generally required all variables to be forward declared and removal of / / format comments.

These files have a "w" appended to the end of their names to denote that they are the Web version of the library file. For information about these file see section 2.5 in the PC section of this manual.

4.5 Compilation Instructions

The source files in this version of *GDAit* are all ANSI C compatible, so they can be compiled with either a C or C++ compiler. The program was compiled with the gcc compiler and the makefile used is included with the source code.

4.6 Installation Instructions

To install the GDAit on a web server:

In the cgi-bin directory:
- Copy the gda_it executable.
- Change the permissions on the executable to 755 or something so that the executable bits are set.

In the document root directory:
- Copy the gda_it parameter file (**gda_it.dat**) to the

In any directory as long as the permissions are suitable:
- Copy the grid shift file (**gridfile.gsb**). This is binary file so be sure to transfer the file as binary.
- Copy the HTML template file (**template.htm**).

Modify the gda_it parameter file so that:
- Line 20 contains the full path and name of the grid shift file.
- Line 21 contains the full path and name of the HTML template file.
- Line 22 contains the string where to **start** inserting the output
- Line 23 contains the string where to **finish** inserting the output

Check the HTML file that contains the Input Form so that:
- gda_it has a forward slash appended to it so that **PATH_TRANSLATED** can find the document root directory, eg . ACTION="/cgi-bin/gda_it/", will pass the **document root** to gda_it.
- If the parameter file is not located in the document root directory, append the subdirectory/s in which it is located, eg . ACTION="/cgi-bin/gda_it/osg/" will pass the **osg** subdirectory of **document root** to gda_it.

References

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

Appendix A. *GDAit* Web Version: HTML Input Form

```
<!-- ** gda_it form starts here ** -->
<!-- Written by David Mitchell -->
<!-- Department of Geomatics, University of Melbourne -->

<!-- Written as part of: -->
<!-- Land Victoria Project Number: LA/11/0002 -->

<!-- MUST include / on end of exe name so that the program -->
<!-- can get the document path using PATH_TRANSLATED environment -->
<!-- variable.  Also append path to location of gda_it.dat file -->
<!-- which contains the location of the grid and template files -->
<!-- as well as the start and end insert strings -->

<!-- Geomatics site path: -->
<FORM METHOD="GET" ACTION="/cgi-bin/gda_it/mitchell/">

<!-- OSG development site path: -->
<!-- <FORM METHOD="GET" ACTION="/cgi-bin/gda_it/"> -->

<!-- OSG live site path ?? (not known as 10/7/98) -->
<!-- <FORM METHOD="GET" ACTION="/cgi-bin/gda_it/"> -->

<!-- input value elements -->

<h4>Enter coordinates of point to be transformed:</h4>
<INPUT TYPE="RADIO" NAME="coordtype" VALUE="3" CHECKED>Map Grid
<INPUT TYPE="RADIO" NAME="coordtype" VALUE="2">Geographic (decimal
degrees)
<br>


<!-- Insert the entry elements in a table to force alignment -->

<P>
<TABLE>
 <TR>
  <TD>Easting / Latitude:</TD>
  <TD><INPUT TYPE="TEXT" NAME="value1" VALUE="0" SIZE=15></TD>
 </TR>
 <TR>
  <TD>Northing / Longitude:</TD>
  <TD><INPUT TYPE="TEXT" NAME="value2" VALUE="0" SIZE=15></TD>
 </TR>
 <TR>
```

```
  <TD>Zone:</TD>
  <TD><INPUT TYPE="TEXT" NAME="value3" VALUE="55" SIZE=2> &nbsp; (Not
required for geographic coordinates)</TD>
 <TR>
</TABLE>


<!-- Display two buttons to select a transformation direction -->
<!-- and display a clear button -->

<h4>Select Transformation:</h4>
<P><P>
<INPUT TYPE="SUBMIT" NAME="direction" VALUE="AGD66 to GDA94">
&#160 &#160
<INPUT TYPE="SUBMIT" NAME="direction" VALUE="GDA94 to AGD66">
&#160 &#160 &#160 &#160
<INPUT TYPE=RESET NAME="RESET" VALUE=" Clear ">
<BR>

</FORM>

<!-- ** gda_it form ends here ** -->
```

Appendix B. Parameter File for *GDAit* Web Version

```
Data file for cgi-bin application "gda_it".

This file is used by the cgi-bin application "gda_it" to locate
its data file, HTML template file and directory, please do
not remove it or the program will not work.

This file must be located in the web document directory.

If you modify the location of the files, make sure they are
appear in the following order:
- line 20 = name and full path of grid shift file
- line 21 = name and full path of HTML template file
- line 22 = "start insert" string in template file
- line 23 = "end insert" string in template file

** IMPORTANT: don't include an exclamation (!) character in the
** start or insert strings. To be safe use only characters in
** range a to z (upper and lower case).

/usr/htdocs/osg/gridfile.gsb
/usr/htdocs/osg/my_osg.htm
--insert the content here open--
--insert the content here close--
```

Appendix C. NTv2 Grid Shift File

C.1 Overview

A Grid Shift file 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.

A Grid Shift file contains one or more rectangular grids which are referred to as "sub grids". Each sub grid consists of nodes which are spaced at regular intervals of latitude and longitude. Each node has a shift value for both latitude and longitude components, as well as a value for the accuracy of each shift. The nodes in a Grid Shift file are written in rows starting in the south east (lower right) corner, and finishing in the north west (upper left) corner. The order and orientation of a sub grid file is represented in Figure C.1. (Refer to section C.3 for information on the use of negative longitude values).

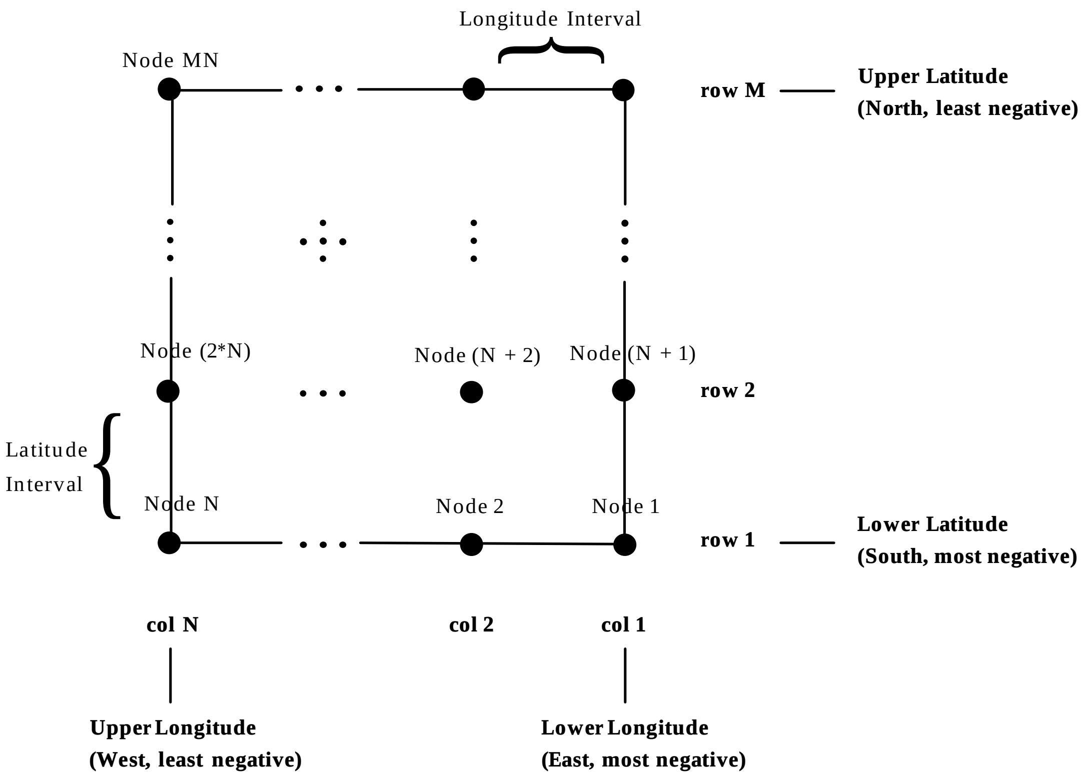


Figure C.1 - Grid Node Order

In its simplest form (Figure C.2), a Grid Shift file defines a single sub grid, however it can contain more than one sub grid (Figure C.3). If multiple sub grids are defined, the sub grids do not have to be contiguous. Sub grids of different densities can overlap each other if certain rules are adhered to (for more information see Junkins and Farley, 1995).

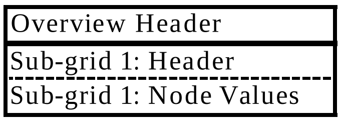


Figure C.2 - Grid Shift file with a single sub grid

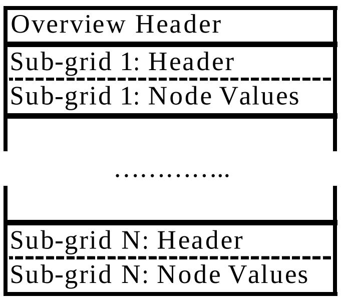


Figure C.3 - Grid Shift file with multiple sub grids

C.2 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 is a pure byte dump. The *NTv2 Developer's Guide* 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 C.1 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 C.2 to C.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).

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

Table C.1 - Grid Shift File Data Types

C.2.1 Grid Shift file Overview

Table C.2 specifies the format of the Overview section of a Grid Shift file and a sample is given in Figure C.4. 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.

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

Table C.2 - Grid Shift File Overview Information

```
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
```

Figure C.4 - Sample Grid Shift file Overview

C.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 C.3 and a sample is given in Figure C.5.

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

Table C.3 - Sub Grid Overview Information

```
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
```

Figure C.5 - Sample Sub Grid Overview

Table C.4 specifies the values at each grid node and a sample is given in Figure C.6. 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 can not be modeled, 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).

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

Table C.4 - 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
  .....................
```

Figure C.6 - Sample Sub Grid Node Values

C.3 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).

In Figure C.1, the affect of this *Positive West* longitude convention is to reverse the direction of the X axis, i.e. values become more positive moving from right to left. This will not change the orientation of objects within the system as Figure C.7 demonstrates.

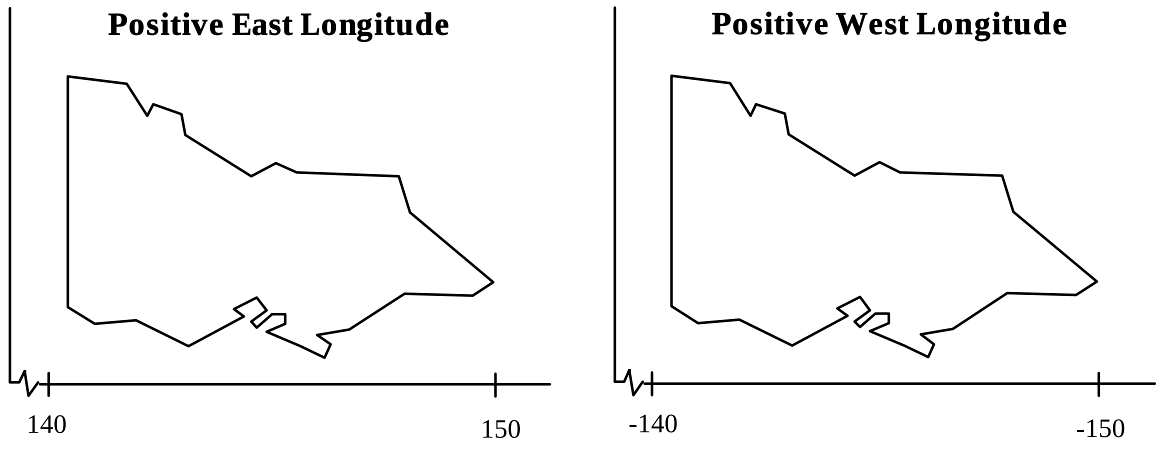


Figure C.7 - Longitude Axis Orientation

The sign reversal of the longitude axis explains why the sub grid parameters Upper and Lower Longitude in Figure C.1 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.

C.4 Difference between Australian and Canadian binary files

NTv2 Grid Shift Files can exist in two forms, binary and ASCII. In the initial implementation of NTv2 in Australia, the only version of a Grid Shift File that strictly adhered to the NTv2 file format was 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. To eliminate confusion, use of *Australian* binary is being phased out, and *Canadian* binary will become the standard for Grid Shift File distribution in Australia. The following explains the difference between the two binary forms and outlines the reason why it occurred.

(**note:** Prior to version 2.0 of *GDAit*, only NTv2 files in ASCII and *Australian* binary could be read. Version 2.0 is capable of reading NTv2 files in both *Australian* and *Canadian* binary format).

C.4.1 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 DON'T 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 (see Figure C.3). 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 C.8 with the number of bytes required being given in brackets.

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

Figure C.8 – Data storage in a Header record

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 C.2) are NUM_OREC, NUM_SREC and NUM_FILE. The record affected in the Sub-grid Header (see Table C.3) 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.

C.4.2 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 has now become apparent that NTv2 binary is 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.

Appendix D. Bi-Linear Interpolation

Bi-linear interpolation is the technique used in *GDAit* to determine the transformation and accuracy components at a non-grid point from the nearest four grid nodes. The procedure is illustrated in Figure D.1 where the transformation components at the interpolation point (p) are required. These are computed from the known transformation components at the four grid nodes (A, B, C and D). To compute the latitude transformation component at p, the equation is :

$$\delta\phi_p = a_0 + a_1X + a_2Y + a_3XY \qquad \ldots(d.1)$$

where

$$a_0 = \delta\phi_A \qquad \ldots(d.2)$$

$$a_1 = \delta\phi_B - \delta\phi_A \qquad \ldots(d.3)$$

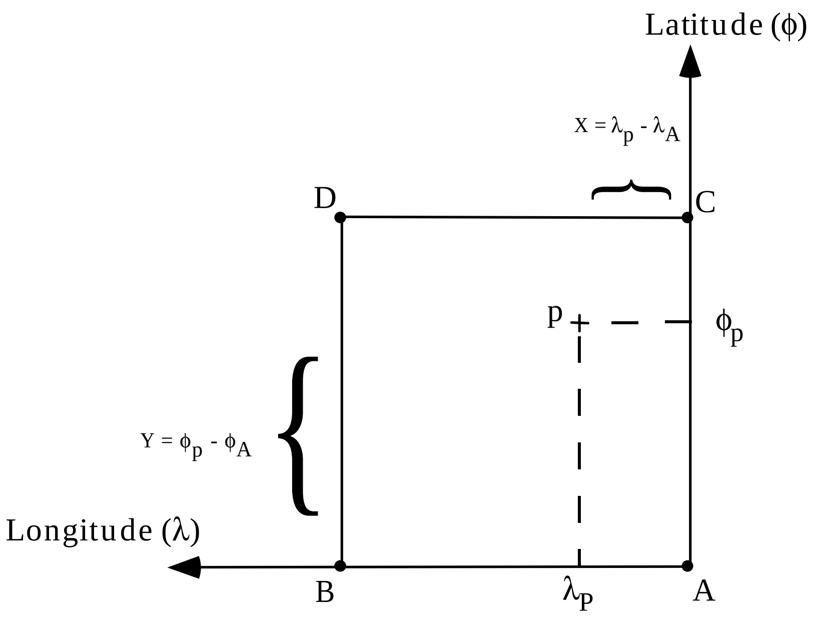
$$a_2 = \delta\phi_D - \delta\phi_A \qquad \ldots(d.4)$$

$$a_3 = \delta\phi_A + \delta\phi_C - \delta\phi_B - \delta\phi_D \qquad \ldots(d.5)$$

$$X = (\lambda_p - \lambda_A) / (\lambda_B - \lambda_A) \qquad \ldots(d.6)$$

$$Y = (\phi_p - \phi_A) / (\phi_C - \phi_A) \qquad \ldots(d.7)$$

$\delta\phi_A$, $\delta\phi_B$, $\delta\phi_C$ and $\delta\phi_D$ are the latitude transformation components at points A to D respectively.



<u>Figure D.1</u> – Bi-Linear Interpolation

By substituting $\delta\lambda$ for $\delta\phi$ in equations (d.1) to (d.5) the longitude transformation component at p can also be computed. Identical equations can also be used to interpolate the accuracy of the transformation in both dimensions.

Appendix E. Sample Transformation Calculations

E.1 Forward Transformation

Coordinates of AGD66 point:

$\phi_{AGD66}(\phi_p) =$	$-37^{o}\ 47'$	= -136020	"
$\lambda_{AGD66}(\lambda_p) =$	$144^{o}\ 57'$	= 521820	" (postive east longitude)
		= - 521820	" (positive west longitude)

Sub Grid File Parameters:

ϕ_{lower} (S_LAT)	-138780	"
ϕ_{upper} (N_LAT)	-134406	"
λ_{lower} (E_LONG)	-526104	"
λ_{upper} (W_LONG)	-519354	"
$\Delta\phi$ (LAT_INC)	54	"
$\Delta\lambda$ (LONG_INC)	54	"

1. Compute Fundamental Grid Parameters:

M (# of rows) = $1 + \text{integer}\ ((\phi_{upper} - \phi_{lower})\ /\ \Delta\phi)$

= 1 + integer((-134406 - -138780)) / 54)

= $\boxed{82}$

N (# of columns) = $1 + \text{integer}\ ((\lambda_{upper} - \lambda_{lower})\ /\ \Delta\lambda)$

= 1 + integer((-519354 - -526104) / 54)

= $\boxed{126}$

2. Determine Grid Nodes to interpolate from (refer to Figure 2.3):

a) Calculate row and column of grid node A (equations 2.1 and 2.2):

i = $1 + \text{integer}(\ (\phi_p - \phi_{lower})\ /\ \Delta\phi)$

= 1 + integer((-136080 - -138780) / 54)

= 1 + integer(51.111)

= $\boxed{52}$

j = $1 + \text{integer}(\ (\lambda_p - \lambda_{lowe}$

= 1 + integer((-521820

= 1 + integer(79.333)

= $\boxed{80}$

b) Use Row and Column to Compute Grid Node Number (equation 2.3):

Node A = N*(i-1) + j = 126*51 + 80	6506
Node B = Node A + 1	6507
Node C = Node A + N	6632
Node D = Node C + 1	6633

3. Retrieve the Values at these four grid nodes (Melboune Grid File, September 11 1998):

	$\delta\phi''$	$\delta\lambda''$	σ_ϕ''	σ_λ''
Node A	5.42432	-4.69423	0.000179	0.000575
Node B	5.42452	-4.69714	0.000391	0.000829
Node C	5.42498	-4.69242	0.000130	0.000908
Node D	5.42430	-4.69563	0.000199	0.000512

4. Compute Coordinates of Grid Node A:

$\phi_A = \phi_{lower} + (i - 1) * \Delta\phi$

$= -138780 + (52 - 1) * 54$

$= \boxed{-136026''}$

$\lambda_A = \lambda_{lower} + (j - 1) * \Delta\lambda$

$= -526104 + (80 - 1) * 54$

$= \boxed{-521838''}$

5. Compute interpolation scale factors (equations c.6 and c.7):

$X = (\lambda_P - \lambda_A) / \Delta\lambda$

$= (-521820 - - 521838) / 54$

$= \boxed{0.333333}$

$Y = (\phi_P - \phi_A) / \Delta\phi$

$= (-136020 - - 136026) / 54$

$= \boxed{0.111111111}$

6. Compute Latitude interpolation parameters (equations c.2 to c.5):

$a_0 = \delta\phi_A$	5.42432 ''	(5.42432)
$a_1 = \delta\phi_B - \delta\phi_A$	0.00020 ''	(5.42452 - 5.42432)
$a_2 = \delta\phi_C - \delta\phi_A$	0.00066 ''	(5.42498 - 5.42432)
$a_3 = \delta\phi_A + \delta\phi_D - \delta\phi_B - \delta\phi_C$	-0.00088 ''	(5.42432 + 5.4243 - 5.42452 - 5.42498)

7. Interpolate value (equation c.1):

$\delta\phi_P = a_0 + a_1X + a_2Y + a_3XY$

$= \boxed{5.424427}$ ''

8. Add interpolated shift to AGD66 latitude to compute GDA94 latitude:

$\phi_{GDA94} = \phi_P + \delta\phi_P$

$= -136020'' + 5.424427''$

$= \boxed{-37^o\ 46'\ 54.57557''}$

9. Repeat steps 6 to 8 to compute GDA94 longitude.

Summary of values to compute latitude and remaining three values:

	Shift		Accuracy	
	ϕ''	λ''	σ_ϕ''	σ_λ''
a_0	5.424320	-4.694230	0.000179	0.000575
a_1	0.000200	-0.002910	0.000212	0.000254
a_2	0.000660	0.001810	-0.000049	0.000333
a_3	-0.000880	-0.000300	-0.000143	-0.000650
δ_P	5.424427	-4.695010	0.000239	0.000673

ϕ_{GDA94}	-37^o 46' 54.57557"	(136014.57557")
λ_{GDA94}	-144^o 57' 04.69501"	(-521824.69501")
σ_ϕ	0.007 m	(0.000239" $* \pi *$ 6378160 / (3600 * 180))
σ_λ	0.016 m	(cos (37 46') * 0.000673" $* \pi *$ 6378160 / (3600*180))

E.2 Reverse Transformation

Coordinates of GDA94 point:

ϕ_{GDA94} = $\boxed{-37^o\ 46'\ 54.57557''}$ = -136014.57557"

λ_{GDA94} = $\boxed{144^o\ 57'\ 04.69501''}$ = - 521824.69501" (positive west longitude)

Sub Grid File Parameters: same as for forward transformation example

1. Compute Fundamental Grid Parameters: same as for forward transformation example

2. Set initial estimate of AGD66 coordinates to the GDA94 coordinates of the point.

$\phi'_{AGD66} = \phi_{GDA94}$

$\lambda'_{AGD66} = \lambda_{GDA94}$

3. Set coordinates of Point P to current estimate of AGD66 coordinates.

$\phi_P = \phi'_{AGD66}$

$\lambda_P = \lambda'_{AGD66}$

4. Determine Grid Nodes to interpolate from (refer to Figure 2.3):

a) Calculate row and column of grid node A (equations 2.1 and 2.2):

i = 1 + integer(51.21156) =	52
j = 1 + integer(79.24639) =	80

b) Use Row and Column to Compute Grid Node Number (equation 2.3):

Node A = N * (i-1) + j = 126 * 51 + 80 = 6506	6506
Node B = Node A + 1 =	6507
Node C = Node A + N =	6632
Node D = Node C + 1 =	6633

5. Retrieve the Values at these four grid nodes (Melboune Grid File, September 11 1998):

	$\delta\phi''$	$\delta\lambda''$	σ_ϕ''	σ_λ''
Node A	5.42432	-4.69423	0.000179	0.000575
Node B	5.42452	-4.69714	0.000391	0.000829
Node C	5.42498	-4.69242	0.000130	0.000908
Node D	5.42430	-4.69563	0.000199	0.000512

6. Compute Coordinates of Grid Node A:

$\phi_A = \phi_{lower} + (i - 1) * \Delta\phi$

= $\boxed{-136026''}$

$\lambda_A = \lambda_{lower} + (j - 1) * \Delta\lambda$

= $\boxed{-521838''}$

7. Compute interpolation scale factors (equations c.6 and c.7):

X = $(\lambda_P - \lambda_A) / \Delta\lambda$

= (-521824.69501 - - 521838) / 54

= $\boxed{0.246389}$

Y = $(\phi_P - \phi_A) / \Delta\phi$

= (-136014.57557 - - 136026) / 54

= $\boxed{0.211564}$

8. Compute interpolation paramaters (equations c.2 to c.5):

	$\Delta\phi$"	$\Delta\lambda$"	σ_ϕ"	σ_λ"
a_0	5.424320	-4.694230	0.000179	0.000575
a_1	0.000200	-0.002910	0.000212	0.000254
a_2	0.000660	0.001810	-0.000049	0.000333
a_3	-0.000880	-0.000300	-0.000143	-0.000650

9. Interpolate values (equation c.1):

	$\Delta\phi$"	$\Delta\lambda$"	σ_ϕ"	σ_λ"
δ_P	5.424463	-4.694580	0.000213	0.000674

10. Subtract interpolated shift to compute estimate of AGD66 coordinates:

$\phi'_{AGD66} = \phi_{GDA94} - \delta\phi_P =$	-136020.000033	"
$\lambda'_{AGD66} = \lambda_{GDA94} - \delta\lambda_P =$	-521820.000430	"

11. Iterate solution:

* Repeat steps 3 to 10 to compute better AGD66 coordinate estimates.

* Repeat three times.

	Iteration		
	1	2	3
ϕ_P	-136020.00003	-136020.00000	-136020.00000
λ_P	-521820.00043	-521820.00000	-521820.00000
i	52	52	52
j	80	80	80
ϕ_A	-136026	-136026	-136026
λ_A	-521838	-521838	-521838
X	0.3333254	0.3333333	0.3333333
Y	0.1111105	0.1111112	0.1111112
$\delta\phi_P$	5.424427	5.424427	5.424427
$\delta\lambda_P$	-4.695010	-4.695010	-4.695010
$\sigma\phi$	0.000239	0.000239	0.000239
$\sigma\lambda$	0.000673	0.000673	0.000673
$\phi'_{AGD66} = \phi_{GDA94} - \delta\phi_p$	-136020.00000	-136020.00000	-136020.00000
$\lambda'_{AGD66} = \lambda_{GDA94} - \delta\lambda_p$	-521820.00000	-521820.00000	-521820.00000

12. Solution:

Coordinate estimates ϕ' and λ' after iteration 3 become the final coordinates.

$\phi_{AGD66} =$	-37° 47' 00"	-136020.00000 "
$\lambda_{AGD66} =$	144° 57' 00"	-521820.00000 "
$\sigma\phi_{AGD66} =$	0.007 m	0.000239 "
$\sigma\lambda_{AGD66} =$	0.016 m	0.000673 "