GUT

User Guide and Algorithm Descriptions

Contents

Document Change Record

Version	Date (yyyy-mm-dd)	Prepared by	Change Description
1.0	2008-07-10	Ian Price	Initial version.
1.1	2008-12-03	Ian Price	Additional content and minor corrections.
1.2	2009-01-20	Ian Price	General revision.

References

[GUT-TUT] "GUT Tutorial", Version 1.0, Rio et.al., Mar-2008.

[SJÖ] "A Reccurence Relation for the β_n-Function", L. Sjöberg, Bull. Geod. 54, pp. 69-72, 1980.

[GOCE-PDH] "GOCE Level 2 Product Data Handbook", Issue 4, Revision 0, 09-06-2008.

Acronyms

API	Application Programming Interface
BRAT	Basic Radar Altimetry Toolbox
DEM	Digital Elevation Model
EFRF	Earth-Fixed Reference Frame
GOCE	Gravity Field and Steady-State Ocean Circulation Explorer
GUT	GOCE User Toolbox
KML	Keyhole Markup Language
MSSH	Mean Sea Surface Height
netCDF	Network Common Data Format
TIFF	Tagged Image File Format
XML	eXtensible Markup Language

1 Introduction

1.1 Scope

This document describes the installation and usage of the GOCE User Toolbox (GUT). It also documents the fundamental data types, file formats, high-level algorithms and independent processing units implemented in the GUT Software.

1.2 Summary

The installation procedure is described in §2. A brief introduction to working with GUT follows in §3. This is designed to familiarize the user with the GUT command-line tool interface. A more elaborate introduction is presented in the GUT Tutorial. The spherical harmonic mathematics, fundamental to GUT algorithms, is summarized in §4. A description of the high-level data types used by GUT, and the file format used to store them are presented in §5 and §6. Detailed descriptions of the core algorithms, as implemented in GUT, are presented in §7. A full list of processing units, their port names and data types, and a brief description of their function is given in §8. This information is targeted at users building their own work-flows. A brief overview of the BratDisplay visualization tool is given in §9. For advanced users, extending GUT with custom software modules is discussed in §10.

2 Installing GUT

GUT is comprised of a command-line tool, a suite of work-flows that describe typical processing tasks, a collection of input data sets and optionally a stand-alone display tool (BratDisplay). The command-line tool has been developed as an OpenSource software project and the suite of work-flows are XML files that are incorporated as an integral part of the GUT source code package. On most platforms installation of the GUT is based on the source package. On a limited set of platforms GUT is also available as a binary package. For all platforms the input data sets are provided as a platform independent binary package that may be added to the basic GUT installation. The optional display tool is provided as a platform specific add-on binary package. The installation procedures for both source and binary packages is described in the following sections.

2.1 Optional Extensions

GUT has two extensions that are enabled by default, but may be disabled when building from the source package. These extensions are GeoTIFF and Fortran support.

The GeoTIFF extension allows gridded data sets to be exported to TIFF image files with special GeoTIFF tags used for recording geolocation meta data. These files are intended for quick-look usage only. The GeoTIFF package is included in the GUT source package but LibTIFF development files must be installed on your system to enable this feature. The latest stable LibTIFF package can be obtained from http://www.remotesensing.org/libtiff. The GeoTIFF extension can be disabled by adding the flag '--disable-geotiff' to the configure script.

GUT can be extended by developing additional processing units in the languages C++, C and if supported, Fortran. If this extension is enabled a Fortran compiler must be installed on your system. The Fortran extension can be disabled by adding the flag '--disable-fortran' to the configure script.

2.2 Installing from source

The GUT source package is applicable to any POSIX platform (i.e. UNIX, Linux, MacOS). To build and install the command-line tool you must have basic software development tools and libraries installed on your system. These are listed below, with optional components in parentheses.

- tar
- gunzip
- C++ Compiler (g++/GCC version 4 or higher recommended)
- make
- unzip decompression utility
- (Fortran90 compiler)
- (libtiff library and header files)

In the installation procedure that follows the use of `this font` indicates a shell command-line (*the* `%` *prefix indicates the shell prompt and should NOT be typed*), and uppercase tokens enclosed by angled brackets are `<VARIABLES>` that you must substitute with an appropriate value.

The build and installation procedure is :

- Create a temporary directory on your system.

  ```
  % mkdir <GUT_TMP_DIR>
  ```

- Obtain the GUT source package (gut-1.0.tar.gz) and the data set package (gut-apriori.zip) and place them in the `<GUT_TMP_DIR>` directory.
- cd to the `<GUT_TMP_DIR>` directory.
- Unpack the source package

  ```
  % tar zxf gut-1.0.tar.gz
  ```

  OR

  ```
  % gunzip gut-1.0.tar.gz
  % tar xf gut-1.0.tar
  ```

- cd into the source package root directory

  ```
  % cd gut-1.0
  ```

- Create a build subdirectory and cd into that directory

  ```
  % mkdir build
  % cd build
  ```

- Configure the source package. By default all features are enabled, so you will need to disable Fortran and/or geotiff support if you do not have the required components installed on your system. The '--help' option to the configure script will provide further details on disabling features or specifying the location of the TIFF library components. **You will need root privileges to install in the default location**. An alternative location can be specified with the '--prefix' option.

```
% ../configure --help
% ../configure --prefix=$HOME/local/GUT
```

- NOTE: If the configure script reports errors you may need to disable features or provide additional information on the command-line.
- Run make to build the command-line tool.

```
% make
```

- If required, change to the superuser account (or use sudo), and run make to install GUT. This installs the executable, work-flow suite and documentation.

```
% su
% make install
    OR
% sudo make install
```

- It is recommended that you add the location of the GUT command-line tool to your PATH. Consult the documentation for your login shell for details. NOTE, the GUT command-line tool executable (gut) is located in the bin subdirectory of the GUT installation (unless an alternative prefix was supplied to the configure script this will be /usr/local/GUT/bin)
- Install the data sets in the data package into the apriori subdirectory of the GUT installation.

```
% cd /usr/local/GUT/apriori
% unzip <GUT_TMP_DIR>/gut-apriori.zip
```

- The base installation is complete and you may now delete the `<GUT_TMP_DIR>` and all its subdirectories.

2.3 Installing the Linux binary package

The GUT binary distribution for the Linux platform includes binary executables for GUT and BratDisplay. These have been compiled on a Debian Etch system and are intended for 32-bit x86 systems using LIBC version 6. You may need to install additional runtime packages, appropriate for your Linux distribution, to use these executables. If this binary distribution of GUT is not suitable for your system you will need to build GUT from source.

The GUT installation is based on a simple directory structure with a single root directory. This directory may be located anywhere on your system, and /usr/local is the recommended location.

In the installation procedure that follows the use of `this font` indicates a shell command-line (*the % prefix indicates the shell prompt and should NOT be typed*), and uppercase tokens enclosed by angled brackets are `<VARIABLES>` that you must substitute with an appropriate value.

The installation procedure is :

- Obtain the GUT Linux binary package (gut-linux-1.0.tar.gz) and the data set package (gut-apriori.zip) and place them in a temporary directory, `<GUT_TMP_DIR>`.
- Select a destination directory, `<GUT_DEST_DIR>`, where GUT will be installed. You will need superuser privileges for the recommend location (/usr/local).
- If the destination directory does not exist, create it with the command

  ```
  % mkdir -p <GUT_DEST_DIR>
  ```
- Change to the destination directory

  ```
  % cd <GUT_DEST_DIR>
  ```
- Unpack the gut-linux-1.0.tar.gz package

  ```
  % tar zxf <GUT_TMP_DIR>/gut-linux-1.0.tar.gz
  ```

  OR

  ```
  % gunzip <GUT_TMP_DIR>/gut-linux-1.0.tar.gz
  %  tar xf <GUT_TMP_DIR>/gut-linux-1.0.tar
  ```
- It is recommended that you add the location of the GUT command-line tool to your PATH. Consult the documentation for your login shell for details. NOTE, the GUT command-line tool executable (gut) is located in the bin subdirectory of the GUT installation (`<GUT_DEST_DIR>`/GUT/bin)
- Install the data sets in the data package into the apriori subdirectory of the GUT installation.

  ```
  % cd <GUT_DEST_DIR>/GUT/apriori
  % unzip <GUT_TMP_DIR>/gut-apriori.zip
  ```
- The installation is complete and you may now delete the binary package files.

2.4 Installing the Windows binary package

The GUT distribution for the Windows platform is a pair of ZIP files. The installation instructions are listed below.

- Obtain the GUT Windows binary package (gut-win32-1.0.zip) and the data set package (gut-apriori.zip) and place them in a temporary folder.
- Unzip the gut-win32-1.0.zip in a folder of your choice with a ZIP utility program (ie. C:\Software). Note that the folder 'GUT' will be created when the package is unzipped.
- Unzip the gut-apriori.zip package into the apriori folder in the GUT folder (ie. C:\Software\GUT\apriori).

- Edit the PATH variable to include the bin folder of the GUT package.
  - Start the Control Panel and select System.
  - Click the Advanced tab and then click the Environment Variables button.
  - Edit the PATH variable in the System Variables list.
  - Add ;"C:\Software\GUT\bin" to the value of the PATH.
- Start the Command Prompt application, type the command gut and press the Enter key. If the PATH was set correctly the gut command-line tool will run and display a usage message. The installation is now complete.

3 Working with GUT

3.1 Getting Started

The GUT command-line tool (gut) supports a wide range of algorithms, each performing a specific task and having well defined inputs and outputs. These algorithms are the building blocks used to assemble dedicated data-chain processors. The building blocks are referred to as *processing units* (or just *units*) and an assembly is referred to as a work-flow. GUT facilitates flexible data processing because the definition of a work-flow is external to the command-line tool. This definition is in the form of an XML file (human readable) that is provided to gut as input. The base GUT installation provides a suite of common-use work-flows, but you can extend the suite by creating additional definition files. This is not particularly complicated, but it is recommended that you start out by using the predefined work-flows. Typically usage of GUT is also demonstrated in the GUT Tutorial.

The GUT command-line tool is primarily a data-processing tool but it supports several other features to assist the user. Each feature is enabled by a specific GUT run-mode. The examples that follow provide a brief introduction to using the most significant run-modes. These examples use `fixed width font` with a `%` prefix to indicate command-lines. These can by typed in your terminal verbatim (without the `%` prefix that represents the prompt).

3.1.1 Usage

Since gut is a command-line tool, it must be run from a *terminal*. In Windows, the terminal is the *Command Prompt* application.

When run without arguments, gut will show the available run-modes and options.

```
% gut
```

3.1.2 Help

The help run mode provides more detail information about the options and other run-modes.

```
% gut --help
```

3.1.3 Version

The GUT installation has strict requirements on its relative directory structure (this structure is assured during installation), but the entire GUT installation can be located (or relocated) anywhere on your system. When executed, gut determines where it is located and uses this information when searching for input files. The default location for work-flow files and data sets are reported, along with version information, when executing this run-mode.

```
% gut --version
```

3.1.4 Work-flow execution

The primary run-mode is for data-processing via a specific work-flow. The arguments that must be supplied to gut are largely determined by the content of the work-flow, but at least the name of a work-flow must be supplied. This name is used to locate a specific work-flow definition file. Any additional command-line arguments are supplied in "*-key value*" pairs (in specific cases the value may be omitted). Since processing is based on data files it is typical for a least one input file to be specified. As an example, the following command-line will compute the Geoid Height on a regular grid from spherical harmonic potential coefficient data stored in the example GOCE Level-2 data product supplied with the base installation. NOTE, before running this command you should *cd* to a directory where you have write permission.

```
% gut geoidheight_gf -InFile GUT_GOCE2_Example.HDR
```

Once completed the file *geoid_height.nc* should have been created in the current directory.

The example command-line above has the minimal amount of information required to execute the geoidheight_gf work-flow, but many other options can be specified. Because the options are specific to the work-flow they are documented within each workflow definition file. This work-flow manual is displayed in the terminal using the man run-mode. This is enabled simply by adding the option '--man' to the command-line.

```
% gut geoidheight_gf -InFile GUT_GOCE2_Example.HDR --man
```

In addition to the displaying the manual, which describes the possible arguments, this will

report the filename of the work-flow definition file located by gut.

3.1.5 Work-flow suite

The workflows run-mode displays a list of all work-flow definition files that are located in the GUT installation.

```
% gut --workflows
```

The suite can be extended be placing additional definition files in the workflow subdirectory of the GUT installation. Note that work-flow definition files DO NOT need to be located in this directory to allow processing (see the run-time help information for details about the file search mechanism).

3.1.6 Checking the Input

The test run-mode performs all steps of the primary run-mode except data processing. This allows the structural validity of a work-flow and the existence of input files to be checked. This run-mode is invoked by adding the option '--test' to the command-line.

```
% gut geoidheight_gf -InFile GUT_GOCE2_Example.HDR --test
```

3.2 Beyond the basics

A great deal of flexibility is provided by GUT by the data-driven nature of the work-flow processing scheme. Developing new work-flows is the simplest mechanism for extending the toolkit and the dot run-mode is intended to assist users with this development. Before presenting this feature it is necessary to describe the structural assembly requirements and inter-unit data-flow mechanism of processing units.

3.2.1 Directed Acyclic Graph

A work-flow is composed of a number of processing units. Each unit obtains or is supplied data at its inputs, processes this data and publishes any results at its outputs. In order to define a complete data-chain processor output data from one unit must be supplied as input to other units. Therefore the connections between input and output must also be defined in the work-flow definition file. This combination of units and connections forms a directed graph (units are nodes, connections are edges and information flows along edges in a specific direction), but there are strict requirements on the structure of this graph. The primary restrictions are that each unit has a unique identifier (name) and that the graph is acyclic (it cannot contain any loops). The requirement for unique unit names ensures that connections can be specified unambiguously. The acyclic requirement ensures that the output data from a unit is not, directly or indirectly, also input data to the same unit. These restrictions are enforced by gut. If satisfied, a dependency analysis can be performed on the graph and the processing order of the units set appropriately.

3.2.2 Input and Output Ports

In order to perform its specific task a processing unit requires input data. The general mechanism for obtaining this data is via a connection to another unit, though some units obtain information from the command-line arguments. The set of inputs and outputs of any given unit are intrinsic properties of that unit, each with a unique name and facilitating the flow of a specific type of data. This information is used when defining the connections of a work-flow and is documented in §8. *It is a requirement that all inputs of all units in a work-flow are connected to precisely one output of the corresponding type*. This is strictly enforced by gut. Outputs do not need to be connected and an output can be connected to more than one input.

3.2.3 Command-line input

A work-flow defines the data-flow within a processing chain, but in general it does not define the data that is to be processed. To facilitate providing data to (and exporting data from) the work-flow, GUT has a specific group of processing units. These units can extract information from the gut command-line and publish that information as an output. They also provide a mechanism for specifying a default value, in case the user did not provide the required command-line information. These units, which has type names beginning with CmdLineArg, are used in the work-flow assembly just like all other types of units.

3.2.4 Structural Verification

The GUT command-line tool will check that all of the restrictions imposed on the work-flow are satisfied, but passing these checks does not guarantee that the resultant process chain is what the work-flow designer intended. The syntax of the work-flow files is simple, but it does not readily convey the graph structure. The dot run-mode aids development and documentation of work-flows by exporting a work-flow file in a format that can be used by the GraphViz software (an open source project). This run mode can be applied to any work-flow file and is usable on incomplete or incorrectly structured work-flows. In this run-mode gut will report errors and export to the dot file as much information as it can.

```
% gut --dot geoidheight_gf
```

The dot file is created in the current directory with a filename that is the work-flow name with the ".dot" extension. The *dot* and *dotty* tools from the GraphViz package will permit visualization of the graph. The image generated by the dot tool from the output of the command above is shown in figure 1.

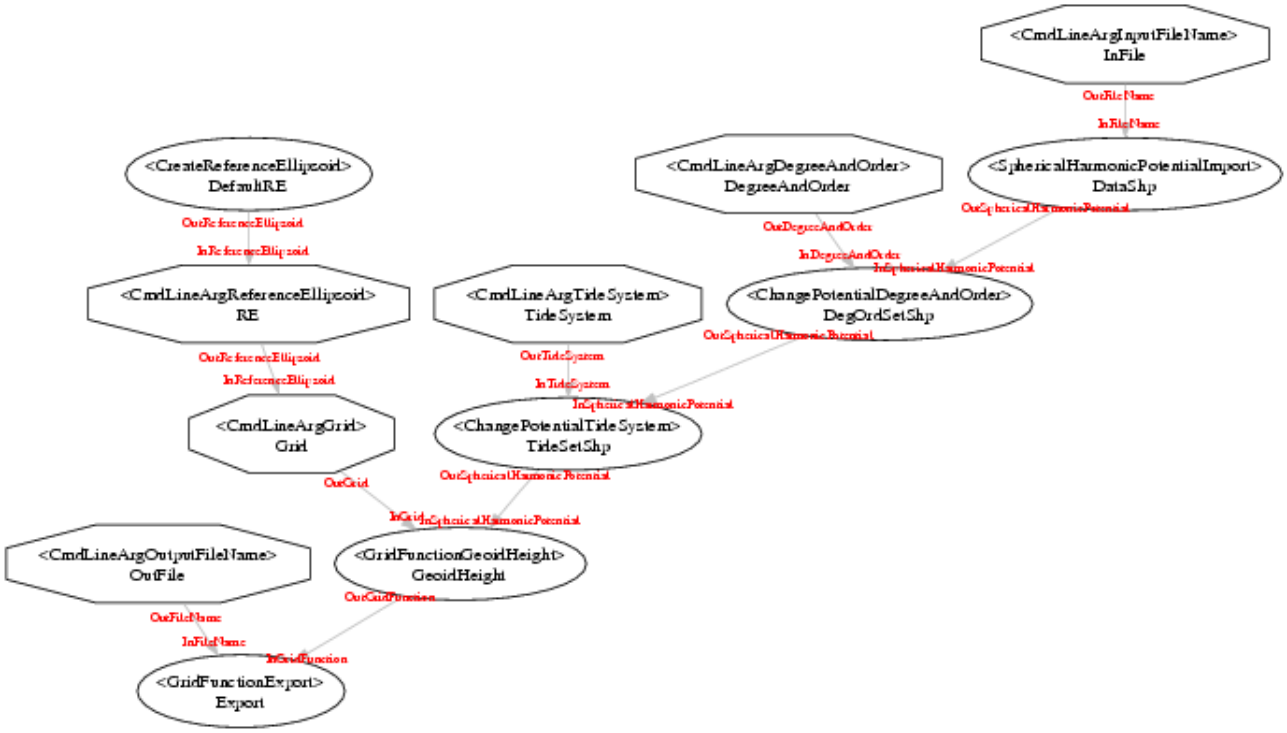

Figure 1: Structure of the geoidheight_gf work-flow

3.2.5 Work-flow definition file content

Each work-flow is defined in an XML file and each file is organized in 3 sections. The first section lists and provides unique identifiers for all of the required processing units. All XML elements in this section are the name of the processing unit. The second section lists all required connections. Each connection is made from a socket and a plug. Each socket (output) defines an unit and an output port for that unit type. Each plug defines an unit and an input port for that unit type. The available processing units, their function and ports are given in §8. The last section is optional (recommended) and provides a short overview of the work-flow. This information is specified as plain text and is the information displayed by the man run-mode. An example work-flow file is shown in Appendix-A.

3.3 Environment Variables

There are a small number of environment variables that you may set in order to customize GUT for your system. These variables, and their function, are listed below.

- **GUT_TMPDIR** : This variable, if set, specifies the directory (folder) in which temporary files will be created if needed by gut when processing. If this variable is not set then the system temporary directory (variable TMP or TEMP or TMPDIR) or the current directory is used. In general all temporary files are removed before gut exits, but terminating the program with a signal (ie. Ctrl-C) may result in temporary files residing on the system.
- **GUT_MEMSIZE_KB** : This variable, if set, should be an integer (in kBytes)

specifying the maximum size of a gridded data file that gut will attempt to manage in system memory. For data size above this limit gut will create a temporary file for storage and swap data between memory and file as needed. This allows gut to process files larger than physical memory but it incurs a performance penalty. The default maximum size is approximately 65536 kB (64Mb). You should set this to a smaller value if your system has limited memory and gut fails to allocate memory for large gridded data sets. You should set the variable to a larger value if your system has adequate memory and gut is found to be swapping to/from disk when processing. *(NOTE: the default limit is approximately equivalent to an equiangular 6'x6' global grid)*.

3.4 Developing new Processing Units

The processing capabilities of GUT are encapsulated in the suite of processing units that are an integral part of the command-line tool. Each of these processing units is defined in a single C++ class and users can extend the GUT feature set by developing software implementations for new processing units. Further discussion of this topic is deferred to §10.

4 Spherical Harmonics

A considerable fraction of the GUT data types and algorithms are described and implemented in terms of arithmetic series of spherical harmonic basis functions. The foundation of spherical harmonic functions are the Associated Legendre Functions, and these functions are well described in many texts. In the field of Geodesy a particular normalization scheme is adopted for these functions and the resulting spherical harmonic basis functions. This scheme is also well described in many texts and the GUT software simply follows this defacto-standard. A brief summary of the relevant equations is provided here merely for consistency.

4.1 General Spherical Harmonics Basis Functions

The spherical harmonic function of degree n, order m, is given by

$$Y_n^m(\theta,\lambda) = N_{nm}\, e^{im\lambda}\, P_n^m(\cos\theta)$$

where $P_n^m(x)$ is an Associated Legendre Function, and N a (complex) normalization coefficient. Note, the notation $P_{nm}(x)$ is also used for these functions, often to avoid confusing the order, m, for a power. In this document both notations are used.

When spherical harmonics are considered as solutions to Laplace's equation and boundary conditions are considered, only integral values of *n* and *m* are relevant, with the additional constraint that $|m| < n$.

The spherical harmonic functions are orthogonal and therefore form a set of basis functions on the sphere. Consequently, an arbitrary function on the sphere can be represented as a linear combination of these basis functions.

$$f(\theta,\lambda)=\sum_{n=0}^{N_{max}}\sum_{m=-n}^{n} N_{nm}\, Y_n^m(\theta,\lambda)$$

The identity,

$$P_n^{-m}(x) \;=\; (-1)^m \frac{(n-m)!}{(n+m)!}\, P_n^m(x)$$

allows the terms $Y_n^m(\theta,\lambda)$ and $Y_n^{-m}(\theta,\lambda)$ to be summed together and then a ℜeal function can be expressed in terms of weighted sine and cosine functions with ℜeal coefficients and a single *Associated Legendre Function*. This forms a new basis set of functions such that the expansion of function $f(\theta,\lambda)$ is now

$$f(\theta,\lambda) \;=\; \sum_{n=0}^{N_{max}}\sum_{m=0}^{n} [C_{nm}\cos(m\lambda)+S_{nm}\sin(m\lambda)]\, P_n^m(\cos\theta)$$

In many disciplines the equation above is normalized. This is a simple transformation to a different set of basis functions, each of which is a scalar multiple of the functions above. Different normalization schemes are applied in different disciplines, based largely on the nature of the functions that are studied. In geodesy the normalization scheme is a simple transformation based on the degree and order. In many texts the normalized coefficients and functions are represented with an over-bar. In this text however, the non-normalized basis functions are only relevant in this summary section, as all algorithms in GUT are implemented in terms of normalized functions. For this reason the *over-bars are shown in this section only and omitted in all other sections*.

The relationship between normalized and non-normalized functions are

$$\begin{aligned} \bar{C}_{n0} &= \frac{1}{\sqrt{2n+1}}\, C_{n0} \\ \bar{S}_{n0} &= 0 \\ \bar{P}_n^0(\cos\theta) &= \sqrt{2n+1}\, P_n^0(\cos\theta) \end{aligned}$$

$$\begin{aligned}
\bar{C}_{nm} &= \sqrt{\frac{(n+m)!}{2(2n+1)(n-m)!}}\, C_{nm} \\
\bar{S}_{nm} &= \sqrt{\frac{(n+m)!}{2(2n+1)(n-m)!}}\, S_{nm} \\
\bar{P}_n^m(\cos\theta) &= \sqrt{\frac{2(2n+1)(n-m)!}{(n+m)!}}\, P_n^m(\cos\theta)
\end{aligned}$$

4.2 Recurrence Relations

A suite of recurrence relations and conditions hold for the Associated Legendre Functions. The following relations are expressed in terms of non-normalized Legendre Functions.

$$\begin{aligned}
(n-m+1)\, P_{n+1}^m(x) &= (2n+1)\, x\, P_n^m(x) - (n+m)\, P_{n-1}^m(x) \\
P_{n+1}^m(x) &= P_{n-1}^m(x) - (2n+1)\sqrt{1-x^2}\, P_n^{m-1}(x) \\
P_n^m(x) &= 0 \qquad \forall \quad |m|>n
\end{aligned}$$

Using these relations, transforming to normalized Associated Legendre Functions and observing that $(1 - x^2)^{1/2} = \sin(\theta)$ when $x = \cos(\theta)$, yields recurrence relations for the normalized Associated Legendre Functions.

$$\bar{P}_n^n(\cos\theta) = \begin{cases} 1 & \text{if } n=0 \\ \sqrt{3}\sin(\theta) & \text{if } n=1 \\ \sqrt{\dfrac{2n+1}{2n}}\sin(\theta)\,\bar{P}_{n-1}^{n-1}(\cos\theta) & \text{otherwise} \end{cases}$$

$$\bar{P}_n^m(\cos\theta) = \begin{cases} \sqrt{\dfrac{(2n+1)(2n-1)}{(n+m)(n-m)}}\cos(\theta)\,\bar{P}_{n-1}^m(\cos\theta) & \text{if } n=m+1 \\ \sqrt{\dfrac{(2n+1)(2n-1)}{(n+m)(n-m)}}\cos(\theta)\,\bar{P}_{n-1}^m(\cos\theta) - & \\ \quad \sqrt{\dfrac{(2n+1)(n+m-1)(n-m-1)}{(2n-3)(n+m)(n-m)}}\cos(\theta)\,\bar{P}_{n-2}^m(\cos\theta) & \text{if } n > m+1 \end{cases}$$

The normalization scheme is such that the following conditions are satisfied.

$$\sum_{m=0}^{n} [\bar{P}_n^m(\cos\theta)]^2 = 2n+1$$

$$\oiint [\bar{P}_n^m(\cos\theta)\,\cos(m\lambda)]^2\, d\sigma = 4\pi$$
$$\oiint [\bar{P}_n^m(\cos\theta)\,\sin(m\lambda)]^2\, d\sigma = 4\pi$$

The derivative of the normalized Legendre Functions can also be expressed via recurrence relations.

$$2\,\frac{\partial \bar{P}_n^m(\cos\theta)}{\partial\theta} = \begin{cases} -\sqrt{2n(n+1)}\;\bar{P}_n^1(\cos\theta) & \text{if m=0} \\ \sqrt{2n(n+1)}\;\bar{P}_n^0(\cos\theta) - \sqrt{(n-1)(n+2)}\;\bar{P}_n^2(\cos\theta) & \text{if m=1} \\ \sqrt{(n+m)(n-m+1)}\;\bar{P}_n^{m-1}(\cos\theta) - \sqrt{(n-m)(n+m+1)}\;\bar{P}_n^{m+1}(\cos\theta) & \text{if m>1} \end{cases}$$

5 Fundamental Data Types

In general, the algorithms that form the building blocks of work-flow processing chains work with high-level data types (objects) as input and output. These data types encapsulate a conceptual unit into a single data object. A reference ellipsoid is one example of a high-level data type. As a conceptual unit it defines a reference surface and a framework for interpreting coordinates. At a lower level, the reference ellipsoid is defined by a few numeric values. Although the low-level details are critical for the implementation of algorithms, the high-level data-object is sufficient to define the interface for the algorithms. The consequence of designing and defining algorithms in terms of high level data-objects yields smaller and simpler interfaces. To support the required algorithms, GUT defines a small set of high-level data types. There is considerable cooperation between these types, and in some cases, one data type incorporates one or more other data types. Each data type is described in the following sections.

The high-level data types applicable to GUT often imply large quantities of data. Moreover, the flexible nature of the processing chain construction makes it impossible for an algorithm designer to preempt the usage of the algorithm. Both of these issues require the data types to provide implicit system resource management for efficient processing.

The fundamental data types implemented for GUT provide this implicit resource management. As a consequence, large quantities of data can be passed as input to an algorithm (within the guise of a single high level data object) without degrading performance or consuming excessive system resources.

5.1 Built-In Data Types

GUT uses a subset of the low-level built-in data types that are provided as part of all modern programming languages in addition to the specialized data types that have been

designed specifically for GUT. These are integers, real numbers and character strings. In the integer category GUT also uses some enumeration types. These are simply a restricted set of integers and they are usually assigned a symbolic name. Within GUT all computation based on real numbers is implemented with double-precision.

Within this document the built-in data types are referred by the names “Int”, “Double”, “String” and “Enumeration”.

5.2 Reference Ellipsoid

The Reference Ellipsoid is the foundation for coordinate systems and transformation between them. This type defines a geometrical ellipsoid surface, and datum, forming the basis of a geodetic coordinate system. Further, it defines a rotation rate and an earth-gravity constant. With the latter two pieces of information, the Reference Ellipsoid provides a model for the gravity and gravitational potential of the Earth, valid for the region outside the ellipsoid.

5.3 Grid

The Grid provides the angular geodetic coordinates for a rectangular array that bounds a contiguous region in latitude-longitude space. All coordinates defined by the grid must be unique and the grid spacing may be regular or irregular. Essentially, the Grid is comprised of a series of latitude ordinates (latitude axis), a series of longitude ordinates (longitude axis) and a Reference Ellipsoid that provides the reference system for these axes. The data in the axes are sorted in strictly increasing order (ie. latitude runs South-to-North and longitude runs West-to-East).

5.4 Grid Function

The Grid Function provides a physical quantity at a location associated with a Grid. The angular coordinates (lat,lon) are directly coupled to the associated grid, but the data is more loosely associated, dependent on the physical quantity represented. For example, a Grid Function that represents an ellipsoid height provides an implicit spatial location (lat,lon,height). Other quantities represent height differences with respect to a surface. This surface may be the surface of the reference ellipsoid, the surface of the solid-Earth, or a geophysical surface (geoid, telluroid, etc).

The Grid Function supports description of a physical quantity by Point or by Area. In the Point scheme, the data represents the value of the physical quantity at the location defined by the corresponding Grid. In the Area scheme, the data represents the 'average' value of the physical quantity in the region defined by the grid 'cell'. In this case, the associated Grid provides the corners for the grid cells.

The Grid Function has the potential to represent very large quantities of data, but very few algorithms ever require ALL data at once. For this reason, the Grid Function provides row, column and cell based data read and write (it does NOT provide a 2-D array interface to the data).

The Grid Function supports the concept of invalid data. This is only applicable to the physical quantity, not the coordinates in the associated Grid. The manner in which this invalid data is handled is algorithm specific.

The Grid Function has an associated Tide-System, though this is not always relevant to all physical quantities. Data that are 'tidal' height fields can be transformed between the Tide-Systems supported by GUT. It is worth mentioning that many data sources do not directly indicate the tide system corresponding to the data, and this makes reliable tracking of this information very difficult. *(Converting to the GUT internal data format is generally recommended as this will allow initial specification of the tide system and consideration of this information by GUT algorithms).*

5.5 Spherical Harmonic Potential

The Spherical Harmonic Potential provides a model for the gravitational potential of the solid-Earth in terms of a band-limited set of normalized coefficients for a spherical harmonic series. The Spherical Harmonic Potential provides interfaces for evaluating the gravity, gravity potential and gravitational potential models. The model for the gravitational potential is given below.

$$V(r,\theta,\lambda)=\left(\frac{GM}{r}\right)\sum_{n=0}^{N_{max}}\left(\frac{a}{r}\right)^{n}\sum_{m=0}^{n}\left[C_{nm}\cos(m\lambda)+S_{nm}\sin(m\lambda)\right]P_{nm}(\cos\theta)$$

The constants GM and a are the Earth Mass-Gravitational Constant and an arbitrary scale length (typically the equatorial radius of the Earth). The coefficients, C_{nm} and S_{nm}, are fully normalized (in the convention used in Geodesy), and P_{nm} are the normalized Associated Legendre functions.

The Solid-Earth tide system adopted is also part of the model. Where gravity, rather than gravitational models are considered, the centrifugal potential, Z, is added. This is given by

$$Z(r,\theta,\lambda)=\frac{\omega^2 r^2\sin^2(\theta)}{2}$$

where ω is the rotation rate of the Earth *(NOTE: the rotation rate of the Earth has been measured to sufficiently high precision that it is treated as a constant, rather than a parameter, in the models implemented in GUT).*

Models for gravity vectors are based on derivatives of V and Z.

5.6 Spherical Harmonic Function

The Spherical Harmonic Function provides a representation of a global function in terms of a band-limited set of normalized coefficients for a spherical harmonic series. This is the spectral-space equivalent of the Grid Function. Like the Grid Function, the Spherical Harmonic Function represents a physical quantity (and tide system) and is defined in association with a reference ellipsoid. The general equation for the spherical harmonic series is

$$f(\theta,\lambda)=\sum_{n=0}^{N_{max}}\sum_{m=0}^{n}[A_{nm}\cos(m\lambda)+B_{nm}\sin(m\lambda)]P_{nm}(\cos\theta)$$

Like the Spherical Harmonic Potential, the fully normalized associated Legendre functions are used in this series. The maximum Degree-and-Order, N_{max}, defines the high-frequency band-limit. The number of coefficients associated with a given N_{max} is $(N_{max}+1)(N_{max}+2)$, though all B_{n0} coefficients are implicitly zero.

5.7 Track

The Track provides the angular geodetic coordinates for a sequence of locations. There are no restrictions on the coordinate of a track. A Reference Ellipsoid provides the reference system for these coordinates.

5.8 Track Function

The Track Function provides a physical quantity (and tide system) at a location associated with a Track. The interpretation of the Track Function data is identical to the data in a Grid Function with the Point scheme; there is no equivalent of the Area scheme.

5.9 Spatial Filter Kernel

The Spatial Filter Kernel provides a convolution kernel with cylindrical symmetry that can be convolved with a Grid Function. A limited number of filter kernel shapes have been implemented. This data type is specifically intended for filtering in the spatial domain.

5.10 Spectral Filter Kernel

The Spectral Filter Kernel provides a convolution kernel with cylindrical symmetry that can modulate a Spherical Harmonic Function. A (more) limited number of filter kernel shapes have been implemented. This data type is specifically intended for filtering in the spectral domain.

6 File Formats

In order to provide efficient import and export of data without loss of either the reference system or meta-data, a set of GUT Internal File Formats were required. These formats provide permanent disk-storage of precisely one high-level data-object. The basis of all GUT internal file formats is the 32-bit netCDF format. This provides efficient, cross-platform binary storage. Where applicable, the existing CF-Convention has been adopted for variable and attribute names. This enables the GUT Internal format to be directly imported and interpreted by existing software.

Import and export in GRAVSOFT grid and point-list format is also supported. However, the lack of meta-data makes the GRAVSOFT format 'lossy'. It is therefore strongly

recommended that GRAVSOFT files only be generated via a conversion from a GUT internal file format.

In addition to the GRAVSOFT and GUT internal (netCDF) formats, a limited number of other file formats can be read (but NOT written). The most important of these is the GOCE Level-2 header (HDR) and data (DBL) file pair. In order to extract a complete set of meta data for a data set, both the HDR and DBL files must be present in the same directory. Since these two files are implicitly treated as a unit, specification of either the HDR or DBL file is sufficient. However, this requires that GOCE Level-2 data files have the appropriate '.HDR' and '.DBL' extensions (with uppercase characters).

6.1 GUT Internal File Formats

The GUT Internal File Formats (netCDF) are based on the CF-Convention and some of the GUT fundamental data types. As described in the previous section, some data types implicitly incorporate another data type and some data types are incorporated into many data types (ie. A Grid Function incorporates a Grid, which incorporates a Reference Ellipsoid. Tracks and Spherical Harmonic Functions also incorporate a Reference Ellipsoid). The file formats build the more complex data types by extension. This means that, for example, a Reference Ellipsoid is described the same way in all file formats that incorporate a Reference Ellipsoid. Because of this it suffices to describe only the more complex file formats in detail.

Internally, GUT works with a specific set of physical quantities and each quantity has a specific unit. GUT provides limited support for scaling data in different unit systems to the internal unit, provided that the source unit is known a-priori or can be determined from the file. When a unit converting scale factor cannot be determined, scaling of the data on import is not performed. Data exported by GUT are always in the internal unit system (in general units are *SI*, though gravity anomalies are an exception, the units in this case being mgal).

6.1.1 netCDF Grid Function

The GUT internal file format for a Grid Function defines a storage scheme for the following data sets.

- Reference Ellipsoid
  - Semi-major axis, a (m)
  - Inverse flattening, f (dimensionless)
  - Mass-Gravity constant, GM ($m^{-1}.s^{-2}$)
  - Rotation rate, ω ($rad.s^{-1}$)
- Latitude Axis
  - Array of N_{LAT} geodetic latitudes in strictly increasing order, lat[i] (degrees).
- Longitude Axis
  - Array of N_{LON} geodetic longitudes in strictly increasing order, lon[j] (degrees).
- Cell Data
  - 2D Array with N_{LAT} rows and N_{LON} columns.
  - Tide-System

All data are stored in IEEE 8-byte floating point format (double precision). Cell data may be point or area values. If point values are stored then the data in cell (i,j) corresponds to geolocation (φ,λ) = (lat[i],lon[j]). If area values are stored then lat[i] and lon[j] correspond to the center of a region and additional arrays are stored to define the geographic boundaries of the cells.

In netCDF terms, the file has two dimensions for point values and three dimensions for area values. The dimensions *lat* and *lon* are always defined and they specify N_{LAT} and N_{LON} respectively. The dimension *nv*, with value 2, is defined for area values. The latitude and longitude axis arrays are stored as the coordinate variables, *lat* and *lon*, following the CF-Convention. The attributes *long_name*, *standard_name* and *units* are specified for these variables. For area values the additional attribute *bounds* is written. This references a 2D array variable that stores the corresponding coordinates of the cell boundaries. These bounds variables are called *lat_bnd* and *lon_bnd* for latitude and longitude respectively. The cell data are stored in a two dimensional variable with the name of the physical quantity of the Grid Function. The attributes *units* and *grid_mapping* are specified for this variable. The *tide_system* attribute is also specified if known and appropriate for the physical quantity. This attribute can have the value *tide-free*, *mean-tide* or *zero-tide*. The *grid_mapping* attribute has the value *crs* and is a reference to another variable. The *crs* variable has zero dimension and all information is stored as attributes. This fully describes the Reference Ellipsoid via the attributes *semi_major_axis*, i*nverse_flattening*, *earth_gravity_constant* and *earth_rotation_rate*. (Additional attributes may be added for non-geocentered and non-rotation axis aligned coordinate systems in the future). The global attribute *Conventions* is also specified and indicates the CF-Convention has been adopted. It is worth mentioning that the variable containing the data is expected to be the first non-coordinate, non-bound, non-crs variable in the file and that GUT only stores one set of cell data in a file. However, any netCDF file that follows the conventions described can be read by GUT.

6.1.2 netCDF Track Function

The GUT internal file format for a Track Function defines a storage scheme for the following data sets.

- Reference Ellipsoid
  - Semi-major axis, a (m)
  - Inverse flattening, f (dimensionless)
  - Mass-Gravity constant, GM ($m^{-1}.s^{-2}$)
  - Rotation rate, ω (rad.s^{-1})
- Station coordinates
  - Array of $N_{STATION}$ geodetic latitudes in station order, lat[i] (degrees).
  - Array of $N_{STATION}$ geodetic longitudes in station order, lon[i] (degrees).
- Station Data
  - Array with $N_{STATION}$ values.
  - Tide-System

All data are stored in IEEE 8-byte floating point format (double precision). The station data, physical_quantity[i], is the point-value at the geolocation (φ,λ) = (lat[i],lon[i]) of the

station.

In netCDF terms, the file has one dimension, *station*. The latitude and longitude coordinates are stored in the variables, *lat* and *lon*, following the CF-Convention. The attributes *long_name*, *standard_name* and *units* are specified for these variables. The station data are stored in a one dimensional variable with the name of the physical quantity of the Track Function. The attributes *units* and *grid_mapping* are specified for this variable. The *tide_system* attribute is also specified if known and appropriate for the physical quantity. This attribute can have the value *tide-free*, *mean-tide* or *zero-tide*. The *grid_mapping* attribute has the value *crs* and is a reference to another variable. The *crs* variable has zero dimension and all information is stored as attributes. This fully describes the Reference Ellipsoid via the attributes *semi_major_axis*, i*nverse_flattening*, *earth_gravity_constant* and *earth_rotation_rate*. (Additional attributes may be added for non-geocentered and non-rotation axis aligned coordinate systems in the future). The global attribute *Conventions* is also specified and indicates the CF-Convention has been adopted. It is worth mentioning that the variable containing the data is expected to be the first non-coordinate, non-crs variable in the file and that GUT only stores one set of station data in a file. However, any netCDF file that follows the conventions described can be read by GUT.

6.1.3 netCDF Spherical Harmonic Function

The GUT internal file format for a Spherical Harmonic Function defines a storage scheme for the following data sets.

- Reference Ellipsoid
  - Semi-major axis, a (m)
  - Inverse flattening, f (dimensionless)
  - Mass-Gravity constant, GM ($m^{-1}.s^{-2}$)
  - Rotation rate, ω (rad.s^{-1})
- Fully normalized Spherical Harmonic Coefficients
  - 2D Array of C_{nm} and S_{nm} coefficients in a triangular packing order.
  - Tide-System

All data are stored in IEEE 8-byte floating point format (double precision). All coefficients from $C/S_{0,0}$ to $C/S_{Nmax,Nmax}$ are stored (even though some are 0 by definition).

In netCDF terms, the file has two dimension, *nm* and *coef*. The value for *coef* is always 2 while the value for *nm* is dependent on the maximum degree and order of the spherical harmonic series (N_{max}). The relation between N_{max} and the value of the *nm* dimension is

$$nm = \frac{(N_{max}+1)(N_{max}+2)}{2}$$

which is guaranteed to be an integer. Solving this expression for Nmax is trivial, but in order to account for limited precision, it is recommended that the following computation be used in software

$$N_{max} = (round)(0.5\sqrt{8nm+1} - 1)$$

The coefficients are stored in a 2D array variable with the name of the physical quantity and dimensions *nm* and *coef*. The indexing in the nm dimension follows a triangular packing scheme with consecutive orders, *m*, of the same degree, *n*, being sequential. The C_{nm} and S_{nm} values are stored at index 0 and 1 in the *coef* dimension. The array indexes for C_{nm} and S_{nm} are therefore (n(n+1)/2+m, 0) and (n(n+1)/2+m, 1) respectively. Of course the range for m is restricted to [0,n].

The attributes *units* and *grid_mapping* are specified for the coefficients variable. The *tide_system* attribute is also specified if known and appropriate for the physical quantity. This attribute can have the value *tide-free*, *mean-tide* or *zero-tide*. The *grid_mapping* attribute has the value *crs* and is a reference to another variable. The *crs* variable has zero dimensions and all information is stored as attributes. This fully describes the Reference Ellipsoid via the attributes *semi_major_axis*, *inverse_flattening*, *earth_gravity_constant* and *earth_rotation_rate*. (Additional attributes may be added for non-geocentered and non-rotation axis aligned coordinate systems in the future). The global attribute *Conventions* is also specified and indicates the CF-Convention has been adopted. It is worth mentioning that the variable containing the data is expected to be the first non-coordinate, non-crs variable in the file and that GUT only stores one set of coefficient data in a file.

6.1.4 netCDF Spherical Harmonic Potential

The GUT internal file format for a Spherical Harmonic Potential defines a storage scheme for the following data sets.

- Reference System
  - Semi-major axis, a (m)
  - Mass-Gravity constant, GM ($m^{-1}.s^{-2}$)
  - Tide System
- Fully normalized Spherical Harmonic Coefficients
  - 2D Array of C_{nm} and S_{nm} coefficients in a triangular packing order.

The structure of this file is closely related to the structure of the Spherical Harmonic Function, differing only in the storage of the reference data. The dimension and naming scheme is identical, but all of the reference information (a, GM, tide-system) are stored as attributes of the crs variable.

6.1.5 netCDF Track, Grid and Reference Ellipsoid formats

The GUT internal file formats for storage of Tracks, Grids and Reference Ellipsoids follow directly from their 'superset' storage formats.

6.2 Support for GRAVSOFT File Formats

This section presents how GUT attempts to interpret the data in GRAVSOFT (text) files. It

does not attempt to describe the GRAVSOFT format specification.

GUT supports reading and writing of the ASCII GRAVSOFT grid format and reading of the ASCII GRAVSOFT point-list format. Since these formats DO NOT indicate either the reference system or the nature of the data contained, care should be taken when using them. When reading it is assumed that the Reference Ellipsoid is GRS80 and the tide system is invalid. On writing the actual reference system is not taken into account. It is generally recommended that GRAVSOFT formats are first converted to the corresponding GUT internal netCDF formats (See the GUT Tutorial for details).

The point-list file format is difficult to reliably interpret due to the multi-column nature of the specification. When a height-field quantity is required, it is read from the height column of the point-list file. For all other quantities the data is read from the column following the height data. The units of the data are assumed to be those used by GUT when exporting data to file.

6.3 GOCE Level-2 Product Files

The GOCE Level-2 product file is an XML formatted file that contains multiple data sets. GUT can extract from this file a Spherical Harmonic Potential or a Grid Function. The possible physical quantities for the Grid Functions are geoid_height, gravity_anomaly, geoid_height_error, vertical_deflection_north and vertical_deflection_east. See the GOCE Level-2 product Specification for further details.

7 Algorithms

From a high-level, GUT is a conglomerate of algorithms that can be linked together to form a single processing operation. The base algorithms vary substantially in their complexity. This section presents algorithmic detail at a level sufficient for the GUT user to fully understand the result of an algorithm. Details relevant to the efficiency of the computation are not discussed unless they have a specific bearing on the interface presented to the user. Many of the building-block algorithms (processing units) available to the GUT user are trivial, and as such, they are briefly described in the GUT User Manual. In this section the focus is on the core algorithms.

7.1 Coordinate Systems and Transformations

There are three coordinate systems used within GUT; geodetic, geocentric and cartesian. Throughout this document the following symbols are used for coordinates.

- cartesian (x,y,z)
- geocentric (r,θ,λ)
- geodetic (h,φ,λ)

The cartesian reference frame is Right-Handed, with the origin at the center of the earth,

on the axis of rotation. The Z-axis is parallel with the axis of rotation and oriented 'positive northward'. The X-axis intersects the prime meridian at the equator. The Y-Axis is orthogonal to the X and Z axes.

The geocentric reference frame is based on the cartesian reference frame. The geocentric coordinates, (radius, colatitude, longitude) are given by :

$$r = \sqrt{x^2 + y^2 + z^2}$$

$$\theta = \arccos\left(\frac{z}{r}\right)$$

$$\lambda = \arctan\left(\frac{y}{x}\right)$$

The geodetic coordinates are coupled to the reference ellipsoid. Geodetic coordinates are transformed to cartesian coordinates by first computing the radius of curvature in the prime vertical, *N*.

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2(\varphi)}}$$

$$x = (N + h)\cos(\varphi)\cos(\lambda)$$

$$y = (N + h)\cos(\varphi)\sin(\lambda)$$

$$z = ((1 - e^2)N + h)\sin(\varphi)$$

where a and e are the semi-major axis and the eccentricity of the ellipsoid respectively.

The transformation from cartesian coordinate to geodetic coordinates involves an iterative method based on the previous equations. The iteration is terminated when both the latitude and the height have converged (with tolerances of 1e-10 radians and 10 microns) or an iteration count is exceeded (20).

7.2 Grid Function Adaption

This algorithm generates a Grid Function on a specified Grid from an existing Grid Function. The basis of this algorithm is bilinear interpolation, with height corrections

applied to account for a change in Reference Ellipsoid when appropriate.

The inputs to this algorithm are :

- A source Grid Function
- A destination Grid

The output from this algorithm is :

- A destination Grid Function

The first part of the algorithm compares the Reference Ellipsoid associated with the source and destination, and the physical quantity that is represented by the source. If the physical quantity is considered independent of the reference ellipsoid, or the source and destination Reference Ellipsoids are the same, then the second phase of the algorithm operates directly on the source Grid Function. If this condition is not met, then the algorithm can *ONLY* proceed if the physical quantity is a height field with respect to the ellipsoid. If the source and destination Reference Ellipsoids differ and the physical quantity is a height field, then the destination Grid is transformed to the source Reference Ellipsoid. This transformation computes the geodetic coordinates (h,φ,λ) of the destination Grid on the surface of the source Reference Ellipsoid. The resulting angular coordinates provide the place in the source Grid Function where the physical quantity must be calculated by interpolation. The height coordinate provides a correction to the result of the interpolation. This slight variation in the direction of the normals on the source and destination reference ellipsoids is ignored.

The bilinear interpolation process for coordinate (φ,λ) attempts to locate the 'cell' in the source Grid that contains this target location. This 'cell' is formed by 4 points in the source Grid. This 'cell' may be formed by the last and first longitudes in the Grid if λ is outside the formal longitude boundaries of the Grid. This wrapping implicitly treats the longitude range of any Grid as 'global'. If the target φ is outside the latitude range of the source Grid, then the 'cell' is formed by the two northern-most (or southern most) latitudes, and the interpolation is actually extrapolation. This extrapolation is limited to target latitudes that extend less than ¾ of the latitude-size of the 'cell'. Beyond this range the values produced by the algorithm are formally invalid. The latitude extrapolation limit is large enough to allow a global cell-centered point Grid to be adapted to include the poles. However, the result of this extrapolation *DOES NOT* ensure that all values for a polar location are the same. The bilinear interpolation operation is decomposed into three linear interpolations; two for determining the values along parallel edges of a cell, and the final interpolation for the value at the target location based on the result of the first two operations. The final result is independent of the selection of latitude-first or longitude first interpolation. In practice both schemes are used, based on the layout of the source data, to improve the efficiency of the calculations. The general linear interpolation equation applied is

$$f(x)=\frac{f(x_a)(x_b-x)+f(x_b)(x-x_a)}{(x_b-x_a)}$$

where x_a and x_b are the coordinates of the cell boundaries, and f is the value of the function defined at those locations.

If the value of the source Grid Function is invalid at any of the four points defining the cell then the result of the interpolation is also invalid.

7.3 Geoid Height

The Geoid Height is calculated at a specific point on the surface of a Reference Ellipsoid from a Spherical Harmonic Potential. The core of this algorithm is applied to both Grid Function and Track Function output.

The inputs to this algorithm are :

- A source Spherical Harmonic Potential
- A destination Grid or Track

The output from this algorithm is :

- A destination Grid Function or Track Function

The disturbing gravitational potential, T, is first expressed as a Spherical Harmonic Potential by subtracting the Spherical Harmonic Potential of the Reference Ellipsoid associated with the destination (Grid or Track). The Spherical Harmonic Potential of the Reference Ellipsoid is expressed as a series with only the normalized coefficients C_{00}, C_{20}, C_{40}, C_{60} and C_{80} being non-trivial. These coefficients are calculated from the following expressions:

$$q_o=(\frac{1}{2})[(1+3(\frac{b}{ae})^2)\arctan(\frac{ae}{b})-3(\frac{b}{ae})]$$

$$K=(\frac{1}{3})[1-\frac{2\omega^2a^3e}{15q_oGM}]$$

$$J_{2n}=(-1)^{(n+1)}\frac{3e^{2n}}{(2n+3)(2n+1)}[1+n(5K-1)]$$

$$C_{2n0}=\frac{-J_{2n}}{\sqrt{4n+1}}$$

where a,b,e are the semimajor axis, semiminor axis and eccentricity of the ellipsoid respectively, ω is the earth rotation rate and GM is the combined Earth-Mass Gravitational constant.

The gravitational potential of the Reference Ellipsoid is transformed to the reference system of the source Spherical Harmonic Potential. This involves scaling the coefficients by a factor that is based on the current and required values of the Earth-Mass Gravity constants, GM^C, GM^R, and the reference scale length, a^C, a^R. The relationship between the current coefficient, C^C_{nm}, and the required coefficient, C^R_{nm}, is given by

$$C^R_{nm}=\left(\frac{GM^C}{GM^R}\right)\left(\frac{a^C}{a^R}\right)^n C^C_{nm}$$

Having transformed the coefficients of the Reference Ellipsoid's gravitational potential model, the disturbing potential is calculated by direct subtraction in a coefficient-wise manner.

At each point of the Grid, or Track, the geoid height, N, is calculated via the Bruns equation.

$$N=\frac{T}{\gamma_o}$$

where γ_o is the normal gravity at the surface of the Reference Ellipsoid. The disturbing potential is evaluated by summation of all terms in the series of the Spherical Harmonic Potential:

$$T(r,\theta,\lambda)=\left(\frac{GM}{r}\right)\sum_{n=0}^{N_{max}}\left(\frac{a}{r}\right)^n\sum_{m=0}^{n}\left[\Delta C_{nm}\cos(m\lambda)+\Delta S_{nm}\sin(m\lambda)\right]P_n^m(\cos\theta)$$

The normal gravity at the surface of the ellipsoid is calculated via

$$\gamma_o=\gamma_a\frac{1+\left(\frac{b\gamma_b}{a\gamma_a}-1\right)\sin^2(\varphi)}{\sqrt{1-e^2\sin^2(\varphi)}}$$

where γ_a and γ_b are the normal gravity at the equator and pole(s) respectively. They are calculated by evaluating the derivative with respect to radius of the Gravity Potential of the Reference Ellipsoid at the appropriate latitudes (0 and 90°). The expression for the Gravity potential of the Reference Ellipsoid is

$$U(r,\theta,\lambda)=[\frac{GM}{r}(1-\sum_{n=1}^{4}(\frac{a}{r})^{2n}J_{2n}P_{2n}(\cos\theta))]+\frac{\omega^2 r^2\sin^2(\theta)}{2}$$

where $P_{2n}(\cos\theta)$ is the Legendre Polynomial of degree $2n$ (and is the Associated Legendre Function P_{2n}^{0}). The derivative is straight-forward and the Legendre Polynomials for $\cos\theta$ equal to 0 and 1 are evaluated via a standard recurrence relation.

The resultant Geoid Height, N, is expressed in units of meters (m).

7.4 Gravity Anomaly

The Gravity Anomaly is calculated as the difference between the magitude of the gravity vector at a point P (on the surface of the Earth), and the magnitude of the reference gravity at the point Q which has the same Gravity Potential as at the point P and lies on the normal to the reference ellipsoid passing through P.

The inputs to this algorithm are :

- A source Spherical Harmonic Potential
- An altitude Grid Function or Track Function

The output from this algorithm is :

- A destination Grid Function or Track Function

The locations of all points, P, where the gravity and gravitational potential are evaluated are defined by the altitude function. This also defines the Grid or Track associated with the output. The geodetic coordinates of the point P are transformed to geocentric coordinates, and the gravity potential, W_P, and gravity, $g_P = \nabla W_P$, are evaluated from the Spherical Harmonic Potential. The gravity potential is given by

$$W(r,\theta,\lambda)=(\frac{GM}{r})\sum_{n=0}^{N_{max}}(\frac{a}{r})^{n}\sum_{m=0}^{n}[C_{nm}\cos(m\lambda)+S_{nm}\sin(m\lambda)]P_n^m(\cos\theta)\ +\ \frac{\omega^2 r^2\sin^2(\theta)}{2}$$

The components of the gradient of this potential, expressed in a local north-oriented right-handed Cartesian reference frame are

$$W_X = \frac{-1}{r}\frac{\partial W}{\partial \theta} = \left(\frac{-GM}{r^2}\right)\sum_{n=0}^{N_{max}}\left(\frac{a}{r}\right)^n \sum_{m=0}^{n}\left[C_{nm}\cos(m\lambda)+S_{nm}\sin(m\lambda)\right]\frac{\partial P_n^m(\cos\theta)}{\partial \theta} - \omega^2 r\sin(\theta)\cos(\theta)$$

$$W_Y = \frac{1}{r\sin(\theta)}\frac{\partial W}{\partial \lambda} = \frac{1}{\sin(\theta)}\left(\frac{-GM}{r^2}\right)\sum_{n=0}^{N_{max}}\left(\frac{a}{r}\right)^n \sum_{m=0}^{n} m\left[C_{nm}\sin(m\lambda)-S_{nm}\cos(m\lambda)\right]P_n^m(\cos\theta)$$

$$W_Z = -\frac{\partial W}{\partial r} = \left(\frac{GM}{r^2}\right)\sum_{n=0}^{N_{max}}(n+1)\left(\frac{a}{r}\right)^n \sum_{m=0}^{n}\left[C_{nm}\cos(m\lambda)+S_{nm}\sin(m\lambda)\right]P_n^m(\cos\theta) - \omega^2 r\sin^2(\theta)$$

In this local coordinate system, W_X is positive northward, W_Y is positive eastward and W_Z positive inward. The magnitude of the gravity vector, g_P, is simply

$$g_P = \sqrt{W_X^2 + W_Y^2 + W_Z^2}$$

The point Q, where $W_P = U_Q$, is determined by iteration. The iteration starts by assuming the point Q has an altitude of zero. Corrections to the guessed solution for the altitude are made by examining the error in the potential, U_Q, and its radial gradient (effectively the Newton-Raphson method for solving non linear equations). Evaluation of the reference gravity, γ, and gravity potential, U, is based on the equations for W and ∇W, but are optimized based on the limited set of non-zero coefficients in the model. The iteration terminates when the error in the potential is less than 10^{-7}J. Since the gradient of the potential is approximate $10 ms^{-1}$, the tolerance in the height is approximately 1 micron.

The resultant gravity anomaly is expressed in units of mgal ($10^{-5} ms^{-1}$).

7.5 Height Anomaly

The Height Anomaly is calculated as the vertical distance between the point P (on the surface of the Earth), the point Q which has the same Gravity Potential as at the point P and lies on the normal to the reference ellipsoid passing through P.

The inputs to this algorithm are :

- A source Spherical Harmonic Potential
- An altitude Grid Function or Track Function

The output from this algorithm is :

- A destination Grid Function or Track Function

The locations of all points, P, where the gravity and gravitational potential are evaluated are defined by the altitude function. This also defines the Grid or Track associated with the output. The process for determining the point Q, given a point P, is precisely the same as used for the Gravity Anomaly (though the gravity vector calculation is omitted since it is not required). The height anomaly, ζ, is then calculated as

$$\zeta = h_P - h_Q$$

where h_P and h_Q are the ellipsoid heights (altitude) of the points P and Q respectively. The resultant height anomaly is expressed in units of meters (m).

7.6 Vertical Deflection North

Deflection of the vertical is calculated according to the definition of Molodenski; the angular separation of the gravity vector at the point P and the normal gravity vector at point Q, such that $U_Q = W_P$.

The inputs to this algorithm are :

- A Spherical Harmonic Potential
- An altitude Grid Function or Track Function

The output from this algorithm is :

- A destination Grid Function of Track Function

The output grid is the same as the input altitude grid. At each input grid location, P, the local gravity vector is calculated from the spherical harmonic potential. The point Q such that W_P is equal to U_Q is located. This point is along the ellipsoid normal through the point P, and thus not strictly on the plumb line through P. The model gravity vector is calculated at the point Q. Both gravity vectors are transformed to the Earth-Fixed Reference Frame (EFRF). The northward deflection is calculated as the difference in the angular distances between these vectors and the Z-axis (North) of the EFRF. The result is reckoned positive northwards and expressed in arcseconds.

7.7 Vertical Deflection East

Deflection of the vertical is calculated according to the definition of Molodenski; the angular separation of the gravity vector at the point P and the normal gravity vector at point Q, such that $U_Q = W_P$.

The inputs to this algorithm are :

- A Spherical Harmonic Potential
- An altitude Grid Function or Track Function

The output from this algorithm is :

- A destination Grid Function or Track Function

The output grid is the same as the input altitude grid. At each input grid location, P, the local gravity vector is calculated from the spherical harmonic potential. The point Q such that W_P is equal to U_Q is located. This point is along the ellipsoid normal through the point P, and thus not strictly on the plumb line through P. The model gravity vector is calculated at the point Q. Both gravity vectors are transformed to the Earth-Fixed Reference Frame (EFRF). The eastward deflection is calculated as the angular separation of the planes defined by great circles through each of these vectors and the Z-axis (North) of the EFRF, scaled by the cosine of the latitude of the point Q. The result is reckoned positive eastwards and expressed in arcseconds.

7.8 Geostrophic Velocity East

Geostrophic velocity North is calculated from numerical approximation of the North-South gradient of a height field.

The inputs to this algorithm are :

- A height field Grid Function
- An equatorial margin

The output from this algorithm is :

- A destination Grid Function

In order to facilitate computation of approximate derivatives and co-location of the resultant grids for both the eastward and northward velocity components, the resultant point grid is interleaved with the input grid (i.e. the midpoint of 4 nearest-neighbours). This always reduces the dimension of the latitude axis by one and will reduce the dimension of the longitude axis by one unless the input grid is global in longitude.

The eastward velocity is given by

$$u_{East} = \frac{-\gamma_o}{fR}\frac{\partial\zeta}{\partial\varphi}$$
$$f = 2\omega\sin(\varphi)$$

where γ_o is the local gravity, ζ is the height field, R is the (local) radius of the Earth, ω is the rotation rate of the Earth and φ is latitude.

The gravity, radius and Coriolis factor (f) are evaluated on the reference ellipsoid at the location of the result grid point. The derivative is approximated by

$$\frac{\partial\zeta}{\partial\varphi} \simeq \frac{\zeta_N-\zeta_S}{\varphi_N-\varphi_S}$$
$$\zeta_N = \frac{\zeta_{NW}+\zeta_{NE}}{2}$$
$$\zeta_S = \frac{\zeta_{SW}+\zeta_{SE}}{2}$$

where NW, NE, SW, SE refer to the North-West, North-East, South-West and South-East nearest-neighbours in the source grid and φ_N and φ_S are the latitudes associated with these neighbouring points. Because the Coriolis factor is zero at the equator, the geostrophic velocity is poorly defined near the equator. The equatorial margin defines a band about the equator (+/-) when the result is not defined.

7.9 Geostrophic Velocity North

Geostrophic velocity North is calculated from numerical approximation of the East-West gradient of a height field.

The inputs to this algorithm are :

- A height field Grid Function
- An equatorial margin

The output from this algorithm is :

- A destination Grid Function

In order to facilitate computation of approximate derivatives and co-location of the resultant grids for both the eastward and northward velocity components, the resultant point grid is interleaved with the input grid (i.e. the midpoint of 4 nearest-neighbours). This always reduces the dimension of the latitude axis by one and will reduce the

dimension of the longitude axis by one unless the input grid is global in longitude.

The northward velocity is given by

$$v_{North} = \frac{\gamma_o}{f R \sin(\theta)} \frac{\partial \zeta}{\partial \lambda}$$

$$f = 2\omega \sin(\varphi)$$

where γ_o is the local gravity, ζ is the height field, R is the (local) radius of the Earth, ω is the rotation rate of the Earth, λ is longitude and φ is latitude and θ is geocentric latitude.

The gravity, radius and Coriolis factor (f) are evaluated on the reference ellipsoid at the location of the result grid point. The derivative is approximated by

$$\frac{\partial \zeta}{\partial \lambda} \simeq \frac{\zeta_E - \zeta_W}{\lambda_E - \lambda_W}$$

$$\zeta_E = \frac{\zeta_{NE} + \zeta_{SE}}{2}$$

$$\zeta_W = \frac{\zeta_{NW} + \zeta_{SW}}{2}$$

where NW, NE, SW, SE refer to the North-West, North-East, South-West and South-East nearest-neighbours in the source grid and λ_E and λ_W are the longitudes associated with these neighbouring points. Because the Coriolis factor is zero at the equator, the geostrophic velocity is poorly defined near the equator. The equatorial margin defines a band about the equator (+/-) when the result is not defined.

7.10 Geostrophic Velocity Along-Track

Geostrophic velocity along-track is calculated from numerical approximation of the station-to-station gradient of a sea surface anomaly. This velocity is resolved into N-S and E-W components expressed at the midpoint of the two stations.

The inputs to this algorithm are :

- A dynamic topography Track Function
- An equatorial margin

The output from this algorithm is :

- A destination Track Function

In order to facilitate computation of approximate derivatives and co-location of the

resultant tracks for both the eastward and northward velocity components, the resultant track is interleaved with the input track (i.e. the midpoint of sequential stations on the input track). This always reduces the number of stations by one. The mid-point location is calculated from the geocentric coordinates of the input stations and is on the great circle joining these stations. The final geodetic location of the output station is defined to be on the ellipsoid. The gradient vector of the height field is assumed to be along-track, and therefore the velocity vector is parallel to the normal to the plane of the great circle joining the stations (reckoned positive in the direction of the cross product of the radial vectors of the stations *i* and *j*; $\mathbf{r_i} \times \mathbf{r_j}$). The magnitude of the along track velocity vector is given by

$$v_{Across-Track} = \frac{\gamma}{fR}\frac{\partial \zeta}{\partial \alpha}$$

$$f = 2\omega \sin(\varphi)$$

where γ is the local gravity, ζ is the height field, R is the (local) radius of the Earth, ω is the rotation rate of the Earth, α is angular separation of the stations and φ is the latitude of the midpoint.

The gravity, radius and Coriolis factor (f) are evaluated on the reference ellipsoid at the location of the mid-point. The derivative is approximated by

$$\frac{\partial \zeta}{\partial \alpha} \simeq \frac{\zeta_j - \zeta_i}{\Delta \alpha_{ij}}$$

where the indexes *i* and *j* refer to sequential stations on the input track. The velocity vector was assumed tangent to the local spherical surface and is resolved into E-W and N-S components based on the local North and local East vectors (tangents on the same spherical surface, hence the across-track velocity is in the plane of the local north and east vectors). Because the Coriolis factor is zero at the equator, the geostrophic velocity is poorly defined near the equator. The equatorial margin defines a band about the equator (+/-) when the result is not defined.

7.11 Mask-Based Substitution

This algorithm produces a Grid Function that is effectively a source Grid Function with selected point replaced with a substitution value. The points where the substitution is made is determined by comparison with a second Grid Function that has the same associated Grid as the source. The second Grid Function is referred to as the Mask, though any Grid Function can serve as a mask in the context of this algorithm. A every point in the source Grid Function the substitution is made if *a comparison* of the corresponding mask value with a *threshold* if logically true. The algorithm supports a variety of comparison operators (ie. greater-than, equal, less-than-or-equal, etc).

The inputs to this algorithm are :

- A source Grid Function
- A mask Grid Function
- A Threshold value
- A Substitution value
- A comparison operator

The output from this algorithm is :

- A destination Grid Function

The substitution concept is straight-forward, however the details are complicated when invalid data are involved, either in the source, the mask or as the threshold. The value to be substituted may also be the invalid value, but this does not effect the decision algorithm. The case where all value are valid is presented below. This is the ideal scenario. This is followed by a discussion of the special cases that apply when invalid values are involved.

$$\begin{aligned} &if \ \ (m_{\varphi,\lambda} \ op \ T) \ \ f^{D}_{\varphi,\lambda} = S \\ &else \ \ f^{D}_{\varphi,\lambda} = f^{S}_{\varphi,\lambda} \end{aligned}$$

where $m_{\varphi,\lambda}$ is the mask value, T is the Threshold, S is the Substitution value, op is the comparison operator and $f^{D}{}_{\varphi,\lambda}$ and $f^{S}{}_{\varphi,\lambda}$ are the destination and source Grid Function values respectively. GUT provides a full set of numeric comparison operators. These are presented in the following table.

Operator	Short Name	Comparison
equal	eq	$m_{\varphi,\lambda} = T$
not equal	ne	$m_{\varphi,\lambda} \neq T$
greater than or equal	ge	$m_{\varphi,\lambda} \geq T$
less than or equal	le	$m_{\varphi,\lambda} \leq T$
greater than	gt	$m_{\varphi,\lambda} > T$
less than	lt	$m_{\varphi,\lambda} < T$

The operation of this algorithm in the presence of invalid data is biased by known use-cases and the requirements of other Grid Function based algorithms in GUT. Based on use-cases, this algorithm is expected to :

- Mask-out land regions by substitution of the invalid value.
- Yield an invalid-free Grid Function by substituting zeros.

The first use-case can be covered by the ideal scenario. The second use case does not formally require any test against a mask, as it is only dependent on the source Grid Function. However, this algorithm aims to facilitate both of these use cases, and others, without enforcing hard requirements for validity. The treatment of point involving invalid data is based on the following rules, listed in order of priority.

- If $f^{S}_{\varphi,\lambda}$ is invalid then the substitution is ALWAYS made.
- If the *Threshold* is invalid, only the *equal* and *not equal* operators are applicable.
- If $m_{\varphi,\lambda}$ is invalid the mask-based comparison is false by definition unless *Threshold* is invalid.

7.12 Mask-Based Merge

This algorithm produces a destination Grid Function by 'tweening' two source Grid Functions with a mask Grid Function used for the 'tween' control value. All three Grid Function must be associated with the same Grid. The source Grid Functions, the primary and the secondary, are expected to be the same physical quantity or both height-fields. The Mask can be any Function, but the algorithm is designed for mask values in the range [0,1]. Values outside this range are clamped to the nearest limit. This algorithm is applicable to both gradient and binary masks.

The inputs to this algorithm are :

- A source Primary Grid Function
- A source Secondary Grid Function
- A Mask Grid Function
- A mask Transparency (0 or 1)

The output from this algorithm is :

- A destination Grid Function

The distinction between the Primary Grid Function and the Secondary Grid Function is merely to define the physical quantity for the destination Grid Function; being that of the Primary. This need for a mask Transparency input is a consequence of this distinction, and merely indicates what mask value results in the destination being equal to the primary (ie. the mask is transparent and the primary is 'visible'). At each point in the destination Grid Function, the value is calculated via the tween function,

$$f^D_{\varphi,\lambda} = \begin{cases} f^P_{\varphi,\lambda}\, m_{\varphi,\lambda} + f^S_{\varphi,\lambda}\,(1-m_{\varphi,\lambda}) & \text{if } T=1 \\ f^P_{\varphi,\lambda}\,(1-m_{\varphi,\lambda}) + f^S_{\varphi,\lambda}\, m_{\varphi,\lambda} & \text{if } T=0 \end{cases}$$

where $m_{\varphi,\lambda}$ is the mask value, T is the Transparency and $f^D_{\varphi,\lambda}$, $f^P_{\varphi,\lambda}$ and $f^S_{\varphi,\lambda}$ are the Destination and Primary and Secondary Grid Function values respectively.

Like Mask-Based Substitution, this algorithm is complicated by the presence of invalid data. The use case that aims to eliminate invalid data is also relevant in this context, and reduces to a single rule.

- If either $m_{\varphi,\lambda}$ or $f^P_{\varphi,\lambda}$ then $f^D_{\varphi,\lambda} = f^S_{\varphi,\lambda}$ otherwise apply the tween function.

7.13 Spatial Filtering

Spatial filtering of a Grid Function is, conceptually, convolution on the sphere. For each point, P, in the Grid associated with the Grid Function, the result is formed by a weighted sum of the data at all points, Q, in the vicinity of P. The weights are defined by a filter kernel and are based on the spherical distance between P and Q. Since this algorithm is convolution on the sphere, the geocentric coordinates of the Grid points are applied when considering spherical distances.

The inputs to this algorithm are :

- A source Grid Function
- A Spatial Filter Kernel

The output from this algorithm is :

- A destination Grid Function

The Spatial Filter Kernel has a specific, finite, range of influence. This means all points, Q, further from P than this range are implicitly assigned a weight of zero. The implementation of this algorithm exploits this fact for the sake of efficiency, as only those points within range need be considered (the kernel footprint). If the data at point P is invalid, the filtered value is also invalid. Within the kernel footprint, only valid data are considered when computing the filtered value. Since the number of points that fall in the kernel footprint is strongly dependent on the latitude of the central point P, the calculation is normalized for each central point. The filtered result, g_P, is calculated as

$$g_P = \frac{\sum_Q w(\alpha_{P,Q}) f_Q}{\sum_Q w(\alpha_{P,Q})}$$

where f_Q is the value of the Grid Function at Q, w is the kernel weight function and $\alpha_{P,Q}$ is the spherical distance between P and Q. The point P is included in the set of points Q.

The footprint of the filter kernel has cylindrical symmetry about the point P on the sphere, but in the Cartesian latitude-longitude space of the Grid Function the footprint looses this symmetry. As a consequence, more points contribute on the polar side of the footprint than on the equatorial side. This introduces a bias into the result. (NOTE: a cell-area weight could be used, but this would be zero for polar points).

GUT provides a set of standard filter kernels. These are:

- Boxcar
- Gaussian
- Truncated Gaussian
- Spherical Cap
- Hanning
- Hamming

All filter kernels are characterized by a scale length. The precise shape and the range of influence of each kernel is based on the scale length. This is given in the following table

Kernel	Scale Length (R)	Shape Function $w(\alpha)$	Range
Boxcar	Radius	$w(\alpha)=1$	R
Gaussian	Half-Width at Half-Maximum	$\sigma=\frac{R}{\sqrt{2\log_e 2}}$ $w(\alpha)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{\alpha^2}{2\sigma^2}}$	10σ
Truncated Gaussian	Half-Width at Half-Maximum	$\sigma=\frac{R}{\sqrt{2\log_e 2}}$ $w(\alpha)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{\alpha^2}{2\sigma^2}}$	2R
Spherical Cap	Radius	$w(\alpha)=\sqrt{R^2-\alpha^2}$	R
Hanning	Half-Width at Half-Maximum	$w(\alpha)=\frac{1}{2}+\frac{1}{2}\cos(\frac{\pi\alpha}{2R})$	2R
Hamming	Half-Width at 54% of Maximum	$w(\alpha)=\frac{54}{100}+\frac{46}{100}\cos(\frac{\pi\alpha}{2R})$	2R

7.14 Spherical Harmonic Synthesis

The transformation of a function from the spatial domain to the spectral domain is based on integration over the sphere of a globally defined function multiplied by a basis function (a spherical harmonic). This exploits the orthogonal nature of the spherical harmonic basis functions, with the result that integration with a single basis function, yields a single coefficient.

$$A_{nm}=\frac{1}{4\pi}\oiint f_{\theta,\lambda}\cos(m\lambda)P_n^m(\cos\theta)\,d\sigma$$

$$B_{nm}=\frac{1}{4\pi}\oiint f_{\theta,\lambda}\sin(m\lambda)P_n^m(\cos\theta)\,d\sigma$$

In order to integrate the function it must be defined over the entire sphere. It is assumed that the values at a Grid Function point are representative in that cell, and can be considered constant. This allows the globalintegration to be reduced to a summation over all cells of the integral within a cell; that integral being independent of the value of the Grid Function.

$$A_{nm}=\frac{1}{4\pi}\sum_{\lambda}\sum_{\theta} f_{\theta,\lambda}\iint_{cell}\cos(m\lambda)P_n^m(\cos\theta)\sin(\theta)\,d\theta\,d\lambda$$

$$B_{nm}=\frac{1}{4\pi}\sum_{\lambda}\sum_{\theta} f_{\theta,\lambda}\iint_{cell}\sin(m\lambda)P_n^m(\cos\theta)\sin(\theta)\,d\theta\,d\lambda$$

The integral over a cell is separable into latitude dependent and longitude dependent components. The cell limits θ_1, θ_2, λ_1 and λ_2 are defined is the mid-points of adjacent cell centers; θ_1 and θ_2 are pushed to the poles at the north and south latitude boundaries of the grid.

$$A_{nm}=\frac{1}{4\pi}\sum_{\lambda}\sum_{\theta} f_{\theta,\lambda}\int_{\lambda_1}^{\lambda_2}\cos(m\lambda)\,d\lambda\int_{\theta_1}^{\theta_2}P_n^m(\cos\theta)\sin(\theta)\,d\theta$$

$$B_{nm}=\frac{1}{4\pi}\sum_{\lambda}\sum_{\theta} f_{\theta,\lambda}\int_{\lambda_1}^{\lambda_2}\sin(m\lambda)\,d\lambda\int_{\theta_1}^{\theta_2}P_n^m(\cos\theta)\sin(\theta)\,d\theta$$

The longitude component can be integrated analytically. The latitude dependent component, which includes a Legendre function, is integrated numerically with a 3-point Simpon's method.

$$K_\theta = \int_{\theta_1}^{\theta_2}P_n^m(\cos\theta)\sin(\theta)\,d\theta \approx$$
$$\frac{1}{6}\left[P_n^m(\cos\theta_1)\sin(\theta_1)+4P_n^m\left(\cos\left(\frac{\theta_1+\theta_2}{2}\right)\right)\sin\left(\frac{\theta_1+\theta_2}{2}\right)+P_n^m(\cos\theta_2)\sin(\theta_2)\right]$$

$$A_{nm}=\frac{1}{4\pi}\sum_{\lambda}\sum_{\theta} f_{\theta,\lambda}\frac{K_\theta}{m}[\sin(m\lambda_2)-\sin(m\lambda_1)]$$

$$B_{nm}=\frac{1}{4\pi}\sum_{\lambda}\sum_{\theta} f_{\theta,\lambda}\frac{K_\theta}{m}[\cos(m\lambda_1)-\cos(m\lambda_2)]$$

7.15 Spherical Harmonic Expansion

The basis of the transformation of a function from the spectral domain to the spatial domain is summation of the series of a Spherical Harmonic Function at coordinates defined by a Grid. If the Reference Ellipsoids associated with the source and destination

differ, then the algorithm can only proceed if the physical quantity is a height field with respect to the Ellipsoid. Under these circumstances a height correction is applied to the result of the summation of the series of the Spherical Harmonic Function.

The inputs to this algorithm are :

- A source Spherical Harmonic Function
- A destination Grid

The output from this algorithm is :

- A destination Grid Function

The algorithm first compares the Reference Ellipsoids associated with the source Spherical Harmonic Function and the destination Grid to determine if the algorithm can proceed, and if so, whether a height correction should be applied. If a height correction is required then the geodetic coordinates of the point on the surface of the source ellipsoid at each destination Grid coordinate is determined. The height of this point provides the height correction. The angular coordinates are then transformed to geocentric coordinates via the source Reference Ellipsoid. This provides the spherical coordinates (q,l) where the Spherical Harmonic Function is evaluated. If a height correction is not required then the geodetic coordinates of the Grid are transformed directly to geocentric coordinates (since the Reference Ellipsoids of the source and destination are the same).

The value of the function is now evaluated via

$$f_{\theta,\lambda} = \sum_{n=0}^{N_{max}} \sum_{m=0}^{n} \left[A_{nm} \cos(m\lambda) + B_{nm} \sin(m\lambda) \right] P_n^m(\cos\theta) \; + \; h_{CORRECTION}$$

where $h_{CORRECTION}$ is zero if not required.

7.16 Spectral Filtering

The Spectral Filtering algorithm operates on a Spectral Harmonic Function by modulating the spectrum of the function. This is simply scaling of the coefficients of the spherical harmonic series with a degree (and perhaps order) dependent weighting function. The weighting function is provided by a Spectral Filter Kernel. GUT implements a small set of spectral filter kernels, all of which have an equivalent kernel in the spatial filtering regime. These kernels are defined with the same names and scale length as their spatial domain counterpart, and are functions of degree, *n*, only.

The inputs to this algorithm are :

- A source Spherical Harmonic Function
- A Spectral Filter Kernel

The output from this algorithm is :

- A destination Spherical Harmonic Function

The coefficients of the destination Spherical Harmonic Function, A^D_{nm} and B^D_{nm}, are the result of a coefficient-wise scaling of the source coefficients, A^S_{nm} and B^S_{nm}, and the weighting function w_n.

$$A^D_{nm} = w_n A^S_{nm}$$

$$B^D_{nm} = w_n B^S_{nm}$$

The weighting function provided by the kernel is a discrete function, typically based on a recurrence relation, and fundamentally a low-pass filter. The weight functions therefore are naturally diminished with increasing order, n, and are artificially clamped to zero if the weight function has an intrinsic negative value. The Spectral Filter Kernels implemented in GUT are Gaussian and Spherical Cap (Pellinen), and the details of these kernels are provided in the following table. It should be noted that the range of the filter is implied by its shape function in the spatial domain.

Kernel	Scale Length (R)	Shape Function w_n
Gaussian	Half-Width at Half-Maximum	$k_o = \frac{\log_e 2}{1-\cos(R)}$ $w_o = \frac{1}{2\pi}$ $w_1 = \frac{\frac{1}{2\pi}(1+e^{-2k_o})}{1-e^{-2k_o}-\frac{1}{k}}$ $w_n = W_{n-2}-\frac{2n-1}{k}w_{n-1}$
Spherical Cap [SJÖ]	Radius	$t_o = \cos(R)$ $w_o = 1$ $w_1 = \frac{1}{2}(1+t_o)$ $w_n = \frac{2n-1}{n+1}t_o w_{n-1}-\frac{n-2}{n+1}w_{n-2}$

8 Processing Units

Each of the following subsections provides a summary of a single processing unit. The section headings are the processing unit type. When work-flows are constructed each instance of a processing unit is given an unique name. In specific cases the name can subtly influence the processing done by the unit (only for command-line argument processing units and then only to specify the command-line flag or augment predefined flags). Some units support specification of a default value. The mechanism for specifying this default is not directly coupled to the processing unit, therefore the processing unit documentation merely indicates if a default can be provided and the text-based formatting requirements. In the current version of GUT the specification of the text containing defaults is always via the text-content of the processing unit XML element in a work-flow file. The names of all input and output ports of a unit are predefined and are documented here. If the data-type associated with a port is not explicit in the port name then it is provided in parentheses after the port name. A brief description of the processing undertaken by the unit is given.

8.1 ChangeFunctionDegreeAndOrder

Produces a Spherical Harmonic Function with a specific maximum degree and order.

Input Ports

- InSphericalHarmonicFunction
- InDegreeAndOrder (Integer)

Output Ports

- OutSphericalHarmonicFunction

The output function is a band-limited copy of the input function. The maximum degree-and-order of the output is actually the minimum of the input degree-and-order and the maximum degree-and-order of the input function. This is because an increase in the band-width would imply zeros for the high frequency coefficients, so the output would be functionally equivalent to the input.

8.2 ChangeGridFunctionTideSystem

Produces a Grid Function with data referenced to a specific tide system.

Input Ports

- InGridFunction
- InTideSystem (Enumeration)

Output Ports

- OutGridFunction

The output function is a transformed copy of the input, where the transformation is dependent on the tide system and physical quantity of the InGridFunction and the InTideSystem. If both source and target tide systems are valid, and the data is 'tidal', then a height correction is made to all data points. The functional form of the correction is the degree-2, order-0 spherical harmonic. The amplitude is calculated by first considering the transform from the source to the tide-free system, and then to the target system. The functions added to the data to transform to the tide-free system are given below.

$$\Delta N = \frac{A}{3}(1-3\cos^2(\theta))$$

Zero-Tide	A = -0.296 k_{LOVE}
Mean-Tide	A = -0.296 (1+k_{LOVE})
k_{LOVE}	0.3022

If the InGridFunction does not have a valid tide system, but is a tidal field, then the data are NOT corrected and the OutGridFunction has the tide system specified by InTideSystem.

8.3 ChangePotentialDegreeAndOrder

Produces a Spherical Harmonic Potential with a specific maximum degree and order.

Input Ports

- InSphericalHarmonicPotential
- InDegreeAndOrder (Integer)

Output Ports

- OutSphericalHarmonicPotential

The output potential is a band-limited copy of the input potential. The maximum degree-and-order of the output is actually the minimum of the input degree-and-order and the maximum degree-and-order of the input potential. This is because an increase in the band-width would imply zeros for the high frequency coefficients, so the output would be functionally equivalent to the input.

8.4 ChangePotentialTideSystem

Produces a Spherical Harmonic Potential with a specific tide system.

Input Ports

- InSphericalHarmonicPotential
- InTideSystem (Enumeration)

Output Ports

- OutSphericalHarmonicPotential

The output potential is the input potential transformed to the required tide-system. The transformation is based on an additive correction to the $C_{2,0}$ coefficient, which is dependent on the input and output tide systems. Three tide systems are supported; tide-free, mean-tide and zero-tide. The transformation is first made to the tide-free system and then to the required system. The following corrections are added to transform to the tide-free system, and subtracted to transform from the tide-free system.

Zero-Tide	4.153×10^{-9}
Mean-Tide	$1.374 \times 10^{-8} + 4.153 \times 10^{-9}$

These numeric values for these corrections are derived from the Bruns equation such that they have the same effect on the geoid height as the *GridFunctionChangeTideSystem* unit.

If the InSphericalHarmonicPotential does not have a valid tide system then the data are NOT corrected and the OutSphericalHarmonicPotential has the tide system specified by InTideSystem.

8.5 ChangeSphericalHarmonicFunctionTideSystem

Produces a Spherical Harmonic Function with data referenced to a specific tide system.

Input Ports

- InSphericalHarmonicFunction
- InTideSystem (Enumeration)

Output Ports

- OutSphericalHarmonicFunction

This unit is the Spherical Harmonic Function equivalent of the *ChangeGridFunctionTideSystem* unit. The tide-system transformation is an additive correction applied to the $C_{2,0}$ coefficient. The values for this correction and are consistent with the *ChangeGridFunctionTideSystem* unit.

8.6 ChangeTrackFunctionReferenceEllipsoid

Produces a Track Function with data referenced to a specific Reference Ellipsoid.

Input Ports

- InTrackFunction
- InReferenceEllipsoid

Output Ports

- OutTrackFunction

This unit transforms the data to a new geodetic coordinate system. If the physical quantity of the data set is a height field with respect to the ellipsoid, the transformation will preserve the spacial position of the data, resulting in a change to the associate track.

8.7 ChangeTrackFunctionTideSystem

Produces a Track Function with data referenced to a specific tide system.

Input Ports

- InTrackFunction
- InTideSystem (Enumeration)

Output Ports

- OutTrackFunction

This unit is the Track Function equivalent of the *ChangeGridFunctionTideSystem* unit.

8.8 CmdLineArgDegreeAndOrder

Extracts a degree-and-order from a command-line flag and its argument.

Output Ports

- OutDegreeAndOrder

The degree-and-order can be specified as an integer, an angular scale or a linear scale, with a different flag used for each of these. The basic flags are predefined, though they can have a single digit suffix (0-9) by including a digit in the processing unit name. The basic flags and arguments are listed below in the order in which they are considered.

Basic Flag	Argument
-DO	Non-negative integral value
-Dkm	Linear scale in kilometers
-Ddeg	Angular scale in degrees

The output is an integer value. If the specification is a linear scale then the value is converted to an integer via the following algorithm.

1. $DegreeAndOrder = \frac{6378.137\pi}{km}$
2. If $DegreeAndOrder$ is not an integral value, round up to the nearest integer.

If the specification is an angular scale then the value is converted to an integer via the following algorithm.

1. $DegreeAndOrder = \frac{180}{deg}$
2. If $DegreeAndOrder$ is not an integral value, round up to the nearest integer.

A default value for the Degree and Order can be specified, and the value “0” is recommended.

8.9 CmdLineArgDouble

Extracts a real number from a command-line flag and its argument.

Output Ports

- OutDouble

The flag extracted from the command-line is the *name* of the processing unit (case-sensitive). The argument must be a real number or the keywork 'NaN' (case-insensitive). The value NaN means Not-a-Number which is used but GUT as the invalid value.

A default value for the number can be provided.

8.10 CmdLineArgGrid

Produces a Grid based on extraction of region, spacing and reference ellipsoid specification via command-line arguments.

Input Ports

- InReferenceEllipsoid

Output Ports

- OutGrid

The Grid is specified in via three independent blocks of information; the region, the spacing and the reference ellipsoid. This can be provided in several ways.

A complete grid can be specified using the -Gf flag. This takes as argument the filename of a file that can be used to extract a Grid (region, spacing and ellipsoid). The -Af flag allows the region and spacing to be extracted from file and combined with the ellipsoid from the *InRefereneceEllipsoid* input. The last mechanism assembles the Grid from independent specifications of the region, spacing and ellipsoid. The region is specified by the latitude and longitude boundaries as a single argument to the flag -R. The recommended format of the argument is "W:E,S:N", though any of the characters ':', ',', or '/' may be used as the field separator. The value for the boundary latitudes and longitudes are degrees in a geodetic coordinate system. The spacing of the Grid is regular in both latitude and longitude, specified as argument to the -I flag, in the format "ΔLon,ΔLat". The spacings are in degrees and should divide the region evenly. The *true* spacing is calculated by dividing the latitude and longitude ranges into *NLAT* and *NLONG* equal-sized intervals based on the best-fit to the specified spacings. The relative error in the specified and calculated spacings must be less than 1%. The ellipsoid is given by the *InRefereneceEllipsoid* input.

The command line flags -R, -I, -Af and -Gf will all be augmented with a single digit suffix (0-9) if the *name* of the unit includes a digit.

A default for the Grid regions and spacing can be provided in the form "W E S N ΔLon Δlat".

8.11 CmdLineArgInputFileName

Extracts the name of an input file from a command-line flag and its argument.

Output Ports

- OutFileName

The command-line flag is given by the *name* of the processing unit. The argument must be the filename of an existing file that the user has permission to read. An absolute or relative path for the file may be specified. If the specification is relative then a search is made for the file, first based on the current working directory and then based on the apriori directory of the GUT installation.

8.12 CmdLineArgInt

Extracts an integral value from a command-line flag and its argument.

Output Ports

- OutInt (Integer)

The command-line flag is given by the *name* of the processing unit. The argument should be an integer in the interval $[-2^{31},+2^{31})$. A default value may be specified.

8.13 CmdLineArgLogicCmpOperator

Extracts a comparison operator from a command-line flag and its argument.

Output Ports

- OutLogicCmpOperator (Enumeration)

The command-line flag is given by the *name* of the processing unit. The argument is a 2-character code that specifies the operator. These are the tokens used in the Fortran-77 language, summarized in the following table.

Operator Name	Code
Equal	eq
Not equal	ne
Greater than	gt
Less than	lt
Greater than or equal	ge
Less than or equal	le

A default value may be specified.

8.14 CmdLineArgOptionalOutputFleName

Extracts a filename for an optional output file from a command-line flag and its argument.

Output Ports

- OutFileName

The command-line flag is given by the *name* of the processing unit. The argument must

be the absolute or relative path of a file that can be created or overwritten. If the path of the filename is a relative specification it is treated as relative to the current working directory. In the context of this processing unit, the term *Optional* simply means that omitting the flag and argument from the command-line, and thus not specifying a filename, is not considered an error. In this case the output is an empty filename and it requires cooperation for the processing unit that interprets the filename to make writing the output file truly optional.

A default value may be specified, but doing so means the filename will *never* be optional.

8.15 CmdLineArgOutputFileName

Extracts a filename for an output file from a command-line flag and its argument.

Output Ports

- OutFileName

The command-line flag is given by the *name* of the processing unit. The argument must be the absolute or relative path of a file that can be created or overwritten. If the path of the filename is a relative specification it is treated as relative to the current working directory.

8.16 CmdLineArgPhysicalQuantity

Extracts a Physical Quantity specification from a command-line flag and its argument.

Output Ports

- OutPhysicalQuantity (Enumeration)

The flag for this processing unit is predefined as -PQ, with the single digit suffix (0-9) if the *name* of the unit includes a digit. The argument is one of a predefined set of tokens, each of which represents a distinct physical quantity. These tokens correspond with variable, standard or long names used for data in GUT internal data files (netCDF), and follow existing naming conventions where appropriate. The list of tokens and common aliases is shown in the following table.

Physical Quantity	Token
Land Mask	land_binary_mask land_mask mask

Height	height height_above_reference_ellipsoid
Orthometric Height	orthometric_height
Normal Height	normal_height
Geoid Height	geoid_height geoid_height_above_reference_ellipsoid geoid (alias)
Sea Surface Height Above Sea Level	ssha sea_surface_height_above_sea_level sea_level_anomaly (alias) sea_surface_height (alias)
Sea Surface Height Above Reference Ellipsoid	ssh sea_surface_height_above_reference_ellipsoid
Sea Level	sea_level sea_level_above_reference_ellipsoid sea_level_height mssh (alias) mss (alias)
Dynamic Topography	dt dynamic_topography sea_surface_height_above_geoid sea_surface_elevation (alias) sea_surface_elevation_anomaly (alias)
Mean Dynamic Topography	mdt sea_level_above_geoid mean_dynamic_topography
Height Anomaly	height_anomaly
Gravity Anomaly	gravity_anomaly free_air_gravity_anomaly
Vertical Deflection North	north_deflection gravity_vector_vertical_northward_deflection
Vertical Deflection East	east_deflection gravity_vector_vertical_eastward_deflection
Gravity	gravity magnitude_of_gravity
Gravity Potential	gravity_potential
Gravitational Potential	gravitational_potential
Geoid Height Error	geoid_height_error geoid_height_above_reference_ellipsoid_error
Geostrophic Velocity North	northward_velocity surface_geostrophic_northward_sea_water_velocity surface_northward_geostrophic_sea_water_velocity
Geostrophic Velocity East	eastward_velocity surface_geostrophic_eastward_sea_water_velocity surface_eastward_geostrophic_sea_water_velocity
Geostrophic Velocity	northward_velocity_anomaly

North Anomaly	surface_geostrophic_northward_sea_water_velocity_assuming_sea_level_for_geoid surface_northward_geostrophic_sea_water_velocity_assuming_sea_level_for_geoid
Geostrophic Velocity East Anomaly	eastward_velocity_anomaly surface_geostrophic_eastward_sea_water_velocity_assuming_sea_level_for_geoid surface_eastward_geostrophic_sea_water_velocity_assuming_sea_level_for_geoid

8.17 CmdLineArgReferenceEllipsoid

Extracts a Reference Ellipsoid specification from a command-line flag and its argument.

Input Ports

- InReferenceEllipsoid

Output Ports

- OutReferenceEllipsoid

The flag for this processing unit is predefined as -Ellipse, with the single digit suffix (0-9) if the *name* of the unit includes a digit. The argument must be one of the predefined tokens given in the table below, a specification of the defining parameters, or a filename. To specify the ellipsoid via parameters, the values for *a*, *1/f* and *GM* must be specified with a ':' separating each of the values. The values can be specified in any order and must be close to the nominal values of standard ellipsoids. The value for ω has the fixed value 7.292115×10^{-5}. If a filename is specified as the argument then the reference ellipsoid will be extracted from the file (ie. a GUT internal netCDF Grid, Grid Function, Track, Track Function or Spherical Harmonic Function). The default value for the Reference Ellipsoid is provided via *InReferenceEllipsoid* input. The default will be either an explicitly created default or extracted from a data source (Potential, Function, Grid, Track, etc).

Token	GM (m^3s^{-2})	a (m)	1/f	ω ($rad.s^{-1}$)
GRIM	398600.4369×10^9	6.37813646×10^6	298.25765	7.292115×10^{-5}
GRS80	398600.5×10^9	6.3781370×10^6	298.257222101	7.292115×10^{-5}
TOPEX	398600.4415×10^9	6.3781363×10^6	298.257	7.292115×10^{-5}
WGS84	398600.5×10^9	6.3781370×10^6	298.257223563	7.292115×10^{-5}
WGS84rev1	398600.4418×10^9	6.3781370×10^6	298.257223563	7.292115×10^{-5}

8.18 CmdLineArgString

Extracts a String (sequence of characters) from a command-line flag and its argument.

Output Ports

- OutString

The command-line flag is given by the *name* of the processing unit. The argument can be any text string. Note, in order to include spaces you will need to "quote the string" when specifying it as a command-line parameter.

A default value may be specified.

8.19 CmdLineArgStringAndDouble

Extracts a String (sequence of characters) and a Real number from a command-line flag and its argument.

Output Ports

- OutString
- OutDouble

The command-line flag is treated as a prefix and a suffix, with the *name* of the processing unit as the prefix and the output string as the suffix. There must not be more than one command-line flag that matches the prefix. The real number is extracted from the argument of the flag. This is identical to the treatment of the argument by the CmdLineArgDouble processing unit.

A default for either the real number, or both the string and the real number may be specified. In the later case the string must precede the number and they must be whitespace separated. If a default is provided then it is valid to specify just the flag on the command-line. Note that the string cannot contain whitespace.

This processing unit is intended for specification of a filter type and scale length.

8.20 CmdLineArgTideSystem

Extracts a tide-system specification from a command-line flag and its argument.

Output Ports

- OutTideSystem (Enumeration)

The flag is predefined as -T with a single digit suffix (0-9) if the name of the processing unit contains a digit. The argument must be one of the tokens *tide-free*, *mean-tide* or *zero-tide*.

A default value for the token may be specified.

8.21 CreateReferenceEllipsoid

Produces a Reference Ellipsoid.

Output Ports

- OutReferenceEllipsoid

The defining properties of the reference ellipsoid can be defined by specification of a default value with the name of a standard reference ellipsoid. The default behaviour is creation of the GUT default Reference Ellipsoid.

8.22 CreateSpatialFilterKernel

Produces a Spatial Filter Kernel of a specific type and scale.

Input Ports

- InFilterType (String)
- InFilterScale (Double)

Output Ports

- OutSpatialFilterKernel

The InFilterType string must be recognized as a valid identifier for the shape function of the filter. The InFilterScale must be a non-negative value for the scale length of the filter in degrees. The interpretation of the scale length is dependent on the filter type. The available filter kernels are shown in the following table.

Kernel	Identifier	Scale Length
Boxcar	box	Radius
Gaussian	g	Half-Width at Half-Maximum
Truncated Gaussian	tg	Half-Width at Half-Maximum
Spherical Cap	sc	Radius
Hanning	han	Half-Width at Half-Maximum
Hamming	ham	Half-Width at 54% of Maximum

8.23 CreateSpectralFilterKernel

Produces a Spectral Filter Kernel of a specific type and scale.

Input Ports

- InFilterType (String)
- InFilterScale (Double)

Output Ports

- OutSpectralFilterKernel

The *InFilterType* string must be recognized as a valid identifier for the shape function of the filter. The *InFilterScale* must be a non-negative value for the scale length of the filter in degrees. The interpretation of the scale length is dependent on the filter type. The available filter kernels are shown in the following table.

Kernel	Identifier	Scale Length
Gaussian	g	Half-Width at Half-Maximum
Spherical Cap	sc	Radius

8.24 FixedValueDouble

Provides a double with a specific value.

Output Ports

- OutDouble

A default value should be provided and this default is the output of the unit. The value must be a real number or the keywork 'NaN' (case-insensitive). The value NaN means Not-a-Number which is used but GUT as the invalid value.

8.25 FixedValueInt

Provides an integral value with a specific value.

Output Ports

- OutInt (Integer)

A default value should be provided and this default is the output of the unit. The value should be an integer in the interval $[-2^{31},+2^{31})$.

8.26 FixedValueLogicCmpOperator

Provides a specific comparison operator.

Output Ports

- OutLogicCmpOperator (Enumeration)

A default operator should be provided and this is the output of the unit. The value is a 2-character code that specifies the operator. These are the tokens used in the Fortran-77 language, summarized in the following table.

Operator Name	Code
Equal	eq
Not equal	ne
Greater than	gt
Less than	lt
Greater than or equal	ge
Less than or equal	le

8.27 FixedValuePhysicalQuantity

Provides a Physical Quantity with a specific value.

Output Ports

- OutPhysicalQuantity (Enumeration)

A default value should be provided and this default is the output of the unit. The value is one of a predefined set of tokens, each of which represents a distinct physical quantity. These tokens correspond with variable names used for data in GUTinternal data files and have previously been described (see *CmdLineArgPhysicalQuantity)*.

8.28 FixedValueArgString

Provides a String (sequence of characters) with a specific value.

Output Ports

- OutString

A default value should be provided and this default is the output of the unit. The value may be the empty string.

8.29 FixedValueTideSystem

Provides a tide-system with a specific value.

Output Ports

- OutTideSystem (Enumeration)

A default value should be provided and this default is the output of the unit. The value must be one of the tokens *tide-free*, *mean-tide* or *zero-tide*.

8.30 GridExport

Export a Grid to file in GUT format.

Input Ports

- InFileName
- InGrid

8.31 GridFromGridFunction

Produces a Grid by extraction of the Grid associated with a Grid Function.

Input Ports

- InGridFunction

Output Ports

- OutGrid

8.32 GridFunctionAdapt

Produces a Grid Function that has adapted its input Grid Function to a specific Grid.

Input Ports

- InGridFunction
- InGrid

Output Ports

- OutGridFunction

This processing unit is a wrapper for the *Grid Function Adaption* algorithm.

8.33 GridFunctionAdd

Produces a Grid Function that is the point-wise addition of its inputs.

Input Ports

- InLhsGridFunction
- InRhsGridFunction

Output Ports

- OutLhsPlusRhsGridFunction

The *Lhs* and *Rhs* components of the port names refer to the Left-Hand-Side operand and the Right-Hand-Side operand of the addition operator (ie. Lhs + Rhs).

The input Grid Functions are required to be associated with the same Grid (that means the same region, spacing and reference ellipsoid) and be either

1. The same physical quantity
2. Height fields

The output Grid Function has the same associated Grid as the inputs and is the point-wise addition of the inputs. The physical quantity of the output is that of the *InLhsGridFunction* input, and is the only thing that makes this addition operator *non-commutative.*

8.34 GridFunctionExport

Export a Grid Function to file in GUT format.

Input Ports

- InFileName
- InGridFunction

8.35 GridFunctionExportGravSoft

Export a Grid Function to file in GRAVSOFT grid format.

Input Ports

- InFileName
- InGridFunction

It should be noted that Grid Functions are only compatible with the GRAVSOFT format if they have a regular spacing in both latitude and longitude. Moreover, this file format does not retain information about the physical quantity represented, nor the reference ellipsoid with which the coordinates are associated.

8.36 GridFunctionExportKmlTiff

Export a Grid Function as an image in TIFF format and export a KML file that defines the TIFF image as a ground overlay.

Input Ports

- InFileName
- InGridFunction
- InHalfRange (Double)

This processing unit is only available if TIFF support was enabled when compiling the GUT software. The image is a North-at-Top, East-at-Right oriented image with each pixel representing one point of an adapted version of the input Grid Function. The adaption migrates the data to the longitude interval [-180,180] degrees with uniform spacing in both latitude and longitude (independent). The colour of a pixel is based on mapping of the function value to a Hot-Cold colour-bar. The colour bar is centered on zero and has a symmetric range, bounded by the input *InHalfRange*. Values above the colour bar range are rendered white and values below are rendered black. Gamma-scaling is applied to each half-range such that greater detail is visible in the middle of the range. A simple KML file is created that defines the TIFF image as a ground overlay. The *InFileName* input specifies the name of the KML file. The filename of the TIFF image is the value of the *InFileName* input with ".tiff" appended. The TIFF image is an output-only format (ie. GUT cannot import a Grid Function from this image file).

8.37 GridFunctionExportTiff

Export a Grid Function as an image in TIFF format.

Input Ports

- InFileName
- InGridFunction
- InHalfRange (Double)

This processing unit is only available if TIFF support was enabled when compiling the GUT software. The image is a North-at-Top, East-at-Right oriented image with each pixel representing one point of the input Grid Function. The colour of a pixel is based on mapping of the function value to a Hot-Cold colour-bar. The colour bar is centered on zero and has a symmetric range, bounded by the input *InHalfRange*. Values above the colour bar range are rendered white and values below are rendered black. Gamma-scaling is

applied to each half-range such that greater detail is visible in the middle of the range. The bottom 4 rows show the colour bar and a black row delineates the grid function and the colour bar. This is an output-only format (ie. GUT cannot import a Grid Function from this image file).

8.38 GridFunctionFilter

Produces a filtered version of the input Grid Function based on filtering in the spatial domain.

Input Ports

- InGridFunction
- InSpatialFilterKernel

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Spatial Filtering Algorithm*.

8.39 GridFunctionGeoidHeight

Produces a Grid Function of the Geoid Height calculated from a Spherical Harmonic Potential at points defined by a specific Grid.

Input Ports

- InSphericalHarmonicPotential
- InGrid

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Geoid Height Algorithm*.

8.40 GridFunctionGeostrophicVelocityEast

Produces a Grid Function of the Eastward component of the geostrophic velocity calculated from a height field Grid Function.

Input Ports

- InGridFunction
- InEquatorialMargin

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Geostrophic Velocity East Algorithm*.

8.41 GridFunctionGeostrophicVelocityNorth

Produces a Grid Function of the Northward component of the geostrophic velocity calculated from a height field Grid Function.

Input Ports

- InGridFunction
- InEquatorialMargin

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Geostrophic Velocity North Algorithm*.

8.42 GridFunctionGravityAnomaly

Produces a Grid Function of the Gravity Anomaly calculated from a Spherical Harmonic Potential at points defined by an altitude Grid Function.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeGridFunction

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Gravity Anomaly Algorithm*.

8.43 GridFunctionGravityAnomalyApprox

Produces a Grid Function of the *approximate* Gravity Anomaly calculated on the Reference Ellipsoid. The algorithm is described in the GOCE Product Data Handbook and has NOT been reproduced here.

Input Ports

- InSphericalHarmonicPotential
- InGrid

Output Ports

- OutGridFunction

This processing unit should be able to reproduce the EGM_GAN_2 gridded data set from the EGM_GCF_2 spherical harmonic potential coefficients in a GOCE Level-2 product.

8.44 GridFunctionHeightAnomaly

Produces a Grid Function of the Height Anomaly calculated from a Spherical Harmonic Potential at points defined by an altitude Grid Function.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeGridFunction

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Height Anomaly Algorithm*.

8.45 GridFunctionImport

Extracts a Grid Function from a file and provides it as output.

Input Ports

- InFileName

Output Ports

- OutGridFunction

All file formats that are recognized by GUT as a source of one or more Grid Functions are supported by this processing unit. These formats are

- GUT internal Grid Function format (netCDF)
- GOCE Level-2 HPF format (XML)
- GRAVSOFT Grid format
- ESA AUX (MSSH and DEM)

With the exception of the GOCE Level-2 HPF format, only one Grid Function data set can be extracted from a file. The GOCE Level-2 HPF format is the unfortunate exception because all components of the product are assembled in a pair of XML files (an header and a data file) and there are 4 independent Grid Function data sets embedded within. The key issue that arises is identification of the *desired* Grid Function. The solution adopted by GUT is use of a default value for the physical quantity desired. This allows definitive identification in the case of the GOCE Level-2 product, but has implication for other file formats. The interpretation of the default value is format dependent and given in the following table.

Format	Interpretation of the default value, PQ
GOCE L2 HPF	• *If* not specified or Unknown then select the geoid_height data. • *Else* select the corresponding data set if available.
GRAVSOFT	• *If* not specified or Unknown then set the physical quantity of the output to be geoid_height. • *Else* set the physical quantity of the output to PQ.
GUT internal	• *If* not specified or Unknown then treat the data in the file as a complete representation of the Grid Function. • Else require that the physical quantity described in the file is consistent with PQ.
ESA AUX	• *If* not specified or Unknown then treat the data in the file as a complete representation of the Grid Function. • Else require that the physical quantity described in the file is consistent with PQ.

8.46 GridFunctionImportSelected

Extracts a Grid Function from a file and provides it as output.

Input Ports

- InFileName
- InPhysicalQuantity

Output Ports

- OutGridFunction

This processing unit is functionally equivalent to the *GridFunctionImport* unit, but differs in that the physical quantity is an explicit input rather than a default value.

8.47 GridFunctionMerge

Produces a Grid Function by merging two input Grid Functions. The merge operation is effectively a weighted sum of the inputs where the weight is provided by a Mask Grid Function.

Input Ports

- InPrimaryGridFunction
- InSecondaryGridFunction
- InMaskGridFunction
- InMaskTransparencyInt

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Mask-Based Merge Algorithm*.

8.48 GridFunctionScale

Produces a Grid Function that is a scalar multiple of the input.

Input Ports

- InGridFunction
- InScalar (Double)

Output Ports

- OutGridFunction

8.49 GridFunctionStats

Calculates rudimentary statistical information for a Grid Function.

Input Ports

- InGridFunction

This processing unit determines or calculates the

- Minimum value
- Maximum value
- Mean & Variance
- 'latitude-weighted' Mean & Variance
- Total number of points and the number of points with valid data.

and reports this information to standard-output.

8.50 GridFunctionSubstitute

Produces a Grid Function from an input by selective substitution of a specific value. The selection is controlled by comparing a mask Grid Function with a fixed threshold.

Input Ports

- InGridFunction
- InMaskGridFunction
- InThresholdDouble
- InSubstituteDouble
- InLogicCmpOperator

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Mask-Based Substitution Algorithm*.

8.51 GridFunctionSubtract

Produces a Grid Function that is the point-wise subtraction of its inputs.

Input Ports

- InLhsGridFunction
- InRhsGridFunction

Output Ports

- OutLhsMinusRhsGridFunction

The *Lhs* and *Rhs* components of the port names refer to the Left-Hand-Side operand and the Right-Hand-Side operand of the subtraction operator (ie. Lhs − Rhs).

The input Grid Functions are required to be associated with the same Grid (that means the same region, spacing and reference ellipsoid) and be either

- The same physical quantity
- Height fields

The output Grid Function has the same associated Grid as the inputs and is the point-wise addition of the inputs. The physical quantity of the output is that of the *InLhsGridFunction* input.

8.52 GridFunctionSurfaceGravitationalPotential

Produces a Grid Function that is the gravitational potential (*ie. does NOT include the centrifugal potential*) at a specified surface, evaluated from a Spherical Harmonic Potential.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeGridFunction

Output Ports

- OutGridFunction

The output Grid Function has the same associated Grid as *InAltitudeGridFunction*.

8.53 GridFunctionSurfaceGravity

Produces a Grid Function that is the magnitude of the gravity vector at a specified surface, evaluated from a Spherical Harmonic Potential.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeGridFunction

Output Ports

- OutGridFunction

The output Grid Function has the same associated Grid as *InAltitudeGridFunction*.

8.54 GridFunctionSurfaceGravityPotential

Produces a Grid Function that is the gravity potential (*ie. DOES include the centrifugal potential*) at a specified surface, evaluated from a Spherical Harmonic Potential.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeGridFunction

Output Ports

- OutGridFunction

The output Grid Function has the same associated Grid as *InAltitudeGridFunction*.

8.55 GridFunctionVerticalDeflectionEast

Produces a Grid Function of the Deflection of the Vertical in the eastward direction calculated from a Spherical Harmonic Potential at points defined by a specific Grid.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeGridFunction

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Vertical Deflection East Algorithm.*

8.56 GridFunctionVerticalDeflectionEastApprox

Produces a Grid Function of the *approximate* Deflection of the Vertical in the eastward direction calculated on the Reference Ellipsoid. The algorithm is described in the GOCE Product Data Handbook and has NOT been reproduced here.

Input Ports

- InSphericalHarmonicPotential
- InGrid

Output Ports

- OutGridFunction

This processing unit should be able to reproduce the EGM_GVE_2 gridded data set from the EGM_GCF_2 spherical harmonic potential coefficients in a GOCE Level-2 product.

8.57 GridFunctionVerticalDeflectionNorth

Produces a Grid Function of the Deflection of the Vertical in the northward direction calculated from a Spherical Harmonic Potential at points defined by a specific Grid.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeGridFunction

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Vertical Deflection North Algorithm.*

8.58 GridFunctionVerticalDeflectionNorthApprox

Produces a Grid Function of the *approximate* Deflection of the Vertical in the northward direction calculated on the Reference Ellipsoid. The algorithm is described in the GOCE Product Data Handbook and has NOT been reproduced here.

Input Ports

- InSphericalHarmonicPotential
- InGrid

Output Ports

- OutGridFunction

This processing unit should be able to reproduce the EGM_GVN_2 gridded data set from the EGM_GCF_2 spherical harmonic potential coefficients in a GOCE Level-2 product.

8.59 GridImport

Extracts a Grid form a file and provides it as output.

Input Ports

- InFileName

Output Ports

- OutGrid

8.60 ReferenceEllipsoidFromGrid

Produces a Reference Ellipsoid by extraction of the Reference Ellipsoid associated with a Grid.

Input Ports

- InGrid

Output Ports

- OutReferenceEllipsoid

8.61 ReferenceEllipsoidFromGridFunction

Produces a Reference Ellipsoid by extraction of the Reference Ellipsoid associated with a Grid Function.

Input Ports

- InGridFunction

Output Ports

- OutReferenceEllipsoid

This unit is essentially for convenience and is functionally equivalent to a *ReferenceEllipsoidFromGrid* unit connected to a *GridFromGridFunction* unit.

8.62 ReferenceEllipsoidFromSphericalHarmonicFunction

Produces a Reference Ellipsoid by extraction of the Reference Ellipsoid associated with a Spherical Harmonic Function.

Input Ports

- InSphericalHarmonicFunction

Output Ports

- OutReferenceEllipsoid

8.63 ReferenceEllipsoidFromTrack

Produces a Reference Ellipsoid by extraction of the Reference Ellipsoid associated with a Track.

Input Ports

- InTrack

Output Ports

- OutReferenceEllipsoid

8.64 ReferenceEllipsoidFromTrackFunction

Produces a Reference Ellipsoid by extraction of the Reference Ellipsoid associated with a Track Function.

Input Ports

- InTrackFunction

Output Ports

- OutReferenceEllipsoid

8.65 ReferenceEllipsoidImport

Extracts a Reference Ellipsoid from a file and provides it as an output.

Input Ports

- InFileName

Output Ports

- OutReferenceEllipsoid

8.66 SetGridFunctionPhysicalQuantity

Produces a Grid Function with a specific Physical Quantity.

Input Ports

- InGridFunction
- InPhysicalQuantity (enumeration)

Output Ports

- OutGridFunction

This unit explicitly sets the value of the physical quantity of a Grid Function, which may implicitly change the units of the data. The data in the resulting output function is numerically identical to the input. This processing unit should be used in association with processing units that do not implicitly manage meta data (i.e. The subtraction of two grid function may produce a result that has a physical interpretation different to the inputs).

8.67 SetGridFunctionReferenceEllipsoid

Produces a Grid Function with a specific Reference Ellipsoid.

Input Ports

- InGridFunction
- InReferenceEllipsoid

Output Ports

- OutGridFunction

This unit explicitly sets the Reference Ellipsoid of a Grid Function. It DOES NOT make any correction to the data points. This unit should only be used for correcting an erroneous Reference Ellipsoid specification in a data set.

8.68 SetGridFunctionTideSystem

Produces a Grid Function with a specific TideSystem.

Input Ports

- InGridFunction
- InTideSystem (enumeration)

Output Ports

- OutGridFunction

This unit explicitly sets the value of the Tide System of a Grid Function. It DOES NOT make any correction to the data points (see *ChangeGridFunctionTideSystem*). This unit should only be used for correcting an erroneous tide system specification in a data set.

8.69 SetGridReferenceEllipsoid

Produces a Grid with a specific Reference Ellipsoid.

Input Ports

- InGrid
- InReferenceEllipsoid

Output Ports

- OutGrid

This unit explicitly sets the Reference Ellipsoid of a Grid. It DOES NOT transform the coordinates. This unit should only be used for correcting an erroneous Reference Ellipsoid specification in a data set.

8.70 SetTrackFunctionReferenceEllipsoid

Produces a Track Function with a specific Reference Ellipsoid.

Input Ports

- InTrackFunction
- InReferenceEllipsoid

Output Ports

- OutTrackFunction

This unit explicitly sets the Reference Ellipsoid of a TrackFunction. It DOES NOT transform the data. This unit should only be used for correcting an erroneous Reference Ellipsoid specification in a data set.

8.71 SetTrackReferenceEllipsoid

Produces a Track with a specific Reference Ellipsoid.

Input Ports

- InTrack
- InReferenceEllipsoid

Output Ports

- OutTrack

This unit explicitly sets the Reference Ellipsoid of a Track. It DOES NOT transform the coordinates. This unit should only be used for correcting an erroneous Reference Ellipsoid

specification in a data set.

8.72 SphericalHarmonicFunctionAdd

Produces a Spherical Harmonic Function that is the sum of its two inputs.

Input Ports

- InLhsSphericalHarmonicFunction
- InRhsSphericalHarmonicFunction

Output Ports

- OutLhsPlusRhsSphericalHarmonicFunction

This unit creates a Spherical Harmonic Function with a maximum degree and order equal to the maximum of the degree and order of the inputs. The resultant coefficients are the point-wise (degree,order) addition of the input coefficients. The physical quantities of the inputs must be either the same, or both must be height fields. The physical quantity of the output is that of the *InLhsSphericalHarmonicFunction* input. The inputs must be associated with the same Reference Ellipsoid.

8.73 SphericalHarmonicFunctionExpansion

Produces a Grid Function by evaluating the series of a Spherical harmonic Function at the locations defined by a Grid.

Input Ports

- InSphericalHarmonicFunction
- InGrid

Output Ports

- OutGridFunction

This processing unit is a wrapper around the *Spherical Harmonic Expansion Algorithm*.

8.74 SphericalHarmonicFunctionExport

Export a Spherical Harmonic Function to file in GUT format.

Input Ports

- InFileName
- InSphericalHarmonicFunction

8.75 SphericalHarmonicFunctionFilter

Produces a Spherical Harmonic Function by filtering in the spectral domain.

Input Ports

- InSphericalHarmonicFunction
- InSpectralFilterKernel

Output Ports

- OutSphericalHarmonicFunction

This processing unit is a wrapper around the *Spectral Filtering Algorithm*.

8.76 SphericalHarmonicFunctionImport

Extracts a Spherical Harmonic Function from a file and provides at as output.

Input Ports

- InFileName

Output Ports

- OutSphericalHarmonicFunction

8.77 SphericalHarmonicFunctionScale

Produces a Spherical Harmonic Function that is a scalar multiple of the input.

Input Ports

- InSphericalHarmonicFunction
- InScalar (Double)

Output Ports

- OutSphericalHarmonicFunction

Each coefficient in the output function is the corresponding coefficient in the input function multiplied by *InScalar*.

8.78 SphericalHarmonicFunctionSubtract

Produces a Spherical Harmonic Function that is the difference of its two inputs.

Input Ports

- InLhsSphericalHarmonicFunction
- InRhsSphericalHarmonicFunction

Output Ports

- OutLhsMinusRhsSphericalHarmonicFunction

This unit creates a Spherical Harmonic Function with a maximum degree and order equal to the maximum of the degree and order of the inputs. The resultant coefficients are the point-wise (degree,order) subtraction of the *InRhsSphericalHarmonicFunction* coefficients from the *InLhsSphericalHarmonicFunction coefficients* (high-frequency coefficients are implicitly zero). The physical quantities of the inputs must be either the same, or both must be height fields. The physical quantity of the output is that of the *InLhsSphericalHarmonicFunction* input. The inputs must be associated with the same Reference Ellipsoid.

8.79 SphericalHarmonicFunctionSynthesis

Produces a Spherical Harmonic Function from a (global) Grid Function.

Input Ports

- InDegreeAndOrder
- InGridFunction

Output Ports

- OutSphericalHarmonicFunction

This processing unit is a wrapper around the *Spherical Harmonic Synthesis Algorithm*.

8.80 SphericalHarmonicPotentialExport

Exports a Spherical Harmonic Potential to file in netCDF format.

Input Ports

- InFileName
- InSphericalHarmonicPotential

8.81 SphericalHarmonicPotentialImport

Extracts a Spherical Harmonic Potential from a file and provides it as output.

Input Ports

- InFileName

Output Ports

- OutSphericalHarmonicPotential

8.82 TideSystemFromGridFunction

Extracts the Tide System from a Grid Function and provides it as output.

Input Ports

- InGridFunction

Output Ports

- OutTideSystem

8.83 TideSystemFromSphericalHarmonicFunction

Extracts the Tide System from a Spherical Harmonic Function and provides it as output.

Input Ports

- InSphericalHarmonicFunction

Output Ports

- OutTideSystem

8.84 TideSystemFromSphericalHarmonicPotential

Extracts the Tide System from a Spherical Harmonic Potential and provides it as output.

Input Ports

- InSphericalHarmonicPotential

Output Ports

- OutTideSystem

8.85 TideSystemFromTrackFunction

Extracts the Tide System from a Track Function and provides it as output.

Input Ports

- InTrackFunction

Output Ports

- OutTideSystem

8.86 TrackExport

Exports a Track to file in GUT format.

Input Ports

- InFileName
- InTrack

8.87 TrackFromTrackFunction

Produces a Track by extraction of the Track associated with a Track Function.

Input Ports

- InTrackFunction

Output Ports

- OutTrack

8.88 TrackFunctionAdd

Produces a Track by point-wise addition of the two inputs.

Input Ports

- InLhsTrackFunction
- InRhsTrackFunction

Output Ports

- OutLhsPlusRhsTrackFunction

The value at each point in the output is the sum of the values at the corresponding input points. The inputs are required to be on the same reference ellipsoid. The physical quantities of the input must be either the same of both height fields. The physical quantity of the output is that of the *InLhsTrackFunction* input.

8.89 TrackFunctionExport

Exports a Track Function to file in GUT internal format.

Input Ports

- InFileName
- InTrackFunction

8.90 TrackFunctionFromGridFunction

Produces a Track Function by interpolation of a Grid Function at locations defined by a Track.

Input Ports

- InGridFunction
- InTrack

Output Ports

- OutTrackFunction

This processing unit calculates the value at each point by bilinear interpolation of a Grid Function. At each point on the Track the interpolation uses the four nearest points of the Grid Function that contain that point. If the Reference Ellipsoid of the Track and Grid function differ and the physical quantity is a height field with respect to the ellipsoid, then the coordinates of the track point are transformed to the geodetic coordinate system of the Grid Function before interpolating and a height correction is applied.

8.91 TrackFunctionGeoidHeight

Produces a Track Function of the Geoid Height calculated from a Spherical Harmonic Potential at points defined by a specific Track.

Input Ports

- InSphericalHarmonicPotential
- InTrack

Output Ports

- OutTrackFunction

This processing unit is a wrapper around the *Geoid Height Algorithm*.

8.92 TrackFunctionGeostrophicVelocityEast

Produces a Track Function of the Eastward component of the Geostrophic velocity calculated from a height Track Function.

Input Ports

- InTrackFunction

- InEquatorialMargin (Double)

Output Ports

- OutTrackFunction

This processing unit is a wrapper around the *Geostrophic Velocity Along-Track Algorithm*.

8.93 TrackFunctionGeostrophicVelocityNorth

Produces a Track Function of the Northward component of the Geostrophic velocity calculated from a height Track Function.

Input Ports

- InTrackFunction
- InEquatorialMargin (Double)

Output Ports

- OutTrackFunction

This processing unit is a wrapper around the *Geostrophic Velocity Along-Track Algorithm*.

8.94 TrackFunctionGravityAnomaly

Produces a Track Function of the Gravity Anomaly calculated from a Spherical Harmonic Potential at points defined by an altitude Track Function.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeTrackFunction

Output Ports

- OutTrackFunction

This processing unit is a wrapper around the *Gravity Anomaly Algorithm*.

8.95 TrackFunctionHeightAnomaly

Produces a Track Function of the Height Anomaly calculated from a Spherical Harmonic Potential at points defined by an altitude Track Function.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeTrackFunction

Output Ports

- OutTrackFunction

This processing unit is a wrapper around the *Height Anomaly Algorithm*.

8.96 TrackFunctionImport

Extracts a Track Function from a file and provides it as output.

Input Ports

- InFileName

Output Ports

- OutTrackFunction

8.97 TrackFunctionImportSelected

Extracts a Track Function from a file and provides at as output.

Input Ports

- InFileName
- InPhysicalQuantity

Output Ports

- OutTrackFunction

This processing unit is functionally equivalent to the Track*FunctionImport* unit, but differs in that the physical quantity is an explicit input rather than a default value.

8.98 TrackFunctionScale

Produces a Track Function that is a scalar multiple of its input.

Input Ports

- InTrackFunction
- InScalar (Double)

Output Ports

- OutTrackFunction

Each value in the output function is the corresponding value in the input function multiplied by *InScalar*.

8.99 TrackFunctionSubtract

Produces a Track by point-wise subtraction of the two inputs.

Input Ports

- InLhsTrackFunction
- InRhsTrackFunction

Output Ports

- OutLhsMinusRhsTrackFunction

The value at each point in the output is the difference of the values at the corresponding input points. The inputs are required to be on the same reference ellipsoid. The physical quantities of the input must be either the same of both height fields. The physical quantity of the output is that of the *InLhsTrackFunction* input.

8.100 TrackFunctionVerticalDeflectionEast

Produces a Track Function of the eastward component of the deflection of the vertical calculated from a Spherical Harmonic Potential at points defined by an altitude Track Function.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeTrackFunction

Output Ports

- OutTrackFunction

This processing unit is a wrapper around the *Vertical Deflection East Algorithm*.

8.101 TrackFunctionVerticalDeflectionNorth

Produces a Track Function of the northward component of the deflection of the vertical calculated from a Spherical Harmonic Potential at points defined by an altitude Track Function.

Input Ports

- InSphericalHarmonicPotential
- InAltitudeTrackFunction

Output Ports

- OutTrackFunction

This processing unit is a wrapper around the *Vertical Deflection North Algorithm*.

8.102 TrackImport

Extracts a Track from a file and provides it as output.

Input Ports

- InFileName

Output Ports

- OutTrack

9 Visualization of Results

In general, GUT exports results in binary files in the netCDF format with the CF-Convention adopted to support interpretation by third-party tools. The visualization tool from the Basic Radar Altimetry Toolbox, BRATDisplay, has implicitly support for the GUT internal netCDF files and as a convenience to users this tool is provided in the binary distributions of GUT. Alternative display mechanisms, specifically for a quick-look at Grid Function results, are provided by the *expimage* and *expkml* work-flows. These work-flows produce an image file in TIFF format that can be viewed with any TIFF compliant image display application. The *expkml* work-flow also produces a KML file that references the image file. The KML file can be loaded by GoogleEarth® and will display the image as an overlay.

9.1 BratDisplay

The BratDisplay tool supports both 2D (map projection) and 3D (ellipsoid surface) rendering of Grid Function data. There are two mechanisms for controlling the input, both based on launching the application with command-line arguments. The first mechanism requires creation of a configuration (parameter) file that references the data file and sets additional display parameters. This file is specified as the only argument on the command-line and is the mechanism used within the scope of BRAT and may therefore be familiar to experienced BRAT users. The alternative mechanism was introduced with BRAT version 1.1.1, and allows a netCDF file to be provided as the command-line argument directly. In addition to the filename, flags and arguments can be specified to control some aspects of the data import and display. The set of command-line flags are shown by BratDisplay when it is run without arguments, but the minimal command-line for displaying a GUT netCDF file is

```
% BratDisplay input_file.nc
```

For more information on BRAT and BratDisplay, consult the BRAT User Guide.

10 Software Extension

Extending the processing capabilities of GUT is a software development activity. The tools required are the same as those required to build GUT from a source package. It is therefore recommended that GUT is built from source on your system and then installed.

10.1 Quick-guide and Overview

The GUT software is written in C++ and extension of GUT is based on features implicit in this language. Simply stated, each processing unit in GUT is a C++ class derived from a base class; the base class supporting the incorporation of the unit in the GUT framework. To extend GUT by developing a new processing unit *a new C++ class must be declared and implemented*. Therefore, some C++ code must be written, though this code is largely

"boiler-plate code" and can be copied from an existing class. The class declaration specifies the unique name of the processing unit and the names and types of the input and output ports. The data processing functionality of the unit is defined in a virtual method called *calculate*. Although C++ should be treated as the preferred language for development, C and Fortran APIs exist to allow C and Fortran functions to be called within the calculate method. This allows the core data processing to be coded in C or Fortran and coupled to the C++ based GUT framework. The APIs provide facilities for passing the input and output data between the C++ and C/Fortran software layers.

The src subdirectory of the GUT installation contains a Makefile and a few source files. Together they allow a custom version of the gut command-line tool to be created that extends the processing unit set with a few trivial examples. These files should serve as templates for creating your own extensions, but as a first step, you should build the example to verify that you have the required software development tools. The build process is :-

- Create a working directory on your system.
- Copy ALL files in `<GUT_ROOT>`/src to your working directory.
- Run 'make' from the working directory and verify that the executable gut_custom was created.
- Run 'gut_custom --version' to verify the file is executable.

Using the files Extension_C.cpp and Extension_C.h as a guide, create the C++ class for your new processing unit. Also develop the core functionality for your processing unit as a function/subroutine in a separate file in your chosen language (C++, C, or Fortran). Make sure to use the appropriate data type for the arguments of your function. When passing GUT high-level data types (ie. GridFunction, etc) you must use an opaque handle passed by reference as the data type for your function. The functions in the GUT API are also defined in terms of these opaque handles. Integers and doubles (double precision) can be passed in the manner conventional for the language your using.

Having developed a new processing unit, you must register it with the GUT framework in order to make use of it in work-flows. The name you chose for your C++ class is also the type-name your processing unit will have in a work-flow (ie. the XML element name), so choose the name carefully (Note that this name must be different from all existing processing units). To register your processing unit you must include the header file of your C++ class and add one line of code in the function *registerProcUnitExtensions* in the file gut.cpp. Use the example extensions as a guide (ie. the class API_Example_C defined in the file Extension_C.h). Finally, edit the Makefile and include your source files in the file list of the variables *c_sources*, *cpp_sources* and *f_sources*. Run 'make' to compile your code and link a new gut_custom executable. Create a work-flow that uses your new processing unit and test it (NOTE. The gut_custom executable will need to be installed in the bin directory of the GUT installation in order to locate work-flow and apriori data files without full path names. In order to test your processing unit you will need to supply complete filenames for all files).

10.2 GUT Data-Model and APIs

In general, GUT processing units work with high-level data type that typically encapsulate a moderately large amount of data. To provide the flexibility for construction of arbitrary work-flows and efficient data handling, all GUT high-level types use a reference counting mechanism and a Copy-On-Write policy. This allows memory resources to be managed efficiently without any action on the part of the code developer. To the C++ developer this means the GUT types can be created, copied, passed to a function as an argument by value (or reference) and returned from functions without any performance penalty or any code for explicitly releasing resources. This is a fundamental part of the GUT software design and makes extending GUT in C++ easy and safe. The C and Fortran APIs are a compromise because they do not naturally collaborate with this data-model. Instead, the C API defines an opaque handle data type (GutHandle_t), which is used for ALL high-level GUT types. These handles provide a mechanism for passing the input and output data of processing unit ports to and from C functions and the macro GUT_HANDLE() facilitates initializing GutHandle_t variables from GUT high-level types. Within the C functions the C API provides a basic set of functions to accessing the low-level data within the high-level type. However, it is the responsibility of the C programmer to use ONLY the API functions that correspond to the true data type of the handle. Failing to do this will result in incorrect processing or run-time crashes. The Fortran API is a thin wrapper on the C API but because different compilers use different symbol naming schemes for C and Fortran functions, the UFNAME() macro should be used when calling API functions. This macro expressing the function or subroutine name in both all uppercase and all lowercase characters. The appropriate selection of one of these names is made based on your Fortran compiler. The C and Fortran APIs are defined in the header files gutC.h and gutF.h respectively.

10.3 Processing Unit Design

Processing units should be designed to do just one elementary task and have as few inputs and outputs possible. Taking this approach will maximize the chance that your processing unit can be used in more contexts than it was originally intended.

The C++ class that defines a processing unit and its inputs and outputs should be kept concise and clear. This is best achieved by coding the core processing in a separate function (and file) and simply calling this function from the calculate method of the processing unit class. In C++ it is convenient to define the core processing function such that it returns the output, but if using the C or Fortran APIs you should make the output an argument of the function. This allows use of the GUT_HANDLE macro, simplifying the code . If need be, declare the output variable as a temporary variable (of GUT high-level types) in the C++ code and set the output port data from the temporary variable after calling your processing function.

10.4 Beyond Processing Units

The C, C++ and Fortran APIs are all part of the libgut.a static library located in the lib subdirectory of the GUT installation. The corresponding header files are located in the include subdirectory of the GUT installation. This library can be used for developing applications completely independently of the GUT work-flow processor based framework. This is perhaps most appropriate for IO utilities that convert your data to GUT internal

netCDF format. Development of this nature is beyond the scope of this document and is recommended only to advanced users with software development experience. In this case the header files (and the source in the GUT source package) will be your guide.

Appendix-A

```
<?xml version="1.0"?>
<workflow>
    <!-- Section 1 : Processing Units -->
    <units>
        <CmdLineArgInputFileName name="InFile"></CmdLineArgInputFileName>
        <CmdLineArgReferenceEllipsoid name="RE" />
        <CmdLineArgGrid name="Grid">0.5 359.5 -89.5 89.5 1.0 1.0</CmdLineArgGrid>
        <CmdLineArgDegreeAndOrder name="DegreeAndOrder" />
        <CmdLineArgTideSystem name="TideSystem" />
        <CreateReferenceEllipsoid name="DefaultRE" />
        <SphericalHarmonicPotentialImport name="DataShp" />
        <ChangePotentialDegreeAndOrder name="DegOrdSetShp" />
        <ChangePotentialTideSystem name="TideSetShp" />
        <GridFunctionGeoidHeight name="GeoidHeight" />
        <CmdLineArgOutputFileName name="OutFile" >geoid_height.nc
             </CmdLineArgOutputFileName>
        <GridFunctionExport name="Export" />
    </units>
    <!-- Section 2 : Connections -->
    <connection>
        <socket unit="InFile" port="OutFileName" />
        <plug unit="DataShp" port="InFileName" />
    </connection>
    <connection>
        <socket unit="DefaultRE" port="OutReferenceEllipsoid" />
        <plug unit="RE" port="InReferenceEllipsoid" />
    </connection>
    <connection>
        <socket unit="RE" port="OutReferenceEllipsoid" />
        <plug unit="Grid" port="InReferenceEllipsoid" />
    </connection>
    <connection>
        <socket unit="DataShp" port="OutSphericalHarmonicPotential" />
        <plug unit="DegOrdSetShp" port="InSphericalHarmonicPotential" />
    </connection>
    <connection>
        <socket unit="DegOrdSetShp" port="OutSphericalHarmonicPotential" />
        <plug unit="TideSetShp" port="InSphericalHarmonicPotential" />
    </connection>
```

```
    <connection>
        <socket unit="TideSetShp" port="OutSphericalHarmonicPotential" />
        <plug unit="GeoidHeight" port="InSphericalHarmonicPotential" />
    </connection>
    <connection>
        <socket unit="TideSystem" port="OutTideSystem" />
        <plug unit="TideSetShp" port="InTideSystem" />
    </connection>
    <connection>
        <socket unit="DegreeAndOrder" port="OutDegreeAndOrder" />
        <plug unit="DegOrdSetShp" port="InDegreeAndOrder" />
    </connection>
    <connection>
        <socket unit="Grid" port="OutGrid" />
        <plug unit="GeoidHeight" port="InGrid" />
    </connection>
    <connection>
        <socket unit="GeoidHeight" port="OutGridFunction" />
        <plug unit="Export" port="InGridFunction" />
    </connection>
    <connection>
        <socket unit="OutFile" port="OutFileName" />
        <plug unit="Export" port="InFileName" />
    </connection>

    <!-- Section 3 : Manual -->
    <manual>
<![CDATA[
Synopsis  : Extract a set of spherical harmonic potential coefficients
            (and GM, R, tide system) from file and calculate the height
            of the geoid on a chosen Grid with a specified expansion of
            the geopotential. The Grid can be specified in one of
            several ways. The default is a global 1x1 degree grid on
            the GRS80 ellipsoid with the potential expanded to the
            degree and order defined by the input file.

Arguments :
       -InFile input_file_name
             Input file containing the geopotential.

       -Gf input_grid_file     (option 1 of 3)
             Specifies the file that defines the output Grid. This can
             be any file from which GUT can extract a grid. Note, this
             includes the ellipsoid.
   OR
       -Af input_grid_file     (option 2 of 3)
             Specifies the file that defines the latitude and longitude
             axes of the output Grid. This can be any file from which
```

GUT can extract a grid. The -Ellipse flag can be used to specify the ellipsoid, otherwise the GUT default of GRS80 is assumed.

OR

-R w:e,s:n (option 3 of 3)

Specifies the latitude and longitude bounds of the equiangular output grid. The longitude limits (w:e) must be in degrees in the range [-360,+360] and the latitude limits (s:n) must be in degrees in the range [-90,+90]. This option is normally use in combination with the -I and -Ellipse options.

-I de:dn (optional)

Specifies the longitude and latitude grid spacing. de and dn are spacings in degrees. The intervals specified by the -R option must be close to integer multiple of these spacings. The spacings will be recalculated to ensure the region is precisely divided.

-Ellipse ellipse (optional)

Set a specific Reference Ellipsoid. If not specified, the GUT default of GRS80 is used. The ellipsoid can be specified as one of ...

* ellipsoid name
  GRS80 TOPEX GRIM WGS84 WGS84rev1
* the parameters
  formatted as inverse_flattening:a:GM
* filename
  extracts the ellipsoid from the meta-data in this file

-DO degree_and_order (option 1 of 3)

Specifies the degree and order of the geopotential expansion. degree_and_order must be a positive integer.

OR

-Dkm scale_length (option 2 of 3)

Specifies the degree and order of the geopotential expansion by specifying a scale length, in km, at the Earth surface.

OR

-Ddeg scale_angle (option 3 of 3)

Specifies the degree and order of the geopotential expansion by specifying a scale angle in degrees.

-T tide-system (optional)

Compute the result in a specific tide-system. This applies a correction to the geopotential before computing the geoid height. tide-system must be one of :

tide-free mean-tide zero-tide

```
       -OutFile output_file_name
             Output filename for resulting netCDF file.

Key Processing Units:   (See the GUT User Guide for more information)
       * SphericalHarmonicPotentialImport
       * ChangePotentialDegreeAndOrder
       * ChangePotentialTideSystem
       * GridFunctionGeoidHeight

]]>
    </manual>
 </workflow>
```