Deliverable: WP4000:

“SYSTEM SPECIFICATION AND ARCHITECTURAL DESIGN”

15th March 2007

GOCE User Toolbox Specifications (GUTS)

Development of Algorithms for the generation of a GOCE User Toolbox and an Absolute Dynamic Topography Product

Ref:
ESA/XGCE-DTEX-EOPS-SW-04-0001 “GOCE User Toolbox Specifications (GUTS)”
Coordinator: Per Knudsen (DNSC)

Deliverable: WP4000: “System Specification and Architectural Design”
Responsible partner: Reading eScience Centre
Prepared by Dan Bretherton

DOCUMENT CHANGE LOG				
Rev.	Date	Sections modified	Comments	Changed by
1	6th December 2006	N/A	Draft version 2.7 submitted to ESA for review	Dan Bretherton & Keith Haines
2	15th March 2007	N/A	Final version 2.8 revised according to ESA review	Dan Bretherton & Keith Haines

List of Contents

APPLICABLE AND REFERENCE DOCUMENTS

1. XGCE-DTEX-EOPS-SW-04-0001, Issue 1.3, Statement of Work
2. XGCE-DTEX-EOPS-SW-04-0001: Development of Algorithms for the Generation of a GOCE User Toolbox and an Absolute Dynamic Topography Product. Proposal in Response to ESA Request for Quotation. Issue 6, 14 December 2005.
3. GO-ID-HPF-GS-0041: Product Specification for L2 Products and Auxiliary Data Products, Issue 5.0, 28.08.2006
4. GO-TN-HPF-GS-0111: GOCE Standards. Issue: 2, 22 / 09 / 2006.
5. GO-MA-HPF-GS-0110: GOCE Level 2 Data Handbook, Issue 3, 22 / 09 / 2006.
6. GOCE User Toolbox : User Toolbox Requirement Document.
7. Toolbox Functionality and Algorithm Specification Document, Revision 2, February 2007

1. INTRODUCTION

The GOCE User Toolbox Specification (GUTS) project has been inter-comparing algorithms for translation of GOCE High level Processing Facility (HPF) data for use in the fields of oceanography and solid Earth science. The toolbox will be composed of a collection of executable tools, sets of auxiliary data and a user interface, with dedicated plug-in points to allow for easy integration of new algorithms, new data and the GOCE HPF product files. The architecture of the toolbox will allow any user to substitute their own auxiliary data or to integrate extra modules to extend the toolbox for individual applications. The ancillary data will include mean sea surface height (MSSH) data from altimetry and dynamic topography (DT) data from models or other oceanographic measurements, as well as a-priori non-GOCE geoid information. Good documentation will be necessary in order to facilitate communication between the various user communities. The software will be developed on an Open Source basis, to allow the flexible use and development of the toolbox by the science community.
The system specification and architectural design for GUTS include the following information:

- Definition of all system external interfaces and formats.
- Development of logical model of system functions.
- Complete specification of output content: data products, metadata, reports and logs.

Within this project the designs and specifications are limited to a logical description of the toolbox architecture and data flow, and do not include software design details. The purpose of this Work Package (WP4000) has been to provide the link between scientists in the GUTS team and the software engineers who will implement the toolbox. The emphasis of this report is on the design and operation of the main user interface and the specification of the main software building blocks. The information and the language used in this report is deliberately aimed at scientists rather than software engineers, because it has been important for the GUTS team as a whole to participate in the development of these design specifications during the GUTS project. For this reason, common software engineering terminology and diagramming techniques such as the Unified Modelling Language (UML) are not used. These will be more appropriate for use in the software architectural design document. Instead, the information here is presented in a way that closely matches the specification of the scientific work-flows and functional algorithms as they are presented in the WP3000 report. It is particularly important that the proposed mode of interaction with the toolbox is fully understood and agreed by scientists, because the users of the toolbox will themselves be scientists.

The GUT design is described in the following sections, which are written as though GUT already exists in order to simplify the language. The first release of GUT will not include the ability to use the error covariance matrix of the GOCE spherical harmonic coefficients in the calculation of geoid or MDT products. This is because the algorithms to do so have not been researched, and because the computational and storage requirements of this functionality are too high for most desktop computers. However, it is anticipated that once an agreed and tested algorithm is available for using the covariances efficiently, the covariance calculations required will, if feasible, be included in future versions of GUT. Provision for the error covariance matrix and associated error products is included in the design presented in this report.

2. OVERVIEW OF GUT DESIGN

This section gives an overview of the toolbox design, with justifications for the key design decisions. The core scientific algorithms are implemented as C and Fortran functions, mostly derived from existing software with the minimum amount of modification. All the existing software being considered for use in GUT at the time of writing is written in Fortran, but the capability for incorporating C will also be maintained. Both languages are appropriate for this purpose because they are free from software licensing costs and are widely used by scientists, maximising access for members of the scientific community involved in oceanography and solid earth studies. The C and Fortran scientific codes are managed using Python, which also provides the main user interface to GUT. The GUI provides access to some or all of the GUT functionality available through the Python interface. Detailed GUI design will be carried out during the implementation phase of GUT development.

There are several advantages to using Python for GUT implementation, as described below.
- Runs on many platforms, including Linux, Windows, MacOS and Solaris
- Provides a powerful, user friendly command interface with scripting ability
- Easily extensible by adding new C and Fortran based functionality
- Provides powerful mathematical capabilities [6]
- Considerable user interaction and programming capability will be required for GUT, more so than for previous ESA toolboxes. Python is highly suitable for such requirements

Python has many useful features but execution speed is not one of them. To achieve acceptable performance, all of the computationally expensive aspects of GUT processing will take place in C and Fortran routines rather than in Python. This approach has been used successfully in other scientific software toolboxes. Two popular Python-based toolboxes are described briefly below.
- VISAN [3,7], the GUI for the Basic Envisat Atmospheric Toolbox (BEAT). According to the developers, the BEAT project "aims to provide scientists with tools for ingesting, processing, and analysing atmospheric remote sensing data." VISAN is used for browsing and plotting data, and incorporates a Python command prompt providing access to the full range of BEAT functionality via the Python application program interface (API).
- Climate Data Analysis Tool (CDAT). CDAT [9] is used for analysis of gridded data in a variety of different formats. Typical operations include plotting, aggregation of multiple files, differencing data sets, calculating departures from a climatology, re-gridding and calculating the mean variance and covariance of a variable.

VISAN is described in its documentation as a "visualization and analysis application for atmospheric data". Access to the atmospheric data is provided by BEAT, and the analysis capability is provided largely by Python, which allows the user to write their own data analysis and processing scripts. In GUT, the emphasis is on the data processing rather than the data access and visualisation, though GUT will have to be able to do these things as well. In other words, GUT users will not be expected to write their own Python scripts for using the GOCE products, because these scripts will be an important part of GUT itself. Of course, users will be able to write scripts and programs in Python and other languages to augment GUT capabilities.

Python is not the only way to implement GUT, and it is possible that the important features of the design specified in this report could be implemented in another way. However, if alternative solutions are evaluated during the implementation phase it is important to consider the key features of the protective *user environment*, which facilitates good science and improves the traceability of results. The user environment consists of the following elements:
- Command interface
- Relationships between *logical data structures*, which include algorithm parameters, spatial and spectral fields, filter matrices and error covariance matrices
- Rules governing execution of scientific work-flows
- Session management
- Internal Data Store (IDS)

All the elements of the user environment are described in more detail in the following sections.

3. TOOLBOX COMPONENTS

3.1. Computationally Expensive Tasks

The parts of GUT that process large data structures and carry out mathematical calculations are written in Fortran and C. These computationally expensive tasks are described by a series of Processing Units, which are defined in Section 8 ("Logical Model of System Functionality"). The Processing Units include data pre-processing tasks and scientific calculations. Existing software will be re-used in the implementation of the Processing Units wherever possible. The scientific software being considered for re-use in GUT is described in the report from WP3000 [2]. All the software under consideration at present is written in Fortran, but the facility to incorporate C/C++ into GUT will be retained to ensure flexibility in the future. Software re-use is discussed in more detail in Section 4 ("Re-use of Existing Software").

3.2. Management of Internal data store

GUT maintains its own internal data store (IDS) on disk. The mathematical functions involved in the scientific work-flows take their input from, and write their output to the IDS. All the spatial data in the IDS is represented on the same set of spatial positions (e.g. a grid), and the reference system is common to the spatial and spectral representations of the data. Where spatial and spectral representations of the same field are both present (as occurs during execution of some of the scientific work-flows) they are consistent with each other. Data structures for the parameters, input data, intermediate stages and final output from scientific work-flow calculations are part of the IDS, which is separate from the external files where the data originates. This may seem like unnecessary duplication of spatial data, but the IDS is necessary because imported data may undergo transformations to make it consistent with data already present. Keeping the transformed fields permanently avoids lengthy transformation tasks being repeated each time a scientific work-flow is executed. The error covariance matrix of the SH coefficients of gravity potential is not part of the IDS; no transformations need to be performed on this large matrix, so there is no reason to create a copy in NetCDF format.

The supported formats for data import and export are discussed in Section 6 ("Specification of Input and Output"), which includes details of existing data handling software that may be suitable for re-use in GUT. Low level data import and export routines from some existing software packages can be re-used in GUT for dealing with a range of file formats.

3.3. Management of High Level Workflows

The high-level scientific work-flows are managed by Python software. Work-flows are executed by means of a sequence of calls to non-Python mathematical subroutines, which interact independently with the internal data store. These non-Python subroutines will be used to implement the Processing Units defined in Section 8 ("Logical Model of System Functionality"). Experts' scientific knowledge is encapsulated by the rules governing the sequences of events, and in the relationships between the logical data structures involved in the work-flows. Subroutines from other languages can be incorporated in the Python language by creating Python extensions [5]. It may not be practical to incorporate all existing software in this way. In cases where this is not practical, Python can also be used to manage binary executables in a way that is similar to a conventional shell scripting language. The Processing Units must be able to access data on the hard disk themselves, because it may not be possible to store all the data they need in memory simultaneously on all computers where GUT will be running. The default grid has 1/30 degree resolution, and some operations, such as grid differencing, have to deal with two gridded fields at the same time. Spatial filter matrices can also be very large.

3.4. Command Interface

Python is suitable as the basis for a command interface to GUT because it allows Python statements to be issued interactively at a command prompt. The command interface is described in Section 5.1 ("Command Interface").

3.5. Graphical Components

A comprehensive GUI can be built on top of the Python application using Python wrappers for wxWidgets [11], an Open Source suite of application programmers' tools that formed a key component of the BEAT and BRAT toolbox GUIs. The GUT visualisation capabilities should be based on VISAN or BRAT, both of which use the Visualisation Toolkit [13] (VTK) for producing images.

4. RE-USE OF EXISTING SOFTWARE

This section discusses how existing software will be incorporated into the Python application. The two main areas of GUT where existing software can be incorporated are data handling (mainly import and export) and scientific algorithms. Existing software for data handling is discussed in Section 6 ("Specification of Input and Output").

As discussed earlier, the computationally expensive tasks described by the Processing Units defined in Section 8 ("Logical Model of System Functionality") will be implemented as Fortran or C subroutines. Several programs containing Fortran code that can be re-used for this purpose have been identified in the WP3000 report. Details of the Fortran programs (if there are any) relevant to each Processing Unit can be found in the WP3000 report in the discussion of the corresponding Functional Algorithm. The numbering of the Processing Units is aligned with the numbering of the Functional Algorithms. WP3000 has established that there is re-usable Fortran code suitable for implementing many of the Processing Units, but there are still a lot of gaps that will need to be filled by new software written in C and Fortran.

Two different approaches to re-using software have been suggested for GUT:

1. Re-use large portions of existing programs or whole programs
2. Re-use of some of the *algorithms* from existing software, possibly taking copies of small subroutines or loops into GUT.

The second approach would involve re-writing all the scientific software in Fortran or C. This is the best solution from the software engineering perspective, and also allows greater flexibility for extending the capabilities of GUT in the future. Consideration of software at the level of small subroutines is a task that will take place during the software design stage of GUT implementation. Therefore, it is beyond the scope of this document and will not be discussed further. The rest of this section will outline a solution involving the re-use of large portions of existing software or whole programs.

Re-using large portions of existing programs or whole programs is a tempting approach because many of the re-usable Fortran programs identified in the WP3000 report perform tasks that fit in well with the tasks performed by individual Processing Units, particularly PU04. Some existing programs provide the functionality of more than one Processing Unit. All existing software should be incorporated into GUT as Python extensions that interact with the internal data store independently, as discussed earlier in Section 3.1 ("Computationally Expensive Tasks"). This will mean that all existing software will need to be modified to a certain extent. All the existing input-output routines will need to be re-written in order to exchange data with the internal data store instead of their own file formats. In cases where a section of an existing program is taken without self-contained input and output functionality, input and output functions will need to be written from scratch. Python code should replace any functionality contained in operating system shell scripts that accompany the existing software, and code modifications may be required during the process of creating Python extensions.

5. USER INTERFACES

5.1. Command Interface

5.1.1. Introduction

The GUT command interface is an element of the user *environment,* which contains the input, output and parameters associated with scientific work-flows. The basic usage procedure involves three main stages.

1. Set up the parameters governing the operation of a particular scientific work-flow
2. Import the necessary data
3. Enter a command to calculate the desired output product

This integrated user environment is a little bit like having your input files, output files and parameter files all in the same directory, but the GUT user environment is more sophisticated than this conventional approach. The most important feature of the GUT environment is that the input, output and algorithm parameters are consistent with each other at all times. This means that the input data files and algorithm parameters do not have to be specified in GUT commands, and all the output can be reproduced exactly without knowing the command that was entered by the user to initiate the calculation. In other words, as a GUT user you can always see where your results came from if the internal data store is retained. Other features of the GUT user interface are described below.

- Users do not interact directly with data and algorithm parameters. Instead, GUT functions called *methods* are used. This ensures that the data and parameters are kept consistent, as described above, and enables the user to be informed about the consequences of important changes they make. Methods are described later in this section.
- Data associated with intermediate stages are saved during execution of work-flows
- A path through a particular work-flow can be executed in full with a single command, or executed in stages with a series of commands.
- The user environment can be saved to disk, and loaded again at a later time.
- The amount of typing is minimised by the use of a name completion facility, which is analogous to name completion in Unix and Linux shell environments. The user can select from a list of all possible commands matching any partially entered command.
- Experts' scientific knowledge is encapsulated in the constraints placed on input data and parameters and the relationships between them.

The Python language can be used to provide a user interface with the above features. One of the unusual features of the language is that it allows statements to be entered interactively at a command prompt, in the same way as operating system shell languages such as the C Shell or the Bash Shell. This makes python suitable as a command interface for applications such as GUT. Using GUT is not like writing a program (or as little like writing a program as possible), but users must have a basic knowledge of programming to be able to understand the Python command syntax. In the following sections, no knowledge of *object oriented* programming methods is assumed, but some object oriented programming terminology is used where appropriate. In the remainder of this report the "`Courier New`" font is used to highlight elements of GUT commands.

5.1.2. Python language elements used in GUT

This section describes the main Python language elements in GUT. The high level scientific work-flows are controlled by manipulating Python language elements called *objects*. An object in GUT consists of the following things:

- *Attributes*, which describe the object and its contents
- *Data*, including input data, results from calculations and parameters for algorithms
- *Methods*. These are things that the object can do

An Object in Python is somewhat similar to a compound variable in a procedural programming language like C. A C *structure* is an example of a compound variable; it is a container for two or more variables, which may be instances of monotonic data types such as integers, arrays, strings or even other structures. The Python object extends this concept by allowing members of the compound variable to be *methods*, which are known as functions or subroutines in other programming languages. The user interacts with GUT by entering commands to execute methods. The methods are used to manipulate, or perform calculations involving the logical data structures belonging to the Objects. The way in which the user interacts with the Objects is described in the Appendix ("Using the command interface"). The five types of Object in GUT are described below.

Main Objects
The most important feature of GUT is called the *Main Object*. A Main Object is a container for everything associated with the scientific work-flows, including the input and output data, algorithm parameters and the algorithms themselves. The only reason why GUT users have to know about Main Objects is that there can be more than one of them present in the user environment during a user session. There are many advantages to this approach. For example, it makes it easy to compare the same output field calculated in different ways or on different grids. However, not all users will find this facility useful, and it can be safely ignored. It is sufficient to understand the following three things about Main Objects in GUT user sessions:

- There must be at least one Main Object.
- It must have a name given to it by the user.
- It can be saved to disk, and loaded again during a GUT session at a later date.

At the start of a GUT session the user must either *create* a new Main Object or *load* a saved one from disk. All the information necessary to repeat any results stored in the logical data structures in the Object is contained within the internal data store belonging to the Object. This allows the relevant calculations to be repeated exactly at a later date if necessary. At the end of a user session the Main Object and its internal data can be saved to disk as a whole, and data from individual fields can be exported to files. The attributes and data are consistent with each other at all times. In addition to its attributes and methods, a GUT Main Object contains four sub-objects, named `Spectral`, `Covar, Filter` and `Spatial.` The five object types represent logical groupings of data structures, parameters and functionality, designed to make the user interface easier to use. Generally speaking, the attributes and methods that belong to the Main Object are concerned with the user environment as a whole. The sub-objects are described below. The full specification of their logical data structures can be found in Section 7 ("Specification of Logical Data Structures").

Spatial Objects
A Spatial Object concerns data in geographical space. Data are presented to the user as a series of one dimensional (1-D) vectors. There are two position vectors (one for latitude values and one for longitude values) and vectors for each of the spatial fields involved in the scientific work-flows. Alternatively, the data can equally well be imagined as one big 2-D array or matrix, with each row being a record representing one position on the globe and each column representing a spatial field. This record structure is similar in concept to the GRAVSOFT file record described in the WP2000 report. It does not matter which view of the data is preferred because the user does not interact directly with the logical data structures in the Objects, which exist only to simplify the understanding of the GUT user environment. The actual data structures in the GUT internal data store are likely to be different to either of these conceptual views, but the user does not need to know about what goes on in there. In this report the spatial fields are described as being stored or located in vectors, to make it easier to understand the link between Spatial Object vectors and Spectral Object arrays of the same name. However, the Spatial Object vectors could equally well be described as fields in the GRAVSOFT-like record structure described above.

Spectral Objects
Spectral Objects deal with data in the spectral domain. The Spectral Object contains a set of SH coefficients representing the gravity potential (GOCE L2 product by default) and the spectral representation of all the spatial fields defined in the Spatial Object. The various sets of SH coefficients appear to the user as separate arrays within the Spectral Object. Most of these arrays will not be populated with data during a typical GUT

session, because they are not all involved with all the scientific work-flows. The names of the SH arrays are the same as the names of the 1-D vectors in the Spatial Object.

Covar Objects

A Covar Object deals with error covariances. Error covariance matrices can be generated for the geoid height, deflection and gravity anomaly output fields in the Spatial Object. The Covar Object is used for specifying a list of geographical positions to be used for error covariance calculations, and for storing the resulting error covariance matrices. Omission error covariance information is also produced in the form of the evaluation of an isotropic, homogenous error function over a range of distances.

Filter Objects

A Filter Object concerns the filtering of spatial and spectral data. Its local data structures include the filter method specifier, the filter scale parameter and the filter matrix, which is populated the first time a particular filter is used. There are five types of spatial filter and two types of spectral filter methods provided. Only one type of filter can be used during any work-flow calculation, i.e. once the filter method has been chosen and its parameters defined, it remains the same for all the calculations involved in the work-flow, right through to the final work-flow product. If a spatial filter type is selected, the filter matrix is valid for the current grid defined in the Spatial Object and will be removed if that grid is changed, subject to user confirmation. Similarly, if one of the spectral filter types is selected the filter matrix is valid only for the current maximum degree and order of the surface SH coefficients in the Spectral Object.

A Python command in GUT is constructed from several separate components. Most GUT methods are specified with respect to a particular, named Main Object, to allow for the possibility of having more than one Main Object present in a GUT user session. The only exceptions are the methods that are used to create a new Main Object, or to load a saved Main Object from disk. These methods are specified with respect to a Python language element known as a *package*, which could be described as a *module* in some other programming languages. The components that make up a command are described below.

- Name of Python package
- Name of Main Object. This is chosen by the user when creating a new Main Object.
- Name of sub-object, if any. There are four sub-objects in a Main Object, whose fixed names are Spatial, Spectral, Covar and Filter
- Name of method
- Parameters. If a method requires extra information to control its operation, these can be supplied in the form of comma separated parameters inside the parentheses that form part of the method name. If no parameters are supplied, the default action of the method is executed

There are three types of Python command in GUT, depending on which of the above components are used. The formats of these three types of command are described below. Each format description is shown with N parameters, where N can be any number including zero. In other words, the format of commands where no parameters are given is not explicitly shown.

- PackageName.MethodName(Parameter1, Parameter2, …, ParameterN)
- MainObjectName.MethodName(Parameter1, Parameter2, …, ParameterN)
- MainObjectName.SubObjectName.MethodName(Parameter1, Parameter2, …, ParameterN)

A number of pre-defined constants are available to GUT users for use as parameters in commands. In addition to GUT constants, one or more parameters can be specified as `None`, Python's null value. This indicates to GUT that the parameter in question should be ignored. This facility is useful when defining a new grid or reference ellipsoid, because different ways of specifying these involve different numbers of parameters. For example, when defining an irregular grid it is necessary to specify `None` as the value for grid cell width and height because these parameters are undefined for an irregular grid.

In this report, complete method names are always written with trailing empty parentheses, `View()` for example. Users will be able to issue GUT commands interactively at a command prompt, one at a time. Alternatively, command sequences can be saved in a text file and run like an executable program, inside or outside the GUT environment. These command sequences are known as *scripts*. GUT itself will be made up primarily of a series of Python scripts, plus a set of components written in other languages. Further explanation of the use of the command interface is given in the Appendix, with examples of commands relating to the execution of the scientific work-flows.

5.1.3. Full specification of command interface

This section specifies the Python methods used to interact with the logical data structures and control the execution of the scientific work-flows. The methods associated with the GUT package and each type of Object are specified below. The GUT package is not associated with any particular Main Object; its methods are used to bring Main Objects into the user environment by creating a new Main Object or loading a saved Main Object from a directory on disk. When a new Main Object is created, its internal data store is populated by the SH coefficients of the gravity potential and the default MSSH data set, which defines the default grid and reference system. All the attributes are set to their default values. A new Main Object is immediately ready for use with work-flows 1a and 1b, but other work-flows require additional data to be imported first.

In the following tables, the "Example of Usage" column shows one or more typical examples of Python commands involving each method. Pre-defined GUT constants are written in upper case letters. Methods that normally expect parameters in the parentheses can also be used with no parameters; this results in the default behaviour of the method. In the tables below, "default behaviour" of a method means the outcome if no parameters are specified.

GUT Package

Name	Example of Usage	Description
`New`	`New("/home/GUT/main_ob1")`	Creates new Main Object with default attributes and data in directory main_ob1
`Load`	`Load("/home/GUT/main_ob1")`	Loads a previously saved Main Object and its internal data store from directory “main_ob1”

Main Object

Most of the Main Object methods are for performing calculations relating to the scientific work-flows defined in WP3000. The names of these methods are derived from the names of the logical data structures provided for storing the intermediate stages and final output from the work-flows. The output from the work-flows is always provided in geographical space, and the results can be found in Spatial Object vectors. Every work-flow product has a unique name that identifies the method that was used to produce it. In the case of work-flows involving intermediate stages in geographical space, the intermediate stages can be found in Spatial Object vectors with names that include an additional identifier beginning with “-Inter”. In the case of work-flows involving data in spectral space, intermediate stages and some final output can also be found in arrays located in the Spectral Object. There is a `Calc` method corresponding to every field in the Spatial Object that is not an input field. Input fields are identified with names beginning with “Input”. It is important to note that all the `Calc` methods in the Main Object refer to fields in the Spatial Object, even though intermediate stages and final output from some of the work-flows also ends up in the Spectral Object. `Calc` methods for "-Inter" fields are provided to allow the user to experiment with one particular part of a work-flow, without incurring the computational cost of going right through to the end product each time. Some `Calc` methods result in output to more than one field. `CalcVelocity()` and `CalcDeflection()` calculate both the northward and eastward components of geostrophic current and gravity field deflection. The methods for calculating error covariances each produce a covariance matrix for a pair of points and the evaluation of an error function for a range of distances.

The table below specifies the methods associated with the Main Object. Most of the methods shown in the table refer to the Main Object attributes. The `Calc` methods for intermediate stages and final output from the scientific work-flows are not listed, to avoid repeating all the data structure names defined in Section 7 ("Specification of Logical Data Structures"). The only `Calc` methods that need mentioning specifically are those involved with Remove-Restore work-flows 4a, 4b, 4c and 4d. The 4a and 4c work-flows are only valid if a spatial filter is specified in the Filter Object, and the 4b and 4d work-flows are only valid if a spectral filter is specified. Attempting to perform calculations involving these work-flows when the wrong type of filter is specified results in an error message.

Name	**Example of Usage**	**Description**
`Annotation`	`Annotation("Some text")`	Defines ASCII string of user's comments. Default behaviour: Display string
`ImportAnnotation`	`ImportAnnotation("file.txt")`	Imports annotations from a text file, including new-line characters. Default behaviour: Try to import from default path.
`Info`	`Info("Ellips")`	Displays the value of the specified Main Object attribute Default: Displays values of all attributes
`Ellips`	`Ellips(GRS80)`	Changes all ellipsoid parameters. Parameters can be specified explicitly by providing them in the following order: $GM, a, \gamma a, f, J_2, \omega$ One or more of them can be None (undefined). Alternatively, one of three constants can be given as a single argument: GRS80, TOPEX or ENVISAT Default Behaviour: The default ellipsoid is used (TOPEX). Adapts internal data dependent on these values if possible and discards data that can not be adapted
`TideSys`	`TideSys(TIDE_FREE)`	Changes value of attribute. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`Save`	`Save("main_path")`	Saves Main Object attributes and data in directory specified by user
`Load`	`Load("main_path")`	Loads a previously saved Main Object from a directory specified by user.
`Calc`	`Calc()`	Starts user dialog session in which user selects desired calculation from a list.

Spatial Object

In addition to methods for changing the values of attributes, the Spatial Object also provides sets of methods with names beginning with "`Convert`", "`Filter`", "`View`", "`Import`", "`Export`" and "`Delete`". These sets of methods are described below.

- The `Filter` methods are used for filtering spatial fields according to the filter parameters defined in the Filter Object. These methods are not used for controlling any of the scientific work-flows, and they are not shown in the table below. The `Filter()` method launches a wizard that allows the user to select the desired field from a numbered list.
- The `View` methods are used to display 2D plots of the spatial fields, and the `View()` method launches a wizard that allows the user to select the field and parameters controlling the type of plot required.

- The `Convert` methods are used for converting spatial data to spectral space. The output of a Spatial Object `Convert` method is an array with the same name as the converted Spatial Object vector. For example, `ConvertMDTC_B-Spectral()` converts the gridded data in the MDTC_B-Spectral vector to SH coefficients by SH analysis, the result of this conversion going into Spectral Object array MDTC_B-Spectral. The maximum degree and order of coefficients used for SH analysis is governed by Spectral Object attribute MaxDegOrdSurface. The `Convert` methods are not used for controlling the scientific work-flows. The names of the `Convert` methods are derived from the names of the Spatial Object vectors, which are specified in Section 7 ("Specification of Logical Data Structures"). The `Convert` methods are not specified in this table, except for the `Convert()` method (with no reference to any particular spatial field), which launches a command based wizard.
- There is an `Import` method for the `Annotation` attribute and every spatial field vector. These methods are used for importing data from external data sources into the internal data store of the Main Object. These methods are not shown in the table below, to avoid repeating information from Section 7. The `Import()` method launches a wizard. Importing data is discussed in more detail in Section 6 ("Specification of Input and Output").
- There is an `Export` method and a `Delete` method corresponding to every spatial field in the Spatial Object. These methods are not shown in the table, to avoid repeating information. The `Export` methods are used to write spatial data from the internal data store to external files, and the `Export()` method launches a wizard. Data export is described in more detail in Section 6 ("Specification of Input and Output"). The `Delete` methods are provided to allow the user to manage the internal data store, to allow a Main Object to be used for calculations involving more than one work-flow without retaining all the input, output and intermediate stages from previous work-flow calculations. Deleting unnecessary fields may be desirable in situations where disk space is limited. The `Delete()` method launches a field deletion wizard

Name	Example of Usage	Description
`Annotation`	`Annotation("Some text")`	Defines ASCII string of user's comments. Default behaviour: Display string
`ImportAnnotation`	`ImportAnnotation("file.txt")`	Imports annotations from a text file, including new-line characters. Default behaviour: Try to import from default path.
`Info`	`Info("Ellips")`	Displays the value of the specified attribute Default: Displays values of all attributes
`GridCalc`	`GridCalc(POINT)`	Changes value of attribute. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`DefineGrid`	`DefineGrid(lat1, lon1,….)`	Specifies grid parameters. Adapts internal data dependent on these values if possible and discards data that can not be adapted
`New`	`New()`	Overwrites Spatial Object with default attributes and data
`Blank`	`Blank()`	Overwrites Spatial Object with a blank object containing no data, with all attributes set to `None`
`MaxDegOrdPotential`	`MaxDegOrdPotential(200)`	Changes value of `MaxDegOrdPotential.` User specifies DO value. Adapts internal data dependent on this value if possible and discards data that can not be adapted

`MaxDegOrdSurface`	`MaxDegOrdSurface(200)`	Changes value of `MaxDegOrdSurface.` User specifies DO value. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxResKMPotential`	`MaxResKMPotential(100)`	Changes value of `MaxDegOrdPotential.` User specifies resolution in Km. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxResDegreesPotential`	`MaxResDegreesPotential(0.5)`	Changes value of `MaxDegOrdPotential.` User specifies resolution in degrees. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxResKMSurface`	`MaxResKMSurface(100)`	Changes value of `MaxDegOrdSurface.` User specifies resolution in KM. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxResDegreesSurface`	`MaxResDegreesSurface(0.5)`	Changes value of `MaxDegOrdSurface.` User specifies resolution in degrees. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`Filter`	`Filter()`	Launches a wizard for filtering a spatial field
`View`	`View()`	Launches a wizard for producing a 1D or 2D plot of a spatial field
`Convert`	`Convert()`	Launches wizard for converting to spectral space
`Import`	`Import()`	Launches data import wizard
`Export`	`Export()`	Launches data export wizard
`Delete`	`Delete()`	Launches a wizard for deleting unused spatial fields in the internal data store

Spectral Object

Any of the 2D arrays of SH coefficients in the Spectral Object can be converted to geographical space using a `Convert` method. The names of the Convert methods are derived from the names of the 2D arrays in the Spatial Object. For example, `ConvertMDTC_B-Spectral()` converts the SH coefficients in the `MDTC_B-Spectral` array to geographical space by SH synthesis, the result of this conversion going into Spatial Object vector `MDTC_B-Spectral`. The maximum degree and order of coefficients used for SH synthesis is governed by Spatial Object attribute `MaxDegOrdSurface`.

There is an `Import` method corresponding to the Annotation attribute, and every array of SH coefficients. This allows the user to supply their own data in spectral form as well as spatial data. The `Import()` method launched a wizard. A set of `Export` methods are provided to allow the user to export data from any of the spectral fields. There is an `Export` method corresponding to all the Spectral Object arrays except for InputPotential. The `Export()` method launches a wizard. Data import and export are discussed in more detail in Section 6 ("Specification of Input and Output"). A set of `Delete` methods are provided to allow the user to delete unused spectral fields in order to save disk space. The `Delete()` method launches a wizard. To avoid repeating all of the field names defined in Section 7 (“Specification of Logical Data Structures”), the `Convert, Import, Export` and `Delete` methods are not shown in the table below.

`Convert`	`Convert()`	Launches wizard for conversion of data to geographical space
`Annotation`	`Annotation("Some text")`	Defines ASCII string of user's comments. Default behaviour: Display string
`ImportAnnotation`	`ImportAnnotation("file.txt")`	Imports annotations from a text file, including new-line characters. Default behaviour: Try to import from default path.
`Info`	`Info("MaxDegOrderSurface")`	Displays the value of the specified attribute Default: Displays values of all attributes
`MaxDegOrdPotential`	`MaxDegOrdPotential(200)`	Changes value of `MaxDegOrdPotential.` User specifies DO value. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxDegOrdSurface`	`MaxDegOrdSurface(200)`	Changes value of `MaxDegOrdSurface.` User specifies DO value. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxResKMPotential`	`MaxResKMPotential(100)`	Changes value of `MaxDegOrdPotential.` User specifies resolution in Km. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxResDegreesPotential`	`MaxResDegreesPotential(0.5)`	Changes value of `MaxDegOrdPotential.` User specifies resolution in degrees. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxResKMSurface`	`MaxResKMSurface(100)`	Changes value of `MaxDegOrdSurface.` User specifies resolution in Km. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`MaxResDegreesSurface`	`MaxResDegreesSurface(0.5)`	Changes value of `MaxDegOrdSurface.` User specifies resolution in degrees. Adapts internal data dependent on this value if possible and discards data that can not be adapted
`Import`	`Import()`	Launches data import wizard
`Export`	`Export()`	Launches data export wizard
`Delete`	`Delete()`	Launches a wizard for deleting unused spectral fields in the internal data store

Covar Object

In addition to the methods defined in the table below, the Covar Object also provides a set of `Export` methods for exporting data from any of the data structures in the Object. The names of these `Export` methods are derived from the names of the data structures, which are defined in Section 7 ("Specification of Logical Data Structures"). The `Export()` method launches a wizard.

Name	**Example of Usage**	**Description**
`Annotation`	`Annotation("Some text")`	Defines ASCII string of user's comments. Default behaviour: Display string
`ImportAnnotation`	`ImportAnnotation("file.txt")`	Imports annotations from a text file, including new-line characters. Default behaviour: Try to import from default path.
`Info`	`Info("InputOmFunctionStart_1")`	Displays the value of the specified attribute Default: Displays values of all attributes
`AddPos`	`AddPos(lat, lon)`	Adds a position to the `Lon` and `Lat` vectors, used for calculation of error covariances
`DelPos`	`AddPos(lat, lon)`	Deletes a spatial position from `Lon` and `Lat`
`Export`	`Export()`	Launches a wizard
`ErrorFunctionStart`		Changes value of attribute
`ErrorFunctionEnd`		Changes value of attribute
`ErrorFunctionInterval`		Changes value of attribute
`InputPotentialCovComPath`	`InputPotentialCovComPath( "/data/goce/covariances.dat")`	Allows user to specify path to error covariance matrix

Filter Object

The following methods are provided for changing Filter Object attributes.

Name	**Example of Usage**	**Description**
`Annotation`	`Annotation("Some text")`	Defines ASCII string of user's comments. Default behaviour: Display string
`ImportAnnotation`	`ImportAnnotation("file.txt")`	Imports annotations from a text file, including new-line characters. Default behaviour: Try to import from default path.
`Info`	`Info("Scale")`	Displays the value of the specified attribute Default: Displays values of all attributes
`Type`	Type(SPATIAL_CAP)	Changes value of attribute. Discards internal data dependent on this value
`Scale`		Changes value of attribute. Discards internal data dependent on this value

5.2. Graphical User Interface (GUI)

Detailed design of the GUI will not be performed as part of the GUTS project, but it is important to establish how it might be implemented and what its capabilities could be. Ideally, the GUT GUI would allow the user to do everything that can be done using the Python command interface, but it would require a lot of effort to design a clear, well laid out, user-friendly GUI that was capable of performing all these functions in accordance with the high level workflows in the WP3000 report [2]. One possible approach would be to adapt the BRAT GUI, which provides the following facilities for altimetry data.

- Data extraction and viewing
- Data processing
- Calculating statistics
- Re-gridding
- Computing basic altimeter parameters
- Calendar and date conversion
- Format conversion

The extra functionality required for GUT could be incorporated by allowing the user to select and then manipulate GUT Objects by editing attributes and executing methods. It is possible to imagine a GUI that allows the user to select the desired object from a list, and then select from a list of the attributes and methods belonging to that object. The method naming convention described earlier would be beneficial in this respect, for the same reason that it will make things easier for command interface users. It might be sensible to restrict GUI users to having only one Main Object in a user session. To avoid the GUT GUI becoming too cluttered and difficult to use, it may be better to leave the calculations involving the high level workflows out of the GUI altogether. It has been suggested that this part of GUT should be a completely separate application, accessible only through Python. Some desirable, non-essential GUI features have also been discussed. These include the management of data downloads from servers selected from a list, and the management of shared work-spaces for GUT objects. A visual display linking the toolbox commands with the scientific work-flow diagrams would be very instructive for novice users.

5.3. Application Program Interface (API)

The API enables programmers to build elements of GUT into their own software applications. The implementation of GUT will be accompanied by detailed documentation of the software architecture, including full specification of the Python language elements that are not normally exposed to the user. This will allow Python programmers to utilise any part of GUT, from the low level algorithms to the high level workflows.

Ideally, the API would also provide access to GUT functionality from other programming languages and proprietary scientific scripting languages such as IDL and MATLAB. However, for the first release of GUT at least, the non-Python API will be restricted to the software used to implement the Processing Units defined in Section 8 ("Logical Model of Sytem Functionality"). The non-Python API will consist of a set of Fortran subroutines that each take input from a NetCDF file and send output to a NetCDF file. Functions for controlling the scientific work-flows will not be part of the non-Python API.

6. SPECIFICATION OF INPUT AND OUTPUT

6.1. Introduction

When a Main Object is created during a GUT user session, its internal data files are created in a directory specified by the user. Later, the `Save` method can be used to save the Main Object itself in that directory. The Main Object, with all its attributes and sub-objects is saved by Python in a process known as *pickling*. The saved Main Object then has a permanent presence on disk, in the directory containing the pickled Python objects and the internal data files. This Main Object can be loaded into another GUT session at a later date by using the `Load` method. Note that the logical data structures in the GUT user environment can not be loaded and saved individually by the user; the internal data store belonging to a Main Object is loaded and saved as a whole. The rest of this section concerns the `Import` and `Export` methods associated with the Spatial, Spectral and Covar objects. These methods deal with *external* data files, which are not part of the internal data store belonging to each Main Object. The `Import` methods read data from external files into the internal data store, and the `Export` methods write data from the internal store to external files.

6.2. Data exports

The user can export data from the Spatial, Spectral and Covar objects, including intermediate stages of scientific work-flows in spatial or spectral form. There is an `Export` method for every vector and array associated with these objects, except for the SH coefficients representing the gravity potential and their error covariance matrix. `Export` method names are identified using the names of the data structures specified in Section 7. The default exported file names are also derived from these data structure names. The default file name is used if the user does not specify a file name. The following file formats can be used for data exports from GUT:

- NetCDF. This is a binary format that is used for each Main Object's internal data store. This format is normally associated with gridded data or point lists, but can be used to store any collection of multidimensional arrays. Therefore, all data structures in the Spatial, Spectral and Covar objects can be exported in this format. Open Source NetCDF software libraries are available for Python, Fortran and C, plus other languages not used in GUT.
- GRAVSOFT. This is an ASCII format, which is described in the WP2000 report. GRAVSOFT is suitable for export of real-space data on grids and in point lists, and surface SH coefficients. Components of the GRAVSOFT software, which is written in Fortran, can be re-used in GUT for the data export routines. A separate metadata file in an ASCII format is required for exports in GRAVSOFT format. The metadata file has the same file name stem as the GRAVSOFT file.

The format of GUT command names is described in Section 5.1 ("Command Interface"). `Export` method names are derived from the data structure names, as described in that section. Therefore, the `Export` method name in the command specifies the data structure to be exported. Export from input fields is allowed, to enable GUT to be used for the simple task of grid adaptation, where external data on a different grid can be imported, adapted to the current Spatial Object grid and then exported. The basic format of an export command is given below.

MainObjectName.SubObjectName.ExportMethodName(FilenameString, FieldIdentifier, FormatSpecifier)

The FormatSpecifier parameter can be either `"NETCDF"` or `"GRAVSOFT"`. These specifiers are Python constants, as described in Section 5.1. An error results if an `Export` command specifies a file format that is not supported for the data structure in question. The following types of data export are supported.

NetCDF and GRAVSOFT formats are suitable for the export of spatial field data from the Spatial Object. Fields can only be exported one at a time, but the same output file can be specified in more than one Export command. GUT exports from the Spatial Object contain the following metadata:

- Attributes from Spatial Object of origin
- Attributes from Main Object of origin
- Data structure identifier (see Section 7 - "Specification of Logical Data Structures"), from which the user can infer details of the origin of the data, such as what type of calculation was used to produce the results.

NetCDF files can store metadata themselves, but the GRAVSOFT files require a separate metadata file to store this information.

Surface SH coefficients from the Spectral Object can be exported in NetCDF or GRAVSOFT format. One GRAVSOFT file contains one array of SH coefficients, but a NetCDF file can contain more than one array. The SH coefficients of the gravity potential can not be exported from the Spectral Object. It does not make sense to provide export facilities for this array because GUT does not modify them in any way. Exports of SH coefficients contain the following metadata:

- Attributes from Spectral Object or origin
- Attributes from Main Object of origin

- Data structure identifier (see Section 7 - "Specification of Logical Data Structures"), from which the user can infer details of the origin of the data, such as what type of calculation was used to produce the results.

A separate metadata file is required for each exported GRAVSOFT file.

Error covariance matrices and covariance function evaluation vectors from the Covar Object can be exported only in NetCDF format. Each output file contains one error covariance matrix or function evaluation vector. These exports include the following metadata:

- Attributes from Covar Object of origin
- Attributes from Main Object of origin
- Data structure identifier (see Section 7 - "Specification of Logical Data Structures"), from which the user can infer details of the origin of the data, such as what type of calculation was used to produce the results.

6.3. Data imports

The subject of importing data into GUT is more complicated than exporting for several reasons:

- The importing routines in GUT deal with adapting data to the grid and/or reference system specified by the attributes in the Main Object and Spatial Object.
- The importing routines need to be supplied with certain parameters that describe the imported data, such as the grid and reference system specifications
- More file formats are supported for importing than for exporting
- There are restrictions on which data structures can receive imported data

These issues are discussed in more detail below.

The following file formats are supported for data import to Spatial Object vectors.

- ICGEM
- NetCDF
- GRAVSOFT
- AVISO. Some AVISO data is distributed in NetCDF format but some data sets are only available in Geophysical Data Record (GDR) format.
- GMT. This is the format used by the General Mapping Tool (GMT). The default land surface height data set will be supplied in this format.

The following file formats are supported for data import to Spectral Object SH coefficient arrays (potential and surface)

- ICGEM
- GRAVSOFT
- NetCDF

The default data sets to be supplied with GUT are specified in WP3000. It does not matter what formats these are supplied in from the point of view of the software design, as long as the formats are supported by GUT.

Data can be imported into any spatial or spectral field, but most of the time it will only be necessary for the user to import into fields used for input to the scientific work-flows. If data imported in spectral form is required in spatial form by one of the work-flows, the user must convert the data to spatial form before using the work-flow. `Convert` methods in the Spectral Object are provided for this purpose. The tables in Section 7 ("Specification of Logical Data Structures") specify the default data set associated with each of the "Input" fields in the Spatial and Spectral Objects. There is an `Import` method associated with each spatial and spectral data structure specified in Section 7 (“Specification of Logical Data Structures”). Additionally, all object types in GUT have a `ImportAnnotation()` method for importing ASCII text into the Annotation attribute. There is no Import method for the error covariance matrix of the GOCE coefficients, because this matrix is not part of the Internal Data Store. The name of each `Import` method is derived from the name of the data structure in question. For example, the

`ImportInputGeoidHeight()` method imports data into the InputGeoidHeight field. The format of an `Import` method command is described below:

MainObjectName.SubObjectName.ImportMethodName(FilenameString, FieldIdentifier, FormatSpecifier)

The FieldIdentifier parameter is for specifying the field or data structure of the required data in the external file. The FormatSpecifier parameter can be either `NETCDF, GMT, AVISO, ICGEM` or `GRAVSOFT`. These specifiers are Python constants, as described in Section 5.1 ("Command Interface").

The default MSSH data set defines the default grid and reference system. When a new Main Object is created, two `Import` methods are executed automatically: `Spatial.ImportInputMSSH()` and `Spectral.ImportPotential().` These import data from the default MSSH data set and the default gravity potential coefficients, which establishes the default grid and reference system and readies the new Main Object for calculations involving work-flows1a and 1b. The user must execute other `Import` methods before using any of the other work-flows.

All the default data sets are supplied with GUT-specific metadata, as described in the previous section ("Data Exports"). The ICGEM files have some information in the header but an additional metadata file is required for GUT-specific information. The NetCDF files store all the required metadata internally. When GUT-specific metadata are supplied with external data files, the Import methods know how to find the important parameters describing the data being imported, and can associate them with the attributes of the GUT Objects. The `Import` methods always look for GUT-specific information in the metadata. If found, the import procedure takes place automatically, the values of the relevant attributes being read from the file header or metadata file. This feature is useful for importing data from the default data sets and for transferring data between two GUT work-flows, where the output from one work-flow is exported, then imported to become the input for another work-flow. If the metadata are not found, as is likely to be the case when importing data from external sources, the user is prompted to enter the relevant parameters describing the imported data, including the grid and reference system specification parameters.

The low level data ingestion routines in the GRAVSOFT software can be re-used in GUT for importing data in GRAVSOFT format. The low level data ingestion routines in BRAT might be useful for GUT imports of altimetry data in NetCDF and other formats. Software written in Perl that is currently being developed by the GOCE HPF team may be suitable for re-use in GUT for importing ICGEM files. Tools for handling GDR (AVISO) data are available from AVISO.

The rules for adapting imported spatial data to a different grid specification are described below. In general, imported data is always adapted to the grid described by the Spatial Object attributes. There are several possible scenarios that need particular mention.

- *Scenario*: The imported data is on a grid and the Spatial Object attributes specify a grid. *Action*: Imported data is adapted to the grid specified by the Spatial Object attributes
- *Scenario*: The imported data is on a grid and the Spatial Object attributes specify a point list. *Action*: Imported data is interpolated to the point list specified by the Spatial Object
- *Scenario*: The imported data is specified for a point list and the Spatial Object attributes specify a grid. *Action*: The import procedure can not be performed according to the general rule, but the user is given the option of interpolating all the data in the Spatial Object to the point list specified in the imported data file.
- *Scenario*: The imported data and the Spatial Object attributes specify two, overlapping regional grids. *Action*: Only data in the overlapping region is taken from the file, and existing data outside the overlapping region is discarded from the Spatial Object, subject to user confirmation. The new Spatial Object grid covers the overlapping region only.
- *Scenario*: The imported data and the Spatial Object attributes specify two regional grids that do not overlap. *Action*: The import operation is not allowed, and an error is reported to the user.

- *Scenario*: The imported data and Spatial Object attributes both specify point lists. *Action*: The import operation is not allowed, and an error is reported to the user.

6.4. Reports and logs

A record of GUT user sessions is stored by each Main Object. The Python methods that form part of the command interface record their use in a log file, along with any warnings and errors reported to the user. The log file is part of the internal data store belonging to the Main Object.

7. SPECIFICATION OF LOGICAL DATA STRUCTURES

This section specifies and labels all the logical data structures that form the conceptual view of the data and parameters presented to the user. There are physical data structures in the internal data store files corresponding to each logical data structure. The data structures are grouped according to the Python objects used to interact with them in the command interface described earlier. These data structures form the basis of the logical model of system functionality described in Section 8. Pre-defined Python constants are written in upper case letters. Unspecified values are shown using Python's null value, `None`.

Main Object

Attributes

Name	Type	Possible values	Default value	Description
`Annotation`	String	ASCII chars, including '\n'	"Default "	Description of the object
`Ellips_GM`	Float		GOCE default ($3.986004415 \cdot 10^{14}$)	*GM*
`Ellips_a`	Float		TOPEX value (6378136.3)	a
`Ellips_gamma_a`	Float		None	γa
`Ellips_f`	Float		TOPEX value (1/ 298.257)	f
`Ellips_J2`	Float		None	J_2
`Ellips_omega`	Float		None	ω
`TideSys`	Flag	Python constants: `TIDE_FREE`, `ZERO_TIDE`, `MEAN_TIDE`	As for GOCE L2 products	Tide system
`GeodeticCalc`	Flag	Python constants: `HEIGHT`, `ANOMALY or DEFLECTION`	Python constant: `HEIGHT`	Controls which geodetic quantities are calculated in work-flows 1a and 1b. Set by choice of `Calc` method by user

Data

Name	Type	Description
Spatial	Python object	Data in the spatial domain
Spectral	Python object	Data in the spectral domain
Covar	Python object	Data associated with error covariance calculations for geodetic quantities
Filter	Python	Parameters associated with the filtering algorithms

Spatial Object

Attributes

Name	Type	Possible values	Default value	Description
`GridType`	Flag	Python constants: `REGULAR`, `UNSTRUCTURED` or `LIST`	Python constant: `REGULAR`	Determines whether or not the Spatial Object holds a regular or unstructured grid or a point list
`GridCalc`	Flag	Python constants: `POINT` or `AVERAGE`	Python constant: `POINT`	Specifies whether values at grid points represent only those points or area averages for the grid cells. Area averages are only allowed if the Spatial Object holds a grid (as opposed to a list of points). Area average calculations are only allowed for geodeditic output fields. Executing `Calc` methods for other fields while area averages are specified results in an error
`MaxDegOrdPotential`	Integer	In range 1 to `Spectral.MaxDegOrdPotential`	`Spectral.MaxDegOrdPotential`	Maximum degree and order for SH synthesis from SH coefficients of gravity potential
`MaxDegOrdSurface`	Integer	In range 1 to `Spectral.MaxDegOrdSurface`	`Spectral.MaxDegOrdPotential`	Maximum degree and order for SH synthesis from surface SH coefficients
`LatMin`	Integer	In range -90 to 90	-80	Grid starting latitude (degrees North)
`LatMax`	Integer	In range -90 to 90	82	Grid ending latitude (degrees North)
`LonMin`	Integer	In range -180 to 180	-180	Grid starting longitude (degrees East)
`LonMax`	Integer	In range -180 to 180	180	Grid ending longitude (degrees East)
`LatCell`	Float		1/30	Regular grid cell height (degrees)
`LonCell`	Float		1/30	Regular grid cell width (degrees)
`InputMSSH_Start`	Integer	4-digit year	1993	Start date for mean

`InputMSSH_End`	Integer	4-digit year	1999	End date for mean
`InputMDT_Start`	Integer	4-digit year	According to default data set	Start date for mean
`InputMDT_End`	Integer	4-digit year	According to default data set	End date for mean
`Annotation`	String	ASCII chars, including '\n'	"Default "	Description of the object

Data

Each spatial field vector has a unique name. There are three categories of vector: input, intermediate and output. Vectors are assigned to one of these categories depending on whether they are an input to a scientific work-flow, a final output or an intermediate stage. Input vector names all begin with "Input". The output vectors are the official GUT work-flow products. They each represent the final output from a scientific work-flow, and are highlighted in bold type in the table below. Vectors storing intermediate stages in a work-flow have names that begin with the name of the final output, plus an extra identifier starting with "-Inter". For example, the final output of Work-flow 4a is stored in a vector called MDTC-A_Spatial, and the two intermediate stages have vectors called MDTC_A-Spatial-InterAprioriMDT and MDTC_A-Spatial- InterMDTCorrection. The names of these data structures may need to be reviewed in the context of their use in the command interface described in Section 5.1 ("Command Interface"), because many of the GUT method names are derived from the names of the data structures they act on. This leads to some long and cumbersome method names such as `ExportMDTC_A-Spatial-InterMDTCorrection()`, although the name completion facility in Python would minimise the amount of typing that would be required in order to enter such a command. The idea of using Work-flow Objects rather than a single Main Object, as discussed in the Appendix ("Using the Command Interface"), may be an appropriate solution to the problem of rationalising the method names used in commands.

Filtering takes place during the execution of some of the scientific work-flows. GUT work-flow products can either be filtered or unfiltered, depending on their position in the work-flow. These products must be clearly separate from the results of "manual" filtering operations, which can be initiated by the use of the Spatial Object `Filter` methods. The user can manually filter any of the input, intermediate or output fields. To avoid confusion with GUT work-flow products, this manual filtering operation results in a new field with the same name plus a preceding "Filtered_". These fields are not shown here. They do not play a part in any of the work-flows, and although they can be exported like any of the other fields, they are not GUT work-flow products.

Type	Name	Description
Vector	Lat	Latitude in degrees Default: Positions in grid of default InputMSSH
Vector	Lon	Longitude in degrees Default: Positions in grid of default InputMSSH
Vector	InputPotential	Gravity potential Default: None. SH coefficients are used in work-flows
Vector	InputLandHeight	Height of land above reference ellipsoid Default: ETOPO2v2 digital elevation model
Vector	InputGeoidHeight	Geoid height Default: EGM_GEO_2
Vector	InputGravityAnomaly	Not part of any work-flow. Import facility provided just to allow grid adaptation followed by export. This facility provides GUT_002 Default: EGM_GAN_2.
Vector	InputEWDeflection	Not part of any work-flow. Import facility provided just to allow grid adaptation followed by export. This facility provides GUT_003 Default: EGM_GVE_2

Vector	InputNSDeflection	Not part of any work-flow. Import facility provided just to allow grid adaptation followed by export. This facility provides GUT_004 Default: EGM_GVN_2
Vector	InputGeoidHeightVarCom	Commission error variances. Not part of any work-flow. Import facility provided just to allow grid adaptation followed by export Default: EGM_GER_2
Vector	InputGravityAnomalyVarCom	Commission error variances. Not part of any work-flow. Import facility provided just to allow grid adaptation followed by export Default: EGM_GER_2
Vector	InputEWDeflectionVarCom	Commission error variances. Not part of any work-flow. Import facility provided just to allow grid adaptation followed by export Default: EGM_GER_2
Vector	InputNSDeflectionVarCom	Commission error variances. Not part of any work-flow. Import facility provided just to allow grid adaptation followed by export Default: EGM_GER_2
Vector	InputMSSH	MSSH for MDT calculations Default: CLS01
Vector	InputAverageSLA	Average SLA for a period taken from a SLA time-series Default time series: ?
Vector	InputAverageADT	Average ADT for a period taken from a ADT time-series Default time series: ?
Vector	InputMDT	MDT Default: OCCAM model at 1/12 degree
Vector	InputMDTS	Satellite MDT Default: None (Should be output from WF3a or WF3b)
Vector	**GeoidHeight**	Geoid Height
Vector	**EWDeflection**	Gravity deflection from the vertical, E-W direction
Vector	**NSDeflection**	Gravity deflection from the vertical, N-S direction
Vector	**GravityAnomaly**	Gravity anomaly
Vector	**GeoidHeightVarCom**	Commission error variances of Geoid Height. This will not be included in the first release of GUT
Vector	**EWDeflectionVarCom**	Commission error variances of Gravity deflection from the vertical, E-W direction. This will not be included in the first release of GUT
Vector	**NSDeflectionVarCom**	Commission error variances of Gravity deflection from the vertical, N-S direction. This will not be included in the first release of GUT
Vector	**GravityAnomalyVarCom**	Commission error variances of Gravity anomaly. This will not be included in the first release of GUT
Vector	**GeoidHeightVarOm**	Omission error variances of Geoid Height
Vector	**EWDeflectionVarOm**	Omission error variances of Gravity deflection from the vertical, E-W direction
Vector	**NSDeflectionVarOm**	Omission error variances of Gravity deflection from the vertical, N-S direction
Vector	**GravityAnomalyVarOm**	Omission error variances of Gravity anomaly

Vector	**MSSH_Average**	MSSH calculated using reference MSSH and average SLA
Vector	**MDT_Average**	MDT calculated using reference MDT and average SLA
Vector	**MDTS_Spatial**	MDTS from WF3a
Vector	MDTS_Spatial-InterMDTS	Unfiltered MDTS
Vector	**MDTS_Spectral**	MDTS from WF3b
Vector	MDTS_Spectral-InterMSSH	Intermediate MSSH. Spatial field not created in work-flow
Vector	MDTS_Spectral-InterGeoidHeight	Intermediate geoid height. Spatial field not created in work-flow
Vector	MDTS_Spectral-InterMDTS	Intermediate MDTS.
Vector	**MDTC_A-Spatial**	MDTC, RR method A in spatial domain
Vector	MDTC_A-Spatial-InterAprioriMDT	A-priori MDT
Vector	MDTC_A-Spatial-InterMDTCorrection	MDT correction
Vector	**MDTC_A-Spectral**	MDTC, RR method A in spectral domain
Vector	MDTC_A-Spectral-InterAprioriMDT	A-priori MDT. The spectral version of this field is calculated in the WF
Vector	MDTC_A-Spectral-InterSmoothAprioriMDT	Smooth A-priori MDT. The spectral version of this field is calculated in the WF
Vector	MDTC_A-Spectral-InterMDTCorrection	MDT correction. The spectral version of this field is calculated in the WF
Vector	**MDTC_B-Spatial**	MTDC, RR method B in spatial domain
Vector	MDTC_B-Spatial-InterUnfilteredMDTCorrection	MSSH – geoid – MDT
Vector	MDTC_B-Spatial-InterMDTCorrection	Filtered {MSSH – geoid –MDT}
Vector	**MDTC_B-Spectral**	MTDC, RR method B in spectral domain
Vector	MDTC_B-Spectral-InterAprioriGeoidHeight	MSSH – MDT, with gaps filled with geoid
Vector	MDTC_B-Spectral-InterGeoidHeight	Intermediate geoid. The spectral version of this field is calculated in the WF
Vector	MDTC_B-Spectral-InterUnfilteredMDTCorrection	A-priori-geoid - geoid. The spectral version of this field is calculated in the WF
Vector	MDTC_B-Spectral-InterMDTCorrection	Filtered {MSSH – geoid. – MDT}. The spectral version of this field is calculated in the WF
Vector	MDTC_B-Spectral-InterAprioriMDT	Filtered a-priori MDT. The spectral version of this field is calculated in the WF
Vector	**NVelocity**	Northwards component of geostrophic current

Spectral Object

Attributes

Name	Type	Possible values	Default value	Description
`MaxDegOrdPotential`	Integer	In range 1 to 250	250	Maximum degree & order

				of SH coefficients representing the gravity potential
`MaxDegOrdSurface`	Integer	In range 1 to default for `MaxDegOrdPotential`	Default for `MaxDegOrdPotential`	Maximum degree & order for SH analysis of spatial fields, i.e. for converting from spatial to spectral space
`Annotation`	String	ASCII chars, including '\n'	"Default "	Description of the object

Data

The Spectral Object has a 2D array equivalent to every vector in the Spatial Object. The names of the arrays in the Spectral Object are the same as the names of the vectors in the Spatial Object. Spatial fields can be converted to an array of SH coefficients during the course of a scientific work-flow. Any spatial field in the Spatial Object can also be manually converted to spectral space using a Spatial Object `Convert` method. The result of these conversions is stored as an array in the Spectral Object. The only data that are supplied in spectral form by default in GUT are the SH coefficients of the gravity potential. To avoid repeating most of the information in the Spatial Object data table above, the only two Spectral Object array that is shown in the table below is for the gravity potential coefficients, because this is the only Spectral Object array that does not have a corresponding vector in the Spatial Object. All the other arrays are undefined until a spatial field is converted to spectral space during a user session.

Type	Name	Description

Covar Object

Attributes

Name	Type	Possible values	Default value	Description
ErrorFunctionStart	Float	?	None	Starting distance for evaluation of omission error covariance function
ErrorFunctionEnd	Float	?	None	Ending distance for evaluation of omission error covariance function
ErrorFunctionInterval	Float	?	None	Interval for evaluation of omission error covariance function
Annotation	String	ASCII chars, including '\n'	"Default "	Description of the object

Data

Name	Type	Description
Lon	Vector	Longitudes for positions for error covariance calculations
Lat	Vector	Latitudes for positions for error covariance calculations
GeoidHeightCovOm	2D Array	Omission error variance-covariance matrix
GeoidHeightCovOmFunction	Vector	Evaluation of omission error covariance function of distance
GravityAnomalyCovOm	2D Array	Omission error variance-covariance matrix
GravityAnomalyCovOmFunction	Vector	Evaluation of omission error covariance function of distance

EWDeflectionCovOm	2D Array	Omission error variance-covariance matrix
EWDeflectionCovOmFunction	Vector	Evaluation of omission error covariance function of distance
NSDeflectionCovOm	2D Array	Omission error variance-covariance matrix
NSDeflectionCovOmFunction	Vector	Evaluation of omission error covariance function of distance
InputPotentialCovCom	2D Array	Error covariance matrix for SH coefficients of the gravity potential. This matrix is not part of the IDS. Instead, the path to the matrix in ICGEM format is stored by Covar Object attribute InputPotentialCovComPath. This data structure will not be included in the first release of GUT Default: EGM_GVC_2
GeoidHeightCovCom	2D Array	Commission error variance-covariance matrix. This will not be included in the first release of GUT
GravityAnomalyCovCom	2D Array	Commission error variance-covariance matrix. This will not be included in the first release of GUT
EWDeflectionCovCom	2D Array	Commission error variance-covariance matrix. This will not be included in the first release of GUT
NSDeflectionCovCom	2D Array	Commission error variance-covariance matrix. This will not be included in the first release of GUT

Filter Object

Attributes

Name	**Type**	**Possible values**	**Default value**	**Description**
`Type`	Flag	Python constants: `SPATIAL_JEKELI`, `SPATIAL_GAUSSIAN`, `SPATIAL_CAP`, `SPATIAL_HANNING`, `SPATIAL_HAMMING`, `SPECTRAL_JEKELI` or `SPECTRAL_PELLINEN`	Python constant: `SPATIAL_JEKELI`	Filter type
`Scale`	Integer	?	?	Length scale
`Annotation`	String	ASCII chars, including '\n'	"Default "	Description of the object

Data

Name	**Type**	**Description**
Matrix	2D Array	Filter matrix

8. LOGICAL MODEL OF SYSTEM FUNCTIONALITY

This section discusses the units of computational work and data flows involved in setting up the internal data store and executing the scientific work-flows. Python syntax is used to specify names of logical data structures according to the

Object they belong to in the user environment. The calculation of commission error variances and covariances is not mentioned explicitly in any of the discussions concerning scientific work-flows. This is because the ability to handle the error covariance matrix of the SH coefficients of gravity potential is not going to be in the first release of GUT, as discussed in the Introduction to this report. For the first release of GUT, calculation of covariance error variances will instead be provided by interpolation of the relevant GOCE L2 products, EGM_GER_2. This can be achieved by importing the relevant products into "Input" logical data structures in the Spatial Object. The `Import` methods in GUT deal with adapting data to the current, specified grid and reference system, as described in Section 6 ("Specification of Input and Output").

8.1. Processing Units

The Processing Units described in this section are related to the Functional Algorithms described in the WP3000 report. They are units of computational work involved in the pre-processing of data and the execution of the scientific work-flows. They are large computational tasks involving one or more spatial or spectral fields. All Processing Units have the following types of input and output.
Input:
- One or more attributes
- One or more data structures representing spatial and/or spectral fields
- Other large data structures such as a filter matrix

Output:
- One or more data structures representing spatial and/or spectral fields
- Other large data structures such as a filter matrix

The Processing Units represent computationally expensive tasks because they each involve a large amount of data. Therefore, they are not likely to be implemented in Python. Instead, they will probably be implemented in C or Fortran as described in Section 3 ("Toolbox Components"). They all exchange data with the internal data store independently of the parent Python application. In the data-flow diagrams in the next section, the output of one Processing Unit is often shown going directly into another Processing Unit. In most cases the output is also shown being directed to a logical data structure corresponding to a scientifically interesting work-flow product. However, in some cases no logical data structure is specified. All transfers of data take place via the internal data store, but logical data structures are not shown unless they correspond to a scientifically interesting intermediate work-flow product.

Each Processing Unit is numbered according to the Functional Algorithm it relates to. The preference selection tasks are not represented by Processing Units because they simply involve assigning values to attributes. That is why there is no Processing Unit related to Functional Algorithm FA01, because selecting the required degree and order for SH synthesis is not a computationally expensive task. However, Functional Algorithms FA02 and FA03, concerning the reference ellipsoid and tide system, are associated with a Processing Unit because they are a mixture of preference selection and data pre-processing. The preference selection part involves assigning values to the attributes defining the ellipsoid and tide system, and the data pre-processing part involves adapting any existing data in the internal data store to the new parameters. Preference selection methods are discussed in the Section 5.1 ("Command Interface") and in the Appendix ("Using the Command Interface"). Processing Units PU15a, PU15b, PU16 and PU17 do not relate to any of the Functional Algorithms.

The Processing Units are described below, with reference to the logical data structures, the Functional Algorithms from WP3000 and the algorithms corresponding to the GUT Products from WP2000. In the Processing Unit definitions, some inputs and outputs are not explicitly mentioned. Firstly, all Processing Units that deal with spatial field vectors have the following attributes as input:
- Spatial Object grid or point list specification parameters
  - `Spatial.GridType`
  - `Spatial.LatMin, Spatial.LatMax, Spatial.LatCell`
  - `Spatial.Lat`

- `Spatial.LonMin, Spatial.LonMax, Spatial.LonCell`
- `Spatial.Lon`

Similarly, all Processing Units that deal with surface SH coefficient arrays in the Spectral Object have input from the `Spectral.MaxDegOrdSurface` attribute, and all Processing Units that deal with SH coefficients of the gravity potential have input from the `Spectral.MaxDegOrdPotential` attribute.

All Processing Units take input from the Main Object reference system parameters, defined by the following attributes:

- o `Main.Ellips_GM`
- o `Main.Ellips_a`
- o `Main.Ellips_gamma_a`
- o `Main.Ellips_f`
- o `Main.Ellips_J2`
- o `Main.Ellips_omega`
- o `Main.TideSys`

Pre-processing Units

PU02: Reference ellipsoid adaptation

Adapts one spatial field vector to a new reference ellipsoid specification, using one of the GUT Product algorithms in the range GUT_104 to GUT_105.

Input:

- One Spatial Object vector
- Reference ellipsoid parameters for the vector
- Current Main Object reference ellipsoid parameters
    - o `Main.Ellips_GM`
    - o `Main.Ellips_a`
    - o `Main.Ellips_gamma_a`
    - o `Main.Ellips_f`
    - o `Main.Ellips_J2`
    - o `Main.Ellips_omega`

Output:

- One Spatial Object vector

PU03a: Tide system adaptation in geographical space

Adapts one spatial field vector to a new tide system, using one of the GUT Product algorithms in the range GUT_106 to GUT_107.

Input:

- One Spatial Object vector
- Tide system specifier for the vector
- Current Main Object tide system specifier
    - o `Main.TideSys`

Output:

- One Spatial Object vector

PU03b: Tide system adaptation in spectral space

Adapts one set of gravity field potential SH coefficients to a new tide system, using the algorithms described in the definition of FA03. There is no GUT Product algorithm corresponding to this Processing Unit.
Input:

- One Spectral Object array
  - `Spectral.InputPotential`
- Tide system specifier for those coefficients
- Current Main Object tide system specifier
  - `Main.TideSys`
- Maximum degree and order of SH coefficients
  - `Spectral.MaxDegOrdPotential`

Output:

- One Spectral Object array
  - `Spectral.InputPotential`

PU07: Time series averaging

Calculates the average value of one spatial field for a period of a time-series, using the algorithm described in the definition of FA07. There is no GUT Product algorithm corresponding to this Processing Unit.
Input:

- Time-series of a spatial field, on a grid or point list
- Spatial field identifier
- Grid or point list specification parameters describing time series
- Start and end dates for calculation of averages

Output:

- A spatial field on a grid or point list

PU08: Grid adaptation

Adapts one gridded spatial field to a new grid or point list specification, using the algorithm for GUT Product GUT_110.
Input:

- A spatial field on a grid
- Spatial field identifier
- Grid specification parameters describing spatial field
- Spatial Object grid or point list specification parameters
  - `Spatial.GridType`
  - `Spatial.LatMin, Spatial.LatMax, Spatial.LatCell`
  - `Spatial.Lat`
  - `Spatial.LonMin, Spatial.LonMax, Spatial.LonCell`
  - `Spatial.Lon`

Output:

- One Spatial Object vector

Work-flow Processing Units

PU04a: SH synthesis from SH coefficients gravity potential

Computes one geodetic field on a grid or point list, starting from the SH coefficients of the gravity field potential. The geodetic field is either geoid height, gravity anomaly or deflection from the vertical. This Processing Unit corresponds to FA04 with either geoid height, gravity anomaly or geoid deflection specified as an option. One GUT Product algorithm in the range GUT_005 to GUT_012 is used during the execution of this Processing Unit.

Input:
- Geodetic output field specifier
    - `Main.GeodeticCalc`
- SH Coefficients of gravity potential
    - `Spectral.InputPotential`
- Maximum degree and order of potential SH coefficients to be used in the synthesis
    - `Spatial.MaxDegOrdPotential`
- Calculation type
    - `Spatial.GridCalc`
- Grid or point list specification
    - `Spatial.GridType`
    - `Spatial.LatMin, Spatial.LatMax, Spatial.LatCell`
    - `Spatial.Lat`
    - `Spatial.LonMin, Spatial.LonMax, Spatial.LonCell`
    - `Spatial.Lon`

Output:
- One Spatial Object vector, or pair of vectors in the case of gravity field deflection

PU4b: SH synthesis from surface SH coefficients

Converts one set of surface SH coefficients to geographical space. This Processing Unit corresponds to FA04 with Dynamic Topography specified as an option, but it is not restricted to SH synthesis of Dynamic Topography fields. PU04b can be used for SH synthesis from any set of surface SH coefficients. An algorithm similar to those defined in GUT_005 to GUT_012 could be used to perform the SH synthesis.

Input:
- Spectral Object array containing surface SH Coefficients
- Maximum degree and order of surface SH coefficients to be used in the synthesis
    - `Spatial.MaxDegOrdSurface`

Output:
- One Spatial Object vector

PU05: Commission error determination

The first release of GUT will not include the ability to calculate commission error variances and covariances, as discussed in the Introduction to this report. This Processing Unit is included for completeness, although it is not used in any of the scientific work-flows described in this report. In the first release of GUT, commission error variance data for the geoid height, deflection and gravity anomaly fields will be provided by allowing the user to import the relevant gridded GOCE error products (EGM_GER_2).

This Processing Unit computes commission error variance and covariance for one geodetic output field. GUT Product algorithms in the range GUT_016 to GUT_031 are used for each execution of this Processing Unit.

Input:
- Geodetic output field specifier
    - `Main.GeodeticCalc`
- SH Coefficients of gravity potential
    - `Spectral.InputPotential`
- Error covariance matrix of SH coefficients of gravity potential
    - `Covar.InputPotentialCovCom`
- Maximum degree and order of potential SH coefficients in Spectral Object
    - `Spectral.MaxDegOrdPotential`

- Maximum degree and order for SH sythesis of potential SH coefficients
  - `Spatial.MaxDegOrdPotential`
- Calculation type
  - `Spatial.GridCalc`
- Grid or point list specification
  - `Spatial.GridType`
  - `Spatial.LatMin, Spatial.LatMax, Spatial.LatCell`
  - `Spatial.Lat`
  - `Spatial.LonMin, Spatial.LonMax, Spatial.LonCell`
  - `Spatial.Lon`
- Specification of a pair of points
  - `Covar.Lat`
  - `Covar.Lon`

Output:

- One Spatial Object vector containing commission error variances
- One Covar Object array containing commission error covariances

PU06: Omission error determination

Computes omission error variance and covariance for one geodetic output field. One of the GUT Product algorithms in the range GUT_032 to GUT_039 is used for this Processing Unit.

Input:

- Geodetic output field specifier
  - `Main.GeodeticCalc`
- SH Coefficients of gravity potential
  - `Spectral.InputPotential`
- Maximum degree and order of potential SH coefficients in Spectral Object
  - `Spectral.MaxDegOrdPotential`
- Maximum degree and order for SH sythesis of potential SH coefficients
  - `Spatial.MaxDegOrdPotential`
- Calculation type
  - `Spatial.GridCalc`
- Grid or point list specification
  - `Spatial.GridType`
  - `Spatial.LatMin, Spatial.LatMax, Spatial.LatCell`
  - `Spatial.Lat`
  - `Spatial.LonMin, Spatial.LonMax, Spatial.LonCell`
  - `Spatial.Lon`
- Specification of a pair of points
  - `Covar.Lat`
  - `Covar.Lon`
- Specification of function evaluation parameters

Output:

- One Spatial Object vector containing omission error variances
- One Covar Object array containing omission error covariances
- One Covar Object vector containing evaluation of omission error function

PU09: Linear filter (spatial)

Filters one spatial field, using one of five types of linear filter. Filtering algorithms are discussed in the WP3000 report.

Input:

- One spatial field vector in Spatial Object
- Filter identifier and scale parameter

  - o `Filter.Type`
  - o `Filter.Scale`
- Filter matrix (if this filter has already been used for the current Spatial Object grid)
  - o `Filter.Matrix`

Output:
- One spatial field vector in Spatial Object
- Filter matrix
  - o `Filter.Matrix`

PU10: Linear filter (spectral)

Filters one array of surface SH coefficients, using either of two types of linear filter. Filtering algorithms are discussed in the WP3000 report.
- One array of surface SH coefficients
- Filter identifier and scale parameter
  - o `Filter.Type`
  - o `Filter.Scale`
- Filter matrix (if this filter has already been used for the current value of `Spectral.MaxDegOrdSurface`)
  - o `Filter.Matrix`

Output:
- One array of surface SH coefficients
- Filter matrix
  - o `Filter.Matrix`

PU11a: Fill gaps on continent with another global field

Fills continental gaps in one spatial ocean field using values from global gridded field. This Processing Unit is designed for filling continental gaps in MSSH. The appropriate algorithm for this processing unit is described in the description of FA11 in the WP3000 report.
Input:
- One Spatial Object vector with gaps over continents
- One Spatial Object global gridded field for filling the continental gaps in the ocean field

Output:
- One Spatial Object vector

PU11b: Fill gaps on continent with zeros

Fills continental gaps in one spatial ocean field with zeros. This Processing Unit is designed for filling continental gaps in MDT. The appropriate algorithm for this processing unit is described in the description of FA11 in WP3000.
Input:
- One Spatial Object vector with gaps over continents

Output:
- One Spatial Object vector

PU12: SH analysis

Calculates surface SH coefficients for one global, gridded field, using GUT Product algorithm GUT_103.
Input:
- One Spatial Object vector
- Maximum degree and order for surface SH coefficients
  - o Spectral.MaxDegOrdSurface

Output:
- One Spectral Object SH coefficient array

PU13: Surface current determination

Calculates northward and eastward components of surface current for one Spatial Object MDT vector. One of two GUT Product algorithms is used during the execution of this Processing Unit: GUT_013 if the Spatial Object data structures specify a grid, and GUT_014 if the Spatial Object data structures specify a point list.
Input:
- Spatial Object MDT vector

Output:
- Spatial Object vector of Northwards component of velocity
- Spatial Object vector of Eastwards component of velocity

PU14a: Sum of two spatial fields

Adds two spatial fields using GUT Product algorithm GUT_100.
Input:
- Two Spatial Object fields

Output:
- One spatial Object field

PU14b: Difference of two spatial fields

Subtracts two spatial fields using GUT Product algorithm GUT_101.
Input:
- Two Spatial Object fields

Output:
- One spatial Object field

PU15a: Sum of two spectral fields

Adds two spectral fields. No algorithm for this Processing Unit is defined in the reports from WP2000 and WP3000.
Input:
- Two Spectral Object fields

Output:
- One Spectral Object field

PU15b: Difference of two spectral fields

Subtracts two spectral fields. No algorithm for this Processing Unit is defined in the reports from WP2000 and WP3000.
Input:
- Two Spectral Object fields

Output:
- One Spectral Object field

PU16: Calculation of geoid height in spectral space

Calculates surface SH coefficients of the geoid height field. No algorithm for this Processing Unit is specified in the reports from WP2000 and WP3000.
Input:
- SH Coefficients of gravity potential
    - `Spectral.InputPotential`
- Maximum degree and order for SH synthesis of geoid height
    - `Spatial.MaxDegOrdPotential`
- Maximum degree and order for SH analysis
    - `Spectral.MaxDegOrdSurface`

Output:
- One array of Surface SH coefficients in the Spectral Object

PU17: Calculation of mean from reference field and average SLA.

Calculates the mean value of a spatial field on a grid or list of points (M), from the reference mean field (REF) and the average SLA for the time period of interest (MSLA). Algorithm: M = REF + MSLA

Input:
- Reference field vector in Spatial Object
- Mean SLA for time period of interest
  - o `Spatial.InputAverageSLA`

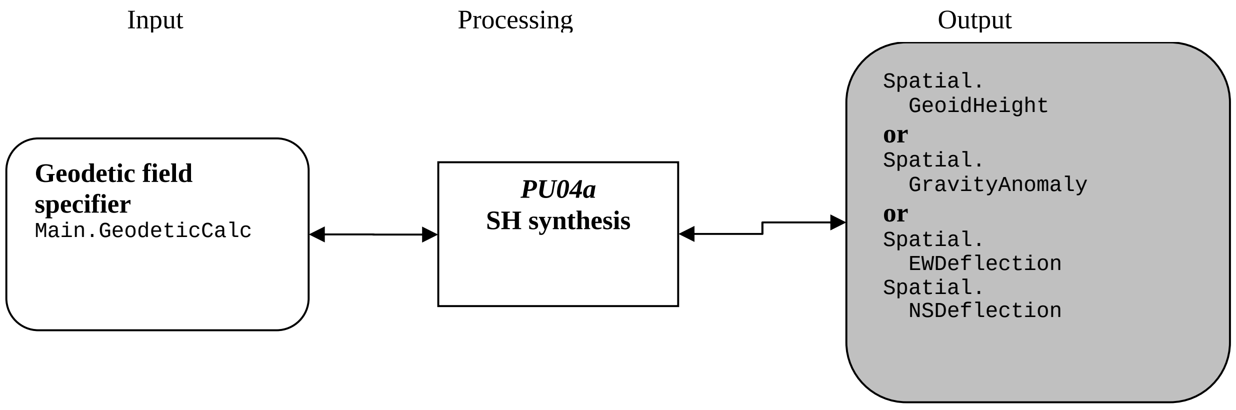
Output
- One Spatial Object vector

8.2. Scientific Data Flows

This section relates the work-flows from WP3000 to the Processing Units and the logical data structures defined in Section 7 ("Specification of Logical Data Structures"), i.e. the attributes, vectors and arrays associated with the various types of Object in the user environment. The figures in this section show a series of *data-flow diagrams* to accompany the scientific work-flows from WP3000. There is a data-flow diagram for each scientific work-flow. Python notation is used to describe the data structures, which are labelled according to the following convention.
ObjectName.DataStructureName

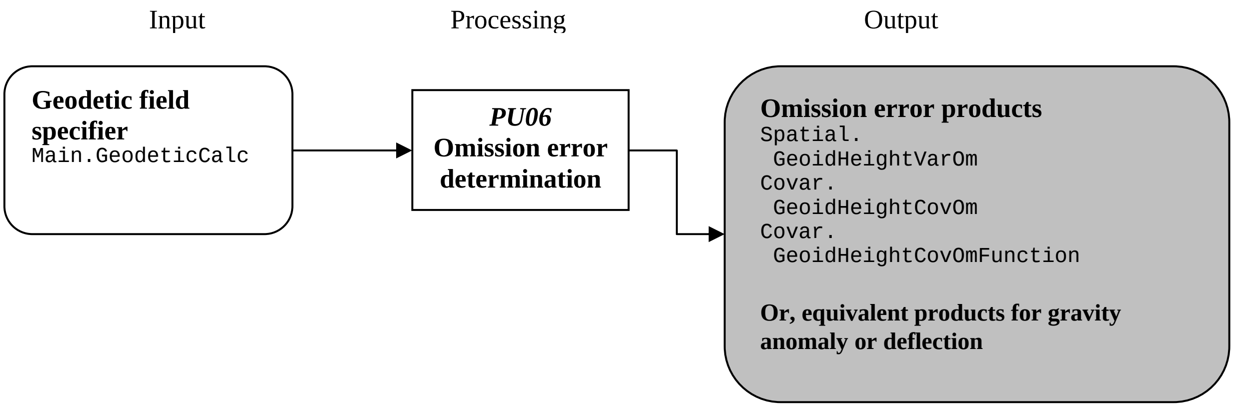
The main difference between the data-flow diagrams and the work-flow diagrams is that preference selection and data pre-processing stages are not shown. The results of preference selection decisions taken by the user are stored in the values of attributes. Similarly, the data pre-processing stages such as grid and reference system adaptation are not shown in the data-flows, because data in the logical data structures has already been through those processes. One advantage of this pre-processing approach is that time consuming processing steps such as grid adaptation do not need to be repeated each time a work-flow calculation is performed. In the data-flow diagrams, all the possible end-points are indicated by shaded boxes. To execute a work-flow up to one of the end points, the `Calc` method corresponding to the desired intermediate or output data structure is used, as described in Section 5.1 ("Command Interface").

All Processing Units take input from, and send output to real data structures in the internal data store. Therefore, all data transfer between processing units shown in the data-flow diagrams takes place via the internal data store. Logical data structures corresponding to these transfers are not shown, except in cases where they correspond to scientifically interesting work-flow products that were included in the work-flow diagrams in the WP3000 report. The scientifically interesting products are directed to intermediate logical data structures, which are accessible to the user. Extra scientifically interesting intermediate data structures can easily be added at a later time. Inputs to, and outputs from Processing Units that come from fixed logical data structures are omitted from the data-flow diagrams. The Process Unit definitions contain details of these fixed inputs and outputs. In cases where the output of one work-flow is to be used as the input to another work-flow, the output from the first work-flow must be exported to an external file, and then imported into the appropriate input logical data structure for the second work-flow.

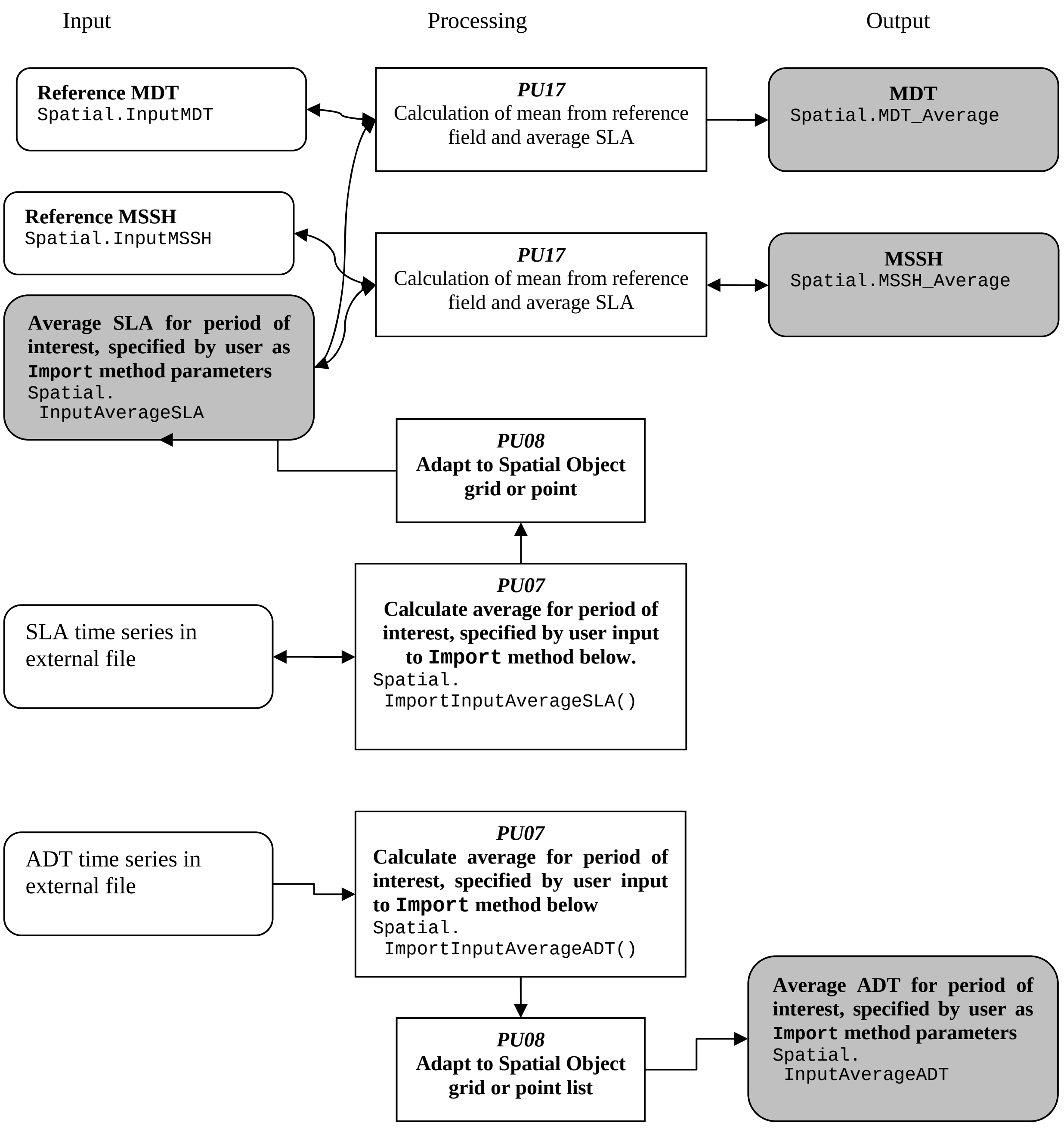


Data-flow 1a: Geoid and gravity field computation.

Input Processing Output

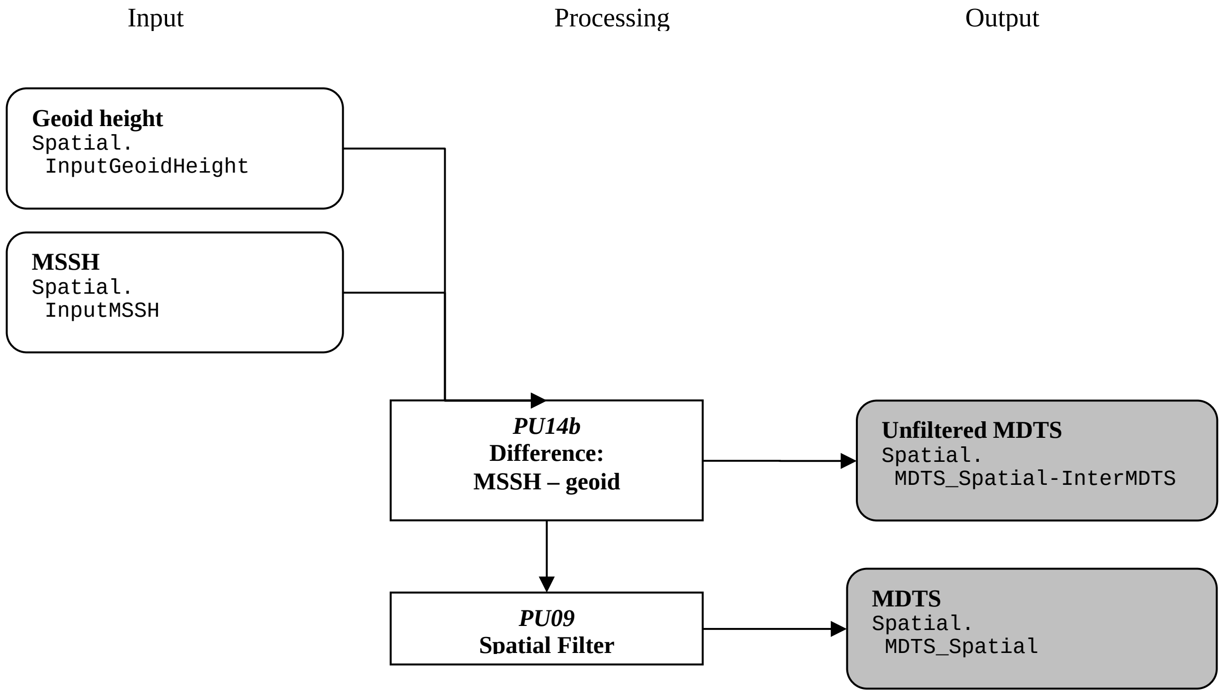


Data-flow 1b. Omission error computation for geoid and gravity field.
Note: Commission error variances for any grid or point list can be obtained by importing GOCE L2 error variance products. The import procedure performs adaptation of the error fields to the desired grid and reference

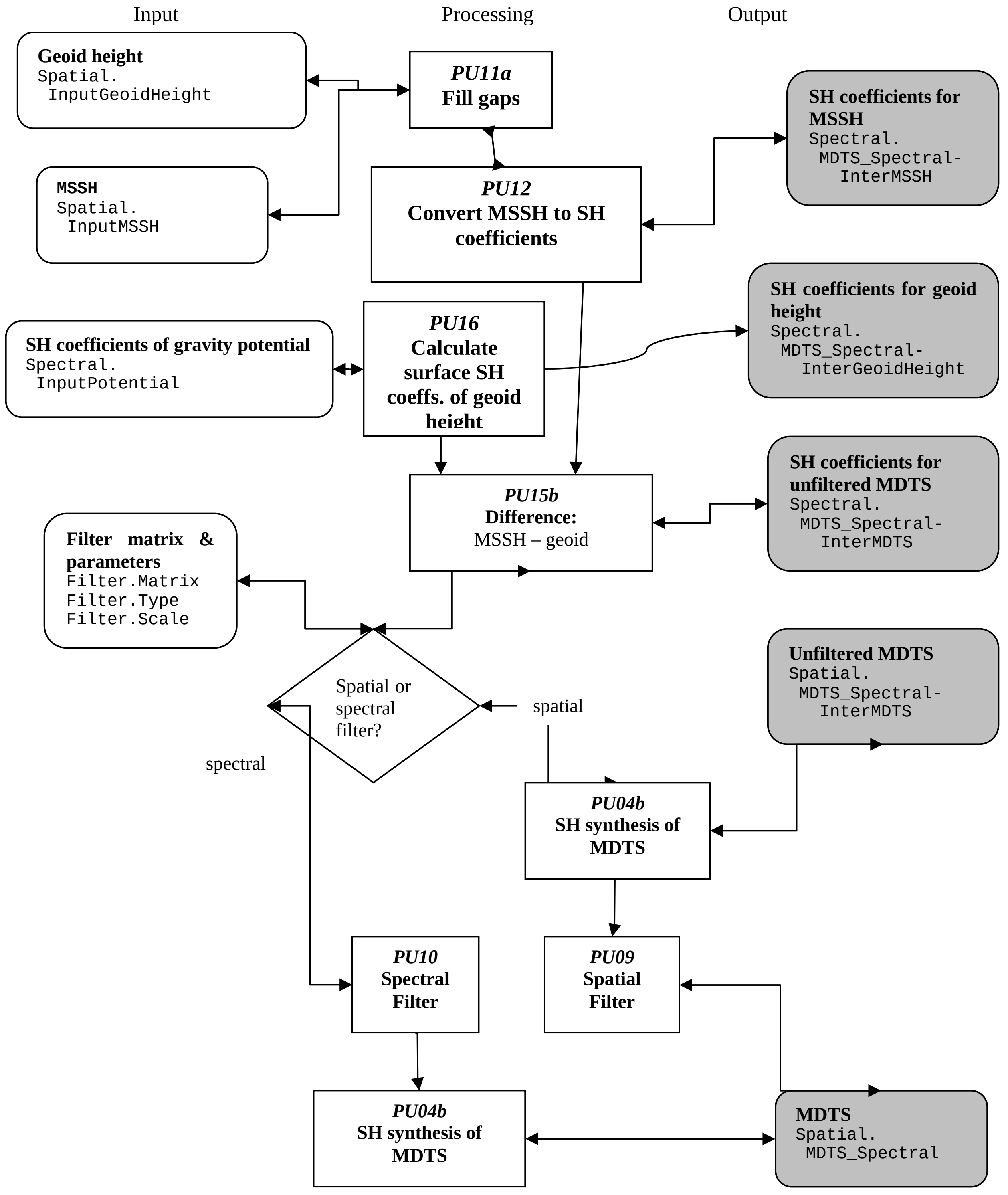


Input
Processing
Output
Reference MDT
Spatial.InputMDT
PU17
Calculation of mean from reference field and average SLA
MDT
Spatial.MDT_Average
Reference MSSH
Spatial.InputMSSH
PU17
Calculation of mean from reference field and average SLA
MSSH
Spatial.MSSH_Average
Average SLA for period of interest, specified by user as Import method parameters
Spatial.
InputAverageSLA
PU08
Adapt to Spatial Object grid or point
SLA time series in external file
PU07
Calculate average for period of interest, specified by user input to Import method below.
Spatial.
ImportInputAverageSLA()
ADT time series in external file
PU07
Calculate average for period of interest, specified by user input to Import method below
Spatial.
ImportInputAverageADT()
PU08
Adapt to Spatial Object grid or point list
Average ADT for period of interest, specified by user as Import method parameters
Spatial.
InputAverageADT

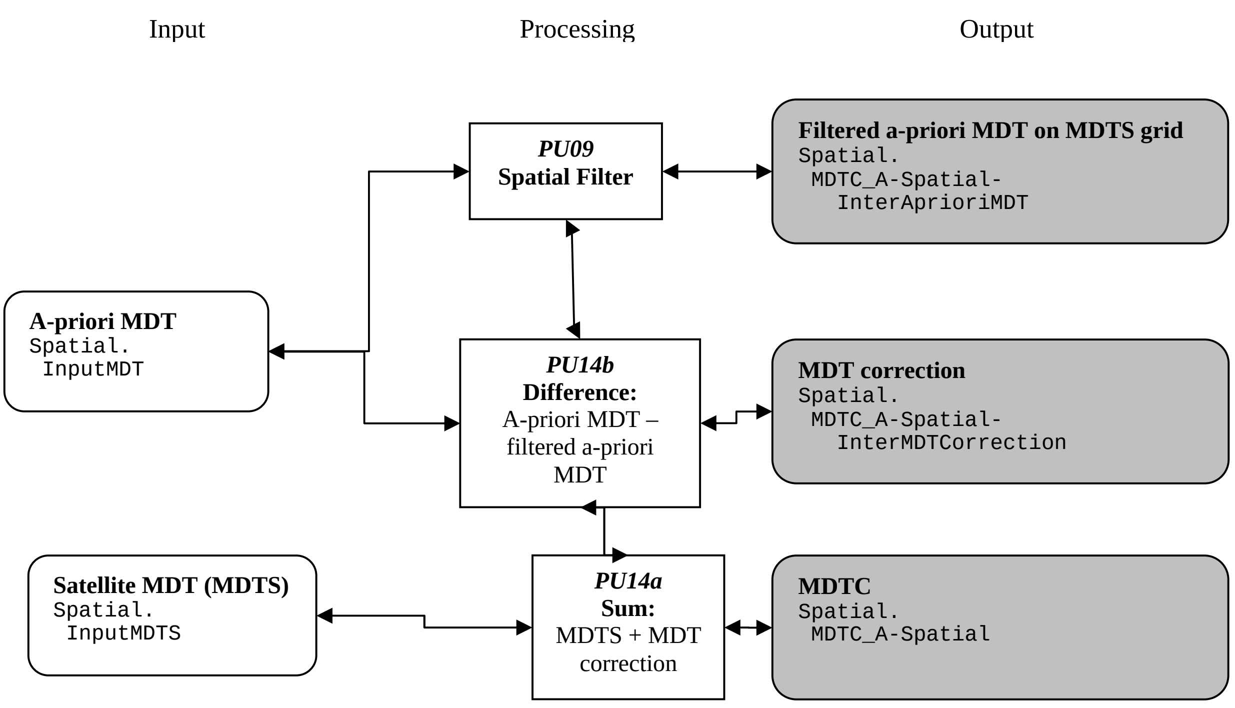
Data-flow 2. Sea surface height and a-priori MDT selection
This data-flow diagram corresponds to Work-flow 2, which covers calculating the average for a period of a time series and also calculating a new mean field from the reference mean field and the average SLA in the period of interest. The Import methods for average SLA and ADT perform averaging and grid adaptation



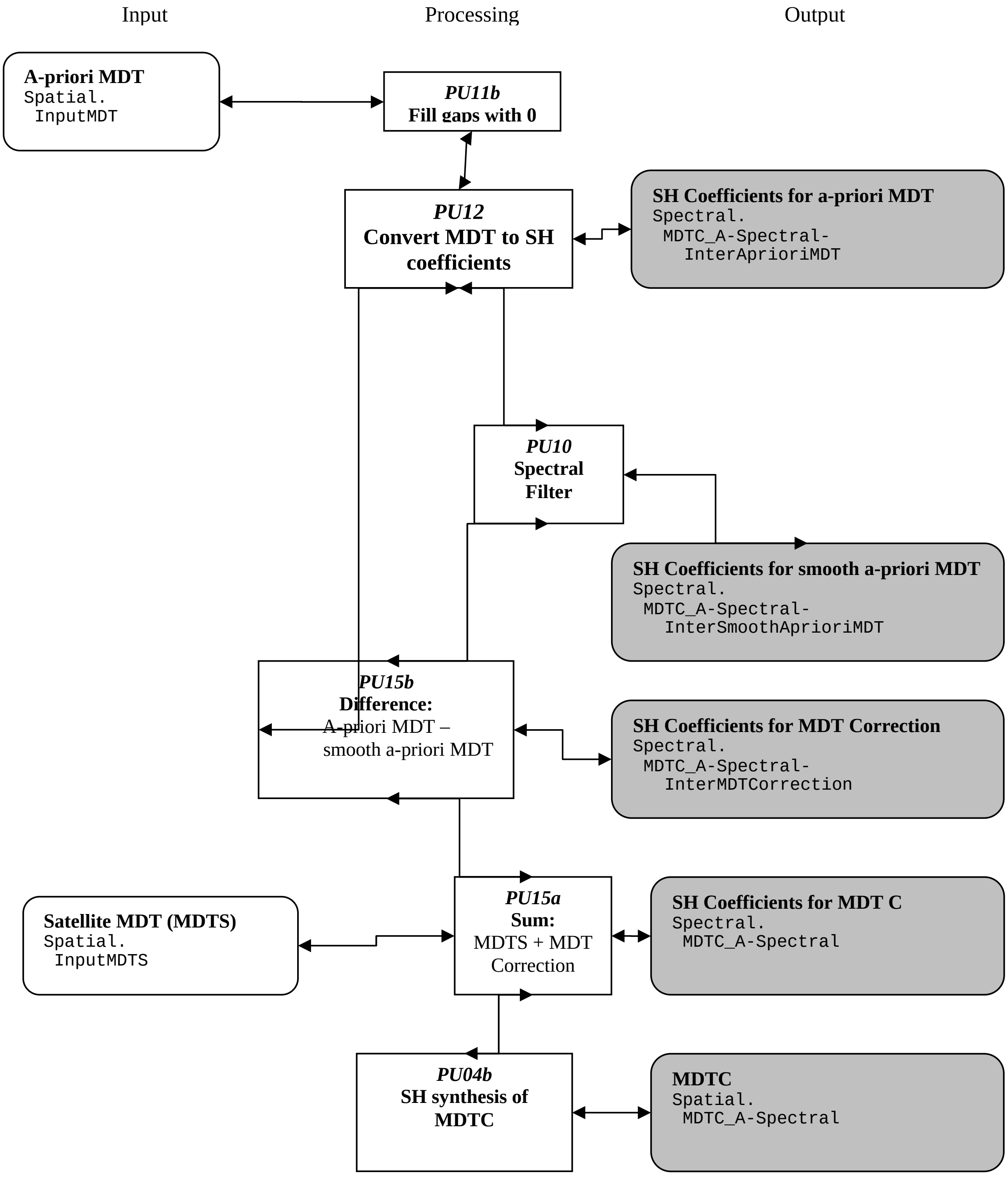
Data-flow 3a. Satellite Dynamic Topography computation in geographical space



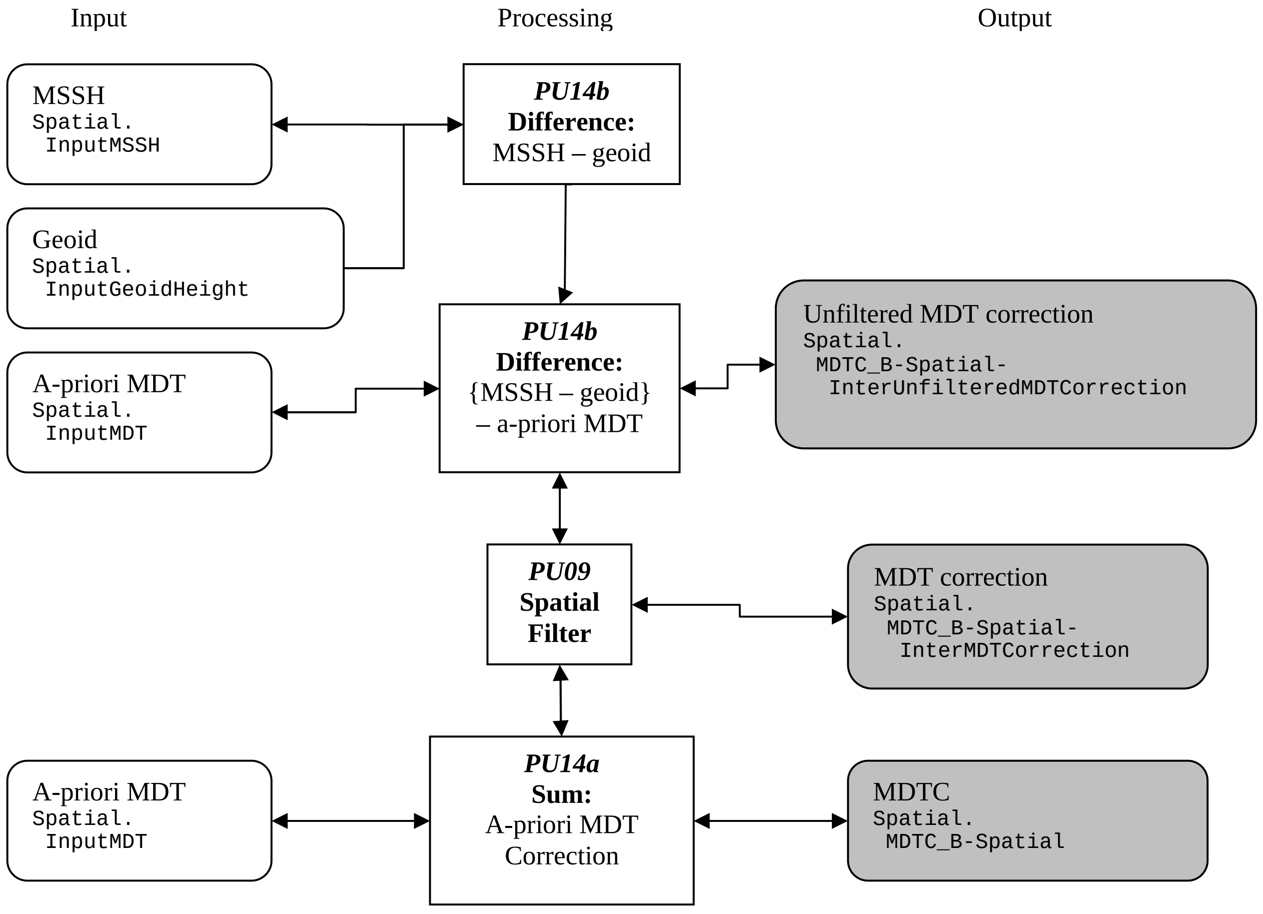
Data-flow 3b. Satellite Dynamic Topography computation in spectral space



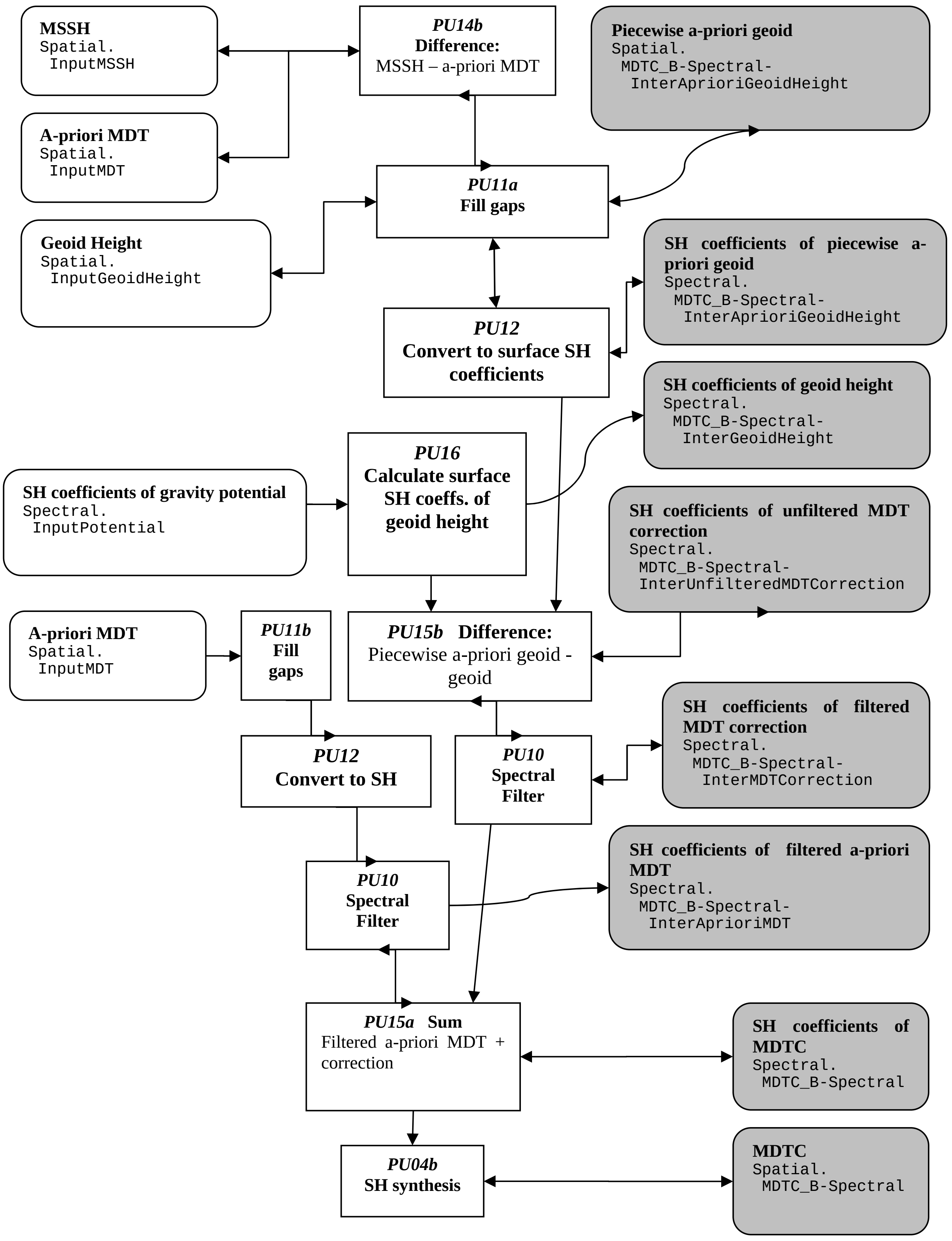
Data-flow 4a: Remove-Restore combined technique A: spatial filtering


Input
Processing
Output
A-priori MDT
Spatial.
InputMDT
PU11b
Fill gaps with 0
PU12
Convert MDT to SH coefficients
SH Coefficients for a-priori MDT
Spectral.
MDTC_A-Spectral-InterAprioriMDT
PU10
Spectral Filter
SH Coefficients for smooth a-priori MDT
Spectral.
MDTC_A-Spectral-InterSmoothAprioriMDT
PU15b
Difference:
A-priori MDT – smooth a-priori MDT
SH Coefficients for MDT Correction
Spectral.
MDTC_A-Spectral-InterMDTCorrection
Satellite MDT (MDTS)
Spatial.
InputMDTS
PU15a
Sum:
MDTS + MDT Correction
SH Coefficients for MDT C
Spectral.
MDTC_A-Spectral
PU04b
SH synthesis of MDTC
MDTC
Spatial.
MDTC_A-Spectral

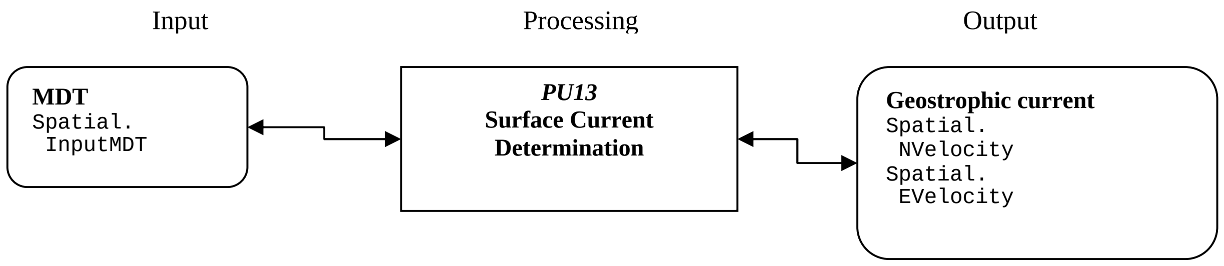
Data-flow 4b: Remove-Restore combined technique A: spectral filtering



Data-flow 4c: Remove-Restore combined technique B: spatial filtering



Workflow 4d: Remove-Restore combined technique B: spectral filtering



Data-flow 5: Dynamic Topography-derived quantities

9. LIST OF ABBREVIATIONS

API	Application Program Interface
ASCII	American Standard Code for Information Interchange
BEAT	Basic Envisat Atmospheric Toolbox
BRAT	Basic Radar Altimetry Toolbox
CDAT	Climate Data Analysis Tool
DT	Dynamic Topography
ESA	European Space Agency
FTP	File Transfer Protocol
GDR	Geophysical Data Record
GMT	Generalised Mapping Tool
GOCE	Gravity Ocean Circulation Explorer
GUI	Graphical User Interface
GUT	GOCE User Toolbox
GUTS	GOCE User Toolbox Specification
HPF	High level Processing Facility
ICGEM	?
IDE	Integrated Development Environment
IDS	Internal Data Store
MDT	Mean Dynamic Topography
MDTC	Combined MDT
MDTS	Satellite MDT
MSSH	Mean Sea Surface Height
NetCDF	Network Common Data Form
RR	Remove-Restore
SH	Spherical Harmonic
SLA	Sea Level Anomaly
SSH	Sea Surface Height
VTK	Visualisation Toolkit
WF	Work-flow
WP	Work-plan

10. REFERENCES

1. Knusden, P. et al, *GUTS User Toolbox Requirements Document*, 2006

2. Siegismund, F. et al, *GUTS Toolbox Functionality and Algorithm Specification Report*
3. Bos, A., Cadot, S., Neimeijer, S., *Envisat Atmospheric Toolbox – Software Specification Document Part B: VISAN*, Science [&] Technology Corporation, March 2002
4. Python.org - http://www.python.org/
5. Python.org - http://docs.python.org/ext/intro.html
6. SciPy.org - http://www.scipy.org/
7. Science [&] Technology bv - http://www.science-and-technology.nl/beat/documentation/visan.html
8. The MathWorks - http://www.mathworks.com/
9. Program for Climate Model Diagnosis and Intercomparison (PCMDI) - http://www-pcmdi.llnl.gov/software-portal/cdat/
10. Unidata - http://ftp.unidata.ucar.edu/software/netcdf/
11. wxWindows.org - http://www.wxwindows.org/
12. CLS - http://www.cls.fr/html/oceano/general/applications/ccn_heracles_en.html
13. VTK.org - http://www.vtk.org/
14. School of Ocean and Earth Science Technology (SOEST), University of Hawaii - http://www.soest.hawaii.edu/gmt/
15. Eclipse.org - http://www.eclipse.org/
16. NetBeans.org - http://www.netbeans.org/
17. ITT Visual Information Solutions - http://www.ittvis.com/idl/
18. GOCE Level 2 Product Handbook

11. Appendix: Using the Command Interface

The user interacts with GUT by entering commands to execute Python methods. We could also describe this as *calling* Python methods, *running* Python methods or *invoking* Python methods. The idea is for the user to set up a Main Object with one particular work-flow in mind, and then to execute a single method that performs calculations leading to the output of the desired product. To allow the user to experiment with one particular part of a scientific work-flow, methods for performing calculations up to intermediate stages are provided. After finishing with one particular work-flow, the user can then turn his or her attention to another work-flow, making any necessary changes to the attributes and data in the Main Object. Alternatively, it may be easier to start again from scratch with a new Main Object. As an extension to the concept of the Main Object, it would be possible to design a user interface based on a set of Work-flow Objects, one for each of the GUT work-flows. A Work-Flow Object would be the same as a Main Object, except that only the attributes, data and methods relating to one particular work-flow would be present. This would simplify the command interface and enable the names of the Python methods used in commands to be rationalized. This is because many of the method names are derived from the names of the logical data structures they act on; a Work-flow Object dealing with only one particular work-flow would need a much reduced set of logical data structures and corresponding methods, which could, as a consequence have shorter names. For example, a Work-flow Object dealing with Work-flow 3a would only need to have one type of output MDT. In the Main Object there are six types of MDT output, all of which must have different names so that the user has a record of which work-flow was used to produce them. It would be useful for two or more Work-flow Objects to be able to share the same internal data store in order to avoid unnecessary duplication of data on disk. Work-flow Objects are not discussed further in this document, but this approach should be considered for implementation in the first release of GUT.

The command interface includes methods for preference selection, data pre-processing and carrying out work-flow calculations. Preference selection tasks are accomplished by assigning values to attributes. Most data pre-processing occurs when data from external sources is imported into the internal data store using `Import` methods. Work-flow calculations are initiated by executing the `Calc` method associated with the desired work-flow output data structure. The main preference selection tasks involve grid parameters, the maximum degree and order of SH coefficients, the reference ellipsoid and the tide system. These are discussed separately below.

Grid specification

The Spatial Object attributes `LatMin, LatMax, LonMin, LonMax, LatCell` and `LonCell` are used to define a regular grid. If all these values are defined then the Spatial Object represents a regular grid, and the value of Spatial Object attribute `GridType` is REGULAR. If `LatCell` and `LonCell` are undefined (i.e. set to `None`, Python's null value) then the Spatial Object represents an irregular or unstructured grid, and the value of the `GridType` attribute is UNSTRUCTURED. If they are all undefined then the Spatial Object represents a set of positions, and the value of the `GridType` attribute is LIST. There are three ways in which these attributes can be changed.

1. Creating a new Spatial Object from scratch using the Spatial Object `New()` method. The Spatial Object is over-written with the default attributes and data (default MSSH and grid).
2. Importing gridded data into a blank Spatial Object. A blank Spatial Object has all its attributes and data set to None, and can be created using the Spatial Object `Blank()` method. Data can be imported into any field. If the file's metadata contains all seven grid specification parameters then these are used to define the values the grid specification attributes. If one or more parameters are missing from the metadata then the `Import` method prompts the user to enter the value.
3. Their values can be changed using the `DefineGrid()` method, which requires the user to specify all seven values, any of which can be specified as None. If data are already present in the Spatial Object when these attributes are changed, grid adaptation of that data takes place.

Maximum degree and order for SH synthesis

This task is referred to as Functional Algorithm FA01 in the WP3000 report. The maximum degree and order for SH synthesis is governed by the value of Spatial Object attributes `MaxDegOrdPotential` and `MaxDegOrdSurface`. The former refers to the SH coefficients representing the gravity potential, and the latter refers to surface SH coefficients. There are three Spatial Object methods that can be used to change the value of `Spatial.MaxDegOrdPotential`.

- `MaxDegOrdPotential,` for specifying the maximum degree and order directly
- `MaxResKMPotential`, for specifying a resolution in kilometres.
- `MaxResDegreesPotential`, for specifying a resolution in degrees latitude.

Three similar methods are provided for changing the value of `Spatial.MaxDegOrdSurface.`

Maximum degree and order for SH analysis

When converting a global, gridded field from geographical space to SH coefficients, the maximum degree and order of the resulting coefficients is governed by the value of Spectral Object attribute `MaxDegOrdSurface.` There are three Spectral Object methods that can be used to change the value of `Spectral.MaxDegOrdSurface`.

- `MaxDegOrdSurface,` for specifying the maximum degree and order directly
- `MaxResKMSurface`, for specifying a resolution in kilometres.
- `MaxResDegreesSurface`, for specifying a resolution in degrees latitude.

Three similar Spectral Object methods are provided for changing the value of `Spectral.MaxDegOrdPotential`. This parameter would not normally be changed manually because it is a property of the SH coefficients of gravity potential. It is not used for SH analysis.

Reference ellipsoid specification

This task is part of Functional Algorithm FA02 in the WP3000 report. The parameters used to specify the reference ellipsoid are stored by the following Main Object attributes:
`Ellips_GM, Ellips_a, Ellips_gamma-a, Ellips_J2, Ellips_omega` and `Ellips_f.` These attributes are located in the Main Object because they apply to both the spatial domain (represented by the Spatial Object) and the spectral domain (represented by the Spectral Object). The attributes do not all have to be defined in order to specify the ellipsoid. Undefined ellipsoid parameters have the value `None`. There are two ways in which the reference ellipsoid can be changed.

1. A single Main Object method for changing all the ellipsoid attributes is provided. This is called `Ellips()`, and takes six parameters, some of which can be given as `None`. Unlike most attributes in GUT, the ellipsoid attributes can not be changed individually using methods with the same names as the attributes. This is because the attributes must be consistent with the data at all times, and changing a single "Ellipse" attribute could result in an invalid ellipsoid specification.
2. When creating a new Spatial Object by importing data from a file, the reference ellipsoid of the Main Object becomes that of the data being imported. The ellipsoid parameters are obtained from the file's metadata if possible, and if not the user is prompted to enter the missing parameters. Subsequent `Import` operations will result, if necessary, in the imported data being adapted to the reference system specified by the Main Object.

Tide System Specification
This task is part of Functional Algorithm FA03 in the WP3000 report. The tide system is specified by Main Object attribute `TideSys`. As with the reference ellipsoid, the `TideSys` attribute can be changed in either of two ways.

1. Using the `TideSys()` method in the Main Object
2. Creating a new Spatial Object by importing spatial data in to a blank Spatial Object. The tide system becomes that of the first field to be imported. Subsequent imports result in data being adapted to the existing tide system.

Many powerful features of the Python language will be available to the GUT user, but it is important to emphasise that it will be easy for novice users to access the GUT work-flows. The best way to illustrate this point is by giving an example sequence of commands. The following three commands, issued at the start of a GUT session, are all that are required in order to calculate geoid heights on the default grid using the default parameters, and to export those results to a file.

```
MyOb = GUT.New()
MyOb.CalcGeoidHeight()
MyOb.Spatial.ExportGeoidHeight("heights.dat", "height_field", GRAVSOFT)
```

Below are some examples of Python command sequences for carrying out some of the work-flows described in the WP3000 report. These examples are intended to give the reader an indication of how the command interface will look, but the exact commands are likely to change during the implementation phase of GUT.

Examples from work-flow 1a: Geoid and gravity field computation.

- Create default Main object with default data
  ```
  MyOb = GUT.New()
  ```
- Load a saved Main Object from directory old_ob
  ```
  MyOb = GUT.Load("old_ob")
  ```

- Geoid heights
  ```
  MyOb.CalcGeoidHeight()
  ```
- Gravity deflections (E-W and N-S deflections are both calculated)
  ```
  MyOb.CalcDeflection()
  ```

- Create new Spatial Object with default grid and data (overwriting existing object)
  ```
  MyOb.Spatial.New()
  ```
- Create a new blank Spatial Object (overwriting existing Object) and specify grid parameters manually
  ```
  MyOb.Spatial.Blank()
  MyOb.Spatial.DefineGrid(min_lat, max_lat,…)
  ```
- Import grid or list of points from a file containing data for one of the input fields. Must start with *blank* Spatial Object, otherwise imported data will be adapted to existing grid and reference system.
  ```
  MyOb.Spatial.Blank()
  ```

```
Myob.Spatial.ImportInputMSSH("mssh.cdf", "mssh_field", NETCDF)
```

- Create default 2-D plot of geoid heights
  `MyOb.Spatial.ViewGeoidHeight()`
- Export heights to a NetCDF file
  `MyOb.Spatial.ExportGeoidHeight("heights.cdf", "height_field", NETCDF)`

Examples from Workflow 1b: Error computation for geoid and gravity field

- Calculation of geoid height omission error variances
  `MyOb.CalcGeoidHeightVarOm()`
- Define pair of positions in Covar Object
  `MyOb.Covar.AddPos(lat1, lon1)`
  `MyOb.Covar.AddPos(lat2, lon2)`
- Calculate geoid height omission error covariances
  `MyOb.CalcGeoidHeightCovOm()`
- Export geoid height omission error variances
  `MyOb.Spatial.ExportGeoidHeightVarOm("heightOmErr.cdf", "err_field", NETCDF)`

Examples from Workflow 2: Sea surface height and a-priori MDT selection

- Import average SLA and ADT for specified period from time-series.
  `MyOb. Spatial.ImportInputAverageSLA("monthly_sla_timeseries.cdf", start_date, end_date)`
  `MyOb. Spatial.ImportInputAverageADT("daily_adt_timeseries.cdf", start_date, end_date)`
- Calculate MSSH as average SLA + Reference MSSH
  `MyOb.CalcMSSH_Average()`
- Calculate MDT as average SLA + Reference MDT
  `MyOb.CalcMDT_Average()`

Examples from Workflow 3a: Satellite Dynamic Topography computation in geographical space

- Calculate MDTS using spatial domain method using default MSSH field and default filter.
  `MyOb.Spatial.ImportInputGeoidHeight("geoid.cdf", "height_field", NETCDF)`
  `MyOb.CalcMDTS_Spatial()`

Examples from Workflow 4b: Remove-Restore combined technique A: spectral filtering

- Calculate the Satellite MDT using work-flow 3b, then export the output
  `MyOb.CalcMDTS_Spectral()`
  `MyOb.Spatial.ExportMDTS_Spectral("temp_mdts.cdf", "mdt_field", NETCDF)`
- Import the MDTS calculated above
  `MyOb.Spatial.ImportInputMDTS("temp_mdts.cdf", "mdt_field", NETCDF)`
- Calculate MDTC by Spectral RR A and spatial filtering, using the output of work-flow 3b as the input MDTS
  `MyOb.CalcMDTC_A-Spectral()`

After reviewing the example commands above, the reader may observe that many of the commands are rather long, giving the mistaken impression that a lot of typing will be required in order to use GUT. Method names such as `"ImportInputLandHeight"` and `"CalcGeoidHeightVarCom"` may seem excessively long, but there are good reasons for having names of this form. The names will be designed specifically for use with a *command completion* facility in Python, which behaves in the same way as file and directory name completion in Linux and Unix shell environments. This is implemented using the `rlcompleter` package in Python. Command completion ensures that GUT users do not need to type out long commands in their entirety. Furthermore, the command completion facility allows the user to select from a list of appropriate methods available for a particular object. The design of the method names allows the user to find and enter the desired command in the minimum number of steps with the minimum amount of typing. For users who prefer a more traditional console application, the following methods launch a command based wizard that allows the user to select from numbered lists of possible actions.

- `Calc()`
- `Convert()`
- `Filter()`
- `Import()` and `Export()`
- `Delete()`
- `View()`

Not all the types of object have all these methods.

Another advantage of longer method names is that they make Python scripts more readable. Python scripts can be written using a conventional text editor or with an integrated development environment (IDE) such as Eclipse [15] or NetBeans [16], many of which allow the programmer to select an appropriate method or variable name from a list. However, IDEs are sophisticated, powerful programmers' tools that may not appropriate for all GUT users.