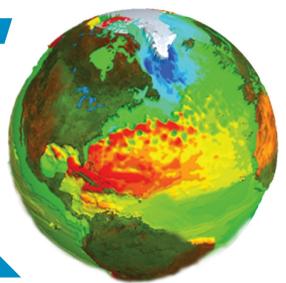


# BRAT

*BASIC RADAR ALTIMETRY TOOLBOX*



**Basic Radar Altimetry Toolbox v1.0**

**User Manual**



## Contents

<a href="#"><b>1. Introduction</b></a> .....	<b>1</b>
<a href="#"><b>2. Data read and processed</b></a> .....	<b>2</b>
<a href="#"><b>3. How to install BRAT</b></a> .....	<b>4</b>
3.1. Windows© binaries.....	<b>4</b>
3.2. Linux binaries.....	<b>4</b>
3.3. From source.....	<b>4</b>
<a href="#"><b>4. How to uninstall BRAT</b></a> .....	<b>5</b>
4.1. Windows© binaries.....	<b>5</b>
4.2. Linux binaries.....	<b>5</b>
4.3. From source.....	<b>5</b>
<a href="#"><b>5. BRAT Graphical User Interface (GUI)</b></a> .....	<b>6</b>
5.1. Overview.....	<b>6</b>
5.2. Starting with BRATGUI.....	<b>6</b>
5.2.1.Create a Workspace.....	<b>6</b>
5.2.2.Create a dataset.....	<b>7</b>
5.2.3.Create an operation.....	<b>9</b>
5.2.4.Create a view.....	<b>25</b>
<a href="#"><b>6. Visualisation interface</b></a> .....	<b>30</b>
6.1. ‘Y=F(X)’.....	<b>30</b>
6.2. ‘Z=F(lon, lat)’.....	<b>32</b>
6.2.1.Display properties.....	<b>33</b>
6.2.2.Color table editor.....	<b>35</b>
6.2.3.Contour table editor.....	<b>38</b>
<a href="#"><b>7. Using BRAT in ‘command lines’ mode with parameter files</b></a> .....	<b>40</b>
7.1. Creating an output NetCDF file.....	<b>40</b>
7.2. Visualising an output NetCDF file through BRAT.....	<b>41</b>
7.3. Using the parameter files to process many datasets.....	<b>42</b>
<a href="#"><b>8. BRATHL Application Programming Interfaces (APIs)</b></a> .....	<b>45</b>
8.1. Data reading function.....	<b>45</b>
8.2. Cycle/date conversion functions.....	<b>45</b>
8.3. Date conversion/computation functions.....	<b>46</b>
8.4. Named structures.....	<b>47</b>
<i>Annex A. List of datasets read by BRAT</i> .....	<b>49</b>
<i>Annex B.Y=F(X) parameter file keys</i> .....	<b>53</b>
<i>Annex C.Z=F(X,Y) parameter file keys</i> .....	<b>55</b>
<i>Annex D.Display parameter file keys</i> .....	<b>58</b>
<i>Annex E.BRATHL-IDL API</i> .....	<b>62</b>
<i>Annex F.BRATHL-MATLAB API</i> .....	<b>75</b>
<i>Annex G.BRATHL-Fortran API</i> .....	<b>87</b>
<i>Annex H.BRATHL-C API</i> .....	<b>92</b>

## 1. Introduction

The Basic Radar Altimetry Toolbox is a collection of tools and tutorial documents designed to facilitate the use of radar altimetry data. The toolbox is able to read most distributed radar altimetry data, from ERS-1 & 2 (ESA), Topex/Poseidon (NASA/CNES), Geosat Follow-On (US Navy), Jason-1 (CNES/NASA), Envisat (ESA) and the future Cryosat (ESA) missions, to process and edit it to some extent and to generate statistics and visualise the results.

The BRAT package contains a set of libraries, command line tools and application program interfaces (APIs), along with a Graphical User Interface.

Its main functions are:

- **Data Import and Quick Look:** basic tools for extracting data from standard formats and generating quick-look images.
- **Data Export:** output of data to NetCDF (ASCII dump provided). Raster images (png, jpeg, bmp, tiff, ppm) can be saved.
- **Statistics:** calculation of statistical parameters from data.
- **Combinations:** computation of combinations of data fields (and saving of those formulas)
- **Resampling:** over and under sampling of data; data binning.
- **Data editing:** data selection using simple criteria, or a combination of criteria (that can also be saved)
- **Exchanges:** data editing and combinations can be exchanged between users
- **Data visualisation:** Display of results, with user-defined preferences. The viewer enables the user to display data stored in the internal format (NetCDF).

The Toolbox is designed with a graphical user interface that enables the operator to easily specify the processing parameters required by each tool. Those parameters can also be defined directly in parameter files. APIs are available with data reading, date and cycle/pass conversion and statistical computation functions for C, IDL and Matlab. For beginners, we recommend using the GUI.

BRAT is provided as open source software, enabling the user community to participate in further development and quality improvement.

## 2. Data read and processed

The Basic Radar Altimetry Toolbox is able to read most distributed radar altimetry data, from ERS-1 & 2 (ESA), Topex/Poseidon (NASA/CNES), Geosat Follow-On (US Navy), Jason-1 (CNES/NASA), Envisat (ESA) and the future Cryosat (ESA) missions. The different types of data readable and processed by the Basic Radar Altimetry Toolbox are listed below (for a description of the exact datasets with their nomenclature, see [Annex A](#)). Note that data stored in arrays (e.g. waveforms) are not available individually (i.e. you can't access one value in the array) through the Graphical User Interface, but "only" through the API ([8.BRATHL Application Programming Interfaces \(APIs\)](#)), except for high-resolution GDR data (10, 18 and 20-Hz data) that you can access individually via the GUI.

### Level 1B/2 data products

Data	satellite(s)	Data center	format
Level 1B & level 2*	Cryosat*	ESA	ESA PDS
RA-2 wind/wave product for Meteo Users (RA2_WWV_2P)	Envisat	ESA	ESA PDS
RA-2 Fast Delivery Geophysical Data Record (RA2_FGD_2P)	Envisat	ESA	ESA PDS
RA-2 Geophysical Data Record (RA2_GDR_2P)	Envisat	ESA	ESA PDS
RA-2 Intermediate Geophysical Data Record (RA2_IGD_2P)	Envisat	ESA	ESA PDS
RA-2 Sensor Data Record (RA2_MWS_2P)	Envisat	ESA	ESA PDS
Interim Geophysical data record (IGDR)	Jason-1, Topex/Poseidon	AVISO PO.DAAC	binary
Geophysical data record (GDR)	Jason-1, Topex/Poseidon	AVISO PO.DAAC	binary
Operational Sensor Data Record (OSDR)	Jason-1	AVISO PO.DAAC	binary
Sensor Geophysical data record (SGDR)	Jason-1	AVISO PO.DAAC	binary
RA OPR	ERS-1 and 2	CERSAT	ESA PDS
Geophysical data record (GDR)	GFO	NOAA	binary

### Higher-level products

Data	satellite(s)	Data center	format
Along-track Delayed-Time and Near Real Time Sea Level Anomalies (DT- & NRT-SLA) (Ssalto/Duacs multimission products)	Cryosat*, Jason-1, Topex/Poseidon, GFO, Envisat, ERS-2, ERS-1	AVISO	NetCDF
Along-track Delayed-Time and Near Real Time Absolute Dynamic Topography (DT- & NRT-ADT) (Ssalto/Duacs multimission products)	Cryosat*, Jason-1, Topex/Poseidon, GFO, Envisat, ERS-2, ERS-1	AVISO	NetCDF
Gridded Delayed-Time and Near Real Time Maps of Sea Level Anomalies (DT- & NRT-MSLA) (Ssalto/Duacs multimission products)	Cryosat*, Jason-1, Topex/Poseidon, GFO, Envisat, ERS-1 & -2, merged	AVISO	NetCDF

\* Data to be published in the future

## Basic Radar Altimetry Toolbox User Manual

Gridded Delayed-Time and Near Real Time Maps of Sea Level Anomalies mapping error (DT- & NRT-MSLA) (Ssalto/Duacs multimission products)	Cryosat*, Jason-1, Topex/Poseidon, GFO, Envisat, ERS-2, ERS-1, merged	AVISO	NetCDF
Gridded Delayed-Time and Near Real Time Maps of Sea Level Anomalies geostrophic velocities (DT- & NRT-MSLA) (Ssalto/Duacs multimission products)	Cryosat*, Jason-1, Topex/Poseidon, GFO, Envisat, ERS-2, ERS-1, merged	AVISO	NetCDF
Gridded Delayed-Time and Near Real TimeMaps of Absolute Dynamic Topography (DT- & NRT-MADT) (Ssalto/Duacs multimission products)	merged	AVISO	NetCDF
Delayed-Time and Near Real Time Absolute Dynamic Topography geostrophic velocities (DT- & NRT-MADT) (Ssalto/Duacs multimission products)	merged	AVISO	NetCDF
Along-track Delayed-Time Sea Level Anomalies (DT-SLA) (monomission product)	Cryosat*, Jason-1, Topex/Poseidon, Envisat, ERS-2	AVISO	NetCDF
Along-track Delayed-Time Corrected Sea Surface Height ( DT-CorSSH) (monomission product)	Cryosat*, Jason-1, Topex/Poseidon, Envisat, ERS-2	AVISO	NetCDF
Along-track Sea Surface Height Anomalies ( AT-SSHA)	Topex/Poseidon, Jason-1	PO.DAAC	binary
Along-track Gridded Sea Surface Height Anomalies (ATG-SSHA)	Topex/Poseidon, Jason-1	PO.DAAC	binary
Gridded Near Real Time Maps of Significant Wave Height (NRT-MSWH ) (mono- and multi-mission products)	Jason-1, Topex/Poseidon, Envisat, GFO, merged	AVISO	NetCDF
Gridded Near Real Time Maps of Wind Speed modulus (NRT-MWind)	Jason-1, Topex/Poseidon, Envisat, GFO, merged	AVISO	NetCDF
Heracles along-track land-ice (multimission products)*	Cryosat*, Envisat	ESA	NetCDF
Heracles crossover land-ice (multimission products)*	Cryosat*, Envisat	ESA	NetCDF
Gridded Heracles SHA land-ice (multimission products)*	Cryosat*, Envisat, merged	ESA	NetCDF
Gridded Heracles Sigma0 land-ice (multimission products)*	Cryosat*, Envisat, merged	ESA	NetCDF
Gridded Heracles Leading Edge Width (LEW) land-ice (multimission products)*	Cryosat*, Envisat, merged	ESA	NetCDF

\* Data to be published in the future

### **3. How to install BRAT**

BRAT binaries are available for Windows<sup>©</sup> XP and Linux (Redhat EL4).

The software can be delivered in two ways: either a single installation package file (for Windows<sup>©</sup> or Linux), or a CD ROM version (containing the installation files for all the platforms but only one version of the common files).

The names of the installation packages have the format: brat-VERSION-PLATFORM-installer.extension, where VERSION is the release of brat, PLATFORM is the destination platform (Windows<sup>©</sup> or Linux) and extension is a more or less platform-specific extension (.exe for Windows<sup>©</sup>, .bin for Linux).

In the CD-ROM, the names of the installation programs have the format setup-PLATFORM.extension, with the same convention as above.

#### **3.1. Windows<sup>©</sup> binaries**

Double-click on the installation package or installation program and follow the instructions.

By default, the software will be put in C:\Program Files\brat-VERSION\ if you have write access to this folder or in your user profile (normally C:\Documents and settings\ACCOUNT\brat-VERSION).

You may choose another folder if you want to.

#### **3.2. Linux binaries**

Execute the installation package or installation program from your file navigator or a console window and follow the instructions. If you procured the installation package via a network it may not be set as executable: you should then run the command 'chmod +x brat-VERSION-linux-installer.bin' in order to make it executable.

By default, the software will be put in /usr/local if you have root permissions or in \$HOME/brat-VERSION.

You may choose another folder if you want to.

#### **3.3. From source**

All installation packages include the source files. If you want to or need to rebuild BRAT, you may install only the source files. To do this first look in the chosen folder and read the README and INSTALL files. The latter one gives you information about the dependencies (what must be installed before) and the specific options which may be used.

As a convenience, you *may* have been given a DVD ROM containing almost all the tools needed, but otherwise everything can be downloaded from the website for each software package.

## 4. How to uninstall BRAT

When BRAT has been installed the whole installation process is registered and everything created at that time can be removed (but not what may be created after that).

### 4.1. Windows® binaries

Go to the control panel, click on ‘Add/Remove programs’ and select the Brat entry. Everything created during installation will then be removed.

Note that there is also a shortcut in the installation folder which you can click on to get the same result.

### 4.2. Linux binaries

In the installation folder (the default one or the one chosen), there is a script called `uninstall-brat-VERSION-linux` which can be executed to remove everything created during the installation.

There is also a shortcut, called ‘Uninstall Basic Radar Altimetry Toolbox.desktop’, which can be used from within your desktop manager (KDE, GNOME) to get the same result.

### 4.3. From source

If you have built BRAT from source code, you may have to remove everything by hand. The folder containing the source files may be deleted entirely

## 5. BRAT Graphical User Interface (GUI)

### 5.1. Overview

The BRAT Graphical User Interface (GUI) is a windowed interface to the BRAT Tools. Note that not all tool functions are accessible from the GUI (some options are only available using the command files directly).

If your screen is 768 pixels high, you might not see the very bottom of the window. You can then maximize it, and you should then see the whole window.

The GUI manipulates objects called 'workspaces'. A workspace contains

- datasets, which are collections of files of the same kind,
- operations, for reading and/or processing and/or selecting data within a dataset,  
An operation produces an intermediate file (NetCDF) and a command file,
- formulas, to enable you to use pre-defined combinations of data fields or to define them yourself and re-use them later.
- views, that plot results of one or more operations  
A 'view' produces a command file and opens the visualisation tool (see chapter 6)

A workspace can be saved for later use. Some or all elements of a workspace can be imported into another workspace.

The 'Logs' tab displays the state of the programmes being run. Several operations and views can be executed at the same time.

When launched, the GUI recalls the last used Workspace, or asks for a new one if no valid one is available. There is no specific tab for the Workspace, only the menu the furthest to the left.

### 5.2. Starting with BRATGUI

#### 5.2.1. Create a Workspace

When you open BRATGUI, the software asks for the name and location of the 'workspace' you will be working in. If there are already one or more workspaces, the last used one is opened by default. You can open another one, or create a new one by choosing 'new' in the 'workspace' menu (the furthest to the left)

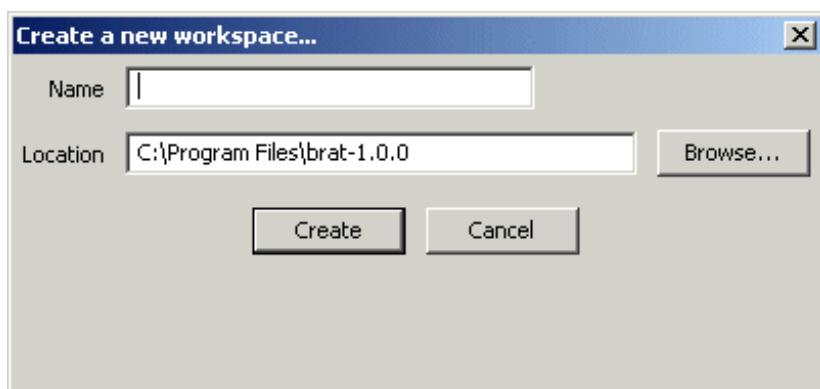


Figure 1: Creating a new workspace window

**It is highly recommended to save the workspace (ctrl+s, or 'save' in the workspace menu) while working.** The workspace will nevertheless be saved when you quit BRATGUI (or, more accurately, you will then be asked whether or not you wish to save the workspace).

You can delete an existing workspace by choosing ‘delete’ in the ‘workspace’ menu, but note that you can only use BRATGUI within a workspace and so you will have to create a new one if none are available.

Items in the ‘Workspace’ menu are:

- **New**: creates a new workspace
- **Open**: opens a previously saved workspace
- **Save**: (or ctrl+s) saves the current workspace and all its datasets, operations, formulas and displays (views)
- **Import**: imports all (datasets, operations, formulas and/or displays) of a previously saved workspace
- **Rename**: renames the current workspace
- **Delete**: deletes the current workspace
- **Recent workspaces**: lists the 2 most recently used workspaces

### 5.2.2. Create a dataset

The first tab opened is ‘datasets’.

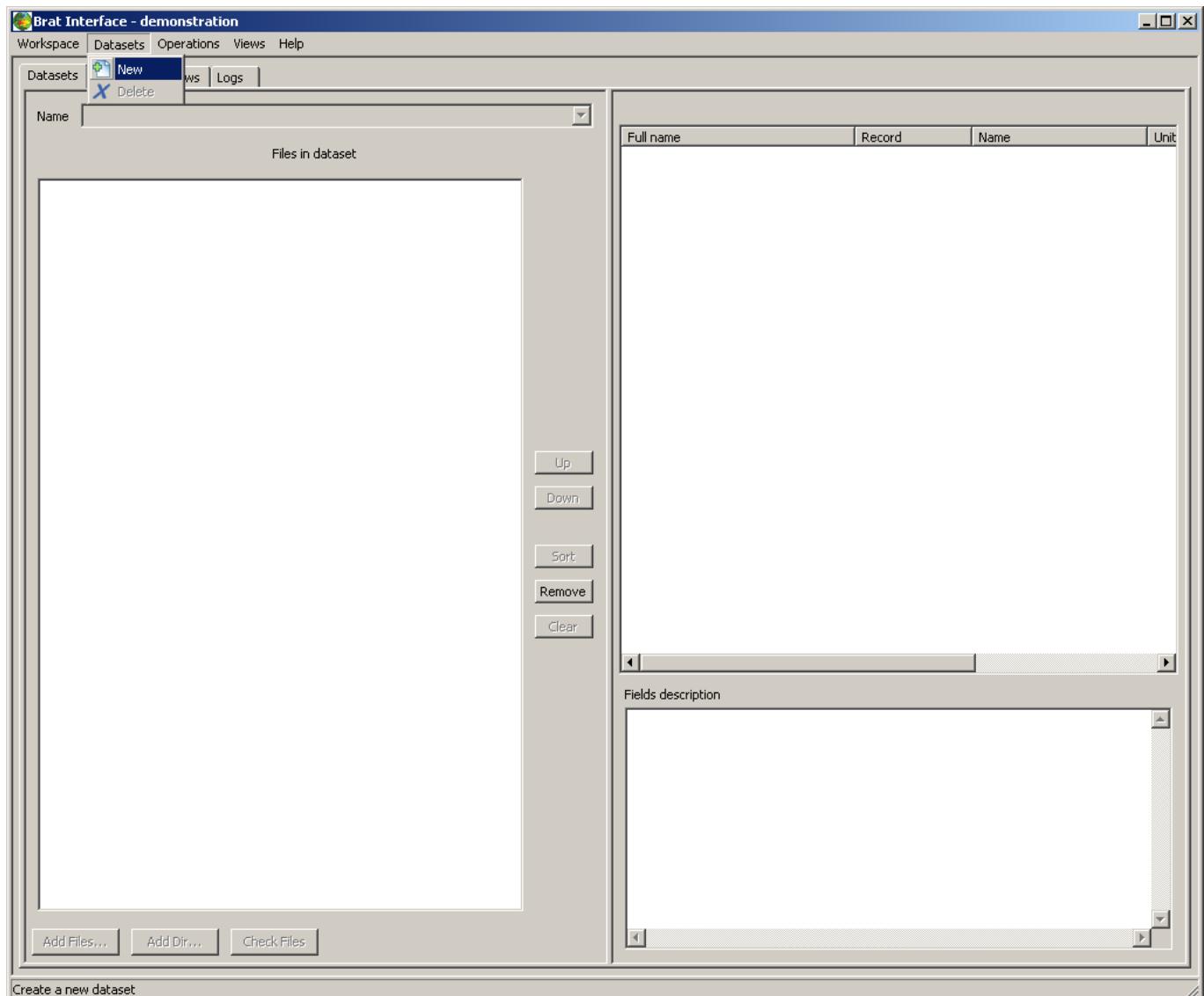


Figure 2: Creating a new dataset

## Basic Radar Altimetry Toolbox User Manual

Choose ‘new’ in the ‘datasets’ menu if no datasets are available or if you wish to work on other data than the one already selected. You can delete an existing dataset by choosing ‘delete’ in the ‘datasets’ menu, but note that you can only use BRATGUI with a dataset which has been defined and so you will have to create a new one if none are available.

The ‘Name’ dropdown list contains all the defined dataset names and allows you to select and rename a dataset. You have to give the dataset a name (with no spaces or special characters in the name). If you change the name within the ‘name’ box, and press the Enter key it **renames** your dataset.

Note that only coherent datasets are possible (i.e. same format, same data product). Use the ‘Check files’ button at the bottom of the window to check for coherence.

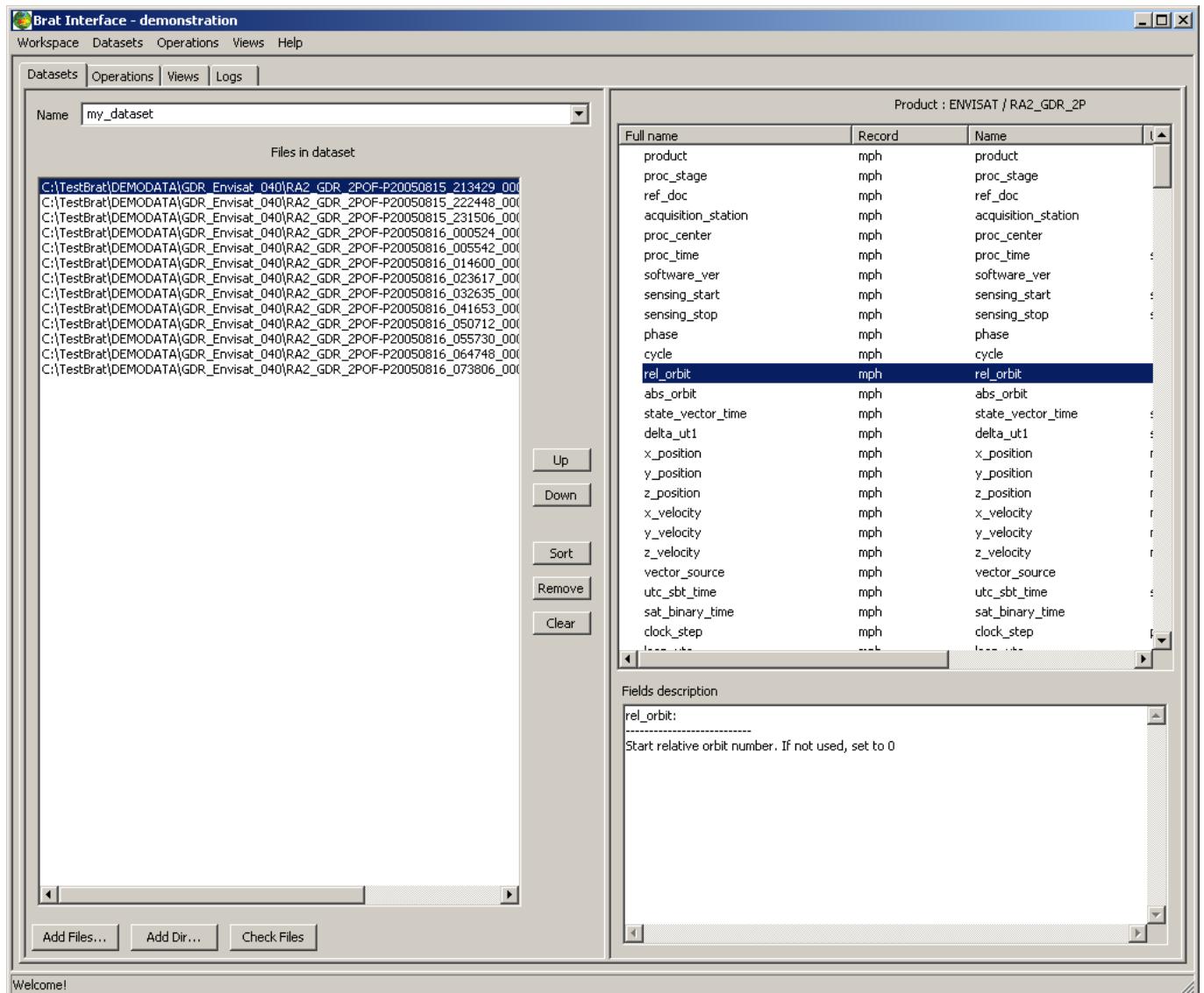


Figure 3: A dataset. On the left, the list of files; right (top) the list of available fields for the selected file format, right (bottom), the description of the selected field as it appears in the data dictionary.

When you have created your dataset and named it, you can then add files, chosen from your hard drive, CD/DVD driver, local network or other medium. If you wish to add a long list of files, the ‘add dir’ button allows you to choose all of the files by simply choosing the folder in which they are stored.

The ‘Clear’ button will remove the **whole** list.

You can delete selected file(s) by using the ‘Remove’ button, or the ‘delete’ key on your keyboard.

The ‘Add files’ button (at the bottom of the window) enables you to select those data files you wish to work on. If there are a lot of files, you should preferably select a whole folder by clicking on ‘Add Dir’

The ‘Up’ ‘Down’ and ‘Sort’ buttons are useful when the order in which files are processed is important (e.g. subtracting one file from another). ‘Up’ moves the selected file upwards in the list, ‘Down’ moves it

downwards. ‘Sort’ puts the whole list into alphabetical or numerical order. It can also be used to check for two occurrences) of the same file, or missing files, or to remove unwanted files from a list.

With this tab window,

- The selected files’ names are on the left;
- The right-hand display lists all fields defined for this kind of data and, below that is a more detailed description of the selected field (extracted from the data dictionary). You can sort the fields alphabetically by clicking on ‘name’, ‘record’, ‘unit’, ‘format’, or ‘dim’ (off screen), at the top of the box, or view a field whose name begins with one or more letters by typing them (fast).

The list of all the fields of the currently selected file is divided into 6 columns:

- **Full name:** the fully described name in the file structure hierarchy and related to the record.
- **Record:** the record containing the field. Many files have only ‘header’ and ‘data’ records while others have more (e.g. Envisat ones)
- **Name:** the short field name
- **Unit:** the unit of the field
- **Format:** the format of the field inside the file. In BRAT all fields are read as floating-point values (double).
- **Dim:** Dimension of the field (number of values in arrays, if the data is stored in an array)

Under the list there is a text box with a detailed description of the currently selected field (as extracted from the data dictionary)

You may define as many datasets as you wish.

Note that if you want the same operation to be applied to several files separately, you will have to define several datasets, or use the parameter files directly with a script (see section 7.3).

### 5.2.3. Create an operation

The second tab is ‘operations’.

## Basic Radar Altimetry Toolbox User Manual

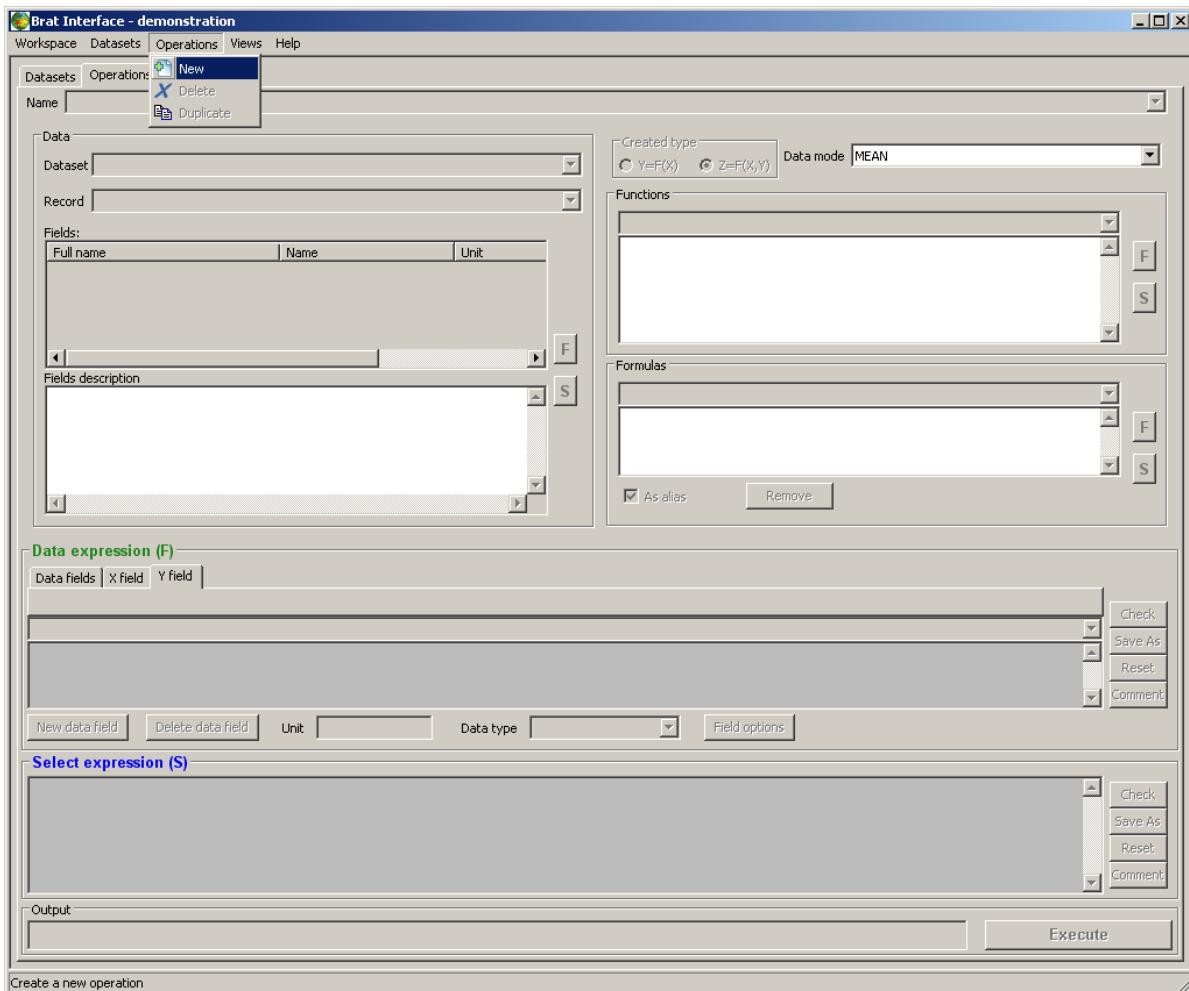


Figure 4: Creating a new 'operation'

If none exist, you have to create a new operation (you may give it any name, but there should be no spaces or special characters in the name). Choose 'new' in the 'Operations' menu.

You may delete an existing operation by choosing 'delete' in the 'Operations' menu, but note that you can only process data with BRATGUI with an operation which has been defined and so you will have to create a new one if none exist.

Otherwise, you may work with a previously saved operation. The 'Name' dropdown list contains all the defined operation names which can be selected and renamed. If you change the name within the 'name' box, it renames your operation and erases the command file that was created with the previous name.

If you want to apply the same operation to different datasets, you will have to re-create it as many times as needed. Use the 'Duplicate' function in the operation menu to duplicate the operation exactly. You can also use the parameter files directly with a script (see section 7.3).

### 5.2.3.1. Data

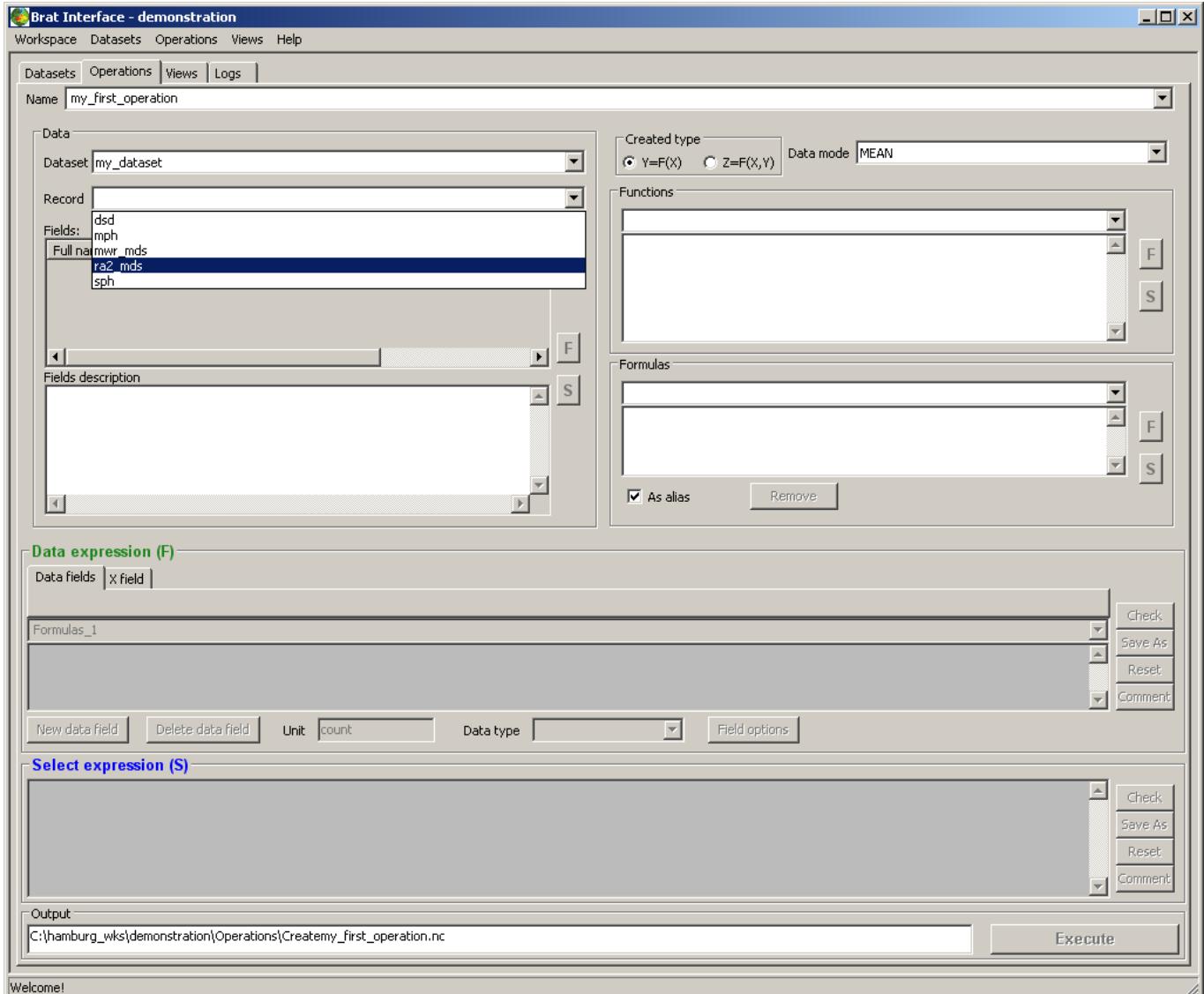


Figure 5: Choosing a record within the selected dataset

Choose a dataset from the list of existing datasets to which to apply the operation. Choose a record within the dataset, containing the data fields which interest you. The description of each field is given below when you click on a particular field.

The 'F' and 'S' button are used to insert a field in either the selected 'data expression (F)' or the selected 'Select expression (S)'

### 5.2.3.2. Created type

You can then decide whether you prefer to choose a 'Y=F(X)' or a 'Z=f(X,Y)' type of operation, i.e.:

- '**Y=F(X)**',  
if you wish to work with one – or several – field(s) with respect to another one;  
typically, this leads to a curve kind of view.  
The BRATCreateYFX program will generate the output of the operation.
- '**Z=f(X,Y)**',  
if you wish to work with one – or more – field(s) with respect to two others;  
typically, with X=longitude and Y=latitude, this leads to a map (any field can be thus processed with respect to any two others – but, for now, it is only possible to display maps within BRAT).  
The BRATCreateZFXY program will generates the output of the operation.

## Basic Radar Altimetry Toolbox User Manual

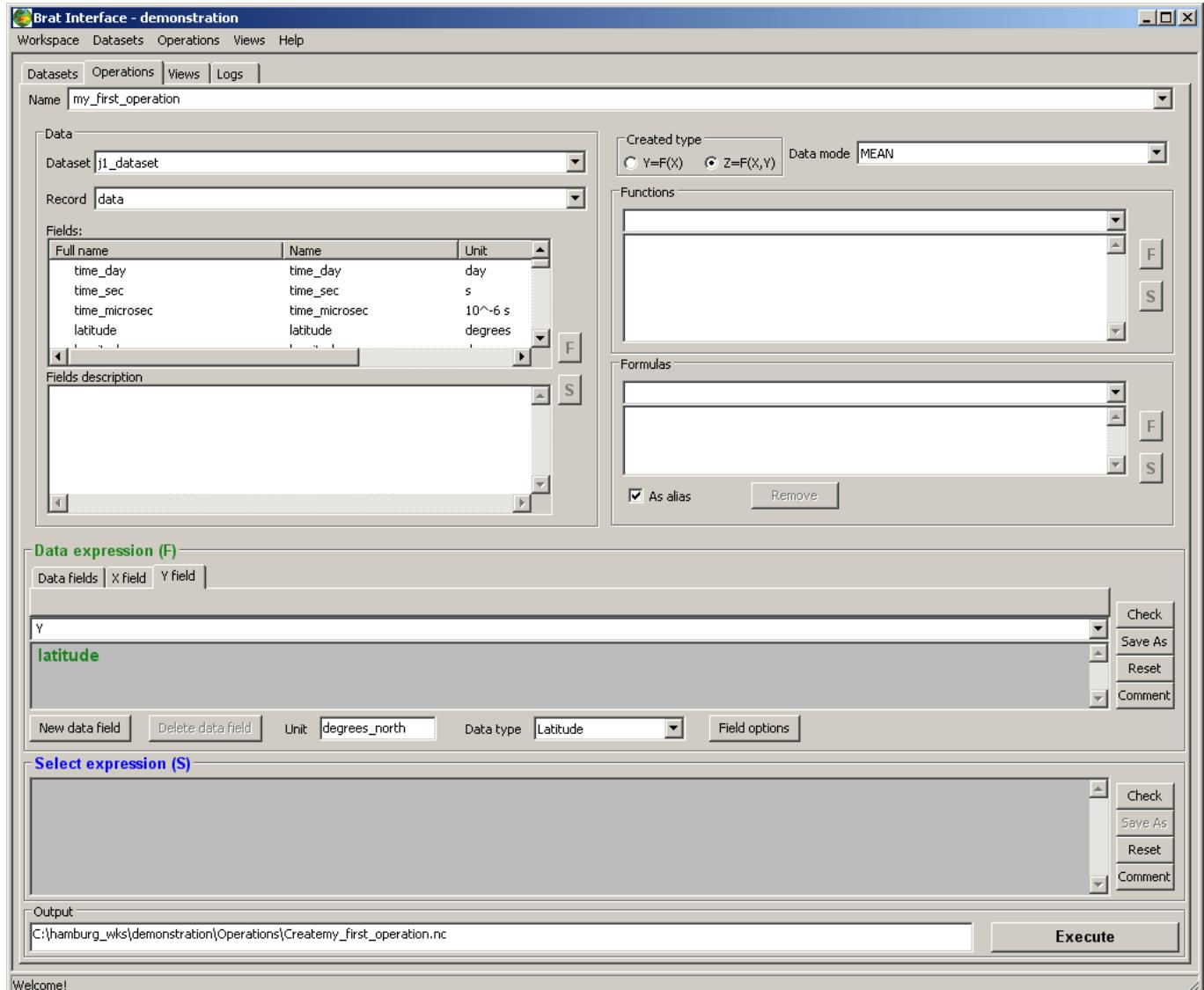


Figure 6: An 'Operations' tab window when the  $Z=F(X,Y)$  type is selected. Note the difference in the 'Data Expression box tabs (in the middle of the window, see section 5.2.3.5): in Figure 5, above, in which only two tabs were available when  $Y=F(X)$  was chosen (data field and X field). Here a third is visible, to define the Y field.

### 5.2.3.3. Data mode

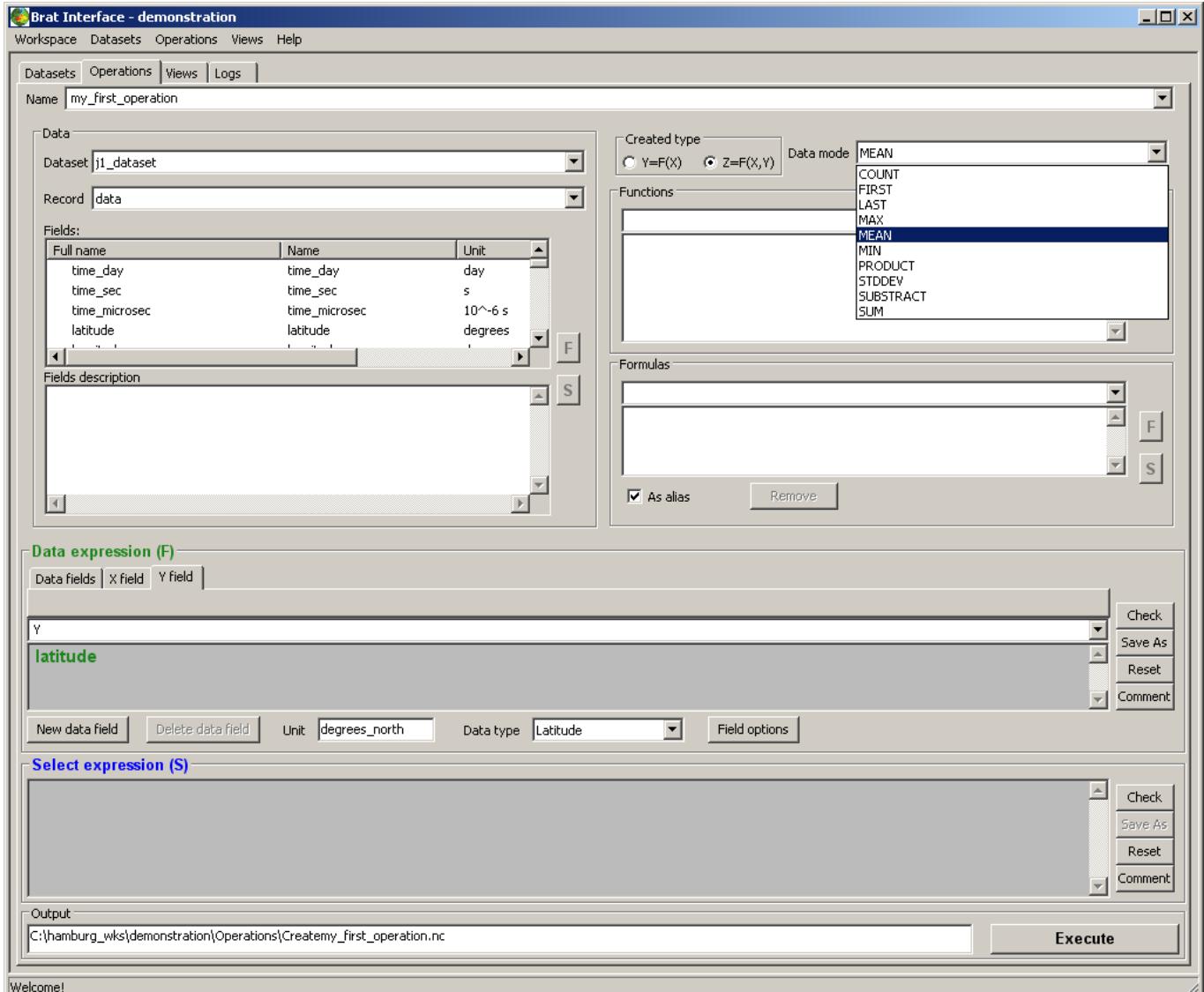


Figure 7: Choice of the data mode

The data mode is used when you have several values of a field for a given (X) or (X,Y). This is typically the case for:

- crossover points between tracks
- several files available for different dates
- sub-sample data

The possible modes are:

- **MEAN** (default) : computes the mean for all values of the field within the dataset at each X (or (X,Y))
- **COUNT**: returns the number of values of the field within the dataset at each X (or (X,Y))
- **FIRST**: returns the first encountered value of the field within the dataset (in the order of the list of files as it appears in the 'dataset' tab)
- **LAST**: returns the last encountered value of the field within the dataset (in the order of the list of files as it appears in the 'dataset' tab)
- **MAX**: gives the maximum value of the field within the dataset
- **MIN**: gives the minimum value of the field within the dataset
- **PRODUCT**: multiplies the selected field for each file within the dataset
- **STDEV**: computes the standard deviation for all values of the field within the dataset at each X (or (X,Y))

- **SUBTRACTION**: subtracts the selected field for each file from the first of the list (file order dependent)
- **SUM**: adds the selected field for each file

The data mode can be used

- to compute statistics.
- to do some arithmetic operations **between files** within a dataset (adding, subtracting or multiplying: SUM, SUBTRACTION, PRODUCT)
- but can also be used for the display (MEAN, FIRST, LAST, MIN, MAX), if you prefer to visualise, for instance, the last value rather than the mean one.

#### 5.2.3.4. Functions

The 'Functions' area provides a simple way of including (and knowing) the available functions and constants which can be used in formulas. By default, no functions are visible, but they appear in the dropdown list if you click on it. For each function, if selected, you will see a short explanation of what it does.

They are available for processing or selecting a data expression. Note that those functions will apply on every field with the same name, for every file within the dataset.

Name	Description	Syntax	Type
!	<b>logical negation operator NOT</b> The logical negation operator (!) reverses the meaning of its operand. The result is <i>true</i> if the converted operand is <i>false</i> ; the result is <i>false</i> if the converted operand is <i>true</i> .	! expr1	Logical operator
!=	<b>not-equal-to operator</b> The not-equal-to operator (!=) returns <i>true</i> if the operands do not have the same value; otherwise, it returns <i>false</i> A != B is true (when no default in A or B) if abs(A-B) >= epsilon	expr1 != expr2	Relational operator
&&	<b>logical AND operator</b> The logical AND operator (&&) returns the boolean value <i>true</i> if both operands are <i>true</i> and returns <i>false</i> otherwise. Logical AND has left associativity.	expr1 && expr2	Logical operator
	<b>logical OR operator</b> The logical OR operator (  ) returns the boolean value <i>true</i> if either one operand is <i>true</i> or both operands are <i>true</i> and returns <i>false</i> otherwise. Logical OR has left associativity	expr1    expr2	Logical operator
<	<b>less than</b> It yields values of the Boolean type. The value returned is <i>false</i> (0) if the relationship in the expression is <i>false</i> ; otherwise, the value returned is <i>true</i> (1).	arithmetic expr1 < arithmetic expr2	Relational operator
<=	<b>less than or equal to</b> It yields values of the Boolean type. The value returned is <i>false</i> (0) if the relationship in the expression is <i>false</i> ; otherwise, the value returned is <i>true</i> (1).	arithmetic expr1 <= arithmetic expr2	Relational operator
==	<b>equal-to operator</b> A == B is true (when there is no default in A or B) if abs(A-B) < epsilon	==	Relational operator

	The equal-to operator returns true (1) if both operands have the same value; otherwise, it returns <i>false</i> (0).		
>	<b>greater than</b> It yields values of the Boolean type. The value returned is false (0) if the relationship in the expression is false; otherwise, the value returned is true (1).	arithmetic expr1 > arithmetic expr2	Relational operators
>=	<b>greater than or equal to</b> It yields values of the Boolean type. The value returned is false (0) if the relationship in the expression is false; otherwise, the value returned is true (1).	arithmetic expr1 >= arithmetic expr2	Relational operators
~	<b>bitwise not operator</b> Takes the value as an integer (a default value if the floating point one is outside the integer range) and reverses each bit.	~ expr1	bitwise operator
&	<b>bitwise and operator</b> Takes the value of each operand as an integer (a default value if the floating point one is outside the integer range) and does an <i>and</i> operation on each corresponding bit <i>And</i> operation: 0011 & 0101 = 0001	expr1 & expr2	bitwise operator
	<b>bitwise or operator</b> Takes the value of each operand as an integer (a default value if the floating point one is outside the integer range) and does an <i>or</i> operation on each corresponding bit <i>Or</i> operation: 0011   0101 = 0111	expr1   expr2	bitwise operator
()	<b>parenthesis operator</b> Isolates an expression (or a sub expression) in order to take it as a whole. Exemple: A * (B + C) multiplies (B + C) by A. without parentheses, B would be multiplied by A and then C added	(expr1)	
DV	Default value	DV	number
PI	PI value	PI	number
PI2	PI/2 value	PI2	number
PI4	PI/4 value	PI4	number
abs	<b>absolute value</b> Calculates the absolute value.	abs(param1)	
ceil	<b>ceiling of a value</b> Calculates the ceiling of a value.	ceil(param1)	
cos	<b>cosine (radian)</b> Calculates the cosine (radian) of a value.	cos(param1)	
cosd	<b>cosine (degree)</b> Calculates the cosine (degree) of a value.	cosd(param1)	
deg2rad	<b>Translates Degree to Radian.</b>	deg2rad(param1)	conversion
deg_normalize	<b>Normalizes longitude (degree)</b> Z = deg_normalize(X, Y) returns a value which makes the following expressions true: Z = Y + n*360, X <= Z < X+360	deg_normalize(param1, param2)	conversion
dv (DV)	<b>Default value</b>	DV	number
exp	<b>exponential</b> Calculates the exponential.	exp(param1)	
floor	<b>floor of a value</b> Calculates the floor of a value	floor(param1)	

frac	<b>fractional parts</b> Calculates the fractional parts of a value.	frac(param1)	
iif	<b>Inline if</b> If the first parameter is true (not 0 and not default value), the second parameter is returned, otherwise the third one is returned. Logically equivalent to: <pre>if (param1 is true)     return param2 else     return param3 end if</pre>	iif(param1, param2, param3)	
iif3	<b>Inline if with default value case</b> If the first parameter is true (not 0 and not default value), the second parameter is returned. If it is 0, the third one is returned, otherwise (it is a default value) the fourth one is returned. Logically equivalent to: <pre>if (param1 is default value)     return param4 else     if (param1 is true)         return param2     else         return param3     end if end if</pre>	iif3(param1, param2, param3, param4)	
int	<b>integer parts</b> Calculates the integer parts of a value.	int(param1)	
is_bounded	<b>Checks whether a value x is included between two values</b> (min/max). is_bounded(min, x, max)	is_bounded(param1 ,param2,param3)	
is_bounded_strict	<b>Checks whether a value x is strictly included between two values</b> (min/max). is_bounded_strict(min, x, max)	is_bounded_strict(p aram1,param2,para m3)	
is_default	<b>Checks whether a value is a default value</b> (1: yes, 0: no)	is_default(param1)	
log	<b>logarithm</b> Calculates the logarithm of a value	log(param1)	
log10	<b>base-10 logarithm</b> Calculates the base-10 logarithm of a value	log10(param1)	
max	<b>Maximum</b> Calculates the larger of two values	max(param1,param 2)	
min	<b>Minimum</b> Calculates the smaller of two values	min(param1,param 2)	
mod	<b>floating-point remainder</b> Calculates the floating-point remainder	mod(param1,param 2)	
rad2deg	<b>Translates Radian to Degree</b>	rad2deg(param1)	conversion
round	<b>rounded value</b> Calculates the rounded value	round(param1)	
sign	<b>Checks the sign</b> of a value (-1: negative, 1: positive or zero)	sign(param1)	
sin	<b>sine (radian)</b>	sin(param1)	

	Calculates the sine (radian) of a value.		
sind	<b>sine (degreee)</b> Calculates the sine (degreee) of a value.	sind(param1)	
sqr	<b>square</b> Calculates the square of a value.	sqr(param1)	
sqrt	<b>square root</b> Calculates the square root of a value.	sqrt(param1)	
tan	<b>tangent (radian)</b> Calculates the tangent (radian) of a value.	tan(param1)	
tand	<b>tangent (degree)</b> Calculates the tangent (degree) of a value.	tand(param1)	
to_date	<b>Date formats conversion</b> Translates a string value into a date value Allowed formats are: YYYY-MM-DD HH:MN:SS.MS string. For instance: '1995-12-05 12:02:10.1230' '1995-12-05 12:02:10' '1995-12-05'  a Julian string: format:positive 'Days Seconds Microseconds' Seconds must be strictly less 86400 and Microseconds must be strictly less than 1000000 For instance: '2530 230 4569'  a Julian string: format:positive decimal Julian day For instance: '850.2536985'  For Julian string, it can contain its reference date at the end by specifying @YYYY where YYYY is the reference year that's must be one of 1950, 1958, 1985, 1990, 2000 The reference year YYYY stands for YYYY-01-01 00:00:00.0 If no reference date is specified the default reference date (1950) is used. For instance: '2530 230 4569@2000' '850.2536985@1990' '850.2536985@1950' is equal to '850.2536985'  Dates prior to 1950-01-01 00:00:00.0 are invalid	to_date(param1)	conversion

*NOTE: except when explicitly stated (as with iif3, is\_default) every expression involving a default value (also called missing value) is a default value. A true expression is an expression which is not 0 and not a default value. The descriptions below are for expressions which do not contain default values (to simplify their writing). For example the result of 'A || B' (A or B) is a default value if B is one even if A is true. 0 and default values are considered as false values (! X is a default value if X is also one, so X is false and ! X too).*

The 'F' and 'S' button are used to insert a function in either the selected 'data expression (F)' or the selected 'Select expression (S)'. The function will appear there with the correct syntax (e.g. function(param1,param2); you then have to replace 'param1', 'param2' etc. by the fields or numbers you wish to apply the function to).

You can use those functions for, among others:

- a test on a flag: `Surface_type == 0` will return only the 'open ocean' flagged Jason-1 GDR data
- boundaries: `is_bounded(-100,SSH,100)` (or: `(SSH >= -100) && (SSH <= 100)`)

### 5.2.3.5. Data expression

When your dataset and the type of operation are chosen, you have access to the definition of the Data expression. You have two or three tabs (Fields, X and Y), depending on the type of operation. The dropdown list contains the names of all the expressions while the text box below allows the value of the expression to be written.

A data expression can be:

- only one field in a dataset (typically, for a map, longitude as x-axis, latitude as y-axis)
- a combination of fields, either `+, -, *, /`, or by using the available functions in the list to the right of the 'data' box (see 5.2.3.4).
- a pre-set combination of fields among the ones you will find in the 'formulas' box (see 5.2.3.7)

Note that, **by default, X is set to longitude and Y to latitude**. You may change this if you want to.

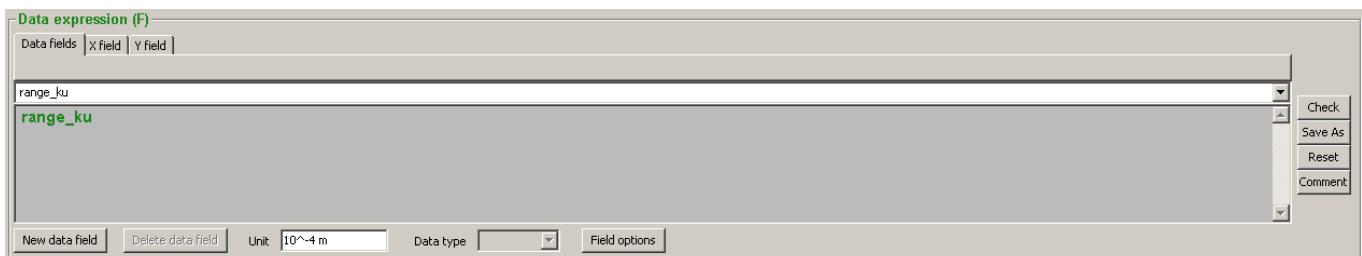


Figure 8: A data expression box with a field included (there is only one data field defining this expression), in the  $Z=F(X,Y)$  case. Note the Unit (default unit as defined in the dataset); if preferred, you could type in 'm' or 'km'. The 'New data field' button enables you to define more than one field to process with respect to the defined X and Y axis.

Give a name to your data expression (for instance, 'my\_first\_field'). Note that if you change the name, it renames your expression. This will then be the default name on the plots, near to the scale if you have not given a title to your field (in the options).

Type your expression (the name of one data field, or a combination of several) in the box below. Alternatively, to appear in the same box, you can select either:

- a field within the dataset/record by selecting it in the list (you can sort the list alphabetically in each column, or type in the first letters to find the right one) and then clicking on the 'F' button to the right of the data fields list to have it inserted in place of your cursor in the data expression box.
- or a ready-made expression by selecting it in the 'formulas' list and then clicking on the 'F' button to the right of the formulas box, or by typing in the alias.

Then click on 'field options'. There you can type in a title for your field. The title will be displayed as the default name of the field in the plots (if no title is entered, it will be the data expression name). If you choose a  $Z=F(X,Y)$  type, you can also choose to smooth and/or extrapolate the data by means of a Loess filter so as to obtain a fully colored plot (and not individual tracks or points). In that case, you will have to fill in the corresponding information in the X and Y fields, too.

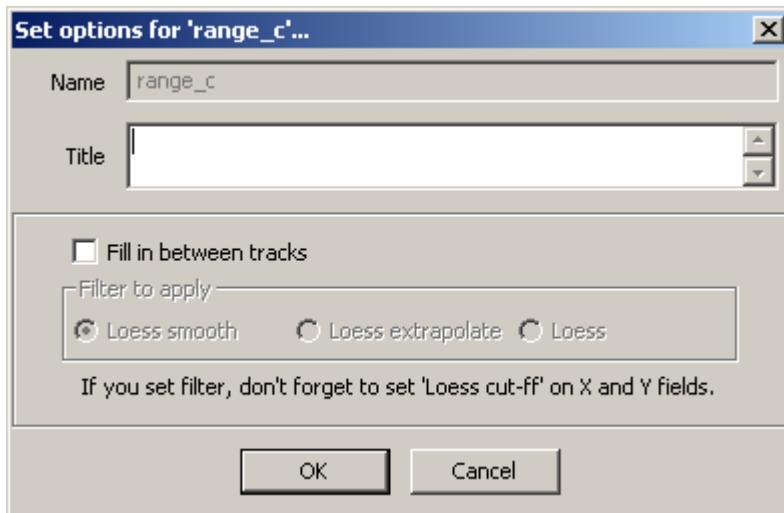


Figure 9: Option for a data field in the  $Z=F(X, Y)$  case.  
For the  $Y=F(X)$  case, only the top two boxes are displayed.

You can define **as many fields as you wish**, by clicking on the ‘new data field’ button and repeating the sequence above. Note that **you must define at least one field**. The choice of the X and Y axes and their options apply to all the fields within the current operation. You cannot choose (for example) different resolutions for X and Y for different fields within the same operation. The data mode is also the same for all fields.

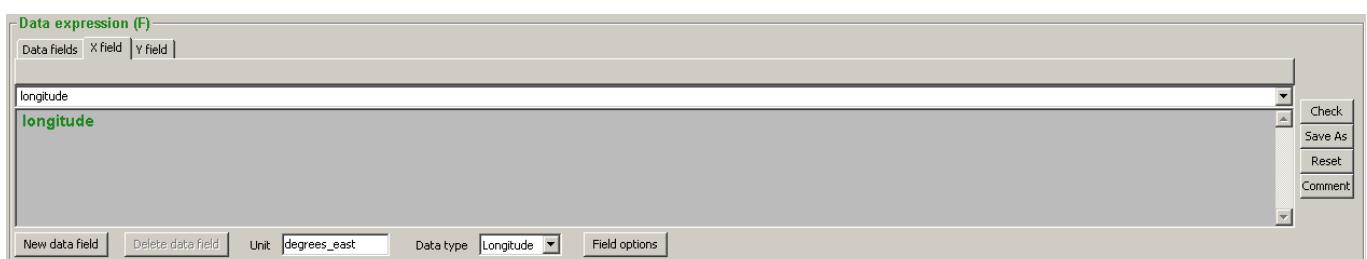


Figure 10: A data expression box with an X field included, for the  $Z=F(X, Y)$  case. Note the Unit (default unit as defined in the dataset) and the data type (used to define default values, min, step and max).

When choosing the field to use as X axis:

- click on the X ‘tab’,
- choose a name ('my\_x\_axis'),
- enter your expression or choose it from the lists (as for the field),
- click on ‘field options’. The option window will enable you to choose:
  - o the title of the axis,

If you choose a  $Z=F(X, Y)$  type, you will also be able to choose :

- o the minimum and maximum values of the axis (typically, this can be used to define a geographical sub-set)
- o the step value, to define the resolution (longitude-latitude) of the output file. The default value is  $1/5^\circ$ . However, note that the smaller the resolution, the longer will be the execution time for the operation.
- o the ‘Loess cut-off’ value, i.e. the number of grid points before the Loess filter becomes equal to zero (odd number)

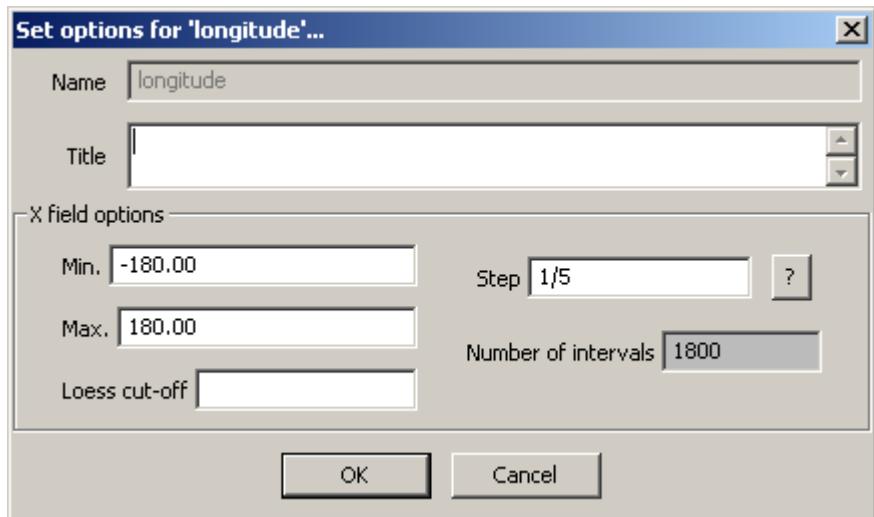


Figure 11: Option for an X field for the  $Z=F(X, Y)$  case. Note the min, max and step values.  
Loess cut-off is used when a filter has been defined in the data field options.  
For the  $Y=F(X)$  case, only the top two boxes (Name and Title) are displayed.

Typical Loess filter cut-off values depend on the Step you choose and on the kind of filter you have selected in your field (Smooth, Extrapolate or Loess). They are odd numbers (if you fill in an even number, the number used will be your number+1).

- **Smooth:** smoothes the values of the data where there are already data (i.e. it will not fill in gaps between tracks)
- **Extrapolate:** fills in the gaps between values (with some overlay on continents)
- **Loess:** smoothes and fills in the gap values (with some overlay on continents)

'Extrapolate tends' to keep data ground tracks visible. 'Smooth' spreads out the data, but tends to level the maxima and minima and to generate 'data' on continents from ocean-only measurements. 'Loess' does both extrapolation and smoothing.

The general rule is that the higher the cut-off value, the more spread out the data will be, since the radius of action of the filter will be greatest.

For good results to render along-track data, values of 31 begins to give rather correct results, even if they still show a hint of ground tracks.

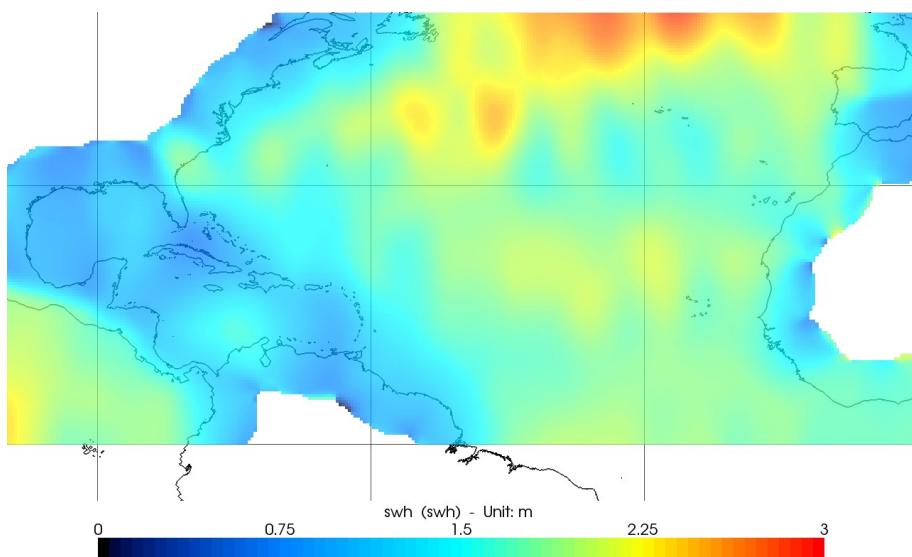
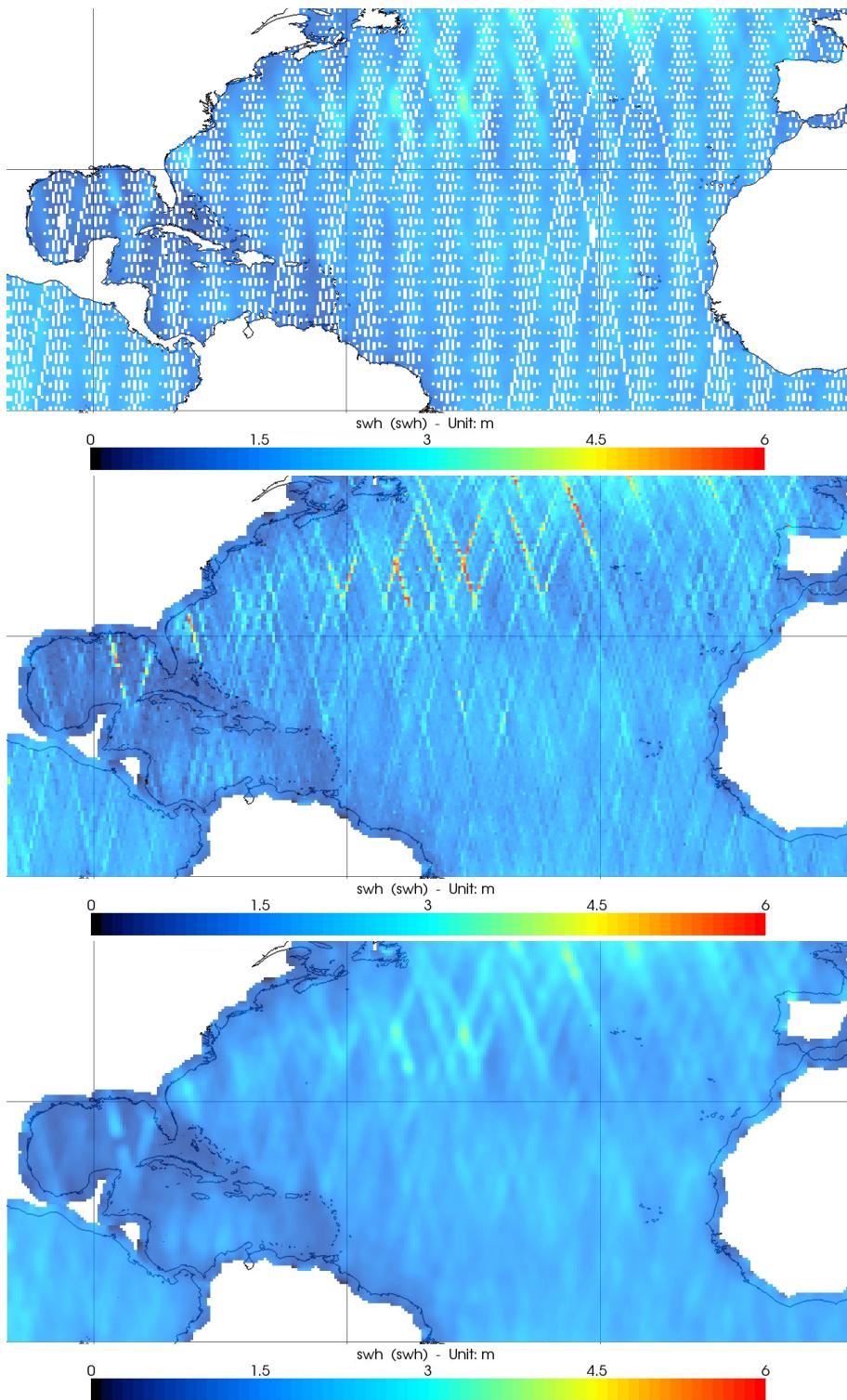


Figure 12: Envisat Significant wave height computed with a 'Loess' filter and a Loess cut-off value of 31 in longitude and latitude.



**Figure 13:** The same data (Envisat Significant wave height) computed with (top to bottom) ‘Smooth’, ‘Extrapolate’ and ‘Loess’ filters, and with (left) a Loess cut-off value of 9 (all with a resolution of  $1/3^\circ$ ). With a Loess cut-off of 9, tracks are still visible.

If you choose a  $Y=F(X)$  type, you can also choose the data type of the axis (lat, lon, time or X for any other kind of data used as X axis)

Note that **you must have defined one X axis**

If relevant ( $Z=F(X,Y)$ ), do the same also for the Y axis. Note that, if you choose  $Z=F(X,Y)$  you **must have defined one Y axis**.

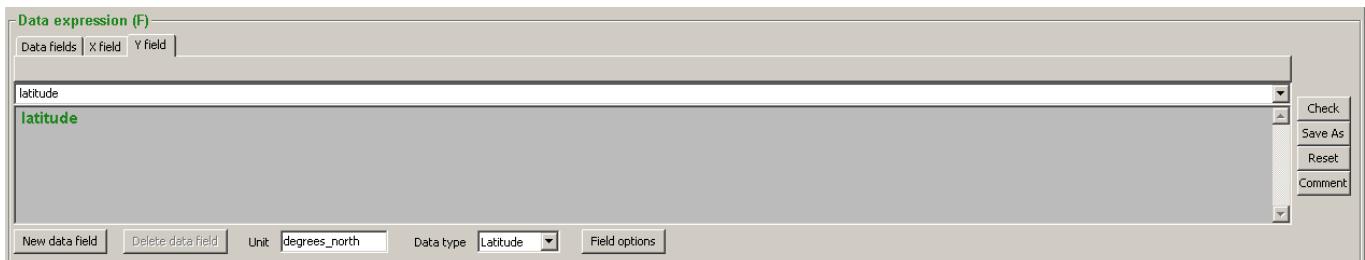


Figure 14: A data expression box with a Y field included. Note the Unit (default unit as defined in the dataset) and the data type (used to define default values, min, step and max).

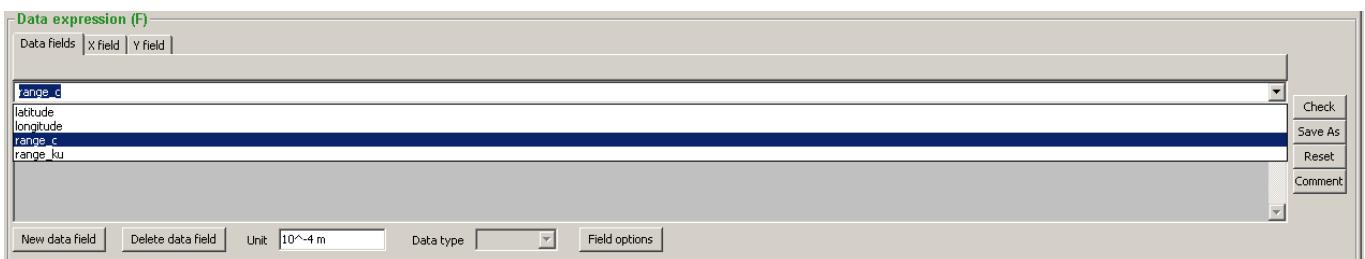


Figure 15: The dropdown list for a data expression with two data fields, one X field, one Y field

Beware when choosing your unit (you need to have a valid unit, i.e. one that is defined in the data dictionary as such; see [5.2.3.6.Units](#) for a list of valid units). If you choose a pre-saved formula: a default 'count' will appear as the unit. If you select one field in the dataset list and insert it by using the 'F' button, it will automatically be filled with the correct unit (but if you finally write your own formula, the final unit might be different). If the unit you defined does not fit the unit of the data, an error will be generated (in the Log tab).

However, note that every operation is computed using SI units even if a sub-unit is defined for the data (e.g. metres instead of cm, mm or km). Thus you can put 'cm' as the unit even if the data are in mm and still end up with correct values.

There are 4 buttons to the right of the Data expression box. These are:

- **Check**, checks whether the expression is well-formed, i.e. if the syntax is correct (but NOT if it is scientifically or even dimensionally correct...).
- **Save as**: saves the current data expression for future use within the workspace. It will then be available in the 'formulas' box
- **Reset** empties the whole box
- **Comment**. The 'Comment' button enables you to associate a more detailed comment with the expression (for convenience, it only appears as comments in files and is never used for computing/viewing), for future reference

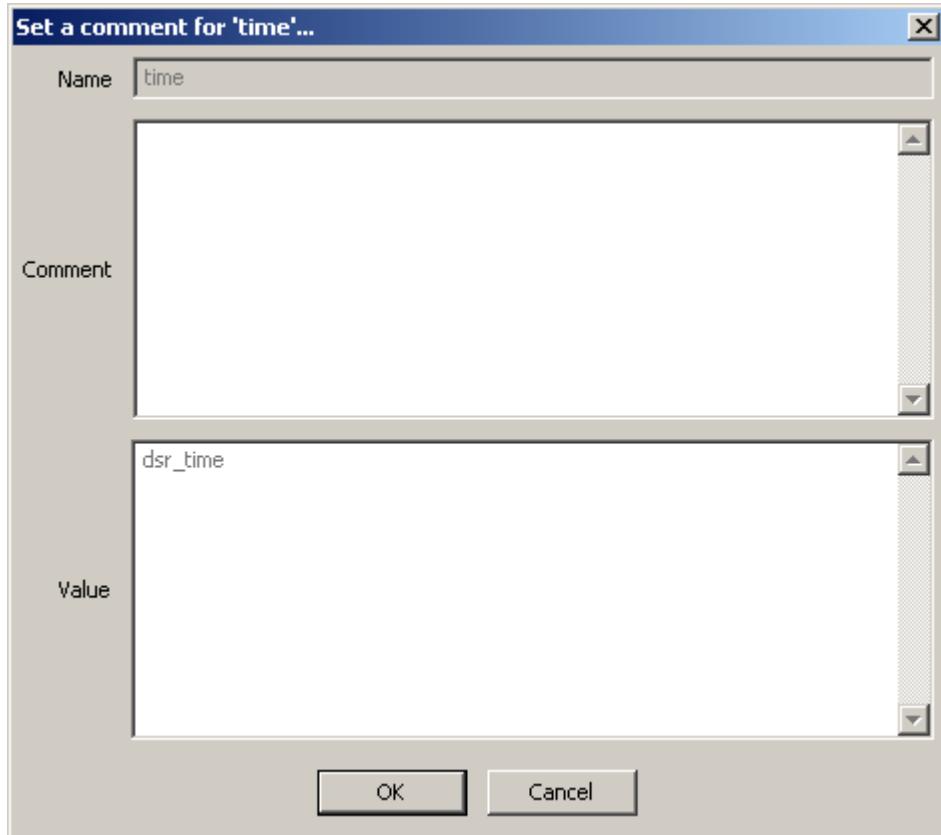


Figure 16: A 'Comment' window. It can be used as a reminder of the expression and its meaning and if need be, of its unit for future use.

#### 5.2.3.6. Units

BRAT is able to understand all SI units and their sub-units, as defined in the International System, i.e. **case sensitive**: “ms” means milliseconds, whereas “Ms” would mean megaseconds), plus “count” for data without dimension, and “dB”.

Typically, the units you might use are:

- metres (m, mm, cm, km,...)
- seconds (s, ms, etc., but also hours, h, days)
- m/s (km/s,...)
- degrees East (longitude)
- degrees North (latitude)
- degrees
- count
- dB

Note that all data fields are converted in SI units in the data dictionary.

Thus practical units such as “TECU” are converted (1 TECU (Total Electron Content Unit) =  $1 \times 10^{16}$  electrons/m<sup>2</sup>).

If you let “count” (which is the default) as unit, the resulting data will be in the basic SI unit (e.g. in metres, even if the field(s) you used was in mm)

#### 5.2.3.7. Formulas

In the Formula box, you will find pre-defined formulas (e.g. Sea Surface Height formulas from the different satellites’ GDR fields) and also Data expressions or Selections previously saved by you within the current workspace (or imported from another workspace).

## Basic Radar Altimetry Toolbox User Manual

The 'F' and 'S' button are used to insert a formula in either the selected 'data expression (F)' or the 'Select expression (S)'. The formula will appear there either as an alias (if you leave the 'as alias' checked), or complete (if you un-check 'as alias').

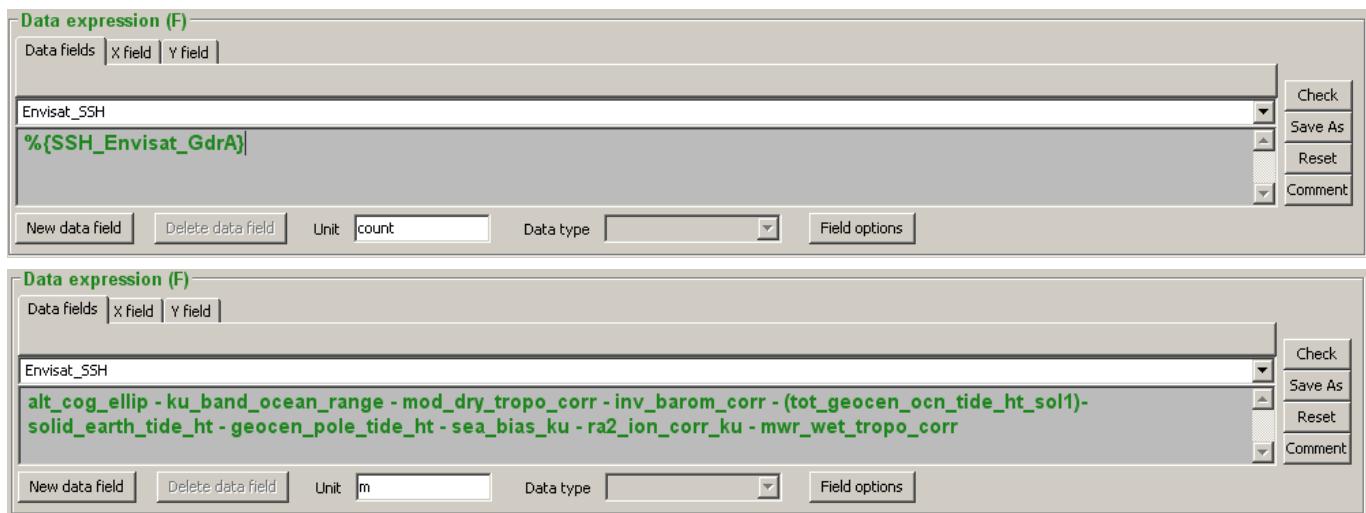


Figure 17: use of a pre-defined formula (Envisat SSH), by inserting its alias (top) and developed version (bottom). Note the unit, set by default to 'count', that you will have to change manually to 'm' (or a sub-unit of the metre)

### 5.2.3.8. Select expression

This box is for selecting data within the dataset: e.g. by date, boundaries, etc. or for editing it; As there is only one selection expression, there is no dropdown list with names, just the text box for entering the value of the expression to be edited.

It is in this text area that fields/functions/formulas are inserted when clicking on the 'S' buttons.

A data measurement in the Dataset files is selected only if the result of this expression is not 0 and not a missing/default value. If no selection expression is defined this is the equivalent of selecting everything (the expression value is '1')

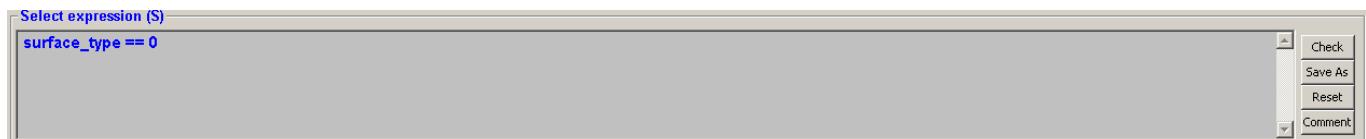


Figure 18: A Selection expression (test on a 'flag' whose value is 0 over the open ocean)

Type in your selection expression. Alternatively, to insert it in the same box, you can select either:

- a field within the dataset/record by selecting it in the list (you can sort the list alphabetically in each column, or type in the first letters to find the right one) and then clicking on the 'S' button to have it inserted at the cursor position in the selection expression box.
- or a ready-made expression by selecting it in the 'formulas' list and then clicking on the 'S' button, or typing in the alias.

Logical operators can be used to combine the conditions.

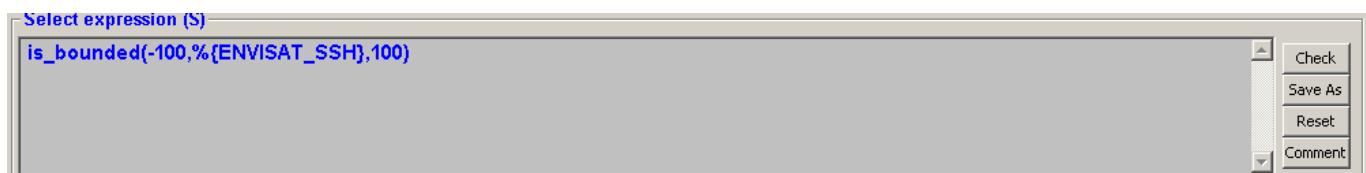


Figure 19: Another selection, using a formula alias, to edit values of SSH which are higher or lower than 100 m. This expression could also have been written '( %{ENVISAT\_SSH} <= 100) && ( %{ENVISAT\_SSH}>= -100)', where && is the logical operator 'AND' (see function list for the complete list of operators)

The selection operates on all available fields within the dataset (you can put in that you wish your X field between min & max, your Y field between min & max and, for instance, your field between -100 and 100).

To the right of the Selection expression box, there are four buttons. These are as follows:

- **Check:** checks whether the expression is well-formed, i.e. if the syntax is correct (but NOT if it is scientifically or even dimensionally correct...).
- **Save as:** saves the current Selection expression for future use within the workspace. It will then be available in the 'formulas' box
- **Reset:** empties the whole box
- **Comment:** The 'Comment' button is used to associate a more detailed comment with the expression (for convenience, it only appears as comments in files and is never used for computing/viewing), for future reference

#### 5.2.3.9. *Output*

Output gives the name of the output (netCDF) file. It is predefined using the name you gave to your operation and cannot be changed within the GUI.

'Execute', bottom right, processes the defined operation on the whole selected dataset.

You may perform several different operations at the same time (i.e. execute one while another is being processed), or an operation and a view (provided you are not trying to visualise the file being processed). However, this will slow down each individual execution. In the 'Log' tab window you can see the current tasks being executed (both operations and views), comments during execution (verbose mode) and errors.

#### 5.2.4. Create a view

The third tab is 'Views'.

If none exist, you have to create a new view (name it as you wish, but without any spaces or special characters in the name). Choose 'new' in the 'Views' menu.

The name is used to call the command file that will be executed to display the data and if need be to retrieve your view in the future. The 'Name' dropdown list contains all the defined view names and can be used to select and rename views.

You can delete an existing view by choosing 'delete' in the 'Views' menu, but note that you can only visualise your data with BRATGUI with a view which has already been defined which means you will have to create a new one if none exist.

## Basic Radar Altimetry Toolbox User Manual

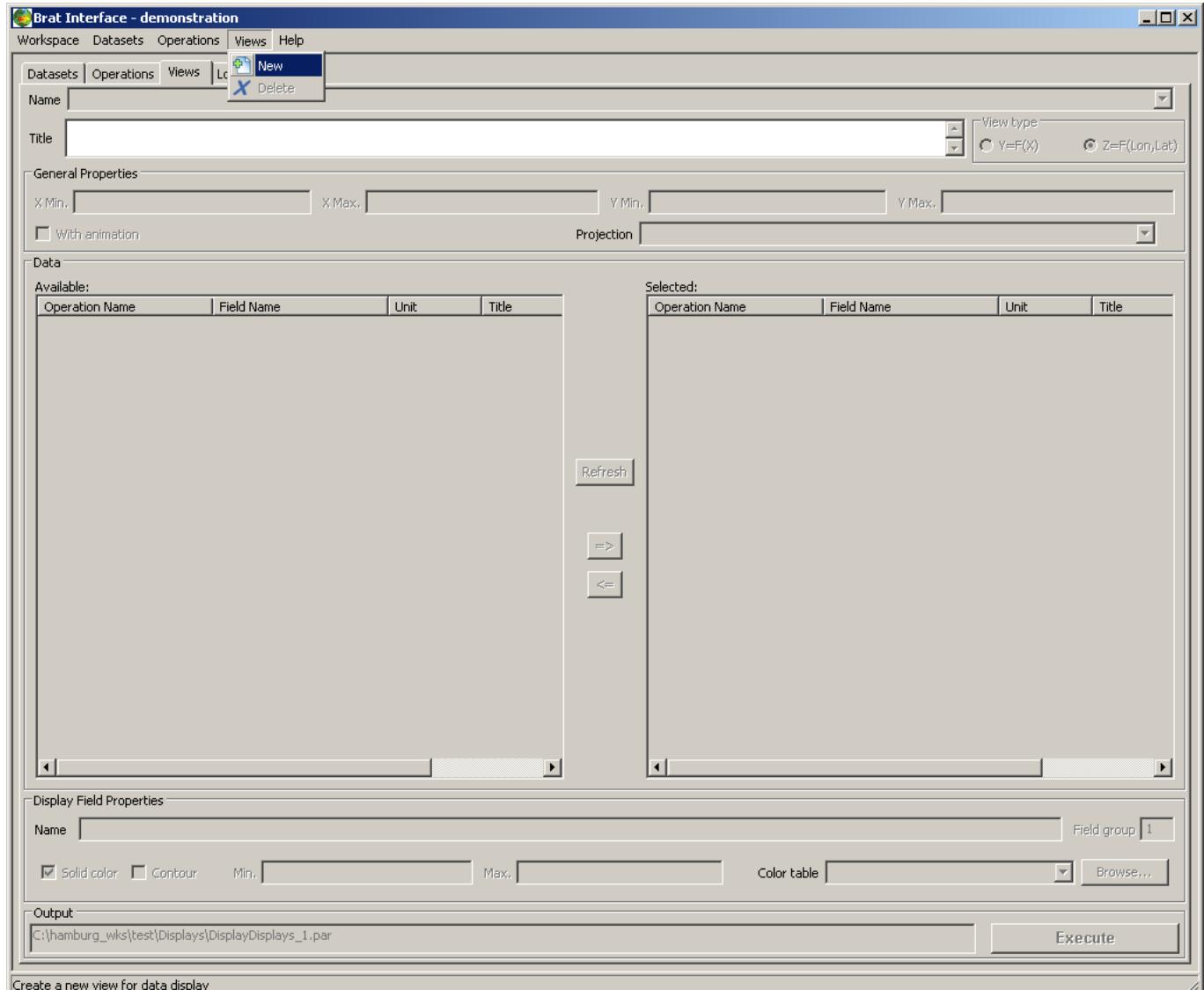


Figure 20: Creating a view, to launch the visualisation tool

Once you have named your view you then have to choose between 'Y=F(X)' and 'Z=F(lon,lat)'. When you have done this, you will have access in the 'Data' box to the available fields corresponding to either choice (output from operations computed **within the workspace**). They are given by operation/file name/field name. You can refresh the list to update it with respect to the latest operations.

In the 'Data' box, select one or several data fields by clicking on them (ctrl + click for several fields) and use the arrow to switch it or them from 'available' (left) to 'selected' (right). The X axis (and Y) has been pre-defined in the 'operation' tab.

You can give your display a title (just below the 'name' of the view).

## Basic Radar Altimetry Toolbox User Manual

### 5.2.4.1. ‘Y=F(X)’

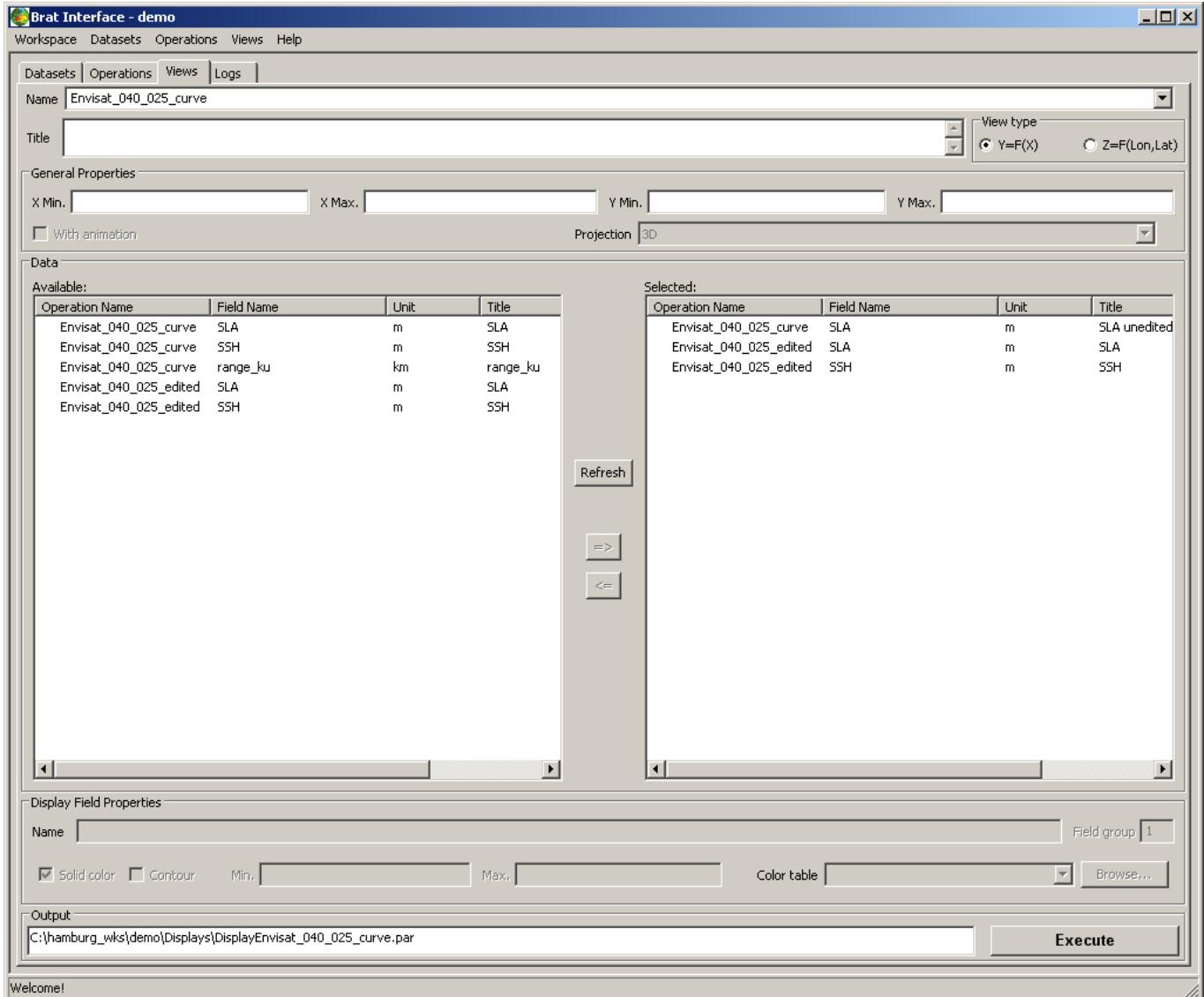


Figure 21: Example of a ‘views’ tab when  $Y=F(X)$  is chosen

For such a plot, you can define a sub-set to be plotted (by X min, X max, Y (=field) min, Y (=field) max). If you click on one of the selected data fields (right-hand list), you can see in the ‘Display field properties’ box below that the name of the represented field is also given. By default, it is the title of the field given in the ‘options’ of the data expression, or the name of the field data expression.

‘Field group’ is used to group the selected fields – or not – in a same plot. By default, each field will have a curve of a given color overlaid in the same plot. However, if you select the second field you wish to visualise, type ‘2’ in ‘Field group’ and the two data fields will be visualised in different adjacent plots. You can thus have as many plots as you have fields.

### 5.2.4.2. 'Z=F(lon,lat)'

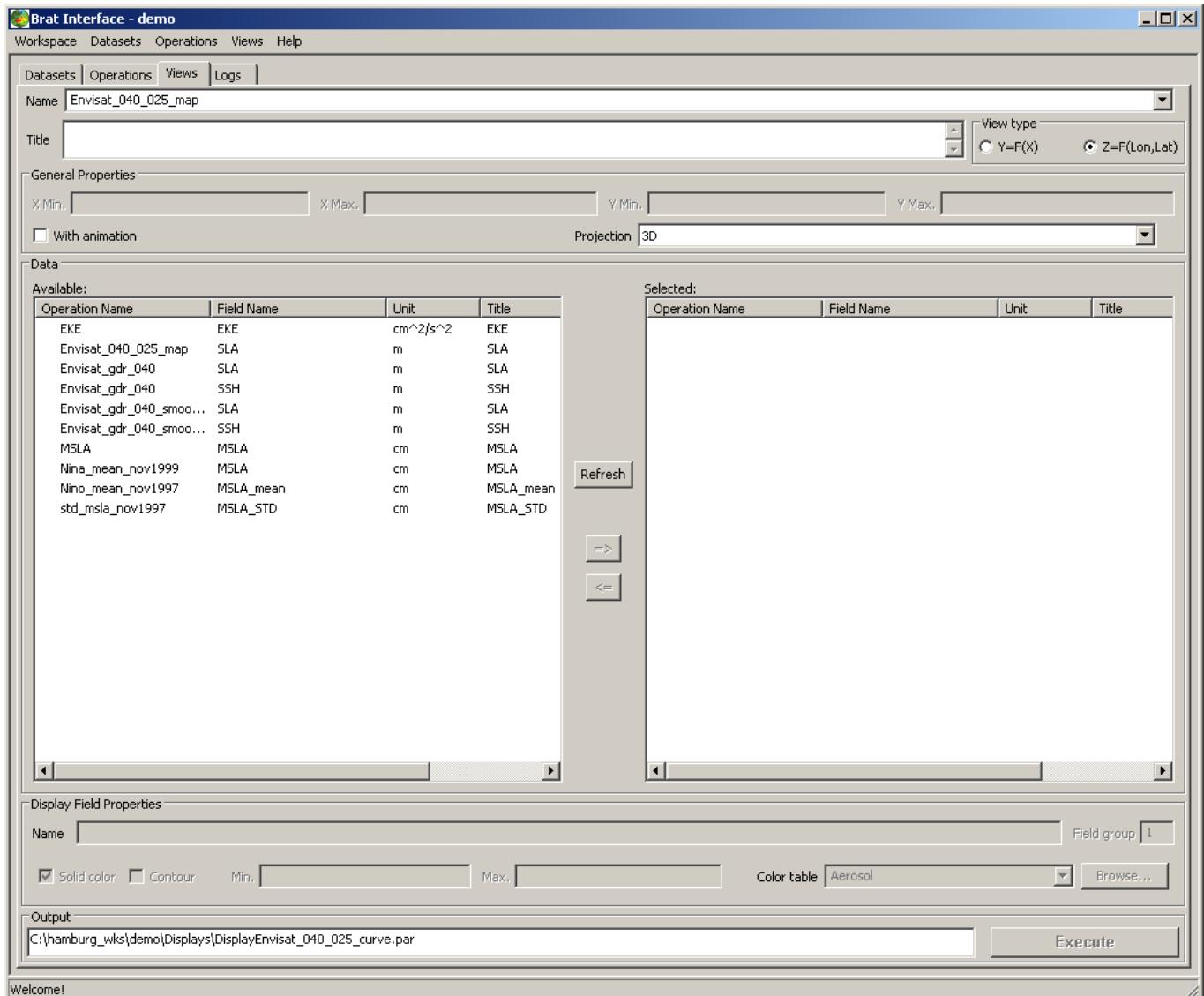


Figure 22: Example of a 'views' tab when  $Z=F(\text{lon}, \text{lat})$  is chosen

You can choose your projection in a list of pre-defined projections (it can be also changed in the visualisation interface). By default this will be a 3D projection.

'With Animation' can be used to animate a series of maps. If you have several identical field names from several operations (e.g. if you have computed the same field at different dates) and if you check this option, you will have access to the 'animation toolbar' in the visualisation interface.

If you click on one of the selected data fields (right-hand list), you will be able to see the following in the 'Display field properties' box below:

- The name of the represented field.  
By default, this will be the title of the field given in the 'options' of the data expression, or the name of the field data expression (if no title was given).
- A choice between 'solid color' and 'contour' representation. It is of course highly recommended to choose at most two different fields to be displayed on the same plot, one represented in solid colors, the other in contours, to be able to see something on the plot.
- Min and max of the color scale
- The color scale, among a pre-defined list of color scales, or in previously made and saved color scale (see 6.2.2).

All those options can also be changed in the visualisation interface.

'Field group' is used to group the selected fields – or not – in a same plot. Typically, if all fields have '1' (default) you will have a color and a contour map plotted one overlaying the other. If you put a different number for each field, you will have as many separate plot windows as you have typed numbers.

**'Execute'** will execute the request as it is in the defined view (request that is written in the command file which name is shown left of the 'execute' button) and launch the visualisation tool (see chapter 6 for a description of this interface). You can see in the 'Log' tab the current executions (both operations and views) and the errors.

## 6. Visualisation interface

The visualisation interface is called by executing a command file from the ‘views’ tab of the GUI. It can also be used with a command file.

The visualisation options are quite different for an ‘Y=f(X)’ (curve) than for a ‘Z=F(lon,lat)’

### 6.1. ‘Y=F(X)’

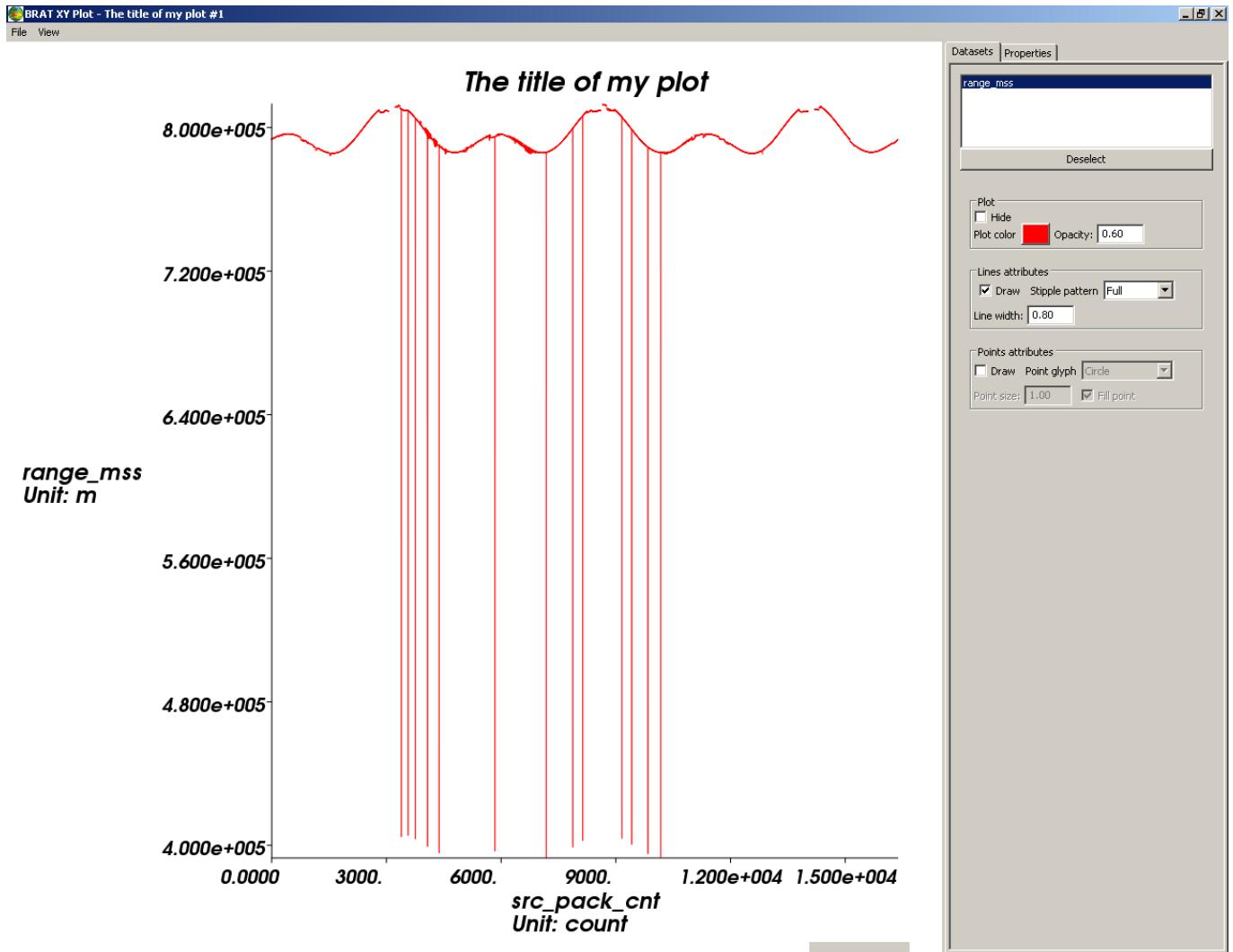


Figure 23: An example Y=F(X) visualisation

In the ‘File’ menu, you can save your plot, in different image format (bmp – windows bitmap – jpeg, png, ppm or tiff), or export it to gnuplot.

The ‘View’ menu enables you to display or not the right-hand panel with the properties.

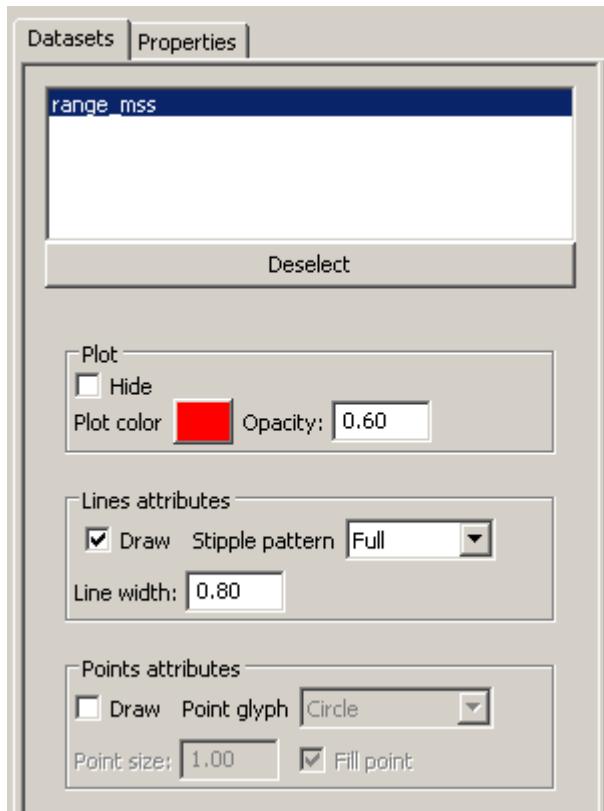
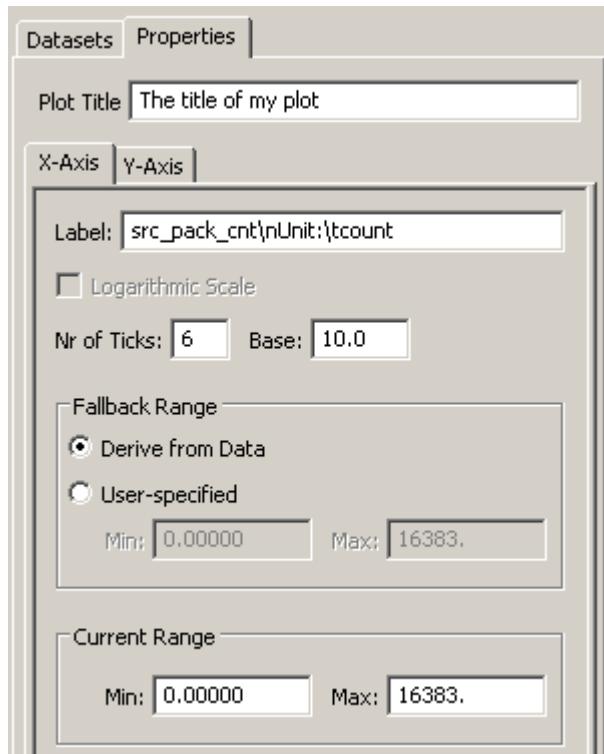


Figure 24: Datasets tab of the visualisation tool



First tab ('datasets') recalls the name of the field as it appears in the Display Field properties of the 'Views' tab.

When a field is selected in this 'datasets' tab, you have some options to choose the color and style (full, dots, etc.) of the line and of the points (none by default, circles, crosses, etc.). If there are several fields to plot, you can thus enhance the legibility of your plot.

Second tab ('properties') enable to choose several options (some being already available within the 'views' tab; however, modifications done only in the visualisation window will not be saved as part of the workspace and thus cannot be recalled for future use. We thus strongly recommend that you choose options as min, max of both axis, units, plot title and axis name within the Operations and Views tabs.

The label of each axis includes by default the name of the plotted field and its unit, with \n for line break and \t for space.

'Fallback range' enables you to select in a more restricted range (e.g. you selected a whole ground track, but finally wish to look only at a -10 + 10°N range). You can also zoom in on a portion of curve using middle button of your mouse.

'Current range' indicates the min max of your current view.

To go back to the first opened view, type on 'r'.

Figure 25: Y-axis properties of a  $Y=F(X)$  plot, with only one field selected for view. Label (including the unit), number of ticks in the axis, min and max of the axis are shown. X-axis properties are similar.

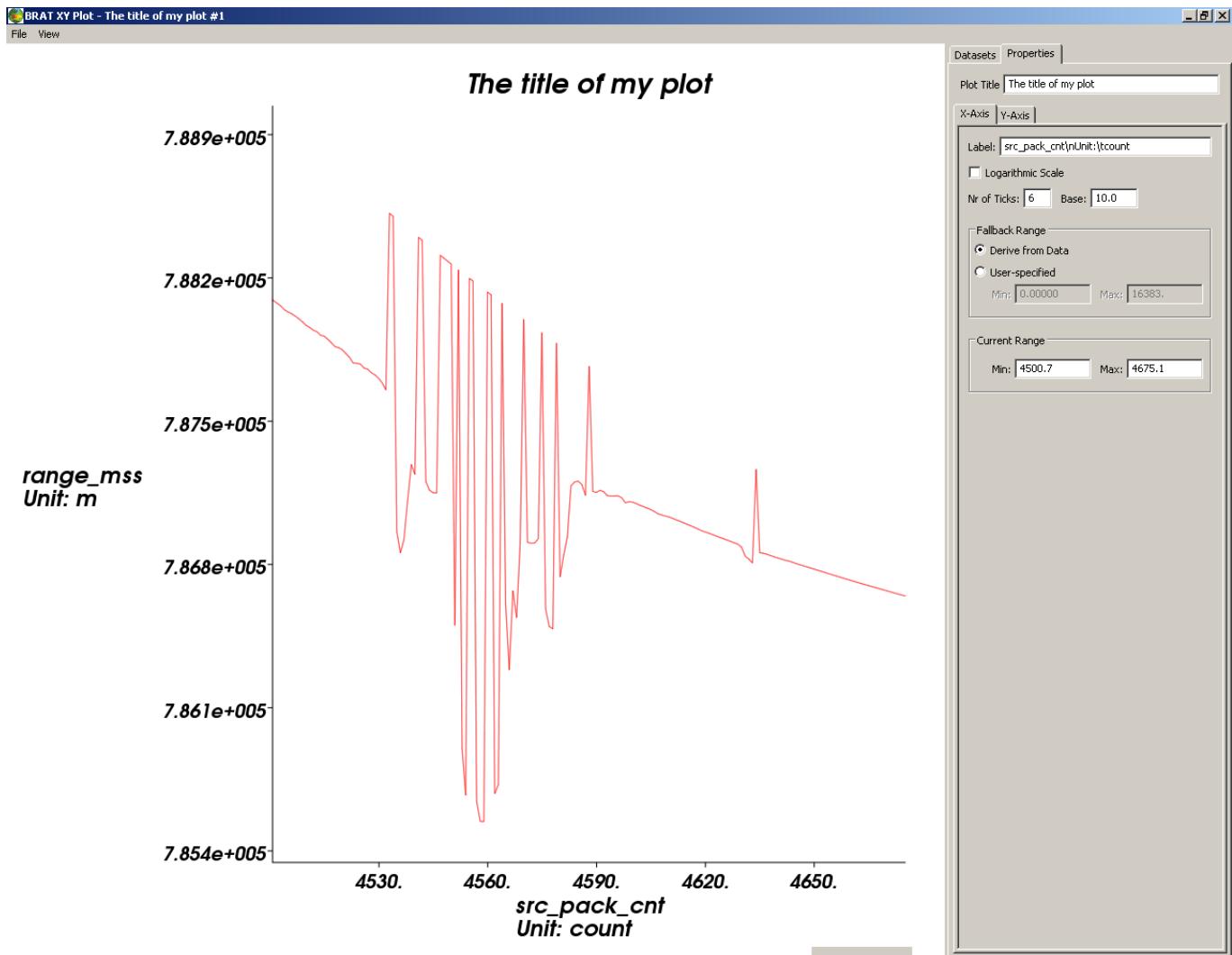


Figure 26: Zoom of the same curve than above

## 6.2. ‘Z=F(lon, lat)’

Note that, even if  $Z=F(X,Y)$  with any data field as  $X$  and as  $Y$  is possible to process within the ‘operations’ tab, only the case  $Z=F(\text{longitude}, \text{latitude})$  (i.e., a map) is possible for now in BRAT.

In the ‘File’ menu, you can save your plot, in different image format (bmp – windows bitmap – jpeg, png, ppm or tiff).

The ‘View’ menu enables you

- to display or not display the right-hand panel with the properties.
- display the color bar or not
- display the animation toolbar (if relevant, i.e. if you are visualising a series of fields with the same name and chose the option ‘With animation’). Once in this toolbar, you can launch the animation of the fields, stop it and control its speed.
- to open the color table editor and the contour table editor



Figure 27: Animation toolbar (available for a series of fields with the same name, option ‘With animation’ chosen). The animation is available as visualisation (not to be saved). ‘Animate’ launch the animation, Reset reset the animation to the first frame. The number after is the number of the frame. ‘Loop’ enables to loop the animation and ‘Speed’ to choose its speed (in frames per second).

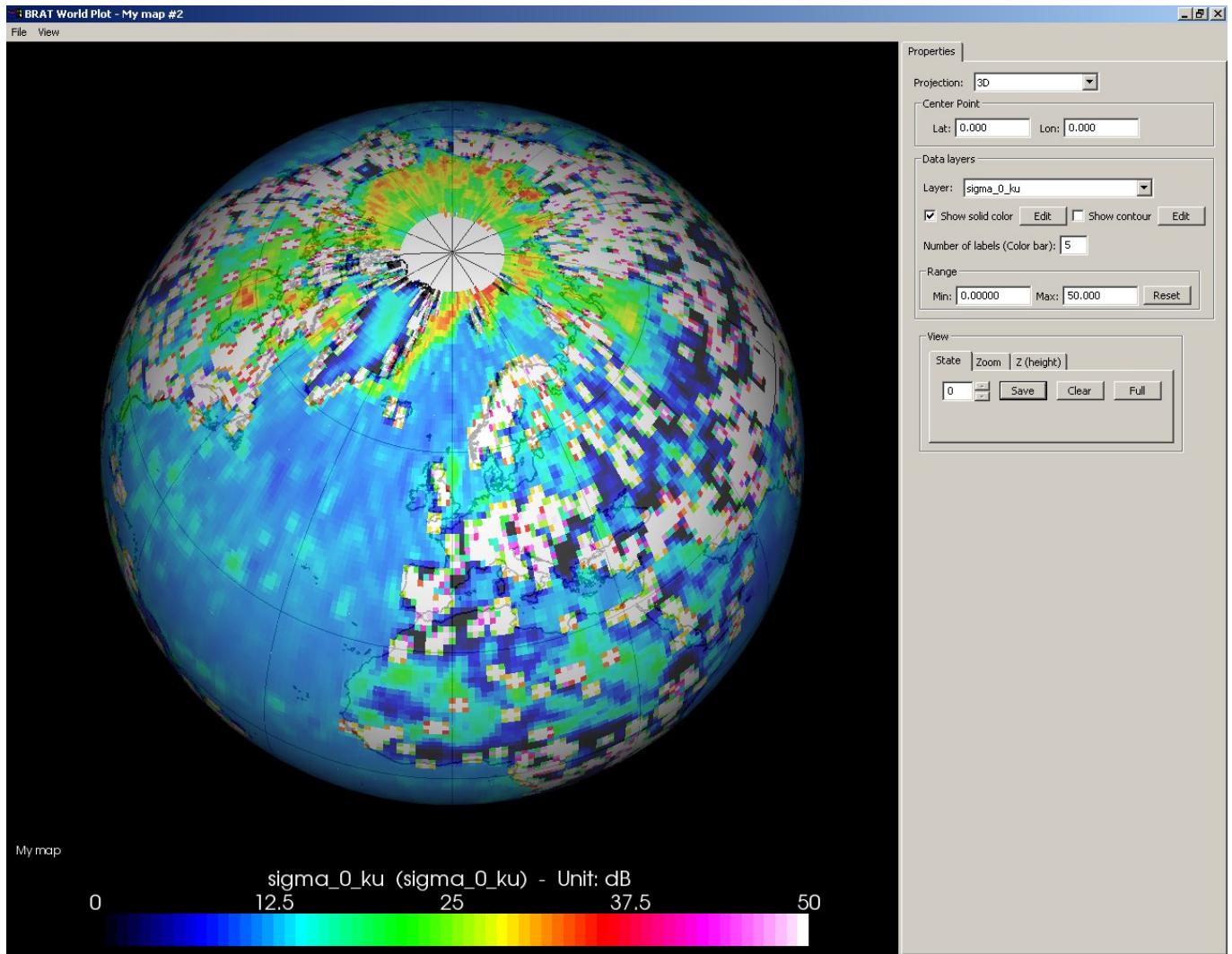


Figure 28: An example  $Z=F(\text{lon}, \text{lat})$  visualisation, with default projection (3D) and the ‘Ozone’ color table

### 6.2.1. Display properties

Available display properties are:

- the projection. Several of them are available (see ‘Create a view’). You can change it on the fly, even if you decided on another one in the ‘view’ tab of the GUI (but your choice will not be saved)

## Basic Radar Altimetry Toolbox User Manual

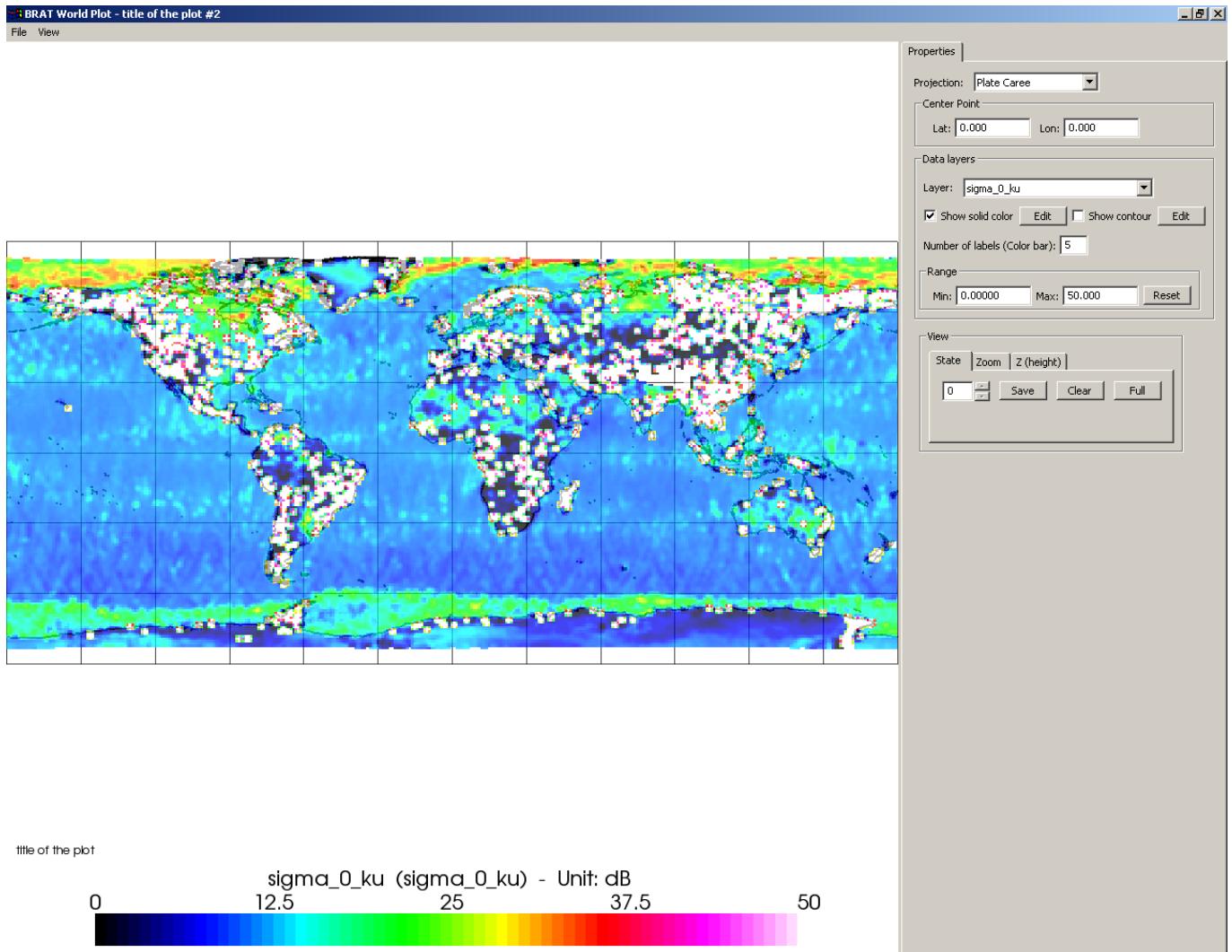


Figure 29: Same plot than above, but with a different projection (Plate Carree)

- Centre point: define the centre of the display (only relevant for 2D maps, not for the '3D' projection)
- Data layers: lists the different fields visualised and if each one is visualised as solid color or as contours.  
Edit open either the color table or the contour table editor (see section 6.2.2 and 6.2.3 below). With these editors you can modify previous definitions (but, once again, your choices will not be saved). If two fields are superimposed, you can switch contours and colors. This is why, in this case, you will have two color tables in your plot (one for each field). The number of labels and the range define those for the color table.

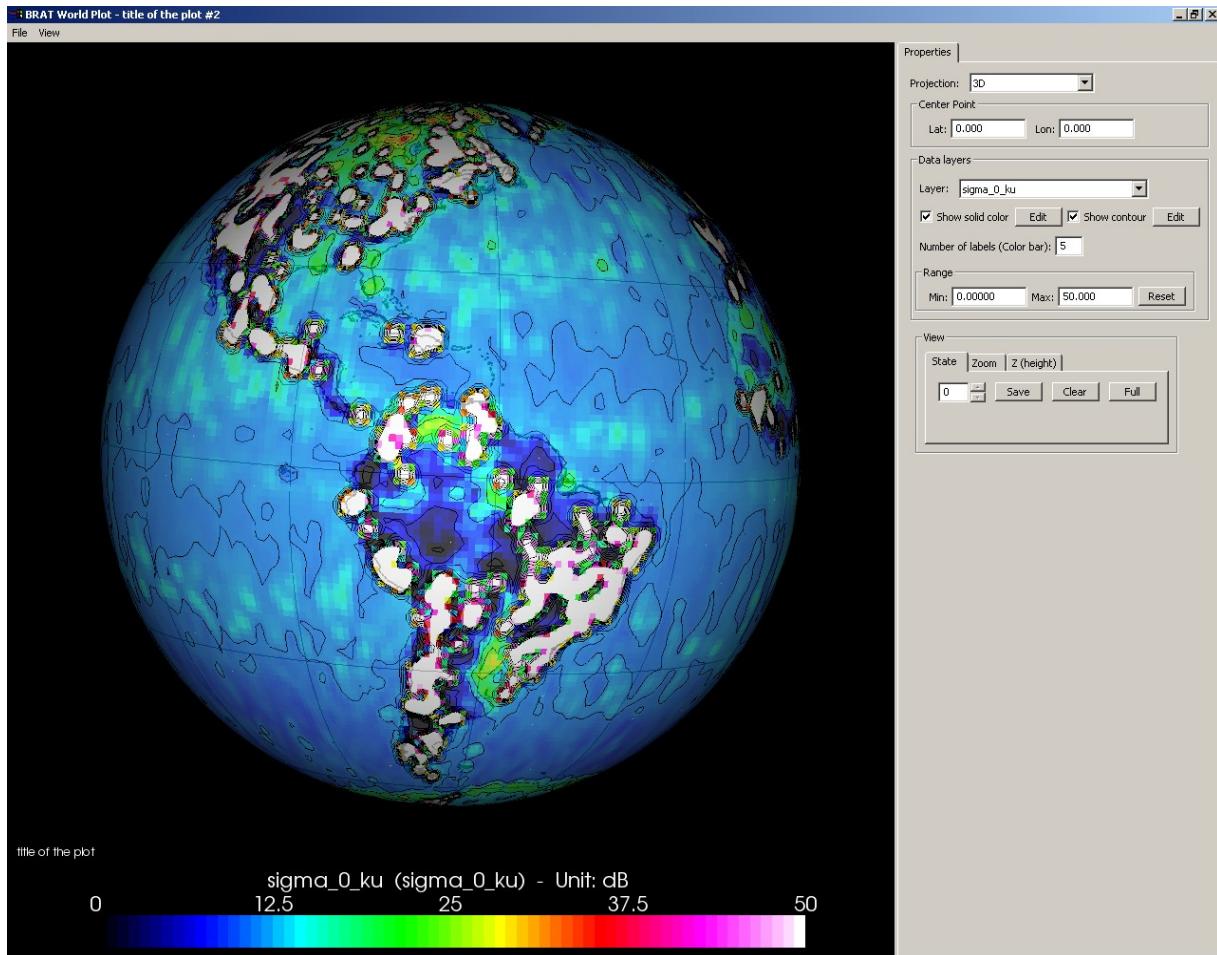


Figure 30: Visualisation with color and contour (for the same field)

- View: There are 3 tabs available
  - o **State:** used to save a particular display for the duration of the session and to recall it by its number.  
Clear erases all the saved displays,  
Full goes back to a full-sized view of the chosen area (if a zoom had been made)
  - o **Zoom:** used to visualise a specified area, defined by its minimum and maximum longitude and latitude respectively (this does not work for the '3D' projection).
  - o **Z-height:** only available for the 3D projection, it is used to render field values at the surface as bumpiness (radius gives the height, factor the scale factor).

### 6.2.2. Color table editor

Several color tables are available within BRAT.  
You can use any one of them. You can also make your own color table.

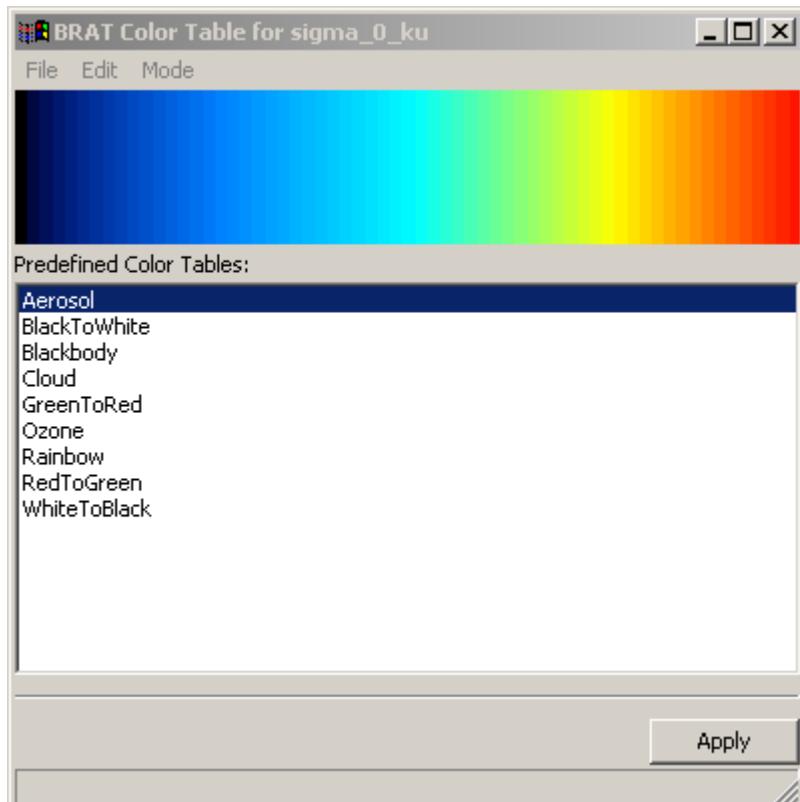


Figure 31: The color table editor, with the list of predefined tables

In the ‘File’ menu of the color table editor, ‘Load color table’ loads a previously made color table. Recent color table recall recently used ones, and ‘save as’ to save the one you’ve just done.

The ‘Edit’ menu enables to change the number of color within an existing color table and the interpolation between the different colors.

The ‘Mode’ menu enables to choose between predefined color tables, two-color gradient color tables or multi-color gradient color tables.

#### 6.2.2.1. two-color gradient color tables

The two-color gradient color table editor enables to make a color table by defining its first and last colors.

Colors are defined by their Red, Green and Blue components and Alpha channel (for transparency). Default is black (RGB=0,0,0) for both and no transparency (A=255). You can click on ‘apply’ to look at the way it shows on your plot. When you are satisfied of your color table, you can save it and recall it in future sessions.

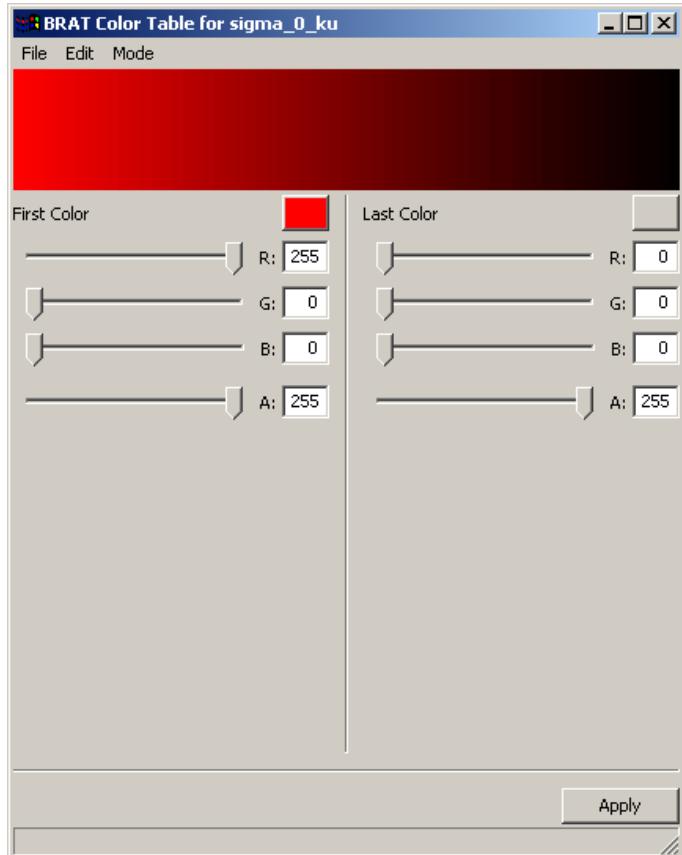


Figure 32: Two-color gradient color table editor

### 6.2.2.2. Multi-color gradient color tables

The multi-color gradient color table editor works much as the two-color one, except that you have to define not only the first and last values, but also define intermediate one(s).

The definition of colors is the same (Red, Green, Blue + Alpha channel) and you also have a cursor beneath the preview of the table that enables you to place your new color in the range.

To add a new color, click on '0' in the 'X-values' list, then on 'Insert color'. You will then have a new value, '1', that you can change by moving the cursor. When you have placed your new value in the range, define your color. Repeat the operation as many time as you wish to add colors. Note that you do not have to define 255 colors (if you want a 255-color table) one by one, since the software interpolates between the values you are giving, so choosing 5 or 7 of them is usually sufficient.

You can click on 'apply' to look at the way it shows on your plot. When you are satisfied with your color table, you can save it and recall it for future sessions.

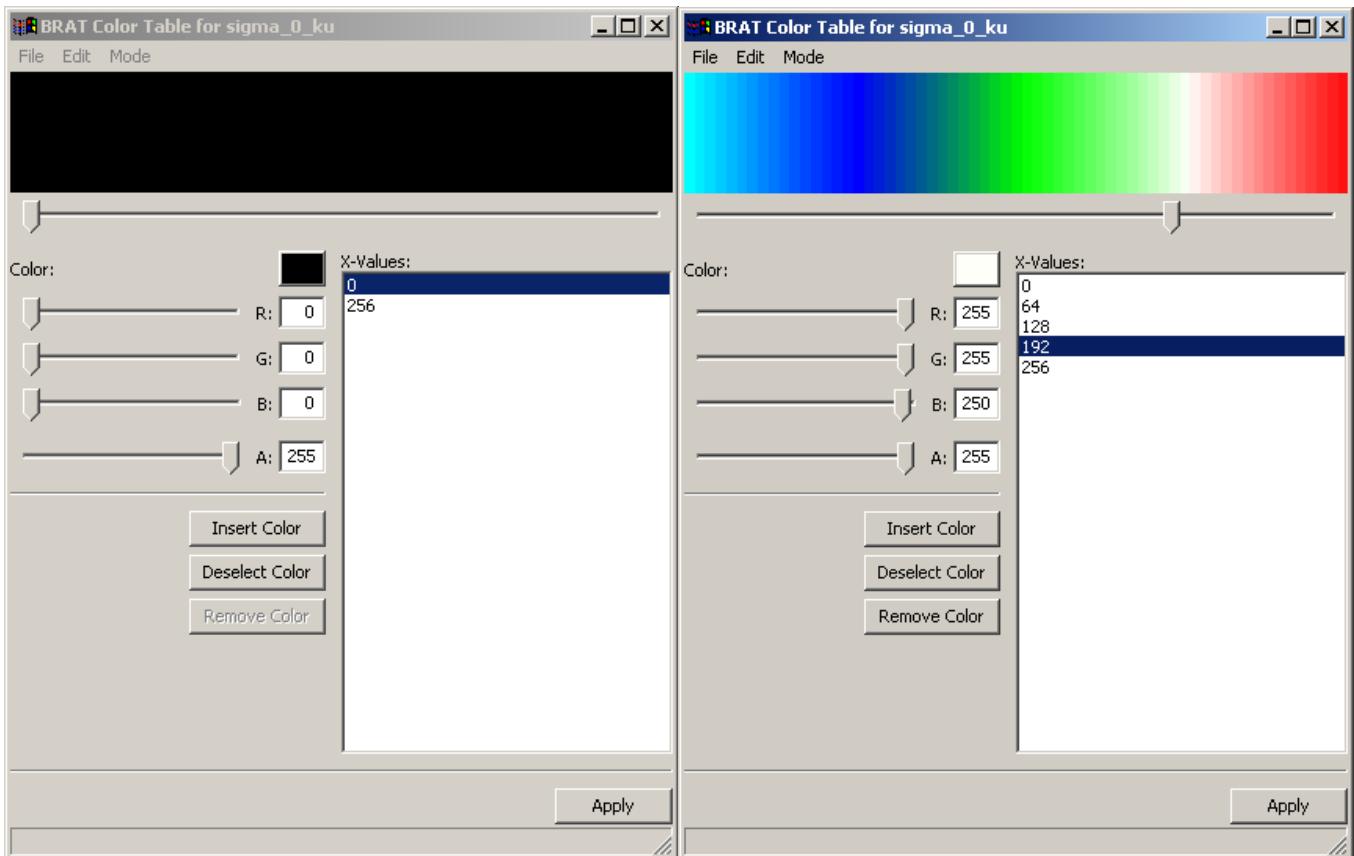


Figure 33: Multi-color gradient, color table editor. When first opening it (left) and after defining 5 colors over the whole range ; equally distributed (on the right)

### 6.2.3. Contour table editor

The contour table editor enables you to choose the range and number of contours that you wish to see on your plot, the width and color of the lines and whether you want labels on the contours or not and if so, which style.

Note that your contour table cannot be saved and re-used for future use.

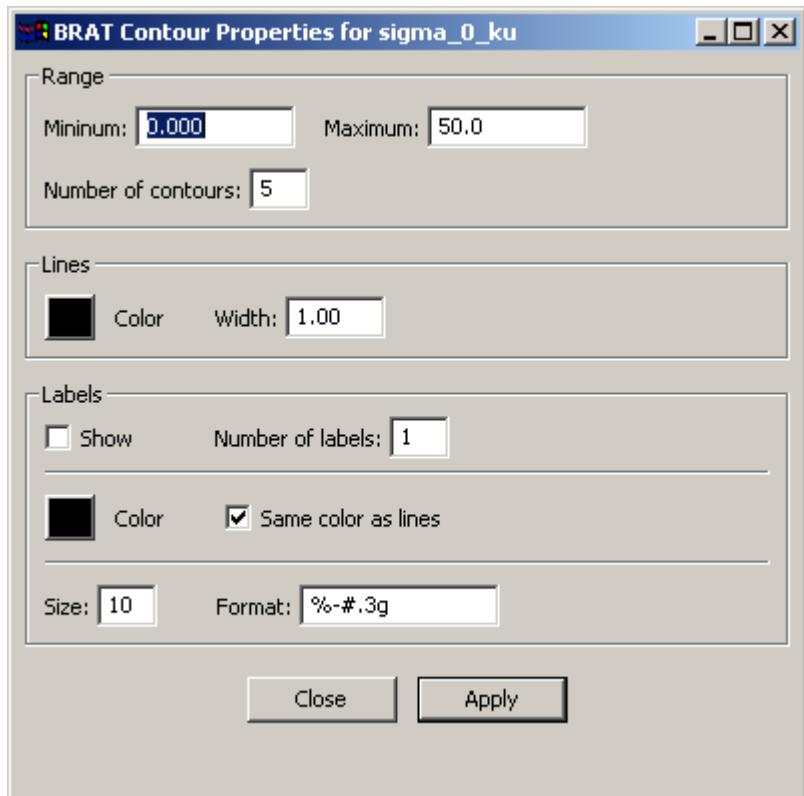


Figure 34: Contour table editor

## 7. Using BRAT in ‘command lines’ mode with parameter files

The GUI is there to ease the use of BRAT. However, everything made with the GUI can be made directly by writing parameter files and execute them and more than what can be done with the GUI is possible with parameter files.

Dictionaries of key functions that can be called within parameter files are available in annex B ([Y=F\(X\)](#)), annex C ([Z=F\(X,Y\)](#)) and annex D ([Display parameter file keys](#)).

‘-h’ option offers help for launching the executable file

‘-k’ offers help on parameter keys

### 7.1. Creating an output NetCDF file

A ‘Create’ parameter file typically consist of:

- the definition of a dataset (a list of files that will be processed),
- the name of the record within the dataset in which the data you are interested in are stored,
- = the definition of an X axis and of one or several ‘Field(s)’; in the Z=F(X,Y) case, also the definition of an Y-axis,
- a selection expression, if need be
- the name and location of the NetCDF output file.

The definition of the axis or of a field includes the name of an existing data field, or the expression that you wish to compute from several of them, a name (without any spaces or special characters), a unit, a title (that may include spaces or special characters), a min and a max and information about a possible filter

```
#----- GENERAL PROPERTIES -----  
  
DATA_MODE=MEAN  
  
#----- DATASET -----  
  
RECORD=ra2_mds  
  
FILE=File1  
FILE=File2  
...  
  
#----- FIELDS -----  
Y=lat  
Y_NAME=lat  
Y_TYPE=Latitude  
Y_UNIT=degrees_north  
Y_TITLE=Latitude  
Y_FILTER=DV  
Y_MIN=DV  
Y_MAX=DV  
Y_INTERVALS=DV  
Y_LOESS_CUTOFF=DV  
  
X=lon  
X_NAME=lon  
X_TYPE=Longitude  
X_UNIT=degrees_east  
X_TITLE=Longitude  
X_FILTER=DV  
X_MIN=DV  
X_MAX=DV  
X_INTERVALS=DV  
X_LOESS_CUTOFF=DV
```

```
FIELD=ra2_wind_sp
FIELD_NAME=my_first_field
FIELD_TYPE=Data
FIELD_UNIT=mm/s
FIELD_TITLE=Altimeter wind speed modulus
FIELD_FILTER=DV
FIELD_MIN=DV
FIELD_MAX=DV
FIELD_INTERVALS=DV
FIELD_LOESS_CUTOFF=DV

FIELD=alt_cog_ellip - ku_band_ocean_range - mod_dry_tropo_corr - inv_barom_corr -
(tot_geocen_ocn_tide_ht_soll + tidal_load_ht + long_period_ocn_tide_ht) -
solid_earth_tide_ht - geocen_pole_tide_ht - sea_bias_ku - ra2_ion_corr_ku -
mwr_wet_tropo_corr
FIELD_NAME=SSH
FIELD_TYPE=Data
FIELD_UNIT=m
FIELD_TITLE=my second field
FIELD_FILTER=DV
FIELD_MIN=DV
FIELD_MAX=DV
FIELD_INTERVALS=DV
FIELD_LOESS_CUTOFF=DV

----- SELECT -----

----- OUTPUT -----

OUTPUT=output_file.nc
```

*Example parameter file for creating a Z=F(X,Y) output*

You create the NetCDF file by typing

**'BratCreateZFXY.exe command\_file.par'**  
**(or 'BratCreateYFX.exe command\_file.par' )**

You will then have a NetCDF file that you can either visualise through the tool provided within BRAT, or with some other tool capable of reading NetCDF.

## 7.2. Visualising an output NetCDF file through BRAT

To visualise an output file, you have to write a second parameter file.

This kind of file is simpler than the one needed to create a NetCDF.

Basically, the commands needed are:

- the name of the file(s) to be displayed
- the title, projection
- the name of the field(s) to be displayed
- some information about the display (min, max, name, whether there is a contour or not, color table...)

```
#!/usr/bin/env BratCreateZFXY
#Type:Z=F(X,Y)
----- DATASET -----

FILE=Createenvisat_cycle.nc

----- GENERAL PROPERTIES -----
```

```
DISPLAY_TITLE=title of the plot  
DISPLAY_GROUPBY_FILE=Y  
DISPLAY_PROJECTION=3D  
  
#----- sigma_0_ku FIELD -----  
  
FIELD=sigma_0_ku  
  
#----- sigma_0_ku FIELDS PROPERTIES -----  
  
DISPLAY_NAME=sigma_0_ku  
FIELD_GROUP=1  
DISPLAY_MINVALUE=0.00000  
DISPLAY_MAXVALUE=50.000  
DISPLAY_CONTOUR=N  
DISPLAY_SOLID_COLOR =Y  
DISPLAY_COLORTABLE=DV
```

*Example ‘display’ parameter file*

You open the visualisation tool by typing:  
**‘BratDisplay.exe command\_file.par’**

### **7.3. Using the parameter files to process many datasets**

A typical case in which using the parameter files will be much easier than using the GUI is when you want to process the same operation on all the altimetry satellite cycles or for a long series of them. Parameter files enable you to write a script that will process the same operation on a number of files.

You can either write the parameter file directly, or you can make the parameter file through the GUI, test it on one cycle and then modify it (right-click) by replacing the cycle number by a character that will be replaced consecutively by a list of cycle numbers through a script;

```
#!/usr/bin/env BratCreateZFXY  
# SRC_DATA_DIR and CYCLE are environment variables that can be set in a shell #  
script  
  
FILE=${SRC_DATA_DIR}/JA1_GDR_2PAP${CYCLE}_001.CNES  
FILE=${SRC_DATA_DIR}/JA1_GDR_2PAP${CYCLE}_002.CNES  
FILE=${SRC_DATA_DIR}/JA1_GDR_2PAP${CYCLE}_003.CNES  
  
RECORD = data  
VERBOSE = 2  
  
ALIAS_NAME = SLA_JASON  
ALIAS_VALUE= altitude - range_ku - model_dry_tropo_corr - inv_bar_corr -  
(ocean_tide_s01 + ocean_tide_equil + load_tide_s01) - solid_earth_tide -  
pole_tide - sea_state_bias_ku - iono_corr_alt_ku - rad_wet_tropo_corr - mss  
  
X      = longitude  
X_TYPE      = longitude  
X_NAME      = Longitude  
X_UNIT      = DV  
X_TITLE      = Longitude  
X_MIN      = DV  
X_MAX      = DV  
X_INTERVALS= 1800  
  
Y      = latitude  
Y_TYPE      = latitude  
Y_NAME      = Latitude  
Y_UNIT      = DV
```

## Basic Radar Altimetry Toolbox User Manual

```
Y_TITLE      = Latitude
Y_MIN        = DV
Y_MAX        = DV
Y_INTERVALS= 900

# SLA_JASON is an alias see ALIAS_NAME and ALIAS_VALUE above
FIELD= ${SLA_JASON}
FIELD_TYPE = data
FIELD_NAME = SLA
FIELD_UNIT = m
FIELD_TITLE= Sea Level Anomalies - Cycle ${CYCLE}
FIELD_FILTER = LOESS_EXTRAPOLATE
X_LOESS_CUTOFF = 5
Y_LOESS_CUTOFF = 5

SELECT = is_bounded(-1.0, ${SLA_JASON},1.0)

OUTPUT      = ${BRATHL_DATA_DIR}/JasonSLA${CYCLE}.nc
OUTPUT_TITLE = Jason - Cycle ${CYCLE}
```

*An example parameter file for creating output NetCDF for several cycles (SLA from Jason-1 GDRs)*

```
REM Set the cycle number
SET CYCLE=109

REM Set the data source path
SET SRC_DATA_DIR=D:\data\gdr_jason\cycle_%CYCLE%

REM Launch 'Brat create Z=F(X,Y)' process
BratCreateZFYX C:\BRAT\MyCmdPath\BratCreateZFYXJasonSLASample.par

REM -----
REM Set another cycle number
SET CYCLE=110

REM Set the data source path
SET SRC_DATA_DIR=D:\data\gdr_jason\cycle_%CYCLE%

REM Launch 'Brat create Z=F(X,Y)' process
BratCreateZFYX C:\BRAT\MyCmdPath\BratCreateZFYXJasonSLASample.par
```

*An example script for DOS (to be inserted in a .bat file) to launch a parameter file over several cycles*

```
#!/bin/bash
# BratCreateZFYXJasonSLASample.sh

# Set the cycle number
export CYCLE=109

# Set the data source path
export SRC_DATA_DIR=/data/gdr_jason/cycle_%CYCLE%

# Launch 'Brat create Z=F(X,Y)' process
BratCreateZFYX BRAT/MyCmdPath/BratCreateZFYXJasonSLASample.par

# -----
# Set the cycle number
export CYCLE=110

# Set the data source path
export SRC_DATA_DIR=/data/gdr_jason/cycle_%CYCLE%
```

## Basic Radar Altimetry Toolbox User Manual

---

```
# Launch 'Brat create Z=F(X,Y)' process  
BratCreateZXY BRAT/MyCmdPath/BratCreateZXYJasonSLASample.par
```

*An example Shell script for Linux for launching a parameter file over several cycles*

## 8. BRATHL Application Programming Interfaces (APIs)

Some functions of BRAT are not available through the GUI, but through C, Fortran, IDL and Matlab APIs. Note that for IDL and Matlab under Windows<sup>®</sup> you need to compile the API before using them.

### 8.1. Data reading function

BRATHL\_READDATA reads data from a set of files; each measurement for a data is a scalar value (a single number). It also gives statistics (e.g. a mean over a geographical area)

Possible arguments of this function are:

[in] **fileNames**: file name string (one file) or file names array

[in] **recordName**: Name of the fields record (for netCdf files the recordName is 'data')

[in] **selection**: Expression involving data fields which has to be true to select returned data. (if the string is empty nothing is selected (in other words all of the data is taken))

[in] **dataExpressions**: Expression string (one expression) or expressions array applied to data fields to build the wanted value.

[in] **units**: Wanted unit for each expression (string (one unit) or units array). (if empty string, no unit conversion is applied to the data of the corresponding expression. When a unit conversion has to be applied, the result of the expression is considered to be the base unit (SI). For example if the wanted unit is grammes/litre, the unit of the expression is supposed to be kilogrammes/m<sup>3</sup> (internally all data are converted to the basic unit of the actual fields unit which is coherent with the above assumption)).

[in/out] **results**: Data read. Must be an array (dim = number of dataExpressions) of values to read.

[in] **ignoreOutOfRange**: Skip excess data. 0=false, other = true

Must be *false* if 'statistics' is *true*.

[in] **statistics**: returns statistics on data instead of data themselves

0=*false*, other = *true*

If statistics is *true*, ignoreOutOfRange must be *false*.

The returned values (5 values) for each expression are:

- Count of valid data taken into account.

Invalid data are those which are equal to the default/missing value

- Mean of the valid data.

- Standard deviation of the valid data

- Minimum value of the valid data

- Maximum value of the valid data

[in] **defaultValue**: value to use for default/missing values

This is the value you want to indicate that a value is missing or invalid.

return 0 or error code.

Syntax: see annexes

- [for IDL](#)
- [for Matlab](#)
- [for Fortran](#)
- [for C](#)

### 8.2. Cycle/date conversion functions

Two functions are available to convert between cycle/pass and date.

Syntax: see annexes

- [for IDL](#)
- [for Matlab](#)
- [for Fortran](#)
- [for C](#)

BRATHL\_CYCLE2YMDHMSM Converts a cycle/pass into a date

Arguments of this function are:

[in] **mission** :

- 0 : Topex/Poseidon
- 1 : Jason-1
- 2 : ERS2
- 3 : Envisat
- 4 : ERS1-A
- 5 : ERS1-B
- 6 : GFO

[in] **cycle** : number of cycles

[in] **pass** : number of passes in the cycle

Outputs are:

[out] dateYMDHMSM : date to convert

BRATHL\_YMDHMSM2CYCLE Converts a date into a cycle/pass

Arguments of this function are:

[in] **mission** : mission type :

- 0 : Topex/Poseidon
- 1 : Jason-1
- 2 : ERS2
- 3 : Envisat
- 4 : ERS1-A
- 5 : ERS1-B
- 6 : GFO

[in] **dateYMDHMSM** : date to convert

Outputs are:

[out] **cycle** : number of cycles

[out] **pass** : number of passes in the cycle

### **8.3. Date conversion/computation functions**

A set of functions is available to convert between the different kinds of date formats:

- days-seconds-microseconds dates:
- Julian decimal dates:
- year, month, day, hour, minute, second, microsecond dates:

Syntax: see annexes

- [for IDL](#)
- [for Matlab](#)
- [for Fortran](#)
- [for C](#)

BRATHL\_DAYOFYEAR Retrieves the day of year of a date

BRATHL\_NOWYMDHMSM Gets the current date/time

BRATHL\_SETREFUSER1 Set user-defined reference dates  
BRATHL\_SETREFUSER2 Set user-defined reference dates

BRATHL_DIFFDSM	Computes the difference between two days-seconds-microseconds dates (date1 - date2) the result is expressed in a decimal number of seconds
BRATHL_DIFFJULIAN	Computes the difference between two decimal Julian dates (date1 - date2) the result is expressed in a decimal number of seconds
BRATHL_DIFFYMDHMSM	Computes the difference between two year, month, day, hour, minute, second, microsecond dates (date1 - date2) the result is expressed in a decimal number of seconds
BRATHL_DSM2JULIAN	Converts a days-seconds-microseconds date into a decimal Julian date, according to refDate parameter
BRATHL_DSM2SECONDS	Converts a days-seconds-microseconds date into seconds, according to refDate parameter
BRATHL_DSM2YMDHMSM	Converts a days-seconds-microseconds date into a year, month, day, hour, minute, second, microsecond date
BRATHL_JULIAN2DSM	Converts a decimal Julian date into a days-seconds-microseconds date, according to refDate parameter
BRATHL_JULIAN2SECONDS	Converts a decimal Julian date into seconds, according to refDate parameter
BRATHL_JULIAN2YMDHMSM	Converts a decimal Julian date into a year, month, day, hour, minute, second, microsecond date
BRATHL_SECONDS2DSM	Converts seconds into a days-seconds-microseconds date, according to refDate parameter
BRATHL_SECONDS2JULIAN	Converts seconds into a decimal Julian date, according to refDate parameter
BRATHL_SECONDS2YMDHMSM	Converts seconds into a decimal Julian date, according to refDate parameter
BRATHL_YMDHMSM2DSM	Converts a year, month, day, hour, minute, second, microsecond date into a days-seconds-microseconds date, according to refDate parameter
BRATHL_YMDHMSM2JULIAN	Converts a year, month, day, hour, minute, second, microsecond date into a decimal Julian date, according to refDate parameter
BRATHL_YMDHMSM2SECONDS	Converts a year, month, day, hour, minute, second, microsecond date into seconds, according to refDate parameter

## 8.4. Named structures

Several structures are also available, to represent the different kinds of date formats

Syntax: see annexes

- [for IDL](#)
- [for Matlab](#)
- [for Fortran](#)
- [for C](#)

BRATHL_DATEYMDHMSM	YYYY-MM-DD HH:MN:SS:MS date structure YEAR MONTH DAY HOUR MINUTE SECOND MUSECOND
--------------------	---

## Basic Radar Altimetry Toolbox User Manual

---

BRATHL_DATEDSM	day/seconds/microseconds date structure REFDATE reference date DAYS numbers of days SECONDS numbers of seconds MUSECONDS numbers of microseconds
	REFDATE is the reference date i.e : 0: 1950-01-01 00:00:00.0 1: 1958-01-01 00:00:00.0 2: 1985-01-01 00:00:00.0 3: 1990-01-01 00:00:00.0 4: 2000-01-01 00:00:00.0 5: user reference 1 6: user reference 2 values of 5 and 6 allow users to set two specific reference dates of their choice (see BRATHL_SETREFUSER1 and BRATHL_SETREFUSER2 functions)
BRATHL_DATESECOND	decimal seconds date structure REFDATE reference date - see :BRATHL_DATEDSM NBSECONDS decimal numbers of seconds (seconds.microseconds)
BRATHL_DATEJULIAN	decimal Julian date structure REFDATE reference date - see :BRATHL_DATEDSM JULIAN decimal Julian day

## Annex A. List of datasets read by BRAT

### Cryosat product overview

product type	description
SIR1LRM_FR	SIRAL FBR-LRM mode product (Rx1 channel)
SIR2LRM_FR	SIRAL FBR-LRM mode product (Rx2 channel)
SIR1FDM_FR	SIRAL FBR-FDM mode product (Rx1 channel)
SIR2FDM_FR	SIRAL FBR-FDM mode product (Rx2 channel)
SIR1SAR_FR	SIRAL FBR-SAR mode product (Rx1 channel)
SIR2SAR_FR	SIRAL FBR-SAR mode product (Rx2 channel)
SIR_SIN_FR	SIRAL FBR-SARin mode product
SIR_SIC3FR	SIRAL FBR-CAL3 mode product
SIR_LRM_1B	SIRAL L1B LRM product
SIR_FDM_1B	SIRAL L1B FDM product
SIR_SAR_1B	SIRAL L1B SAR mode product
SIR_SIN_1B	SIRAL L1B SARin mode product
SIR1LRM_0M	SIRAL MON-LRM/TRK product (Rx1 channel)
SIR2LRM_0M	SIRAL MON-LRM/TRK product (Rx2 channel)
SIR1SAR_0M	SIRAL MON-SAR product (Rx1 channel)
SIR2SAR_0M	SIRAL MON-SAR product (Rx2 channel)
SIR_SIN_0M	SIRAL MON-SARin product
SIR_SIC40M	SIRAL MON-CAL4 product
SIR1LRC11B	SIRAL CAL1-LRM product (Rx1 channel)
SIR2LRC11B	SIRAL CAL1-LRM product (Rx2 channel)
SIR1SAC11B	SIRAL CAL1-SAR product (Rx1 channel)
SIR2SAC11B	SIRAL CAL1-SAR product (Rx2 channel)
SIR_SIC11B	SIRAL CAL1-SARin product
SIR_SICC1B	SIRAL complex CAL1-SARin product
SIR1SAC21B	SIRAL CAL2-SAR product (Rx1 channel)
SIR2SAC21B	SIRAL CAL2-SAR product (Rx2 channel)
SIR1SIC21B	SIRAL CAL2-SARin product (Rx1 channel)
SIR2SIC21B	SIRAL CAL2-SARin product (Rx2 channel)
SIR_SIC31B	SIRAL CAL3 product
SIR_LRM_2_	SIRAL L2 product from LRM processing
SIR_FDM_2_	SIRAL L2 product from fast delivery ocean processing
SIR_SIN_2_	SIRAL L2 product from SARin processing
SIR_SID_2_	SIRAL L2 product from SARin degraded processing
SIR_SAR_2A	SIRAL L2 product from SAR step 1 processing
SIR_SAR_2B	SIRAL L2 product from SAR step 2 processing
SIR_GDR_2A	SIRAL L2 consolidated product including SAR step 1 data (SIR_SAR_2A)
SIR_GDR_2B	SIRAL L2 consolidated product including SAR step 2 data (SIR_SAR_2B)
SIR_LRMI2_	SIRAL intermediate L2 product from LRM processing
SIR_FDMI2_	SIRAL intermediate L2 product from fast delivery ocean processing
SIR_SINI2_	SIRAL intermediate L2 product from SARin processing

SIR_SIDI2_	SIRAL intermediate L2 product from SARin degraded processing
SIR_SARI2A	SIRAL intermediate L2 product from SAR step 1 processing
SIR_SARI2B	SIRAL intermediate L2 product from SAR step 2 processing

### Envisat product overview

product type	description
RA2_FGD_2P	RA-2 Fast Delivery Geophysical Data Record
RA2_GDR_2P	RA-2 Geophysical Data Record
RA2_IGD_2P	RA-2 Intermediate Geophysical Data Record
RA2_MWS_2P	RA-2 Sensor Data Record
RA2_WWV_2P	RA-2 wind/wave product for Meteo Users

### Jason-1 product overview

product type	description
JA1 OSD_2P	The Operational Sensor Data Record (OSDR), produced on a NRT basis
JA1 IGD_2P	The Interim Geophysical Data Record (IGDR)
JA1 GDR_2P	The Geophysical Data Record (GDR)
JA1 SDR_2P	The Sensor Geophysical Data Record (SGDR)

### Topex/Poseidon product overview

Topex/Poseidon radar altimetry products

product type	description
CYCLE_HEADER_FILE	The GDR-M cycle header file
PASS_FILE	The GDR-M passfile
XNG_FILE	The crossover point file

### ERS-1 and 2 product overview

ERS-1 and ERS-2 radar altimetry products

product type	description
OPR_pass_file	Same as the off-line intermediate product but enhanced with all geophysical corrections and precise orbit altitude.

### GFO product overview

product type	description
GDR	The GDR is generated from GFO Sensor Data Records (SDRs), precise laser orbit ephemerides provided by NASA Goddard Space Flight Center and Raytheon ITSS, environmental corrections, and ancillary geophysical variables.

### PODAAC product overview

Physical Oceanography Distributed Active Archive Center radar altimetry products for Jason-1 and Topex/Poseidon

product type	description
J1SSHA_CYCLE_HEADER_FILE	The PODAAC JASON-1 SSHA cycle header file

TPSSHA_CYCLE_HEADER_FILE	The PODAAC TOPEX/POSEIDON SSHA cycle header file
J1SSHA_PASS_FILE	The PODAAC JASON-1 SSHA pass file
TPSSHA_PASS_FILE	The PODAAC TOPEX/POSEIDON SSHA pass file
J1SSHA_ATG_FILE	The PODAAC JASON-1 Along Track Gridded SSHA file
TPSSHA_ATG_FILE	The PODAAC TOPEX/POSEIDON Along Track Gridded SSHA file

## HDF products (including NetCDF)

HDF products are self describing products.

This means that when an HDF file is opened one can retrieve the product structure from the file itself. For this reason, BRAT will not store fixed product format descriptions for HDF files in the Data Dictionary (you will therefore also not find HDF product format descriptions in this documentation). What BRAT will do is use the underlying HDF4 and HDF5 libraries to retrieve the product format dynamically once an HDF file is opened. Based on this format BRAT will create, on the fly, a mapping of the HDF product structure to one that is based on the Data Dictionary data types

However, to be properly interpreted in the toolbox, a HDF product needs a description module to be added.

For example, in order to (really) read a NetCDF files we need to:

- 1- Access to NetCDF attributes
- 2- Identify default/missing values (see `_FillValue` standard attribute)
- 3- Convert data to its actual value (not the value stored in file): see `scale_factor` and `add_offset` standard attributes.
- 4- Interpret the structure of file to compute actual values of data (and not solely returning the NetCDF variables values 'as is').
- 5- Avoid making available variables belonging to data structure (which are not the data themselves)

## Aviso Altimetry data in NetCDF

product type	description
NRT- or DT-MSLA (h)	Ssalto/Duacs multimission Near real-time or Delayed time Maps of sea level anomalies (gridded)
NRT- or DT-MSLA (uv)	Ssalto/Duacs multimission Near real-time or Delayed time Geostrophic velocities associated to the Maps of sea level anomalies (gridded)
NRT- or DT-MSLA (err)	Ssalto/Duacs multimission Near real-time or Delayed time Maps of sea level anomalies Formal mapping error (gridded)
NRT- or DT-SLA	Ssalto/Duacs multimission Near real-time or Delayed time Sea level anomalies (along-track)
NRT- or DT-NRT- or DT-MADT (h)	Ssalto/Duacs multimission Near real-time or Delayed time Maps of absolute dynamic topography (gridded)
NRT- or DT-MADT (uv)	Ssalto/Duacs multimission Near real-time or Delayed time Geostrophic velocities associated to the Maps of absolute dynamic topography (gridded)
NRT- or DT-ADT	Ssalto/Duacs multimission Near real-time or Delayed time Absolute dynamic topography (along-track)
Monomission DT-SLA	Delayed time Sea level anomalies (along-track)
Monomission DT-CorSSH	Delayed time Corrected sea surface height (along-track)

## Basic Radar Altimetry Toolbox User Manual

---

NRT-MSWH	Near real-time Maps of Significant wave height (gridded)
NRT-MWind	Near real-time Maps of Wind speed modulus (gridded)

## Annex B.Y=F(X) parameter file keys

*NOTE: A help on parameter file keywords can be obtained by: "BratCreateYFX -k"*

FILE	Type : Str Count : [1-n] Input file name.
RECORD	Type : Str Count : 1 Record set name to take into account for a file.
OUTPUT	Type : Str Count : 1 Name of created/modified file.
OUTPUT_TITLE	Type : Str Count : [0-1] Title of created/modified file (string describing the content and which should appear as a graphic title, for example). (Default="")
SELECT	Type : Expr Count : [0-n] True for record values selected. (Default=1)
FIELD	Type : Expr Count : [1-20]=X Expression of fields of *RECORD* to take into account.
FIELD_NAME	Type : Name Count : X Name of the *FIELD* data
FIELD_TYPE	Type : KW1 Count : X Type of *FIELD* data.
FIELD_UNIT	Type : Unit Count : X Unit of *FIELD* expression.
FIELD_TITLE	Type : Str Count : X Long name describing *FIELD*. The one which should appear in graphics on axis or legends, for example.
DATA_MODE	Type : KW2 Count : [0-1] Keyword to indicate how data are stored/computed. (Default=MEAN)
X	Type : Expr Count : 1 Expression of fields of *RECORD* to take into account.
X_NAME	Type : Name Count : 1 Name of the *X* data
X_TYPE	Type : KW1 Count : 1 Type of *X* data (normally X, T or longitude).
X_UNIT	Type : Unit Count : 1 Unit of *X* expression
X_TITLE	Type : Str Count : 1 Long name describing *X*. The one which should appear in graphics on axis or legends, for example.
ALIAS_NAME	Type : Name Count : [0-n]=N Name of an alias. An alias is a value which can be used anywhere in another value of field by mean of %{{NAME}} construct. Names are case sensitive. If a name reference (%{{XXX}}) does not correspond to an actually defined alias, the expansion is an empty string. (Default=None)
ALIAS_VALUE	Type : Str Count : N The value of the alias. ALIAS_VALUE keyword must have at least as many occurrences as the ALIAS_NAME one.
VERBOSE	Type : Int Count : [0-1] Amount of output: 0=None...5=Debug. (Default=0)

=====

Description of types:

Name	String beginning with a letter and containing only letters, digits and '_'
Int	Integer
Expr	Combination of fields of the current record.
	An expression which can contain function calls like trigonometric, conversion, test...
Str	String. Leading and trailing blanks are ignored.
Unit	Unit string conforming to Udunits package and the special keyword 'DATE' which means that the data is a date.
KW1	Keywords: X/Y/Z/T/Latitude/Longitude/Data
KW2	Keywords: FIRST/LAST/MIN/MAX/MEAN/STDDEV/COUNT

## Annex C.Z=F(X,Y) parameter file keys

NOTE: A help on parameter file keywords can be obtained by: "BratCreateZFXY -k"

FILE	Type : Str	Count : [1-n]
	Input file name.	
OUTPUT	Type : Str	Count : 1
	Name of created/modified file.	
OUTPUT_TITLE	Type : Str	Count : [0-1]
	Title of created/modified file (string describing the content and which should appear as a graphic title, for example).	
	(Default="")	
SELECT	Type : Expr	Count : [0-n]
	True for record values selected.	
	(Default=1)	
RECORD	Type : Str	Count : 1
	Record set name to take into account for a file.	
DATA_MODE	Type : KW2	Count : [0-1]
	Keyword to indicate how data are stored/computed.	
	(Default=MEAN)	
POSITION_MODE	Type : KW3	Count : [0-1]
	How position is computed.	
	(Default=NEAREST)	
OUTSIDE_MODE	Type : KW4	Count : [0-1]
	How data outside limits are managed.	
	(Default=STRICT)	
X	Type : Expr	Count : 1
	Expression of fields of *RECORD* to take into account.	
X_NAME	Type : Name	Count : 1
	Name of the *X* data	
X_TYPE	Type : KW1	Count : 1
	Type of *X* data (normally X, T or longitude).	
X_UNIT	Type : Unit	Count : 1
	Unit of *X* expression	
X_TITLE	Type : Str	Count : 1
	Long name describing *X*. The one which should appear in graphics on axis or legends, for example.	
X_INTERVALS	Type : Int	Count : 1
	Number of intervals between Min and Max for *X*.	
	(Default=180 for lat 360 for lon)	
X_MIN	Type : Flt	Count : 1
	Min value for *X* expression storage.	
	(Default=-90 for lat, -180 for lon)	
X_MAX	Type : Flt	Count : 1
	Max value for *X* expression storage.	
	(Default=90 for lat, 180 for lon)	
X_LOESS_CUTOFF	Type : Int	Count : 1
	Distance (in dots) where LOESS filter reaches 0 along X axis. Must be an odd integer. If 1 or 0, Distance computation is disabled. Needed only if at least one filter is asked.	
	(Default=0)	
Y	Type : Expr	Count : 1
	Expression of fields of *RECORD* to take into account.	
Y_INTERVALS	Type : Int	Count : 1
	Number of intervals between Min and Max for *Y*.	
	(Default=180 for lat 360 for lon)	

Y_NAME	Type : Name Count : 1 Name of the *Y* data.
Y_TYPE	Type : KW1 Count : 1 Type of *Y* data (normally X, T or longitude).
Y_UNIT	Type : Unit Count : 1 Unit of *Y* expression.
Y_TITLE	Type : Str Count : 1 Long name describing *Y*. The one which should appear in graphics on axis or legends, for example.
Y_MIN	Type : Flt Count : 1 Min value for *Y* expression storage. (Default=-90 for lat, -180 for lon)
Y_MAX	Type : Flt Count : 1 Max value for *Y* expression storage. (Default=90 for lat, 180 for lon)
Y_LOESS_CUTOFF	Type : Int Count : 1 Distance (in dots) where LOESS filter reaches 0 along Y axis. Must be an odd integer. If 1 or 0, Distance computation is disabled. Needed only if at least one filter is asked. (Default=0)
FIELD	Type : Expr Count : [1-20]=X Expression of fields of *RECORD* to take into account.
FIELD_NAME	Type : Name Count : X Name of the *FIELD* data
FIELD_TYPE	Type : KW1 Count : X Type of *FIELD* data.
FIELD_UNIT	Type : Unit Count : X Unit of *FIELD* expression.
FIELD_TITLE	Type : Str Count : X Long name describing *FIELD*. The one which should appear in graphics on axis or legends, for example.
FIELD_FILTER	Type : KS1 Count : X How to filter the data.
ALIAS_NAME	Type : Name Count : [0-n]=N Name of an alias. An alias is a value which can be used anywhere in another value of field by mean of %{{NAME}} construct. Names are case sensitive. If a name reference (%{XXX}) does not correspond to an actually defined alias, the expansion is an empty string. (Default=None)
ALIAS_VALUE	Type : Str Count : N The value of the alias. ALIAS_VALUE keyword must have at least as many occurrences as the ALIAS_NAME one.
VERBOSE	Type : Int Count : [0-1] Amount of output: 0=None...5=Debug. (Default=0)

=====

#### Description of types:

Name	String beginning with a letter and containing only letters, digits and '_'
Flt	Floating point number
Int	Integer
Expr	Combination of fields of the current record. An expression which can contain function calls like trigonometric, conversion, test...

Str	String. Leading and trailing blanks are ignored.
Unit	Unit string conforming to Udunits package and the special keyword 'DATE' which means that the data is a date.
KW1	Keywords: X/Y/Z/T/Latitude/Longitude/Data
KW2	Keywords: FIRST/LAST/MIN/MAX/MEAN/STDDEV/COUNT
KW3	Keywords: EXACT/NEAREST EXACT: Measures which are exactly on boundaries (grid lines) are kept others are ignored NEAREST: Get the nearest boundary.
KW4	Keywords: STRICT/RELAXED/BLACK_HOLE STRICT: Measure outside limits are ignored RELAXED: Measure outside limits are ignored if they are farther than a half step from the limit. BLACK_HOLE: Everything outside the limit is considered to be on the limit.
KS1	Set of keywords from: NONE, LOESS_SMOOTH, LOESS_EXTRAPOLATE, LOESS (LOESS means LOESS_SMOOTH and LOESS_EXTRAPOLATE)

## Annex D. Display parameter file keys

NOTE: A help on parameter file keywords can be obtained by: "BratDisplay -k"

FILE	Type : Str	Count : [1-n]
	Input file name.	
FIELD	Type : Expr	Count : [1-23]=X
	Expression of fields of *RECORD* to take into account.	
FIELD_GROUP	Type : Int	Count : X
	Group id from where belongs *FIELD*. generally used to group many fields in one plot.	
DISPLAY_PROPERTIES	Type : Bool	Count : [0-1]
	Indicates if property panel is shown. (Default=No)	
DISPLAY_TITLE	Type : Str	Count : [0-1]
	Title of the plot to be displayed. (Default="")	
DISPLAY_ANIMATIONBAR	Type : Bool	Count : [0-1]
	Keyword to indicate if property panel is shown. (Default=No)	
DISPLAY_COLORBAR	Type : Bool	Count : [0-1]
	Keyword to indicate if color bar (legend) is shown. (Default=Yes)	
DISPLAY_CENTERLAT	Type : Flt	Count : [0-1]
	Latitude of the projection's center point. (Default=0)	
DISPLAY_CENTERLON	Type : Flt	Count : [0-1]
	Longitude of the projection's center point. (Default=0)	
DISPLAY_PROJECTION	Type : KW9	Count : [0-1]
	Projection to use for mapping the world globe. (Default=3D)	
DISPLAY_COASTRESOLUTION	Type : KW6	Count : [0-1]
	Resolution of the coast line drawn on the map. Recommended value: low. (Default=low)	
DISPLAY_ZOOM_LON1	Type : Flt	Count : [0-1]
	Zoom area west side. (Default=-180)	
DISPLAY_ZOOM_LON2	Type : Flt	Count : [0-1]
	Zoom area east side. (Default=180)	
DISPLAY_ZOOM_LAT1	Type : Flt	Count : [0-1]
	Zoom area south side. (Default=-90)	
DISPLAY_ZOOM_LAT2	Type : Flt	Count : [0-1]
	Zoom area north side. (Default=90)	
DISPLAY_GROUPBY_FILE	Type : Bool	Count : [0-1]
	For world plot. When several files are in input, this parameter indicates if fields are displayed in the same plot (group field by file) or in different plots (one plot by file). (Default=Yes)	
DISPLAY_XMINVALUE	Type : Flt	Count : [0-1]
	Minimum X coordinate value to use in XY plot. (Default=min of data values for X axis)	
DISPLAY_XMAXVALUE	Type : Flt	Count : [0-1]

	Maximum X coordinate value to use in XY plot. (Default=max of data values for X axis)	
DISPLAY_YMINVALUE	Type : Flt	Count : [0-1]
	Minimum Y coordinate value to use in XY plot. (Default=min of data values for Y axis)	
DISPLAY_YMAXVALUE	Type : Flt	Count : [0-1]
	Maximum Y coordinate value to use in XY plot. (Default=max of data values for Y axis)	
DISPLAY_XLABEL	Type : Str	Count : [0-1]
	X axis label to be displayed. (Default=field title or field name)	
DISPLAY_YLABEL	Type : Str	Count : [0-1]
	Y axis label to be displayed. (Default=field title or field name)	
DISPLAY_XTICKS	Type : Int	Count : [0-1]
	Number of ticks for the X axis. (Default=6)	
DISPLAY_YTICKS	Type : Int	Count : [0-1]
	Number of ticks for the Y axis. (Default=6)	
DISPLAY_NAME	Type : Str	Count : [0-n]=W
	Field name to be displayed.	
DISPLAY_OPACITY	Type : Flt	Count : 0 or W
	Opacity of the color value map image: 1.0 color is totally opaque 0.0 is completely transparent. (Default=0.7)	
DISPLAY_MINVALUE	Type : Flt	Count : 0 or W
	Minimum color table value to use in plot. (Default=min of data values)	
DISPLAY_MAXVALUE	Type : Flt	Count : 0 or W
	Maximum color table value to use in plot. (Default=max of data values)	
DISPLAY_NUMCOLORLABELS	Type : Int	Count : 0 or W
	Number of labels shown on the plot's color bar. (Default=2)	
DISPLAY_COLORTABLE	Type : Str	Count : 0 or W
	Name of a predefined color table: Aerosol Blackbody BlackToWhite Cloud Ozone GreenToRed Rainbow RedToGreen WhiteToBlack or name of a file containing the color table definition (absolute or relative path). (Default=Aerosol)	
DISPLAY_COLORCURVE	Type : KW5	Count : 0 or W
	Set the color table on a specific curve. (Default=Linear)	
DISPLAY_CONTOUR	Type : Bool	Count : 0 or W
	Indicates if the contour layer of the field is shown or not. (Default=No)	
DISPLAY_CONTOUR_NUMBER	Type : Int	Count : 0 or W
	Number of contour lines to generate (equally spaced contour values between specified range)	

	See DISPLAY_CONTOUR_MINVALUE and DISPLAY_CONTOUR_MAXVALUE). (Default=5)		
DISPLAY_CONTOUR_LABEL	Type : Bool Count : 0 or W Indicate if the contour labels (value) are shown or not. (Default=No)		
DISPLAY_CONTOUR_LABEL_NUMBER	Number of labels on each contour. (Default=1)	Type : Int	Count : 0 or W
DISPLAY_CONTOUR_MINVALUE	Minimum value to use to contour calculation. Default values are the same as the color scale one. (Default=min of data values)	Type : Flt	Count : 0 or W
DISPLAY_CONTOUR_MAXVALUE	Maximum value to use to contour calculation. Default values are the same as the color scale one. (Default=max of data values)	Type : Flt	Count : 0 or W
DISPLAY_SOLID_COLOR	Type : Bool Count : 0 or W Indicates if color layer of the field is shown or not. (Default=Yes)		
DISPLAY_COLOR	Type : KW7 Count : 0 or W Color name of the XY plot field. (Default=random color)		
DISPLAY_POINTS	Type : Bool Count : 0 or W Indicates if points are displayed in a XY plot (for the field). (Default=No)		
DISPLAY_LINES	Type : Bool Count : 0 or W Indicates if line is displayed in a XY plot (for the field). (Default=Yes)		
DISPLAY_POINTSIZE	Type : Flt Size of the points (XY plot, for the field). (Default=1.0)	Count : 0 or W	
DISPLAY_LINEWIDTH	Type : Flt Width of the line (XY plot, for the field). (Default=0.8)	Count : 0 or W	
DISPLAY_STIPPLEPATTERN	Type : KW10 Count : 0 or W Stipple pattern for the line (field) (XY plot). (Default=Full)		
DISPLAY_POINTGLYPH	Type : KW8 Glyph of the points (field) (XY plot). (Default=Circle)	Count : 0 or W	
DISPLAY_POINTFILLED	Type : Bool Count : 0 or W Indicates if points are filled or not. (Default=Yes)		
ALIAS_NAME	Type : Name Count : [0-n]=N Name of an alias. An alias is a value which can be used anywhere in another value of field by mean of %{{NAME}} construct. Names are case sensitive. If a name reference (%{XXX}) does not correspond to an actually defined alias, the expansion is an empty string. (Default=None)		
ALIAS_VALUE	Type : Str Count : N The value of the alias. ALIAS_VALUE keyword must have at least as many occurrences as the ALIAS_NAME one.		
VERBOSE	Type : Int Count : [0-1] Amount of output: 0=None...5=Debug. (Default=0)		

=====

Description of types:

Name	String beginning with a letter and containing only letters, digits and '_'
Bool	Boolean true if : YES/Y/TRUE/T/OUI/O/VRAI/V/1 false if : NO/N/FALSE/F/NON/N/FAUX/0
Flt	Floating point number
Int	Integer
Expr	Combination of fields of the current record. An expression which can contain function calls like trigonometric, conversion, test...
Str	String. Leading and trailing blanks are ignored.
KW5	Keywords: cosine, linear, sqrt (square root)
KW6	Keywords: In increasing resolution: crude, low, intermediate, full
KW7	Keywords: AQUAMARINE, BLACK, BLUE, BLUE VIOLET, BROWN, CADET BLUE, CORAL, CORNFLOWER BLUE, CYAN, DARK GREY, DARK GREEN, DARK OLIVE GREEN, DARK ORCHID, DARK SLATE BLUE, DARK SLATE GREY, DARK TURQUOISE, DIM GREY, FIREBRICK, FOREST GREEN, GOLD, GOLDENROD, GREY, GREEN, GREEN YELLOW, INDIAN RED, KHAKI, LIGHT BLUE, LIGHT GREY, LIGHT STEEL BLUE, LIME GREEN, MAGENTA, MAROON, MEDIUM AQUAMARINE, MEDIUM BLUE, MEDIUM FOREST GREEN, MEDIUM GOLDENROD, MEDIUM ORCHID, MEDIUM SEA GREEN, MEDIUM SLATE BLUE, MEDIUM SPRING GREEN, MEDIUM TURQUOISE, MEDIUM VIOLET RED, MIDNIGHT BLUE, NAVY, ORANGE, ORANGE RED, ORCHID, PALE GREEN, PINK, PLUM, PURPLE, RED, SALMON, SEA GREEN, SIENNA, SKY BLUE, SLATE BLUE, SPRING GREEN, STEEL BLUE, TAN, THISTLE, TURQUOISE, VIOLET, VIOLET RED, WHEAT, WHITE, YELLOW, YELLOW GREEN.
KW8	Keywords: ARROW, CIRCLE, CROSS, DASH, DIAMOND, HOOKEDARROW, SQUARE, THICKARROW, THICKCROSS, TRIANGLE
KW9	Keywords: 3D, Azimuthal Equidistant, Lambert Cylindrical, Lambert Azimuthal, Mercator, Mollweide, Plate Caree, Robinson
KW10	Keywords: DASHTINY, DASH, DASHDOT, DOT, FULL

## Annex E.BRATHL-IDL API

The BRAT-IDL API consists of a handful of IDL 'named structures' and functions.

```
=====
'named structures':
=====
```

BRATHL\_DATEYMDHMSM  
BRATHL\_DATEDSM  
BRATHL\_DATESECOND  
BRATHL\_DATEJULIAN

BRATHL\_DATEYMDHMSM named structure

---

This structure represents a YYYY-MM-DD HH:MN:SS:MS date structure :

YEAR  
MONTH  
DAY  
HOUR  
MINUTE  
SECOND  
MUSECOND

Example :

```
MyDate={BRATHL_DATEYMDHMSM}
```

```
MyDate.YEAR=2003  
MyDate.MONTH=12  
MyDate.DAY=5  
MyDate.HOUR=18  
MyDate.MINUTE=0  
MyDate.SECOND=21  
MyDate.MUSECOND=1069
```

BRATHL\_DATEDSM named structure

---

This structure represents a day/seconds/microseconds date structure:

REFDATE	reference date
DAYS	numbers of days
SECONDS	numbers of seconds
MUSECONDS	numbers of microseconds

REFDATE is the reference date i.e :

- 0: 1950-01-01 00:00:00.0
- 1: 1958-01-01 00:00:00.0
- 2: 1985-01-01 00:00:00.0
- 3: 1990-01-01 00:00:00.0
- 4: 2000-01-01 00:00:00.0

5: user reference 1  
6: user reference 2

values of 5 and 6 allow the user to set two specific reference date of his choice  
(see BRATHL\_SETREFUSER1 and BRATHL\_SETREFUSER2 functions)

:

Example :

```
MyDate={BRATHL_DATEDSM}
```

```
MyDate.REFDATE=3
MyDate.DAYS=423
MyDate.SECONDS=5
MyDate.MUSECONDS=0
```

BRATHL\_DATESECONDS named structure

---

This structure represents a decimal seconds date structure:

REFDATE	reference date - see :BRATHL_DATEDSM
NBSECONDS	decimal numbers of seconds (seconds.microseconds)

:

Example :

```
MyDate={BRATHL_DATESECONDS}
```

```
MyDate.REFDATE=0
MyDate.NBSECONDS=56236.0253
```

BRATHL\_DATEJULIAN named structure

---

This structure represents a decimal julian date structure:

REFDATE	reference date - see :BRATHL_DATEDSM
JULIAN	decimal julian day

:

Example :

```
MyDate={BRATHL_DATESECONDS}
```

```
MyDate.REFDATE=0
MyDate.JULIAN=123.569
```

---

=====

Functions

=====

---

Date conversion/computation functions

---

BRATHL\_DAYOFYEAR

BRATHL\_DIFFDSM  
BRATHL\_DIFFJULIAN  
BRATHL\_DIFFYMDHMSM

BRATHL\_DSM2JULIAN  
BRATHL\_DSM2SECONDS  
BRATHL\_DSM2YMDHMSM

BRATHL\_JULIAN2DSM  
BRATHL\_JULIAN2SECONDS  
BRATHL\_JULIAN2YMDHMSM

BRATHL\_SECONDS2DSM  
BRATHL\_SECONDS2JULIAN  
BRATHL\_SECONDS2YMDHMSM

BRATHL\_NOWYMDHMSM

BRATHL\_YMDHMSM2DSM  
BRATHL\_YMDHMSM2JULIAN  
BRATHL\_YMDHMSM2SECONDS

BRATHL\_SETREFUSER1  
BRATHL\_SETREFUSER2

=====

Cycle/date conversion functions

=====

BRATHL\_CYCLE2YMDHMSM  
BRATHL\_YMDHMSM2CYCLE

=====

Data reading function

=====

BRATHL\_READDATA

### BRATHL\_DAYOFYEAR

---

Retrieves the day of year of a date

BRATHL\_DAYOFYEAR(BRATHL\_DATEYMDHMSM dateYMDHMSM, ULONG dayOfYear)

[in] dateYMDHMSM : date

[out] dayOfYear : day of year of the date parameter

returns 0 or error code (see Date error codes in brathl general documentation)

Example :

MyDate={BRATHL\_DATEYMDHMSM}

MyDate.YEAR=2003

MyDate.MONTH=12

MyDate.DAY=5

MyDate.HOUR=18

MyDate.MINUTE=0

MyDate.SECOND=21

MyDate.MUSECOND=1069

```
dayOfYear=0L  
r = BRATHL_DAYOFYEAR(MyDate, dayOfYear)  
print, r, dayOfYear
```

### BRATHL\_DIFFDSM

---

Computes the difference between two dates (date1 - date2)  
the result is expressed in a decimal number of seconds

```
BRATHL_DIFFDSM(BRATHL_DATEDSM date1, BRATHL_DATEDSM date2, DOUBLE diff)
```

[in] date1  
[in] date2  
[out] diff : difference in seconds (date1 - date2)

return 0 or error code (see Date error codes in brathl general documentation)

Example:

```
d1={BRATHL_DATEDSM}  
d1.REFDATE=3  
d1.DAYS=423  
d1.SECONDS=5  
d1.MUSECONDS=0
```

```
d2={BRATHL_DATEDSM}  
d2.REFDATE=2  
d2.DAYS=36  
d2.SECONDS=54  
d2.MUSECONDS=2536
```

```
diff = 0.0D  
r = BRATHL_DIFFYMDHMSM(d1, d2, diff)  
print, r, diff
```

### BRATHL\_DIFFJULIAN

---

Computes the difference between two dates (date1 - date2)  
the result is expressed in a decimal number of seconds

```
BRATHL_DIFFJULIAN(BRATHL_DIFFJULIAN date1, BRATHL_DIFFJULIAN date2, DOUBLE diff)
```

[in] date1  
[in] date2  
[out] diff : difference in seconds (date1 - date2)

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DIFFDSM

### BRATHL\_DIFFYMDHMSM

-----  
Computes the difference between two dates (date1 - date2)  
the result is expressed in a decimal number of seconds

BRATHL\_DIFFYMDHMSM(BRATHL\_DIFFYMDHMSM date1, BRATHL\_DIFFYMDHMSM date2,  
DOUBLE diff)

[in] date1  
[in] date2  
[out] diff : difference in seconds (date1 - date2)

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DIFFDSM

### BRATHL\_DSM2JULIAN

---

Converts a days-seconds-microseconds date into a decimal julian date, according to refDate parameter

BRATHL\_DSM2JULIAN(BRATHL\_DATEDSM dateDSM, INT refDate, BRATHL\_DATEJULIAN  
dateJulian);

[in] dateDSM : date to convert  
[in] refDate : date reference conversion  
[out] dateJulian : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example :

dIn={BRATHL\_DATEDSM}

dIn.REFDATE=3  
dIn.DAYS=423  
dIn.SECONDS=5  
dIn.MUSECONDS=0

dOut={BRATHL\_DATEJULIAN}

refDateDestination = 0

r = BRATHL\_DSM2JULIAN(dIn, refDateDestination, dOut)  
print, r, dOut.REFDATE, dOut.JULIAN

### BRATHL\_DSM2SECONDS

---

Converts a days-seconds-microseconds date into secnods, according to refDate parameter

BRATHL\_DSM2SECONDS(BRATHL\_DATEDSM dateDSM, INT refDate, BRATHL\_DATESECOND  
dateSeconds);

[in] dateDSM : date to convert  
[in] refDate : date reference conversion  
[out] dateSeconds : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

### BRATHL\_DSM2YMDHMSM

---

Converts a days-seconds-microseconds date into a year, month, day, hour, minute, second, microsecond date

BRATHL\_DSM2YMDHMSM(BRATHL\_DATEDSM                   dateDSM,                   BRATHL\_DATEYMDHMSM  
dateYMDHMSM);

[in] dateDSM : date to convert

[in] refDate : date reference conversion

[out] dateYMDHMSM : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example:

```
dIn={BRATHL_DATEDSM}
```

```
dIn.REFDATE=3
```

```
dIn.DAYS=423
```

```
dIn.SECONDS=5
```

```
dIn.MUSECONDS=0
```

```
dOut={BRATHL_DATEYMDHMSM}
```

```
refDateDestination = 0
```

```
r = BRATHL_DSM2YMDHMSM(dIn, dOut)
```

```
print, r, dOut.YEAR, dOut.JULIAN, dOut.MONTH, dOut.DAY, dOut.HOUR, dOut.MINUTE,  
dOut.SECOND, dOut.MUSECOND
```

### BRATHL\_JULIAN2DSM

---

Converts a decimal julian date into a days-seconds-microseconds date, according to refDate parameter

BRATHL\_JULIAN2DSM(BRATHL\_DATEJULIAN   dateJulian,   INT   refDate,   BRATHL\_DATEDSM  
dateDSM);

[in] dateJulian : date to convert

[in] refDate : date reference conversion

[out] dateDSM : result of conversion

return 0 or error code (see Date error codes in brathl general documentation)

BRATHL\_DSM2YMDHMSM(BRATHL\_DATEDSM                   dateDSM,                   BRATHL\_DATEYMDHMSM  
dateYMDHMSM);

[in] dateDSM : date to convert

[in] refDate : date reference conversion

[out] dateYMDHMSM : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

#### BRATHL\_JULIAN2SECONDS

---

Converts a decimal julian date into seconds, according to refDate parameter

BRATHL\_JULIAN2SECONDS(BRATHL\_DATEJULIAN                   dateJulian,                   INT                   refDate,  
BRATHL\_DATESECOND dateSeconds);

[in] dateJulian : date to convert

[in] refDate : date reference conversion

[out] dateSeconds : result of conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

#### BRATHL\_JULIAN2YMDHMSM

---

Converts a decimal julian date into a year, month, day, hour, minute, second, microsecond date

BRATHL\_JULIAN2YMDHMSM(BRATHL\_DATEJULIAN                   dateJulian,                   BRATHL\_DATEYMDHMSM  
dateYMDHMSM);

[in] dateJulian : date to convert

[in] refDate : date reference conversion

[out] dateYMDHMSM : result of conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2YMDHMSM

#### BRATHL\_SECONDS2DSM

---

Converts seconds into a days-seconds-microseconds date, according to refDate parameter

BRATHL\_SECONDS2DSM(BRATHL\_DATESECOND dateSeconds, INT refDate, BRATHL\_DATEDSM  
dateDSM);

[in] dateSeconds : date to convert

[in] refDate : date reference conversion

[out] dateDSM : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

#### BRATHL\_SECONDS2YMDHMSM

---

Converts seconds into a a decimal julian date, according to refDate parameter

BRATHL\_SECONDS2YMDHMSM(BRATHL\_DATESECOND dateSeconds, INT refDate, BRATHL\_DATEJULIAN dateJulian)

[in] dateSeconds : date to convert  
[in] refDate : date reference conversion  
[out] dateJulian : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

BRATHL\_NOWYMDHMSM

---

Gets the current date/time,

LIBRATHL\_API int32\_t brathl\_NowYMDHMSM(brathl\_DateYMDHMSM \*dateYMDHMSM);

[out] dateYMDHMSM : current date/time

BRATHL\_NOWYMDHMSM(BRATHL\_DATEYMDHMSM dateYMDHMSM)

Example: see BRATHL\_DSM2JULIAN

```
dOut={BRATHL_DATEYMDHMSM}
r = BRATHL_NOWYMDHMSM(dOut)
print, r, dOut.YEAR, dOut.JULIAN, dOut.MONTH, dOut.DAY, dOut.HOUR, dOut.MINUTE,
dOut.SECOND, dOut.MUSECOND
```

BRATHL\_YMDHMSM2DSM

---

Converts a year, month, day, hour, minute, second, microsecond date into a days-seconds-microseconds date,  
according to refDate parameter

BRATHL\_YMDHMSM2DSM(BRATHL\_DATEYMDHMSM dateYMDHMSM, INT refDate, BRATHL\_DATEDSM dateDSM)

[in] dateYMDHMSM : date to convert  
[in] refDate : date reference conversion  
[out] dateDSM : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

BRATHL\_YMDHMSM2JULIAN

---

Converts a year, month, day, hour, minute, second, microsecond date into a decimal julian date, according to refDate parameter

BRATHL\_YMDHMSM2JULIAN(BRATHL\_DATEYMDHMSM      dateYMDHMSM,      INT      refDate,  
BRATHL\_DATEJULIAN dateJulian)

[in] dateYMDHMSM : date to convert

[in] refDate : date reference conversion

[out] dateJulian : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

#### BRATHL\_YMDHMSM2SECONDS

---

Converts a year, month, day, hour, minute, second, microsecond date into a seconds, according to refDate parameter

BRATHL\_YMDHMSM2SECONDS(BRATHL\_DATEYMDHMSM      dateYMDHMSM,      INT      refDate,  
BRATHL\_DATESECOND dateSeconds)

[in] dateYMDHMSM : date to convert

[in] refDate : date reference conversion

[out] dateSeconds : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

#### BRATHL\_SETREFUSER1

#### BRATHL\_SETREFUSER2

---

Set user-defined reference dates

BRATHL\_SETREFUSER1(STRING dateRef)

[in] dateRef : date to set - format: YYYY-MM-DD HH:MN:SS.MS

return 0 or error code (see Date error codes in brathl general documentation)

Example:

```
dateRefUser1 = '2001 01 12 14:57:23:1456'  
dateRefUser2 = '2005 11 14'
```

```
brathl_setrefuser1(dateRefUser1)
brathl_setrefuser2(dateRefUser2)
```

```
MyDate={BRATHL_DATEDSM}
```

```
. Set user-defined ref. date 2001 01 12 14:57:23:1456
```

```
MyDate.REFDATE=5
```

```
MyDate.DAYS=423
```

```
MyDate.SECONDS=5
```

```
MyDate.MUSECONDS=0
```

```
AnotherDate={BRATHL_DATEDSM}
```

```
. Set user-defined ref. date 2005 11 14
```

```
AnotherDate.REFDATE=6
```

```
AnotherDate.DAYS=423
```

```
AnotherDate.SECONDS=5
```

```
AnotherDate.MUSECONDS=0
```

```
; ref. date for MyDate is now 2005 11 14
```

```
MyDate.REFDATE=6
```

```
brathl_setrefuser2('2005 05 18 13:08:00')
```

```
; ref. date for MyDate and AnotherDate is now 2005 05 18 13:08:00
```

## BRATHL\_CYCLE2YMDHMSM

---

Converts a cycle/pass into a date

BRATHL\_CYCLE2YMDHMSM(INT mission, ULONG cycle, ULONG pass, BRATHL\_DATEYMDHMSM dateYMDHMSM)

[in] mission : mission type :

0 : Topex/Poseidon

1 : Jason-1

2 : ERS2

3 : Envisat

4 : ERS1-A

5 : ERS1-B

6 : GFO

[in] cycle : number of cycle to convert

[in] pass : number of pass in the cycle to convert

[out] dateYMDHMSM : date corresponding to the cycle/pass

return 0 or error code (see Cycle/date conversion error codes in brathl general documentation)

Example:

```
cycle=120L
```

```
pass=153L
```

```
mission=3
```

```
dOut={BRATHL_DATEYMDHMSM}

r = BRATHL_CYCLE2YMDHMSM(mission, cycle, pass, dOut)
print, "result ", r

print, "mission ", mission , " cycle ", cycle, " pass ", pass
print, "Y", dOut.year, " M ", dOut.month, " D ", dOut.day, " H ", dOut.hour, " MN ", dOut.minute, " S ",
dOut.second, " MS ", dOut.muSecond
```

### BRATHL\_YMDHMSM2CYCLE

---

Converts a date into a cycle/pass

BRATHL\_YMDHMSM2CYCLE(INT mission, BRATHL\_DATEYMDHMSM dateYMDHMSM, ULONG cycle, ULONG pass)

[in] mission : mission type :

- 0 : Topex/Poseidon
- 1 : Jason-1
- 2 : ERS2
- 3 : Envisat
- 4 : ERS1-A
- 5 : ERS1-B
- 6 : GFO

[in] dateYMDHMSM : date to convert

[out] cycle : number of cycle

[out] pass : number of pass in the cycle

return 0 or error code (see Cycle/date conversion error codes in brathl general documentation)

Example:

```
cycle=0L
pass=0L
mission=1

dIn={BRATHL_DATEYMDHMSM}
dIn.YEAR=2003
dIn.MONTH=12
dIn.DAY=5
dIn.HOUR=18
dIn.MINUTE=0
dIn.SECOND=21
dIn.MUSECOND=1069
```

```
r = BRATHL_YMDHMSM2CYCLE(mission, dIn, cycle, pass)
print, "result ", r
```

```
print, "Y", dOut.year, " M ", dOut.month, " D ", dOut.day, " H ", dOut.hour, " MN ", dOut.minute, " S ",
dOut.second, " MS ", dOut.muSecond
print, "mission ", mission , " cycle ", cycle, " pass ", pass
```

## BRATHL\_READDATA

---

Read data from a set of files

Each measure for a data is a scalar value (a single number)

[in] fileNames : file name string (one file) or file names array

[in] recordName : Name of the fields record (for netCdf files recordName is 'data')

[in] selection : Expression involving data fields which has to be true to select returned data.

(if empty string no selection is done (all data is selected))

[in] dataExpressions : Expression string (one expression) or expressions array applied to data fields to build the wanted value.

[in] units : Wanted unit for each expression (string (one unit) or units array).

(if empty string, no unit conversion is applied to the data of the corresponding expression. When a unit conversion has to be applied, the result of the expression is considered to be the base unit (SI). For example if the wanted unit is gram/l, the unit of the expression is supposed to be kilogram/m3 (internaly all data are converted to base unit of the actual fields unit which is coherent with the above assumption)).

[in/out] results : Data read. Must be an array (dim = number of dataExpressions) to values to read.

[in] ignoreOutOfRange : Skip excess data.

0=false, other = true

Must be false if statistics is true.

[in] statistics : returns statistics on data instead of data themselves

0=false, other = true

If statistics is true, ignoreOutOfRange must be false.

The returned values (5 values) for each

expression are:

- Count of valid data taken into account.

Invalid data are those which are equal to the default/missing value

- Mean of the valid data.

- Standard deviation of the valid data

- Minimum value of the valid data

- Maximum value of the valid data

[in] defaultValue : value to use for default/missing values

This is the value you want to indicate that a value is missing or invalid.

return 0 or error code.

Example:

```
; Set data input file  
files=SINDGEN(3)  
files[0]="/data/samples/JA1_GDR_2PaP124_001.CNES"  
files[1]="/data/samples/JA1_GDR_2PaP124_002.CNES"  
files[2]="/data/samples/JA1_GDR_2PaP124_003.CNES"
```

```
; Set record name  
record="data"
```

```
; Set data selection - (set selection = "" to retrieve all data row)  
selection="latitude > 20 && latitude < 30"
```

```
; Set expressions (here 2 expressions)
expr=SINDGEN(2)
; A compute expression
expr[0]="latitude + longitude"
; A single expression
expr[1]="swh_ku"

; Set units for each expression
units=SINDGEN(2)
; Convert unit to radians for expression 1 (latitude + longitude)
units[0]="radians"
; No unit conversion for expression 2 (swh_ku) - result will be in SI unit.
units[1]=""

; Create results array (dimension is number of expression)
dataResults=DINDGEN(2)

ignoreOutOfrange=0

; No statistics
statistics=0

; Default value is 0
defaultValue=0

; Call ReadData function

r = BRATHL_READDATA(files, record, selection, expr, units, dataResults, ignoreOutOfrange, statistics,
defaultValue)
print, "return code ", r

print, size(dataResults)

print, "NDIMS", size(dataResults, /N_DIMENSIONS)
print, "DIMS" , size(dataResults, /DIMENSIONS)
print, "NELTS", size(dataResults, /N_ELEMENTS)
print, "TYPE", size(dataResults, /TYPE)

dim = size(dataResults, /DIMENSIONS)

; Print data value on the screen
for i = 0, 1 do begin
  for j = 0,dim[1] - 1 do begin
    print, "Data", i,j, data[i,j]
  endfor
endfor
```

## Annex F.BRATHL-MATLAB API

The BRATHL-MATLAB API consists of just a handful of Matlab structures and functions.

```
=====
    structures
=====
```

BRATHL\_DATEYMDHMSM = 0  
BRATHL\_DATEDSM = 1  
BRATHL\_DATESECOND = 2  
BRATHL\_DATEJULIAN = 3

To create a structure, use BRATHL\_CREATESTRUCT (see description below)

BRATHL\_DATEYMDHMSM structure

```
-----
```

This structure represents a YYYY-MM-DD HH:MN:SS:MS date structure :

YEAR  
MONTH  
DAY  
HOUR  
MINUTE  
SECOND  
MUSECOND

Example :

```
MyDate=BRATHL_CREATESTRUCT(0)
```

```
MyDate.YEAR=2003
MyDate.MONTH=12
MyDate.DAY=5
MyDate.HOUR=18
MyDate.MINUTE=0
MyDate.SECOND=21
MyDate.MUSECOND=1069
```

BRATHL\_DATEDSM structure

```
-----
```

This structure represents a day/seconds/microseconds date structure:

REFDATE	reference date
DAYS	numbers of days
SECONDS	numbers of seconds
MUSECONDS	numbers of microseconds

REFDATE is the reference date i.e :

- 0: 1950-01-01 00:00:00.0
- 1: 1958-01-01 00:00:00.0
- 2: 1985-01-01 00:00:00.0
- 3: 1990-01-01 00:00:00.0
- 4: 2000-01-01 00:00:00.0

5: user reference 1  
6: user reference 2

values of 5 and 6 allow the user to set two specific reference date of his choice  
(see BRATHL\_SETREFUSER1 and BRATHL\_SETREFUSER2 functions)

:

Example :

MyDate=BRATHL\_CREATESTRUCT(1)

MyDate.REFDATE=3  
MyDate.DAYS=423  
MyDate.SECONDS=5  
MyDate.MUSECONDS=0

BRATHL\_DATESECONDS structure

---

This structure represents a decimal seconds date structure:

REFDATE reference date - see :BRATHL\_DATEDSM  
NBSECONDS decimal numbers of seconds (seconds.microseconds)

:

Example :

MyDate=BRATHL\_CREATESTRUCT(2)

MyDate.REFDATE=0  
MyDate.NBSECONDS=56236.0253

BRATHL\_DATEJULIAN structure

---

This structure represents a decimal julian date structure:

REFDATE reference date - see :BRATHL\_DATEDSM  
JULIAN decimal julian day

:

Example :

MyDate=BRATHL\_CREATESTRUCT(3)

MyDate.REFDATE=0  
MyDate.JULIAN=123.569

=====

Functions

=====

=====

structure creation functions

=====

BRATHL\_CREATESTRUCT

=====

Date conversion/computation functions

=====

BRATHL\_DAYOFYEAR

BRATHL\_DIFFDSM

BRATHL\_DIFFJULIAN

BRATHL\_DIFFYMDHMSM

BRATHL\_DSM2JULIAN

BRATHL\_DSM2SECONDS

BRATHL\_DSM2YMDHMSM

BRATHL\_JULIAN2DSM

BRATHL\_JULIAN2SECONDS

BRATHL\_JULIAN2YMDHMSM

BRATHL\_SECONDS2DSM

BRATHL\_SECONDS2JULIAN

BRATHL\_SECONDS2YMDHMSM

BRATHL\_NOWYMDHMSM

BRATHL\_YMDHMSM2DSM

BRATHL\_YMDHMSM2JULIAN

BRATHL\_YMDHMSM2SECONDS

BRATHL\_SETREFUSER1

BRATHL\_SETREFUSER2

=====

Cycle/date conversion functions

=====

To convert cycle <-> date, these functions use an ascii parameter file (ascii file) with records :

field 1 : Name of the mission

field 2 : Cycle reference

field 3 : Pass reference

field 4 : Reference date in decimal julian day

Each field has to be separated by, at least, a non-numeric character

The file can contained several records for a same mission.

Only the field with the greatest date is taken into account

You can add records.

You can add comments, commented lines start by '#' character.

If the file doesn't exist, default values are :

Name	Cycle	Pass	Reference date
Jason-1	99	230	19987.9081795
Topex/Poseidon	442	230	19987.9127535
ERS2	66	598	18831.768334
ERS1-A	15	1	15636.938955
ERS1-B	42	108	16538.6732895
ENVISAT	30	579	19986.106016

BRATHL\_CYCLE2YMDHMSM

## BRATHL\_YMDHMSM2CYCLE

### BRATHL\_DAYOFYEAR

---

Retrieves the day of year of a date

dayOfYear = BRATHL\_DAYOFYEAR(BRATHL\_DATEYMDHMSM dateYMDHMSM)

[in] dateYMDHMSM : date

[out] dayOfYear : day of year of the date parameter

Example :

```
MyDate={BRATHL_DATEYMDHMSM}
```

```
MyDate.YEAR=2003
```

```
MyDate.MONTH=12
```

```
MyDate.DAY=5
```

```
MyDate.HOUR=18
```

```
MyDate.MINUTE=0
```

```
MyDate.SECOND=21
```

```
MyDate.MUSECOND=1069
```

```
dayOfYear=0L
```

```
r = BRATHL_DAYOFYEAR(MyDate, dayOfYear)
```

```
print, r, dayOfYear
```

### BRATHL\_DIFFDSM

---

Computes the difference between two dates (date1 - date2)

the result is expressed in a decimal number of seconds

BRATHL\_DIFFDSM(BRATHL\_DATEDSM date1, BRATHL\_DATEDSM date2, DOUBLE diff)

[in] date1

[in] date2

[out] diff : difference in seconds (date1 - date2)

return 0 or error code (see Date error codes in brathl general documentation)

Example:

```
d1={BRATHL_DATEDSM}
```

```
d1.REFDATE=3
```

```
d1.DAYS=423
```

```
d1.SECONDS=5
```

```
d1.MUSECONDS=0
```

```
d2={BRATHL_DATEDSM}
```

```
d2.REFDATE=2
```

```
d2.DAYS=36  
d2.SECONDS=54  
d2.MUSECONDS=2536
```

```
diff = 0.0D  
r = BRATHL_DIFFYMDHMSM(d1, d2, diff)  
print, r, diff
```

### BRATHL\_DIFFJULIAN

---

Computes the difference between two dates (date1 - date2)  
the result is expressed in a decimal number of seconds

BRATHL\_DIFFJULIAN(BRATHL\_DIFFJULIAN date1, BRATHL\_DIFFJULIAN date2, DOUBLE diff)

[in] date1  
[in] date2  
[out] diff : difference in seconds (date1 - date2)

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DIFFDSM

### BRATHL\_DIFFYMDHMSM

---

Computes the difference between two dates (date1 - date2)  
the result is expressed in a decimal number of seconds

BRATHL\_DIFFYMDHMSM(BRATHL\_DIFFYMDHMSM date1, BRATHL\_DIFFYMDHMSM date2,  
DOUBLE diff)

[in] date1  
[in] date2  
[out] diff : difference in seconds (date1 - date2)

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DIFFDSM

### BRATHL\_DSM2JULIAN

---

Converts a days-seconds-microseconds date into a decimal julian date, according to refDate parameter

BRATHL\_DSM2JULIAN(BRATHL\_DATEDSM dateDSM, INT refDate, BRATHL\_DATEJULIAN dateJulian);

[in] dateDSM : date to convert  
[in] refDate : date reference conversion  
[out] dateJulian : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example :

```
dIn={BRATHL_DATEDSM}  
  
dIn.REFDATE=3  
dIn.DAYS=423  
dIn.SECONDS=5  
dIn.MUSECONDS=0  
  
dOut={BRATHL_DATEJULIAN}  
  
refDateDestination = 0  
  
r = BRATHL_DSM2JULIAN(dIn, refDateDestination, dOut)  
print, r, dOut.REFDATE, dOut.JULIAN
```

### BRATHL\_DSM2SECONDS

---

Converts a days-seconds-microseconds date into secnods, according to refDate parameter

**BRATHL\_DSM2SECONDS(BRATHL\_DATEDSM dateDSM, INT refDate, BRATHL\_DATESECOND dateSeconds);**

[in] dateDSM : date to convert  
[in] refDate : date reference conversion  
[out] dateSeconds : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

### BRATHL\_DSM2YMDHMSM

---

Converts a days-seconds-microseconds date into a year, month, day, hour, minute, second, microsecond date

**BRATHL\_DSM2YMDHMSM(BRATHL\_DATEDSM dateDSM, BRATHL\_DATEYMDHMSM dateYMDHMSM);**

[in] dateDSM : date to convert  
[in] refDate : date reference conversion  
[out] dateYMDHMSM : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example:

```
dIn={BRATHL_DATEDSM}  
  
dIn.REFDATE=3  
dIn.DAYS=423  
dIn.SECONDS=5  
dIn.MUSECONDS=0
```

```
dOut={BRATHL_DATEYMDHMSM}  
refDateDestination = 0  
  
r = BRATHL_DSM2YMDHMSM(dIn, dOut)  
print, r, dOut.YEAR, dOut.JULIAN, dOut.MONTH, dOut.DAY, dOut.HOUR, dOut.MINUTE,  
dOut.SECOND, dOut.MUSECOND
```

### **BRATHL\_JULIAN2DSM**

---

Converts a decimal julian date into a days-seconds-microseconds date, according to refDate parameter

```
BRATHL_JULIAN2DSM(BRATHL_DATEJULIAN dateJulian, INT refDate, BRATHL_DATEDSM  
dateDSM);
```

[in] dateJulian : date to convert  
[in] refDate : date reference conversion  
[out] dateDSM : result of conversion  
return 0 or error code (see Date error codes in brathl general documentation)

```
BRATHL_DSM2YMDHMSM(BRATHL_DATEDSM dateDSM, BRATHL_DATEYMDHMSM  
dateYMDHMSM);
```

[in] dateDSM : date to convert  
[in] refDate : date reference conversion  
[out] dateYMDHMSM : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

### **BRATHL\_JULIAN2SECONDS**

---

Converts a decimal julian date into seconds, according to refDate parameter

```
BRATHL_JULIAN2SECONDS(BRATHL_DATEJULIAN dateJulian, INT refDate,  
BRATHL_DATESECOND dateSeconds)
```

[in] dateJulian : date to convert  
[in] refDate : date reference conversion  
[out] dateSeconds : result of conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

### **BRATHL\_JULIAN2YMDHMSM**

---

Converts a decimal julian date into a year, month, day, hour, minute, second, microsecond date

```
BRATHL_JULIAN2YMDHMSM(BRATHL_DATEJULIAN dateJulian, BRATHL_DATEYMDHMSM  
dateYMDHMSM);
```

[in] dateJulian : date to convert

[in] refDate : date reference conversion

[out] dateYMDHMSM : result of conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2YMDHMSM

#### BRATHL\_SECONDS2DSM

---

Converts seconds into a days-seconds-microseconds date, according to refDate parameter

BRATHL\_SECONDS2DSM(BRATHL\_DATESECOND dateSeconds, INT refDate, BRATHL\_DATEDSM dateDSM);

[in] dateSeconds : date to convert

[in] refDate : date reference conversion

[out] dateDSM : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

#### BRATHL\_SECONDS2JULIAN

---

Converts seconds into a decimal julian date, according to refDate parameter

BRATHL\_SECONDS2JULIAN(BRATHL\_DATESECOND dateSeconds, INT refDate,  
BRATHL\_DATEJULIAN dateJulian)

[in] dateSeconds : date to convert

[in] refDate : date reference conversion

[out] dateJulian : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

#### BRATHL\_SECONDS2YMDHMSM

---

Converts seconds into a a decimal julian date, according to refDate parameter

BRATHL\_SECONDS2YMDHMSM(BRATHL\_DATESECOND dateSeconds, INT refDate,  
BRATHL\_DATEJULIAN dateJulian)

[in] dateSeconds : date to convert

[in] refDate : date reference conversion

[out] dateJulian : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

**BRATHL\_NOWYMDHMSM**

---

Gets the current date/time,

**LIBRATHL\_API int32\_t brathl\_NowYMDHMSM(brathl\_DateYMDHMSM \*dateYMDHMSM);**

[out] dateYMDHMSM : current date/time

**BRATHL\_NOWYMDHMSM(BRATHL\_DATEYMDHMSM dateYMDHMSM)**

Example: see BRATHL\_DSM2JULIAN

```
dOut={BRATHL_DATEYMDHMSM}  
r = BRATHL_NOWYMDHMSM(dOut)  
print, r, dOut.YEAR, dOut.JULIAN, dOut.MONTH, dOut.DAY, dOut.HOUR, dOut.MINUTE,  
dOut.SECOND, dOut.MUSECOND
```

**BRATHL\_YMDHMSM2DSM**

---

Converts a year, month, day, hour, minute, second, microsecond date into a days-seconds-microseconds date,  
according to refDate parameter

**BRATHL\_YMDHMSM2DSM(BRATHL\_DATEYMDHMSM dateYMDHMSM, INT refDate, BRATHL\_DATEDSM dateDSM)**

[in] dateYMDHMSM : date to convert

[in] refDate : date reference conversion

[out] dateDSM : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

**BRATHL\_YMDHMSM2JULIAN**

---

Converts a year, month, day, hour, minute, second, microsecond date into a decimal julian date,  
according to refDate parameter

**BRATHL\_YMDHMSM2JULIAN(BRATHL\_DATEYMDHMSM dateYMDHMSM, INT refDate, BRATHL\_DATEJULIAN dateJulian)**

[in] dateYMDHMSM : date to convert

[in] refDate : date reference conversion

[out] dateJulian : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

BRATHL\_YMDHMSM2SECONDS

---

Converts a year, month, day, hour, minute, second, microsecond date into a seconds, according to refDate parameter

BRATHL\_YMDHMSM2SECONDS(BRATHL\_DATEYMDHMSM      dateYMDHMSM,      INT      refDate,  
BRATHL\_DATESECOND dateSeconds)

[in] dateYMDHMSM : date to convert

[in] refDate : date reference conversion

[out] dateSeconds : result of the conversion

return 0 or error code (see Date error codes in brathl general documentation)

Example: see BRATHL\_DSM2JULIAN

BRATHL\_SETREFUSER1

BRATHL\_SETREFUSER2

---

Set user-defined reference dates

BRATHL\_SETREFUSER1(STRING dateRef)

[in] dateRef : date to set - format: YYYY-MM-DD HH:MN:SS.MS

return 0 or error code (see Date error codes in brathl general documentation)

Example:

```
dateRefUser1 = '2001 01 12 14:57:23:1456'  
dateRefUser2 = '2005 11 14'
```

```
brathl_setrefuser1(dateRefUser1)  
brathl_setrefuser2(dateRefUser2)
```

MyDate={BRATHL\_DATEDSM}

. Set user-defined ref. date 2001 01 12 14:57:23:1456

MyDate.REFDATE=5

MyDate.DAYS=423

MyDate.SECONDS=5

MyDate.MUSECONDS=0

AnotherDate={BRATHL\_DATEDSM}

. Set user-defined ref. date 2005 11 14

AnotherDate.REFDATE=6

AnotherDate.DAYS=423

AnotherDate.SECONDS=5

AnotherDate.MUSECONDS=0

; ref. date for MyDate is now 2005 11 14

MyDate.REFDATE=6

brathl\_setrefuser2('2005 05 18 13:08:00')

; ref. date for MyDate and AnotherDate is now 2005 05 18 13:08:00

## BRATHL\_CYCLE2YMDHMSM

---

Converts a cycle/pass into a date

BRATHL\_CYCLE2YMDHMSM(INT mission, ULONG cycle, ULONG pass, BRATHL\_DATEYMDHMSM dateYMDHMSM)

[in] mission : mission type :

0 : Topex/Poseidon

1 : Jason-1

2 : ERS2

3 : Envisat

4 : ERS1-A

5 : ERS1-B

6 : GFO

[in] cycle : number of cycle to convert

[in] pass : number of pass in the cycle to convert

[out] dateYMDHMSM : date corresponding to the cycle/pass

return 0 or error code (see Cycle/date conversion error codes in brathl general documentation)

Example:

cycle=120L

pass=153L

mission=3

dOut={BRATHL\_DATEYMDHMSM}

```
r = BRATHL_CYCLE2YMDHMSM(mission, cycle, pass, dOut)
print, "result ", r
```

```
print, "mission ", mission , " cycle ", cycle, " pass ", pass
```

```
print, "Y", dOut.year, " M ", dOut.month, " D ", dOut.day, " H ", dOut.hour, " MN ", dOut.minute, " S ",
dOut.second, " MS ", dOut.muSecond
```

## BRATHL\_YMDHMSM2CYCLE

---

Converts a date into a cycle/pass

BRATHL\_YMDHMSM2CYCLE(INT mission, BRATHL\_DATEYMDHMSM dateYMDHMSM, ULONG cycle, ULONG pass)

[in] mission : mission type :  
0 : Topex/Poseidon  
1 : Jason-1  
2 : ERS2  
3 : Envisat  
4 : ERS1-A  
5 : ERS1-B  
6 : GFO

[in] dateYMDHMSM : date to convert

[out] cycle : number of cycle

[out] pass : number of pass in the cycle

return 0 or error code (see Cycle/date conversion error codes in brathl general documentation)

Example:

```
cycle=0L  
pass=0L  
mission=1
```

```
dIn={BRATHL_DATEYMDHMSM}  
dIn.YEAR=2003  
dIn.MONTH=12  
dIn.DAY=5  
dIn.HOUR=18  
dIn.MINUTE=0  
dIn.SECOND=21  
dIn.MUSECOND=1069
```

```
r = BRATHL_YMDHMSM2CYCLE(mission, dIn, cycle, pass)  
print, "result ", r
```

```
print, "Y", dOut.year, " M ", dOut.month, " D ", dOut.day, " H ", dOut.hour, " MN ", dOut.minute, " S ",  
dOut.second, " MS ", dOut.muSecond  
print, "mission ", mission , " cycle ", cycle, " pass ", pass
```

## Annex G.BRATHL-Fortran API

The BRATHL-C API consists of just a handful of Fortran functions.

Below is the list of Fortran APIs functions.

A description of each function is detailed in the BRATHL documentation in html or latex format (search for refman-html or refman-latext sub-directories in your BRATHL directories installation). Note: When installing BRAT Toolbox, you have to selected 'Documentations' component.

=====

Date conversion/computation functions

=====

brathl\_DayOfYear

brathl\_DiffDSM  
brathl\_DiffJULIAN  
brathl\_DiffYMDHMSM

brathl\_DSM2Julian  
brathl\_DSM2Seconds  
brathl\_DSM2YMDHMSM

brathl\_JULIAN2DSM  
brathl\_JULIAN2Seconds  
brathl\_JULIAN2YMDHMSM

brathl\_SECONDS2DSM  
brathl\_SECONDS2Julian  
brathl\_SECONDS2YMDHMSM

brathl\_NowYMDHMSM

brathl\_YMDHMSM2DSM  
brathl\_YMDHMSM2Julian  
brathl\_YMDHMSM2Seconds

Date conversion/computation example:

```
PROGRAM TESTDATE_F
IMPLICIT NONE

INCLUDE "brathlf.inc"
INTEGER IREFDATESRC
DOUBLE PRECISION ISECONDS
INTEGER IREFDATEDEST
INTEGER O DAYS
INTEGER O SECONDS
INTEGER OM USECONDS

INTEGER Y
INTEGER M
INTEGER D
INTEGER H
INTEGER MN
INTEGER SEC
INTEGER MS

INTEGER RESULT
CHARACTER*128 ERRSTR
CHARACTER*28 REFUSER
```

## Basic Radar Altimetry Toolbox User Manual

---

```
INTEGER TMP

REFUSER = '1952 02 18'
CALL BRATHLF_SETREFUSER1(REFUSER)
IREFDATESRC = REF20000101
C      IREFDATEDEST = REF19500101
IREFDATEDEST = REFUSER1

ISECONDS = 86460.16936D0
ODAYS = 0
OSECONDS = 0
OMUSECONDS = 0

& RESULT = BRATHLF_SECONDS2DSM(IREFDATESRC, ISECONDS, IREFDATEDEST,
& ODAYS, OSECONDS, OMUSECONDS)

IF (RESULT .NE. BRATHL_SUCCESS) THEN
  CALL BRATHLF_ERRNO2STRING(RESULT, ERRSTR)
  WRITE(*,*) 'ERROR: ' // ERRSTR
  STOP
END IF

WRITE(*,*) ' IREFDATESRC:', IREFDATESRC, ' ISECONDS:', ISECONDS,
& ' IREFDATEDEST:', IREFDATEDEST, ' ODAYS:', ODAYS, ' OSECONDS:',
& OSECONDS, ' OMUSECONDS:', OMUSECONDS
C -----
& RESULT = BRATHLF_DSM2SECONDS(IREFDATESRC, ODAYS, OSECONDS,
& OMUSECONDS, IREFDATEDEST, ISECONDS)

IF (RESULT .NE. BRATHL_SUCCESS) THEN
  CALL BRATHLF_ERRNO2STRING(RESULT, ERRSTR)
  WRITE(*,*) 'ERROR: ' // ERRSTR
  STOP
END IF

WRITE(*,*) ' IREFDATESRC:', IREFDATESRC, ' ISECONDS:', ISECONDS,
& ' IREFDATEDEST:', IREFDATEDEST, ' ODAYS:', ODAYS, ' OSECONDS:',
& OSECONDS, ' OMUSECONDS:', OMUSECONDS
C -----
RESULT = brathlf_DSM2YMDHMSM(IREFDATESRC, ODAYS, OSECONDS,
& OMUSECONDS, Y, M, D, H, MN, SEC, MS)

IF (RESULT .NE. BRATHL_SUCCESS) THEN
  CALL BRATHLF_ERRNO2STRING(RESULT, ERRSTR)
  WRITE(*,*) 'ERROR: ' // ERRSTR
  STOP
END IF

WRITE(*,*) ' IREFDATESRC:', IREFDATESRC, ' Y:', Y,
' M:', M, ' D:', D, ' H:', H, ' MN:', MN, ' SEC:', SEC, ' MS:', MS,
& ' ODAYS:', ODAYS, ' OSECONDS:',
& OSECONDS, ' OMUSECONDS:', OMUSECONDS
C -----
RESULT = brathlf_YMDHMSM2DSM( Y, M, D, H, MN, SEC, MS,
& IREFDATEDEST, ODAYS, OSECONDS, OMUSECONDS,)

IF (RESULT .NE. BRATHL_SUCCESS) THEN
  CALL BRATHLF_ERRNO2STRING(RESULT, ERRSTR)
  WRITE(*,*) 'ERROR: ' // ERRSTR
  STOP
END IF

WRITE(*,*) ' IREFDATESRC:', IREFDATESRC, ' Y:', Y,
' M:', M, ' D:', D, ' H:', H, ' MN:', MN, ' SEC:', SEC, ' MS:', MS,
& ' ODAYS:', ODAYS, ' OSECONDS:',
& OSECONDS, ' OMUSECONDS:', OMUSECONDS
C -----
END
=====
```

### Cycle/date conversion functions

---

To convert cycle <-> date, these functions use an ascii parameter file (ascii file) with records:

field 1 : Name of the mission  
field 2 : Cycle reference  
field 3 : Pass reference  
field 4 : Reference date in decimal julian day

Each field has to be separated by, at least, a non-numeric character

The file can contained several records for a same mission.  
Only the field with the greatest date is taken into account

You can add records.

You can add comments, commented lines start by '#' character.

If the file doesn't exist, default values are :

Name	Cycle	Pass	Reference date
Jason-1	99	230	19987.9081795
Topex/Poseidon	442	230	19987.9127535
ERS2	66	598	18831.768334
ERS1-A	15	1	15636.938955
ERS1-B	42	108	16538.6732895
ENVISAT	30	579	19986.106016

brathl\_Cycle2YMDHMSM  
brathl\_YMDHMSM2Cycle

### Cycle/date conversion example:

```
PROGRAM TESTCYCLE_F
IMPLICIT NONE

INCLUDE "brathlf.inc"
INTEGER C
INTEGER P
INTEGER MISSION

INTEGER Y
INTEGER M
INTEGER D
INTEGER H
INTEGER MN
INTEGER SEC
INTEGER MS

INTEGER RESULT
CHARACTER*128 ERRSTR

MISSION = ENVISAT

C = 120
P = 153

&      RESULT = BRATHLF_CYCLE2YMDHMSM(MISSION, C, P,
&      Y, M, D, H, MN, SEC, MS)

IF (RESULT .NE. BRATHL_SUCCESS) THEN
  CALL BRATHLF_ERRNO2STRING(RESULT, ERRSTR)
  WRITE(*,*) 'ERROR: ' // ERRSTR
  STOP
END IF
```

## Basic Radar Altimetry Toolbox User Manual

---

```
      WRITE(*,*) ' MISSION:', MISSION, ' CYCLE:', C,
&      ' PASS:', P,
&      ' Y:', Y,
&      ' M:', M, ' D:', D, ' H:', H, ' MN:', MN, ' SEC:', SEC, ' MS:', MS
C -----
      RESULT = BRATHLF_YMDHMSM2CYCLE(MISSION,
&      Y, M, D, H, MN, SEC, MS, C, P)

      IF (RESULT .NE. BRATHL_SUCCESS) THEN
          CALL BRATHLF_ERRNO2STRING(RESULT, ERRSTR)
          WRITE(*,*) 'ERROR: ' // ERRSTR
          STOP
      END IF

      WRITE(*,*) ' MISSION:', MISSION, ' CYCLE:', C,
&      ' PASS:', P,
&      ' Y:', Y,
&      ' M:', M, ' D:', D, ' H:', H, ' MN:', MN, ' SEC:', SEC, ' MS:', MS

      END
```

---

### Data reading function

---

#### brathl\_ReadData

##### Example:

```
PROGRAM P
IMPLICIT NONE
CHARACTER*(100) NAMES(10)
CHARACTER*(10) Record
CHARACTER*(120) Selection
CHARACTER*(200) Expressions(20)
CHARACTER*(20) Units(20)
REAL*8 Result(1000,20)
LOGICAL*4 Ignore
LOGICAL*4 Statistics
REAL*8 Default

INTEGER*4 NbValues
INTEGER*4 NbResults
INTEGER*4 ReturnCode

INCLUDE "brathlf.inc"

NAMES(1) = 'JA1_GDR_2PaP124_001.CNES'
NAMES(2) = 'JA1_GDR_2PaP124_002.CNES'
NAMES(3) = 'JA1_GDR_2PaP124_003.CNES'
Record      = 'data'
Selection    = 'latitude > 20'
Expressions(1) = 'latitude + longitude'
Units(1) = 'radians'
Expressions(2) = 'swh_ku'
Units(2) = 'm'
NbValues = 1000
NbResults   = -1
Ignore       = .false.
Statistics   = .false.
Default      = 1.0E100

ReturnCode   = brathlf_ReadData(3,
$                                NAMES,
$                                Record,
$                                Selection,
$                                2,
$                                Expressions,
$                                Units,
$                                Result,
$                                NbValues,
```

## Basic Radar Altimetry Toolbox User Manual

---

```
$                               NbResults,  
$                               Ignore,  
$                               Statistics,  
$                               Default)  
print *, NbResults  
print *, ReturnCode  
END
```

## Annex H.BRATHL-C API

The BRATHL-C API consists of just a handful of C structures and functions.

Below is the list of C APIs functions.

A description of each function is detailed in the BRATHL documentation in html or latex format (search for refman-html or refman-latext sub-directories in your BRATHL directories installation). Note: When installing BRAT Toolbox, you have to selected 'Documentations' component.

=====

Date conversion/computation functions

=====

brathl\_DayOfYear

brathl\_DiffDSM  
brathl\_DiffJULIAN  
brathl\_DiffYMDHMSM

brathl\_DSM2Julian  
brathl\_DSM2Seconds  
brathl\_DSM2YMDHMSM

brathl\_JULIAN2DSM  
brathl\_JULIAN2Seconds  
brathl\_JULIAN2YMDHMSM

brathl\_SECONDS2DSM  
brathl\_SECONDS2Julian  
brathl\_SECONDS2YMDHMSM

brathl\_NowYMDHMSM

brathl\_YMDHMSM2DSM  
brathl\_YMDHMSM2Julian  
brathl\_YMDHMSM2Seconds

Date conversion/computation example :

```
#include <brathl.h>
#include <brathl_error.h>

void PrintfDateDSM(brathl_DateDSM *d);
void PrintfDateSecond(brathl_DateSecond *d);
void PrintfDateJulian(brathl_DateJulian *d);
void PrintfDateYMDHMSM(brathl_DateYMDHMSM *d);

int main (int argc, char *argv[])
{
    double diff = 0;
    brathl_DateSecond dateSeconds;
    brathl_DateDSM dateDSM;
    brathl_DateDSM dateDSM2;
    brathl_DateJulian dateJulian;
    brathl_DateJulian dateJulian2;
    brathl_DateYMDHMSM dateYMDHMSM;
    brathl_DateYMDHMSM dateYMDHMSM2;
    brathl_refDate refDate = REF19500101;
    brathl_refDate refDateDest = REF19500101;
```

## Basic Radar Altimetry Toolbox User Manual

---

```
char Buff[1024];

memset(brathl_refDateUser1, '\0', BRATHL_REF_DATE_USER_LEN - 1);

memset(&dateSeconds, '\0', sizeof(dateSeconds));
memset(&dateDSM, '\0', sizeof(dateDSM));
memset(&dateDSM2, '\0', sizeof(dateDSM2));
memset(&dateJulian, '\0', sizeof(dateJulian));
memset(&dateJulian2, '\0', sizeof(dateJulian2));
memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));
memset(&dateYMDHMSM2, '\0', sizeof(dateYMDHMSM2));

puts ("saisir Référentiel Source : \n"
      "1 --> 1950\n"
      "2 --> 1958\n"
      "3 --> 1990\n"
      "4 --> 2000\n"
      "5 --> user 1\n"
      "x Exit\n");

c = getchar();
getchar();

switch (c)
{
    case 'X' :
    case 'x' :
        return 0;
    case '1' : refDate = REF19500101; break;
    case '2' : refDate = REF19580101; break;
    case '3' : refDate = REF19900101; break;
    case '4' : refDate = REF20000101; break;
    case '5' :
        refDate = REFUSER1;
        puts ("saisir la date du référentiel au format YYYY MM DD hh:mm:s:ms ");
        gets (Buff);
        strncpy (brathl_refDateUser1, Buff, BRATHL_REF_DATE_USER_LEN - 1);

        break;
    default : refDate = REF19500101;
}

puts ("saisir Référentiel Destination : \n"
      "1 --> 1950\n"
      "2 --> 1958\n"
      "3 --> 1990\n"
      "4 --> 2000\n"
      "5 --> user 1\n"
      "x Exit\n");

c = getchar();
getchar();

switch (c)
{
    case 'X' :
    case 'x' :
        return 0;
    case '1' : refDateDest = REF19500101; break;
    case '2' : refDateDest = REF19580101; break;
    case '3' : refDateDest = REF19900101; break;
    case '4' : refDateDest = REF20000101; break;
    case '5' :
        refDateDest = REFUSER1;
        puts ("saisir la date du référentiel au format YYYY MM DD hh:mm:s:ms ");
        //fgets (brathl_refDateUser1, strlen(refDateUser), stdin);
        gets (Buff);
        strncpy (brathl_refDateUser1, Buff, BRATHL_REF_DATE_USER_LEN - 1);
```

```

        break;
    default : refDateDest = REF19500101;
}

printf("ref. dest %d %s\n", refDateDest, brathl_refDateUser1 );

do
{
    puts ("\nConversion : \n"
        "1 - Seconds --> DSM\n"
        "2 - DSM -->Seconds\n"
        "3 - Julian --> DSM\n"
        "4 - DSM -->Julian\n"
        "5 - YMDHMSM --> DSM\n"
        "6 - DSM -->YMDHMSM\n"
        "7 - Seconds --> Julian\n"
        "8 - Julian --> Seconds\n"
        "9 - Seconds --> YMDHMSM\n"
        "A - YMDHMSM --> Seconds\n"
        "B - Julian --> YMDHMSM\n"
        "C - YMDHMSM -->Julian\n"
        "D - diff Date1 - Date2 (YMDHMSM) \n"
        "E - diff Date1 (ref. src) - Date2 (ref. dest) (DSM)\n"
        "F - diff Date1 (ref. src) - Date2 (ref. dest) (Julian)\n"
        "N - Now --> YMDHMSM\n"
        "Q - YMDHMSM --> Quantieme\n"
        "x Exit\n");
}

c = getchar();
getchar();

switch (c)
{
    case '1' : // Seconds --> DSM
        memset(&dateSeconds, '\0', sizeof(dateSeconds));
        memset(&dateDSM, '\0', sizeof(dateDSM));

        dateSeconds.refDate = refDate;

        puts ("nbSeconds :");
        gets (Buff);
        sscanf(Buff, "%lf", &dateSeconds.nbSeconds);

        result = brathl_Seconds2DSM(&dateSeconds, refDateDest, &dateDSM);
        printf("result %d %s\n", result, brathl_Errno2String(result));
        PrintfDateSecond(&dateSeconds);
        PrintfDateDSM(&dateDSM);
        break;
    case '2' : // DSM -->Seconds
        memset(&dateSeconds, '\0', sizeof(dateSeconds));
        memset(&dateDSM, '\0', sizeof(dateDSM));

        dateDSM.refDate = refDate;

        puts ("D S M :");
        gets (Buff);
        sscanf(Buff, "%ld%*c%ld%*c%ld ",
               &dateDSM.days,
               &dateDSM.seconds,
               &dateDSM.muSeconds );

        result = brathl_DSM2Seconds(&dateDSM, refDateDest, &dateSeconds);
        printf("result %d %s\n", result, brathl_Errno2String(result));
        PrintfDateSecond(&dateSeconds);
        PrintfDateDSM(&dateDSM);
        break;
    case '3' : // Julian --> DSM
        memset(&dateDSM, '\0', sizeof(dateDSM));

```

## Basic Radar Altimetry Toolbox User Manual

---

```
        memset(&dateJulian, '\0', sizeof(dateJulian));

        dateJulian.refDate = refDate;

        puts ("julian :");
gets (Buff);
        sscanf(Buff, "%lf", &dateJulian.julian);

        result = brathl_Julian2DSM(&dateJulian, refDateDest, &dateDSM);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateJulian(&dateJulian);
        PrintfDateDSM(&dateDSM);
        break;

case '4' : // DSM -->Julian
        memset(&dateJulian, '\0', sizeof(dateJulian));
        memset(&dateDSM, '\0', sizeof(dateDSM));

        dateDSM.refDate = refDate;

        puts ("D S M :");
gets (Buff);
        sscanf(Buff, "%ld%*c%ld%*c%ld ",
               &dateDSM.days,
               &dateDSM.seconds,
               &dateDSM.muSeconds );

        result = brathl_DSM2Julian(&dateDSM, refDateDest, &dateJulian);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateJulian(&dateJulian);
        PrintfDateDSM(&dateDSM);
        break;

case '5' : // YMDHMSM --> DSM
        memset(&dateDSM, '\0', sizeof(dateDSM));
        memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));

        puts ("YYYY MM DD hh:mn:s:ms :");
gets (Buff);
        sscanf(Buff, "%4d%*c%2d%*c%2d%*c"
                           "%2d%*c%2d%*c%2d%*c%6d",
               &dateYMDHMSM.year, &dateYMDHMSM.month, &dateYMDHMSM.day,
               &dateYMDHMSM.hour, &dateYMDHMSM.minute, &dateYMDHMSM.second,
               &dateYMDHMSM.muSecond);

        result = brathl_YMDHMSM2DSM(&dateYMDHMSM, refDateDest, &dateDSM);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateYMDHMSM(&dateYMDHMSM);
        PrintfDateDSM(&dateDSM);
        break;

case '6' : // DSM -->YMDHMSM
        memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));
        memset(&dateDSM, '\0', sizeof(dateDSM));

        puts ("D S M :");
gets (Buff);
        sscanf(Buff, "%ld%*c%ld%*c%ld ",
               &dateDSM.days,
               &dateDSM.seconds,
               &dateDSM.muSeconds );

        result = brathl_DSM2YMDHMSM(&dateDSM, &dateYMDHMSM);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateYMDHMSM(&dateYMDHMSM);
        PrintfDateDSM(&dateDSM);
        break;

case '7' : // Seconds --> Julian
```

## Basic Radar Altimetry Toolbox User Manual

---

```
memset(&dateSeconds, '\0', sizeof(dateSeconds));
        memset(&dateJulian, '\0', sizeof(dateJulian));

        dateSeconds.refDate = refDate;

        puts ("nbSeconds :");
gets (Buff);
        sscanf(Buff, "%lf", &dateSeconds.nbSeconds);

        result = brathl_Seconds2Julian(&dateSeconds, refDateDest, &dateJulian);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateSecond(&dateSeconds);
        PrintfDateJulian(&dateJulian);
        break;

case '8' : // Julian --> Seconds
memset(&dateSeconds, '\0', sizeof(dateSeconds));
        memset(&dateJulian, '\0', sizeof(dateJulian));

        dateJulian.refDate = refDate;

        puts ("julian :");
gets (Buff);
        sscanf(Buff, "%lf", &dateJulian.julian);

        result = brathl_Julian2Seconds(&dateJulian, refDateDest, &dateSeconds);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateSecond(&dateSeconds);
        PrintfDateJulian(&dateJulian);
        break;

case '9' : // Seconds --> YMDHMSM
memset(&dateSeconds, '\0', sizeof(dateSeconds));
        memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));

        dateSeconds.refDate = refDate;

        puts ("nbSeconds :");
gets (Buff);
        sscanf(Buff, "%lf", &dateSeconds.nbSeconds);

        result = brathl_Seconds2YMDHMSM(&dateSeconds, &dateYMDHMSM);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateSecond(&dateSeconds);
        PrintfDateYMDHMSM(&dateYMDHMSM);
        break;

case 'A' : // YMDHMSM --> Seconds
case 'a' : // YMDHMSM --> Seconds
memset(&dateSeconds, '\0', sizeof(dateSeconds));
        memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));

        puts ("YYYY MM DD hh:mn:s:ms :");
gets (Buff);
        sscanf(Buff, "%4d%*c%2d%*c%2d%*c"
                "%2d%*c%2d%*c%2d%*c%6d",
                &dateYMDHMSM.year, &dateYMDHMSM.month, &dateYMDHMSM.day,
                &dateYMDHMSM.hour, &dateYMDHMSM.minute, &dateYMDHMSM.second,
&dateYMDHMSM.muSecond);

        result = brathl_YMDHMSM2Seconds(&dateYMDHMSM, refDateDest, &dateSeconds);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateSecond(&dateSeconds);
        PrintfDateYMDHMSM(&dateYMDHMSM);
        break;

case 'B' : // Julian --> YMDHMSM
case 'b' : // Julian --> YMDHMSM
memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));
        memset(&dateJulian, '\0', sizeof(dateJulian));
```

```

        dateJulian.refDate = refDate;

        puts ("julian :");
gets (Buff);
        sscanf(Buff, "%lf", &dateJulian.julian);

        result = brathl_Julian2YMDHMSM(&dateJulian, &dateYMDHMSM);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateJulian(&dateJulian);
        PrintfDateYMDHMSM(&dateYMDHMSM);
        break;

case 'C' : // YMDHMSM --> Julian
case 'c' : // YMDHMSM --> Julian
        memset(&dateJulian, '\0', sizeof(dateJulian));
        memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));

        puts ("YYYY MM DD hh:mn:s:ms :");
gets (Buff);
        sscanf(Buff, "%4d%*c%2d%*c%2d%*c"
                "%2d%*c%2d%*c%2d%*c%6d",
                &dateYMDHMSM.year, &dateYMDHMSM.month, &dateYMDHMSM.day,
                &dateYMDHMSM.hour, &dateYMDHMSM.minute, &dateYMDHMSM.second,
&dateYMDHMSM.muSecond);

        result = brathl_YMDHMSM2Julian(&dateYMDHMSM, refDateDest, &dateJulian);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateJulian(&dateJulian);
        PrintfDateYMDHMSM(&dateYMDHMSM);
        break;

case 'D' : // diff Date1 (ref. src) - Date2 (ref. dest) (YMDHMSM)
case 'd' : // diff Date1 (ref. src) - Date2 (ref. dest) (YMDHMSM)
        memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));
        memset(&dateYMDHMSM2, '\0', sizeof(dateYMDHMSM2));

        puts ("Date 1 YYYY MM DD hh:mn:s:ms :");
gets (Buff);
        sscanf(Buff, "%4d%*c%2d%*c%2d%*c"
                "%2d%*c%2d%*c%2d%*c%6d",
                &dateYMDHMSM.year, &dateYMDHMSM.month, &dateYMDHMSM.day,
                &dateYMDHMSM.hour, &dateYMDHMSM.minute, &dateYMDHMSM.second,
&dateYMDHMSM.muSecond);

        puts ("Date 2 YYYY MM DD hh:mn:s:ms :");
gets (Buff);
        sscanf(Buff, "%4d%*c%2d%*c%2d%*c"
                "%2d%*c%2d%*c%2d%*c%6d",
                &dateYMDHMSM2.year, &dateYMDHMSM2.month, &dateYMDHMSM2.day,
                &dateYMDHMSM2.hour, &dateYMDHMSM2.minute,
                &dateYMDHMSM2.second, &dateYMDHMSM2.muSecond);

        diff = 0;

        result = brathl_DiffYMDHMSM(&dateYMDHMSM, &dateYMDHMSM2, &diff);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateYMDHMSM(&dateYMDHMSM);
        PrintfDateYMDHMSM(&dateYMDHMSM2);
printf("\t----> Difference : %lf \n", diff);
        break;

case 'E' : // diff Date1 (ref. src) - Date2 (ref. dest) (DSM)
case 'e' : // diff Date1 (ref. src) - Date2 (ref. dest) (DSM)
        memset(&dateDSM, '\0', sizeof(dateDSM));
        memset(&dateDSM2, '\0', sizeof(dateDSM2));

        dateDSM.refDate = refDate;
        dateDSM2.refDate = refDateDest;

```

## Basic Radar Altimetry Toolbox User Manual

---

```
    puts (" Date 1 D S M :");
    gets (Buff);
    sscanf(Buff, "%ld%*c%ld%*c%ld ",
           &dateDSM.days,
           &dateDSM.seconds,
           &dateDSM.muSeconds );

    puts (" Date 2 D S M :");
    gets (Buff);
    sscanf(Buff, "%ld%*c%ld%*c%ld ",
           &dateDSM2.days,
           &dateDSM2.seconds,
           &dateDSM2.muSeconds );

diff = 0;

result = brathl_DiffDSM(&dateDSM, &dateDSM2, &diff);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateDSM(&dateDSM);
PrintfDateDSM(&dateDSM2);
printf("\t----> Difference : %lf \n", diff);
break;

case 'F' : // diff Date1 (ref. src) - Date2 (ref. dest) (Julian)
case 'f' : // diff Date1 (ref. src) - Date2 (ref. dest) (Julian)
memset(&dateDSM, '\0', sizeof(dateDSM));
memset(&dateDSM2, '\0', sizeof(dateDSM2));

dateJulian.refDate = refDate;
dateJulian2.refDate = refDateDest;

puts ("Date 1 julian :");
gets (Buff);
sscanf(Buff, "%lf", &dateJulian.julian);

puts ("Date 2 julian :");
gets (Buff);
sscanf(Buff, "%lf", &dateJulian2.julian);

diff = 0;

result = brathl_DiffJulian(&dateJulian, &dateJulian2, &diff);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateJulian(&dateJulian);
PrintfDateJulian(&dateJulian2);
printf("\t----> Difference : %lf \n", diff);
break;

case 'N' : // Now --> YMDHMSM
case 'n' : // Now --> YMDHMSM
memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));

result = brathl_NowYMDHMSM(&dateYMDHMSM);
printf("result %d %s\n", result, brathl_Errno2String(result));
PrintfDateYMDHMSM(&dateYMDHMSM);
break;

case 'Q' : // YMDHMSM --> Quantième
case 'q' : // YMDHMSM --> Quantième
memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));

puts ("YYYY MM DD hh:mn:s:ms :");
gets (Buff);
sscanf(Buff, "%4d%*c%2d%*c%2d%*c"
           "%2d%*c%2d%*c%2d%*c%6d",
           &dateYMDHMSM.year, &dateYMDHMSM.month, &dateYMDHMSM.day,
           &dateYMDHMSM.hour, &dateYMDHMSM.minute, &dateYMDHMSM.second,
           &dateYMDHMSM.muSecond);
```

```
    uint32_t quantieme;
    result = brathl_Qantieme(&dateYMDHMSM, &quantieme);
    printf("result %d %s\n", result, brathl_Errno2String(result));
    PrintfDateYMDHMSM(&dateYMDHMSM);
    printf("\t----> Quantieme : %ld \n", quantieme);
    break;

    default : break;
}

if ((c != 'X') && (c != 'x'))
{
    puts("Press enter key to continue");
    getchar();
}

} while ((c != 'X') && (c != 'x'));

return 0;
}

//-----
void PrintfDateDSM(brathl_DateDSM *d)
{
    printf("\tbrathl_DateDSM days %ld seconds %ld museconds %ld ref. %d %s\n",
        d->days, d->seconds, d->muSeconds, d->refDate, brathl_refDateUser1);
}

//-----
void PrintfDateSecond(brathl_DateSecond *d)
{
    printf("\tbrathl_DateSecond nbSeconds %lf ref. %d %s\n",
        d->nbSeconds, d->refDate, brathl_refDateUser1);
}

//-----
void PrintfDateJulian(brathl_DateJulian *d)
{
    printf("\tbrathl_DateJulian julian %lf ref. %d %s\n",
        d->julian, d->refDate, brathl_refDateUser1);
}

//-----
void PrintfDateYMDHMSM(brathl_DateYMDHMSM *d)
{
    printf("\tbrathl_DateYMDHMSM year %ld month %ld day %ld hour %ld minute %ld second %ld
musecond %ld ref. %s\n",
        d->year, d->month, d->day, d->hour, d->minute, d->second, d->muSecond,
        brathl_refDateUser1);
}

=====
```

### Cycle/date conversion functions

---

To convert cycle <-> date, these functions use an ascii parameter file (ascii file) with records :

field 1 : Name of the mission  
field 2 : Cycle reference

field 3 : Pass reference

field 4 : Reference date in decimal julian day

Each field has to be separated by, at least, a non-numeric character

The file can contained several records for a same mission.

Only the field with the greatest date is taken into account

You can add records.

You can add comments, commented lines start by '#' character.

If the file doesn't exist, default values are :

Name	Cycle	Pass	Reference date
Jason-1	99	230	19987.9081795
Topex/Poseidon	442	230	19987.9127535
ERS2	66	598	18831.768334
ERS1-A	15	1	15636.938955
ERS1-B	42	108	16538.6732895
ENVISAT	30	579	19986.106016

brathl\_Cycle2YMDHMSM

brathl\_YMDHMSM2Cycle

### Cycle/date conversion example

```
#include <brathl.h>
#include <brathl_error.h>

void PrintfDateDSM(brathl_DateDSM *d);
void PrintfDateSecond(brathl_DateSecond *d);
void PrintfDateJulian(brathl_DateJulian *d);
void PrintfDateYMDHMSM(brathl_DateYMDHMSM *d);

int main (int argc, char *argv[])
{
    uint32_t cycle = 0;
    uint32_t pass = 0;
    int32_t result = BRATHL_SUCCESS;
    char c;

    double diff = 0;

    brathl_mission mission;
    brathl_DateYMDHMSM dateYMDHMSM;
    char Buff[1024];

    memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));

    puts ("saisir la mission : \n"
          "1 --> TOPEX\n"
          "2 --> JASON1\n"
          "3 --> ERS2\n"
          "4 --> ENVISAT\n"
          "5 --> ERS1_A\n"
          "6 --> ERS1_B\n"
          "7 --> GFO\n"
          "x Exit\n");

    c = getchar();
    getchar();
```

## Basic Radar Altimetry Toolbox User Manual

---

```
switch (c)
{
    case 'X' :
    case 'x' :
        return 0;
    case '1' : mission = TOPEX; break;
    case '2' : mission = JASON1; break;
    case '3' : mission = ERS2; break;
    case '4' : mission = ENVISAT; break;
    case '5' : mission = ERS1_A; break;
    case '6' : mission = ERS1_B; break;
    case '7' : mission = GFO; break;

    break;
default : mission = TOPEX;
}

do
{
    puts ("\nConversion Cycle <--> Date: \n"
        "1 - Cycle --> Date YMDHMSM\n"
        "2 - Date YMDHMSM -->Cycle\n"
        "x Exit\n");

    c = getchar();
    getchar();

    switch (c)
    {
        case '1' : // Cycle --> Date YMDHMSM
            memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));
            cycle = pass = 0;

            puts ("Cycle Pass:");
            gets (Buff);
            sscanf(Buff, "%ld%c%ld ", &cycle, &pass);
            result = brathl_Cycle2YMDHMSM(mission, cycle, pass, &dateYMDHMSM);

            printf("result %d %s\n", result, brathl_Errno2String(result));
            printf("\tcycle %d pass %d\n", cycle, pass);
            PrintfDateYMDHMSM(&dateYMDHMSM);
            break;
        case '2' : // Date YMDHMSM -->Cycle
            memset(&dateYMDHMSM, '\0', sizeof(dateYMDHMSM));
            cycle = pass = 0;

            puts ("YYYY MM DD hh:mn:s:ms :");
            gets (Buff);
            sscanf(Buff, "%4d%c%2d%c%2d%c"
                "%2d%c%2d%c%2d%c%6d",
                &dateYMDHMSM.year, &dateYMDHMSM.month, &dateYMDHMSM.day,
                &dateYMDHMSM.hour, &dateYMDHMSM.minute, &dateYMDHMSM.second,
                &dateYMDHMSM.muSecond);

            result = brathl_YMDHMSM2Cycle(mission, &dateYMDHMSM, &cycle, &pass);

            printf("result %d %s\n", result, brathl_Errno2String(result));
            printf("\tcycle %d pass %d\n", cycle, pass);
            PrintfDateYMDHMSM(&dateYMDHMSM);
            break;

        default : break;
    }

    if ((c != 'X') && (c != 'x'))
    {
        puts("Press enter key to continue");
        getchar();
    }
} while ((c != 'X') && (c != 'x'));

return 0;
}
```

## Basic Radar Altimetry Toolbox User Manual

---

```
-----  
void PrintfDateDSM(brathl_DateDSM *d)  
{  
    printf("\tbrathl_DateDSM days %ld seconds %ld museconds %ld ref. %d %s\n",  
        d->days, d->seconds, d->muSeconds, d->refDate, brathl_refDateUser1);  
}  
-----  
  
void PrintfDateSecond(brathl_DateSecond *d)  
{  
    printf("\tbrathl_DateSecond nbSeconds %lf ref. %d %s\n",  
        d->nbSeconds, d->refDate, brathl_refDateUser1);  
}  
-----  
void PrintfDateJulian(brathl_DateJulian *d)  
{  
    printf("\tbrathl_DateJulian julian %lf ref. %d %s\n",  
        d->julian, d->refDate, brathl_refDateUser1);  
}  
-----  
void PrintfDateYMDHMSM(brathl_DateYMDHMSM *d)  
{  
    printf("\tbrathl_DateYMDHMSM year %ld month %ld day %ld hour %ld minute %ld second %ld musecond %ld  
ref. %s\n",  
        d->year, d->month, d->day, d->hour, d->minute, d->second, d->muSecond, brathl_refDateUser1);  
}
```

=====

### Data reading function

=====

#### brathl\_ReadData

##### Example:

```
#include <stdio.h>  
#include <stdlib.h>  
#include "brathl.h"  
#include "brathl_error.h"  
  
int main(int argc, char **argv)  
{  
    char                 *Names[10];  
    int32_t              ReturnCode;  
    double               *Data[2] = {NULL,NULL};  
    int32_t              Sizes[2] = {-1, -1};  
    char                 *Expressions[2];  
    char                 *Units[2];  
    int32_t              ActualSize;  
  
    Names[0]             = "JA1_GDR_2PaP124_001.CNES";  
    Names[1]             = "JA1_GDR_2PaP124_002.CNES";  
    Names[2]             = "JA1_GDR_2PaP124_003.CNES";  
  
    Expressions[0]        = "latitude + longitude";  
    Units[0]              = "radians";  
    Expressions[1]        = "swh_ku";  
    Units[1]              = "m";  
  
    ReturnCode            = brathl_ReadData(3, Names,  
                                            "data",  
                                            "latitude > 20",  
                                            2,  
                                            Expressions,  
                                            Units,  
                                            Data,  
                                            Sizes,
```

```
    &ActualSize,  
    0,  
    0,  
    0);  
  
printf("Return code      : %d\n", ReturnCode);  
printf("Actual number of data: %d\n", ActualSize);  
return 0;  
}
```