SOFTWARE USER MANUAL

(BRAT version 4.2.1)

version : 2.0

date : 14/06/2018
# Document Information

<table>
<thead>
<tr>
<th>Contract Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contract Number:</td>
</tr>
<tr>
<td>Contract Issuer:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
<th>Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prepared by</td>
<td>BRAT development team</td>
<td>-</td>
</tr>
<tr>
<td>Reviewed by</td>
<td>Ana Friaças</td>
<td>PA Manager</td>
</tr>
<tr>
<td>Approved by</td>
<td>Miguel Terra-Homem</td>
<td>Project Manager</td>
</tr>
</tbody>
</table>
## Document Change Log

<table>
<thead>
<tr>
<th>Issue</th>
<th>Author</th>
<th>Section</th>
<th>Change Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>BRAT Dev team</td>
<td>All</td>
<td>SUM version aligned with the new BRAT GUI.</td>
<td>02/05/2016</td>
</tr>
<tr>
<td>1.1</td>
<td>BRAT Dev team</td>
<td>All</td>
<td>SUM version aligned with the new BRAT GUI (BRAT V4.0.0-beta).</td>
<td>30/06/2016</td>
</tr>
<tr>
<td>1.2</td>
<td>BRAT Dev team</td>
<td>All</td>
<td>SUM version aligned with the new BRAT GUI (BRAT V4.0.0).</td>
<td>15/09/2016</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sec. 4.2.5.1</td>
<td>Added section 4.2.5.1 Export</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sec. 18</td>
<td>Added Annex I: BRAT-Python Algorithms</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sec. 19</td>
<td>Added Annex J: Compilation in GPOD Environment</td>
<td></td>
</tr>
<tr>
<td>1.3</td>
<td>BRAT Dev Team</td>
<td>Sec. 3</td>
<td>Update on the installation notes</td>
<td>25/10/2016</td>
</tr>
<tr>
<td>1.4</td>
<td>BRAT Dev Team</td>
<td>Sec. 3</td>
<td>Update on the installation notes</td>
<td>17/01/2017</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sec. 3.6</td>
<td>Added section regarding RADS service</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sec. 4.2.2.1</td>
<td>Added section &quot;RADS datasets&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sec. 4.3.3</td>
<td>Added section &quot;RADS Datasets tab&quot;</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>BRAT Dev Team</td>
<td>Sec. 3.6, Sec. 4.2.2.1</td>
<td>Minor updates for BRAT V4.1.0</td>
<td>17/04/2017</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated RADS sections</td>
<td></td>
</tr>
<tr>
<td>1.6</td>
<td>BRAT Dev Team</td>
<td>All</td>
<td>Reviewed and updated for BRAT V4.2.0</td>
<td>04/09/2017</td>
</tr>
<tr>
<td>1.7</td>
<td>BRAT Dev Team</td>
<td></td>
<td>Added sections 3.7 and 4.1.1</td>
<td>04/10/2017</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated Figure 5, Figure 13 and Figure 27</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Major updates on sections 4.3.4, 4.3.5.1, 4.3.5.3, 4.3.5.4, 6 and 7</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Minor updates on sections 4.2.4, 4.2.4.2.1, 4.2.4.2.2, 4.2.4.3, 4.2.5, 4.3.1 and 4.3.6</td>
<td></td>
</tr>
<tr>
<td>1.8</td>
<td>BRAT Dev Team</td>
<td></td>
<td>Updated Figure 1</td>
<td>02/11/2017</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated section 4.1.1</td>
<td></td>
</tr>
<tr>
<td>1.9</td>
<td>BRAT Dev Team</td>
<td></td>
<td>Added section 4.2.3.1</td>
<td>10/05/2018</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Added section 4.2.4.2.3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated section 4.3.5.4.5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Updated Figure 25</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>BRAT Dev Team</td>
<td></td>
<td>Updated section 4.2.3 to explain the options for applying an area filter without time variable</td>
<td>14/06/2018</td>
</tr>
</tbody>
</table>
# Table of Contents

1. **INTRODUCTION** ............................................................................................................. 10
   1.1. Project history and background ................................................................................. 10
   1.2. Global overview .......................................................................................................... 10
   1.3. Toolbox contents ....................................................................................................... 11
2. **DATA READ AND PROCESSED** .................................................................................... 13
   2.1. Background ................................................................................................................ 13
   2.2. Level 1B/2 data products ........................................................................................... 13
   2.3. Higher level products ............................................................................................... 14
3. **HOW TO INSTALL AND UNINSTALL BRAT** ................................................................. 16
   3.1. Supported platforms ................................................................................................. 16
   3.2. The BRAT distribution DVD .................................................................................... 16
   3.3. MS Windows ............................................................................................................. 16
      3.3.1. Installing the binary distribution ........................................................................ 16
      3.3.2. Installing from source ...................................................................................... 17
      3.3.3. Uninstalling ...................................................................................................... 17
   3.4. Linux .......................................................................................................................... 17
      3.4.1. Installing the binary distribution ........................................................................ 17
      3.4.2. Installing from source ...................................................................................... 18
      3.4.3. Uninstalling ...................................................................................................... 19
   3.5. Mac OS X .................................................................................................................. 19
      3.5.1. Installing the binary distribution ........................................................................ 19
      3.5.2. Installing from source ...................................................................................... 20
      3.5.3. Uninstalling ...................................................................................................... 20
   3.6. The RADS Service .................................................................................................... 20
      3.6.1. Installing the RadsService ................................................................................ 20
      3.6.1.1. Required Permissions .................................................................................. 20
      3.6.1.2. Installation and configuration procedures .................................................... 21
      3.6.2. Uninstalling the RadsService .......................................................................... 21
   3.7. Troubleshooting OpenGL issues .............................................................................. 21
   3.8. Sample Build in Debian 8 ........................................................................................ 22
4. **BRAT GRAPHICAL USER INTERFACE (GUI)** ............................................................... 23
   4.1. Overview ................................................................................................................... 23
      4.1.1. The Application Settings (Options) .................................................................... 23
   4.2. Starting with BRAT GUI .......................................................................................... 24
      4.2.1. Create a workspace ......................................................................................... 25
      4.2.2. Create a dataset ............................................................................................... 25
8.2. Visualising an output netCDF file through BRAT ............................................................. 70
8.3. Using the parameter files to process many datasets ............................................................ 71
9. BRATHL APPLICATION PROGRAMMING INTERFACES (APIS) ........................................ 73
  9.1. Data reading function ........................................................................................................... 73
  9.2. Cycle/date conversion functions ......................................................................................... 74
  9.3. Date conversion/computation function .............................................................................. 75
  9.4. Named structures ................................................................................................................ 76
10. ANNEX A: list of datasets read by BRAT ............................................................................. 78
    10.1. Cryosat product overview .............................................................................................. 78
    10.2. Cryosat Ocean products overview ................................................................................... 78
    10.3. Jason-2 product overview ............................................................................................. 79
    10.4. Envisat product overview ............................................................................................... 79
    10.5. Jason-1 product overview ............................................................................................. 79
    10.6. Topex/Poseidon product overview .................................................................................. 79
    10.7. ERS-1 and 2 product overview ...................................................................................... 80
    10.8. GFO product overview ................................................................................................... 80
    10.9. PODAAC product overview ........................................................................................... 80
    10.10. River and Lake product overview .................................................................................. 80
    10.11. NetCDF products ........................................................................................................... 80
      10.11.1. Aviso Altimetry data in netCDF ............................................................................... 81
      10.11.2. ERS REAPER data in netCDF ................................................................................ 81
      10.11.3. Sentinel 3 data in netCDF ..................................................................................... 82
11. ANNEX B: Y=X parameter file keys ..................................................................................... 83
12. ANNEX C: Z=F(X,Y) parameter file keys ............................................................................. 86
13. ANNEX D: Display parameter file keys .................................................................................. 91
14. ANNEX E: BRATHL-MATLAB API ...................................................................................... 99
15. ANNEX F: BRATHL-Fortran API .......................................................................................... 114
16. ANNEX G: BRATHL-C API .................................................................................................. 121
17. ANNEX H: BRATHL-PYTHON API ...................................................................................... 140
18. ANNEX I: BRAT-PYTHON ALGORITHMS ......................................................................... 151
19. Annex J: Compilation in GPOD Environment ........................................................................ 156
    19.1. Dependencies ................................................................................................................. 156
    19.2. Source and Build directories ......................................................................................... 156
    19.3. Configure and make ....................................................................................................... 156

List of Tables

Table 1: Level 1B/2 data products ............................................................................................... 13
Table 2: Higher level products ................................................................................................. 14
Table 3: BRAT functions ........................................................................................................................................ 45
Table 4: BRAT algorithms .................................................................................................................................... 50
Table 5: 10.1. Cryosat product overview ........................................................................................................ 78
Table 6: Cryosat Ocean products overview ......................................................................................................... 78
Table 7: Jason-2 product overview ..................................................................................................................... 79
Table 8: Envisat product overview ...................................................................................................................... 79
Table 9: Jason-1 product overview ...................................................................................................................... 79
Table 10: Topex/Poseidon radar altimetry products ............................................................................................ 79
Table 11: ERS-1 and ERS-2 radar altimetry products ......................................................................................... 80
Table 12: GFO product overview ........................................................................................................................ 80
Table 13: Physical Oceanography Distributed Active Archive Center radar altimetry products for Jason-1 and Topex/Poseidon .................................................................................................................. 80
Table 14: ENVISAT-ERS Exploitation River and Lake Products ........................................................................ 80
Table 15: Aviso Altimetry data in netCDF .......................................................................................................... 81
Table 16: ERS REAPER data in netCDF ............................................................................................................. 81
Table 17: Sentinel 3 data in netCDF .................................................................................................................. 82

List of Figures

Figure 1: The start-up section of the BRAT settings dialog .............................................................................. 24
Figure 2: 'Create a new workspace' window. You can choose to save it wherever you want on your hard drive or local network, and name it as you prefer (preferably in such a way you will remember what's in it). .................................................................................................................. 25
Figure 3: The Dataset tab as it appears when opening a new Workspace. The “New” button enables to create a new dataset. ........................................................................................................................................ 26
Figure 4: Several datasets. On the top, the list of files; on the centre, the description of the netCDF data file, bottom (left) enumeration of the available fields inside the netCDF file, bottom (right) field description. The satellite tracks for the selected data file are plotted on the map at the right (red line). 27
Figure 5: The filters tab. The “New Area Selection” tool enables to draw a selection over the map or enter its values in the coordinates fields; the “Create area” button creates a new area from current map selection .................................................................................................................. 28
Figure 6: The 'Operations' tab in the advanced mode. The “Create operation” button enables to create a new operation. ........................................................................................................................................ 30
Figure 7: On the top, the dataset dropdown list; below, the tree with records and data fields. .................. 31
Figure 8: Dialog shown when "Window>Workspace views" menu is triggered that allows the selection and visualization of a certain operations ........................................................................................................ 34
Figure 9: A 'Display' window with one view created. Note the list of available views ........................................ 35
Figure 10: Examples of standard and RADS datasets with netCDF data selected. ......................................... 37
Figure 11: Filters tab showing applied filter ..................................................................................................... 39
Figure 12: Operations tab, with an operation being built .................................................................................. 40
Figure 13: Operations tab, with the Workspace Elements pane detached and the main map extended to the boundaries of the application window ........................................................................... 42
Figure 14: Example of menu that appears by right-click on a data expression (‘SLA’). Note that here one data field (‘equator_time’) is selected (left-click); if no data field is selected, this item is inactive ....... 43
Figure 15: "Show Aliases" pop-up window. Here for a Jason2 NetCDF file. Note the 'Syntax' column, where the alias syntax is given, while the 'Value' column gives the original field name (or combination).

Figure 16: The 'Formulas' pop-up window, with the list of available formulas, top (sorted in alphabetical order).

Figure 17: use of a pre-defined formula (Ocean_data_editing_GFO_from_cycle_83). Note the use in this particular expression of the formula as alias %{Ocean_data_editing_GFO_from_cycle_83}.

Figure 18: Insert Algorithm pop-up, with the BratAlgoGeoVelGridV selected.

Figure 19: Operation resulting from the insertion of algorithms.

Figure 20: Choice of the data computation.

Figure 21: Configuration parameters for output grid data.

Figure 22: Example of the definition of an alias. This example is for Envisat RA2 and MWR products, by default for data within the "ra2_mds" record. "ku_band_ocean_range" is the name given by default in the documentation and thus in BRAT. To keep it simpler, we call it here "range".

Figure 23: An example Y=F(X) visualisation with two curves.

Figure 24: Data Options tab of the visualisation tool.

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: Two curves overlaid, with different point glyphs defined.

Figure 27: Map plot type to display a simple z=f(lon,lat) graph type.

Figure 28: The "Data Options" tab.

Figure 29: You can also trigger the Globe Plot for this type of data by clicking under the “3D” button.

Figure 30: Plotting a z=f(x,y) graph.

Figure 31: Same plot but with a hidden spectrogram plot by clicking under the 2D button.

Figure 32: Example parameter file for creating a Z=F(X,Y) output.

Figure 33: Example 'display' parameter file.

Figure 34: An example parameter file for creating output netCDF for several cycles (SLA from Jason-1 GDRs).

Figure 35: An example script for DOS (to be inserted in a .bat file) to launch a parameter file over several cycles.

Figure 36: An example Shell script for Linux for launching a parameter file over several cycles.
1. INTRODUCTION

1.1. Project history and background

The Broadview Radar Altimetry Toolbox (BRAT) and the Radar Altimetry Tutorial (RAT) were originally produced by CLS and S&T in 2006-2011 under contract with ESA and CNES (the toolbox name at the time was called Basic Radar Altimetry Toolbox). Since April 2015 under ESA contract within the SEOM program, with additional support from CNES, the current consortium formed by DEIMOS Engenharia S.A., isardSAT UK, and TU Delft is continuing the work, updating the content of the tutorial and redesigning and improving the toolbox.

1.2. Global overview

The Broadview Radar Altimetry Toolbox (BRAT) is a collection of tools and tutorial documents designed to facilitate the processing of radar altimetry data. BRAT is able to handle most distributed radar altimetry data formats, providing support for ingesting, processing, editing (to a certain extent), generating statistics, visualising and exporting the results.

BRAT consists of several modules operating at different levels of abstraction. These modules can be Graphical User Interface (GUI) applications, command-line tools, interfaces to existing applications (such as IDL and MATLAB) or application program interfaces (APIs) to programming languages such as C, Fortran and Python.

The main BRAT 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 the netCDF binary format, ASCII text files, or GeoTiff+GoogleEarth (KMZ/KML export); raster images (PNG, JPEG, BMP, TIFF, and PNM) of visualisations can be saved.
- Statistics: calculation of statistical parameters from data.
- Combinations: computation of formulas involving 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).
- APIs are available with data reading, date and cycle/pass conversion and statistical computation functions for C, Fortran, IDL, (only using previous versions of BRAT), MATLAB and Python, allowing the integration of BRAT functionality in custom applications. For the most common use cases (selection, combinations, visualisations, etc.), command-line tools are available that can be configured by creating parameter files. For beginners, we recommend using the BRAT GUI application, which enables the operator to easily specify the processing parameters required by each tool (and then invoke those tools at the push of a button).
- Download and periodic synchronization of satellite products with RADS database.

APIs are available with data reading, date and cycle/pass conversion and statistical computation functions for C, Fortran, IDL, (only using previous versions of BRAT), MATLAB and Python, allowing the integration of BRAT functionality in custom applications. For the most common use cases (selection, combinations, visualisations, etc.), command-line tools are available that can be configured by creating parameter files. For beginners, we recommend using the BRAT GUI application, which enables the operator to easily specify the processing parameters required by each tool (and then invoke those tools at the push of a button).

BRAT is provided as Open Source Software, enabling the user community to participate in further development and quality improvement.
1.3. Toolbox contents

BRAT consists of the following parts:

- **BRAT Library**
  The core part of the toolbox is the BRAT library package itself. This package provides data ingestion functionality for each of the supported data products. The data access functionality is provided via two different layers, called CODA and BRATHL.

- **CODA**
  The first BRAT layer (formerly known as BRATLL) is implemented using the Common Data Access framework CODA. CODA allows direct access to product data, supporting a very wide range of products and formats. It provides a single consistent hierarchical view on data independent of the underlying storage format.

  The version of CODA that comes with BRAT supports over 200 altimetric product files. All product file data is accessible via the CODA C library. Furthermore, the version of CODA in BRAT also comes with a set of command-line tools (codacheck, codacmp, codadump, and codafind). Typically, BRAT users will not need to deal with the CODA library directly (although it is included if it is needed), but the CODA command-line tools can be useful for investigating or debugging product data files directly.

  More information about the CODA framework and tools can be found in the CODA documentation, supplied in the BRAT doc/coda/ directory (HTML format). Be aware that in order for the CODA command-line tools to function correctly in a BRAT environment, the user must manually set the CODA_DEFINITION path environment variable to include the location of the BRAT data directory (i.e. the bin/data/ subdirectory of the BRAT installation root directory). This is necessary because the CODA command-line tools need to be told where to find the BRAT product format definition files. In order to check if everything is set properly, the command:

  `codadd list`

will yield a list of all the products CODA recognises. (For a correct BRAT configuration, this list will e.g. include JASON and River_Lake products.)

  More information about the specific altimetry product formats made accessible from BRAT through CODA can be found in the CODA definitions documentation, supplied in the BRAT doc/codadef/ directory (HTML format), and in Chapter 2, Data read and processed and Annex A, List of Datasets read by BRAT.

- **BRATHL**
  The second layer of BRAT provides an abstraction to the product data to make it easier for the user to get the most important data from a product. A single function will allow the user to ingest selected altimetric product data values (from one or more files), into an array. It is also possible (in the same function call) to request statistics on the ingested data and to perform calculations on the data values (e.g. field1 + field2). In addition to the ingestion function, a number of date and cycle data structures and conversion functions are also available.

  The BRATHL library is implemented in C++, and built on top of the CODA framework (plus various other third-party libraries). It is possible to develop programs that make direct use of the C++ classes that make up the BRATHL library, but this is mainly intended for the (rare) case in which users need to develop BRATHL itself.

  Instead, the simple public BRATHL functionality described earlier is accessible via C, Fortran, IDL, (only if using BRAT v3.1), MATLAB and Python interfaces.

  More information about the various BRATHL APIs can be found in Chapter 9, BRATHL Application Programming Interfaces (APIs).

  More information about the C++ BRATHL API can be found in the BRAT reference manual, supplied in the BRAT doc/ directory (PDF format).

- **BRAT Console Applications**
  Most BRAT users will not be programers and will interact with the BRAT library via the use of one or more of the supplied executable applications.
The toolbox contains a number of console applications that are to be run from the command-line. These applications shield the user from the library and the programming level by providing a set of the most commonly needed BRAT functionalities (data computations, data conversions, etc.). These functionalities are in turn user-configurable by so-called parameter files that can easily be created, stored, and shared.

The console applications included in BRAT are: BratCreateYFX, BratCreateZFXY, BratListFieldNames, BratShowInternalFile, BratStats, BratExportAscii and BratExportGeoTiff.

In addition, BRAT also contains the lower-level CODA console applications mentioned in Section 1.2.1.1, as well as the similarly low-level ncdump and ngen utilities. These latter two are part of the netCDF library and can be used to inspect (ncdump) or create (ngen) data files in the netCDF format.

More information about the BRAT Console Applications can be found in Chapter 8, Using BRAT in ‘command lines’ mode with parameter files.

- **BRAT GUI Applications**

  In order to provide a truly pleasant, user-friendly interface to the BRAT functionality, BRAT also contains two applications that present a Graphical User Interface (GUI). It is expected that most BRAT users will primarily interact with BRAT through these applications.

  - **Brat**

    Brat is the main BRAT application. It allows the user to create and manage Workspaces, Datasets, Operations and Views at a very high level of abstraction, and with all the power and convenience of a modern-day graphical user interface. Brat is built on top of the BRAT Console Applications, which it invokes ‘under the hood’, shielding the user from having to deal with command line options or parameter files directly.

    There is a price to pay for the convenience of Brat: not all functionality of the console applications is available through Brat. If the users reach the limits of what can be done with Brat, they will have to learn to work with the console applications after all. For a majority of important uses, however, the functionality of Brat should be sufficient.

    More information about Brat can be found in Chapter 4 BRAT Graphical User Interface (GUI).

  - **Scheduler**

    Scheduler enables BRAT user to delay the execution of an Operation (e.g. having it running at night). It will be available through Brat application, and also through its own icon/executable (to check and modify a scheduled task, in particular).

    More information about Scheduler can be found in Chapter 7 BRAT scheduler interface.
2. DATA READ AND PROCESSED

2.1. Background

The Broadview 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), Cryosat (ESA) and), Jason-2 (CNES/NASA/EUMETSAT/NOAA) and the to be launched Sentinel-3 (ESA/EU) missions. The different types of data readable and processed by the Broadview Radar Altimetry Toolbox are listed below (for a description of the exact datasets with their nomenclature, see 85, List of datasets read by BRAT).

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 (See Chapter 9, 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.

NetCDF COARDS-CF compliant data can be read by BRAT. Note, however, that no warning/error message will be issued if different data are mixed, thus leading to incoherent datasets.

2.2. Level 1B/2 data products

Table 1: Level 1B/2 data products

<table>
<thead>
<tr>
<th>Data</th>
<th>Satellite(s)</th>
<th>Data center</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1B &amp; level 2</td>
<td>Cryosat</td>
<td>ESA</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>Level 1B &amp; Level 2 Ocean Products</td>
<td>Cryosat</td>
<td>ESA</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>RA-2 wind/wave product for Meteo Users</td>
<td>Envisat</td>
<td>ESA</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>(RA2_WWWV_2P)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA-2 Fast Delivery Geophysical Data Record</td>
<td>Envisat</td>
<td>ESA</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>(RA2_FGD_2P)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA-2 Geophysical Data Record (RA2_GDR_2P)</td>
<td>Envisat</td>
<td>ESA</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>RA-2 Intermediate Geophysical Data Record</td>
<td>Envisat</td>
<td>ESA</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>(RA2_IGD_2P)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RA-2 Sensor Data Record (RA2_MWS_2P)</td>
<td>Envisat</td>
<td>ESA</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>Interim Geophysical data record (IGDR)</td>
<td>Jason-1, Topex/Poseidon</td>
<td>AVISO PO.DAAC</td>
<td>Binary</td>
</tr>
<tr>
<td>Geophysical data record (GDR)</td>
<td>Jason-1, Topex/Poseidon</td>
<td>AVISO PO.DAAC</td>
<td>Binary</td>
</tr>
<tr>
<td>Operational Sensor Data Record (OSDR)</td>
<td>Jason-1</td>
<td>AVISO PO.DAAC</td>
<td>Binary</td>
</tr>
<tr>
<td>Sensor Geophysical data record (SGDR)</td>
<td>Jason-1</td>
<td>AVISO PO.DAAC</td>
<td>Binary</td>
</tr>
<tr>
<td>Operational / Interim / Geophysical data record (O/I/GDR)</td>
<td>Jason-2</td>
<td>AVISO EUMETSAT NOAA</td>
<td>netCDF</td>
</tr>
<tr>
<td>Sensor (Interim) Geophysical data record (S(I)GDR)</td>
<td>Jason-2</td>
<td>AVISO EUMETSAT NOAA</td>
<td>netCDF</td>
</tr>
<tr>
<td>Sea Surface Height Anomaly Operational / Interim / Geophysical data record (SSHA O/I/GDR)</td>
<td>Jason-2</td>
<td>AVISO EUMETSAT NOAA</td>
<td>netCDF</td>
</tr>
</tbody>
</table>
### Data

<table>
<thead>
<tr>
<th>Data</th>
<th>Satellite(s)</th>
<th>Data center</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topex waveforms</td>
<td>Topex/Poseidon</td>
<td>PO.DAAC</td>
<td>Binary</td>
</tr>
<tr>
<td>RA OPR</td>
<td>ERS-1 and 2</td>
<td>CERSAT</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>RA WAP</td>
<td>ERS-1 and 2</td>
<td>CERSAT</td>
<td>ESA PDS</td>
</tr>
<tr>
<td>ERS REAPER Level 2 Products</td>
<td>ERS-1 and 2</td>
<td>ESA</td>
<td>netCDF</td>
</tr>
<tr>
<td>Geophysical data record (GDR)</td>
<td>GFO</td>
<td>NOAA</td>
<td>Binary</td>
</tr>
<tr>
<td>Level 1 &amp; Level 2 Products</td>
<td>Sentinel 3’</td>
<td>ESA</td>
<td>netCDF</td>
</tr>
</tbody>
</table>

#### 2.3. Higher level products

**Table 2: Higher level products**

<table>
<thead>
<tr>
<th>Data</th>
<th>Satellite(s)</th>
<th>Data center</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Along-track Delayed-Time and Near Real Time Sea Level Anomalies (DT- &amp; NRT-SLA) (Ssalto/Duacs multimission products)</td>
<td>Cryosat, Jason-1, Jason-2, Topex/Poseidon, GFO, Envisat, ERS-2, ERS-1</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Along-track Delayed-Time and Near Real Time Absolute Dynamic Topography (DT- &amp; NRT-ADT) (Ssalto/Duacs multimission products)</td>
<td>Cryosat, Jason-1, Jason-2, Topex/Poseidon, GFO, Envisat, ERS-2, ERS-1</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Gridded Delayed-Time and Near Real Time Maps of Sea Level Anomalies (DT- &amp; NRT-MSLA) (Ssalto/Duacs multimission products)</td>
<td>Merged</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Gridded Delayed-Time and Near Real Time Maps of Sea Level Anomalies mapping error (DT- &amp; NRT-MSLA) (Ssalto/Duacs multimission products)</td>
<td>Merged</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Gridded Delayed-Time and Near Real Time Maps of Sea Level Anomalies geostrophic velocities (DT- &amp; NRT-MSLA) (Ssalto/Duacs multimission products)</td>
<td>Merged</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Gridded Delayed-Time and Near Real Time Maps of Absolute Dynamic Topography (DT- &amp; NRT-MADT) (Ssalto/Duacs multimission products)</td>
<td>Merged</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Delayed-Time and Near Real Time Absolute Dynamic Topography geostrophic velocities (DT- &amp; NRT-MADT) (Ssalto/Duacs multimission products)</td>
<td>Merged</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Along-track Delayed-Time Sea Level Anomalies (DT-SLA) (monomission product)</td>
<td>Cryosat, Jason-1, Jason-2, Topex/Poseidon, Envisat, ERS-2</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Along-track Delayed-Time Corrected Sea Surface Height (DT-CorSSH) (monomission product)</td>
<td>Cryosat, Jason-1, Jason-2, Topex/Poseidon, Envisat, ERS-2</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Along-track Sea Surface Height Anomalies (AT-SSHA)</td>
<td>Topex/Poseidon, Jason-1</td>
<td>PO.DAAC</td>
<td>Binary</td>
</tr>
<tr>
<td>Along-track Gridded Sea Surface Height Anomalies (ATG-SSHA)</td>
<td>Topex/Poseidon, Jason-1</td>
<td>PO.DAAC</td>
<td>Binary</td>
</tr>
<tr>
<td>Gridded Near Real Time Maps of Significant Wave Height (NRT-MSWH) (monom- and multi-mission products)</td>
<td>Jason-1, Jason-2, Topex/Poseidon, Envisat, GFO, merged</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Gridded Near Real Time Maps of Wind Speed modulus (NRT-MWind)</td>
<td>Jason-1, Jason-2, Topex/Poseidon, Envisat, GFO, merged</td>
<td>AVISO</td>
<td>netCDF</td>
</tr>
<tr>
<td>Data</td>
<td>Satellite(s)</td>
<td>Data center</td>
<td>Format</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>------------------------------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>Heracles along-track land-ice (multimission products)†</td>
<td>Cryosat, Envisat</td>
<td>ESA</td>
<td>netCDF</td>
</tr>
<tr>
<td>Heracles crossover land-ice (multimission products) ‡</td>
<td>Cryosat, Envisat</td>
<td>ESA</td>
<td>netCDF</td>
</tr>
<tr>
<td>Gridded Heracles SHA land-ice (multimission products) ‡</td>
<td>Cryosat, Envisat, merged</td>
<td>ESA</td>
<td>netCDF</td>
</tr>
<tr>
<td>Gridded Heracles Sigma0 land-ice (multimission products) ‡</td>
<td>Cryosat, Envisat, merged</td>
<td>ESA</td>
<td>netCDF</td>
</tr>
<tr>
<td>Gridded Heracles Leading Edge Width (LEW) land-ice (multimission products) ‡</td>
<td>Cryosat, Envisat, merged</td>
<td>ESA</td>
<td>netCDF</td>
</tr>
<tr>
<td>River &amp; Lake products</td>
<td>Envisat</td>
<td>ESA</td>
<td>Binary</td>
</tr>
</tbody>
</table>
3. HOW TO INSTALL AND UNINSTALL BRAT

3.1. Supported platforms

BRAT binaries are available as single-file installer packages for the three major operating systems, in 32 and 64 bit processor architectures: Windows\(^1\), Linux\(^2\), and Mac OS X\(^3\). These standalone installers can be downloaded from the BRAT Website (http://www.altimetry.info/toolbox/) or copied from the top-level directory of the BRAT Distribution DVD.

On not directly supported platforms and for certain purposes, BRAT will have to be compiled from source. A source archive is therefore also available, but as compilation is a rather complex affair it is highly recommended to try one of the binary installers first.

3.2. The BRAT distribution DVD

The BRAT Distribution DVD contains:

- The binary installers for the supported platforms.
- The source archive.
- A copy of all the BRAT documentation (also already included in the binary installers).
- A large directory of sample data files (which is too large to be included in the binary installers).

3.3. MS Windows

3.3.1. Installing the binary distribution

BRAT supports Windows XP and higher, 32 and 64 bit. The binary distribution contains pre-built versions of the full toolbox as well as all the BRAT documentation and examples. For the MATLAB and Python interfaces, pre-built versions are included that will work with MATLAB V8.1/R2013a or higher and Python 3.0 or higher. For the IDL interface, BRAT version 3.1.0 should be used; it will work with IDL 6.3 or higher.

The BRAT Windows binary installers are found in the files:

- `brat-4.2.1-Win32-installer.exe` (32 bit)
- `brat-4.2.1-x64-installer.exe` (64 bit)

In order to install BRAT, select and double-click the installer file that matches the architecture of your Windows version and follow the instructions.

By default, BRAT will be installed in `C:/Program Files/BRAT-4.2.1/\(^4\)`, or in the user's local profile directory when installed as a user without Administrator privileges. It is also possible to specify a custom installation location during the installation process.

After installation, the BRAT Console and GUI applications are immediately ready for use. A shortcut to the BRAT application will have been placed on the desktop and is also accessible via the Start > Programs > Broadview Radar Altimetry Toolbox<version><architecture> menu. In order to use the Console Applications, open a command window and call the applications directly from their installed location.

---

\(^1\) Windows® is a registered trademark of Microsoft Corporation in the United States and other countries.

\(^2\) Linux® is the registered trademark of Linus Torvalds in the U.S. and other countries.

\(^3\) Mac OS X® is a registered trademark of Apple Inc. in the U.S. and other countries.

\(^4\) This is valid for 32 bit installers in 32 bit systems and 64 bit installers in 64 bit systems. 32 bit installers in 64 bit systems will install BRAT by default in `C:/Program Files (x86)/BRAT-4.2.1/`. 
There are a number of optional software prerequisites to using BRAT after installation:

- If you plan on using the C interface, you should have a C or C++ compiler installed on your system. The C interface has been verified to work with Microsoft Visual Studio 13 and 15, but it is expected to be compatible without major issues with the same tools that could build BRAT 3.1.0, in the same or higher versions, with the exception of Visual Studio 6 and earlier.

- If you plan on using the Fortran interface, you should have a FORTRAN 77 or Fortran 90 compiler installed on your system.

- If you plan on using the IDL interface, besides installing BRAT 3.1.0 side-by-side with BRAT 4.2.1, you need a recent version of IDL for Windows: The IDL interface has been verified to work with IDL version 6.3 and higher.

- If you plan on using the MATLAB interface you need a recent version of MATLAB for Windows: The MATLAB interface will only work with MATLAB version V8.1/R2013a or higher.

- If you plan on using the Python interface you need to have a version of Python 3.x installed on your Windows system, and it must match the installed BRAT architecture (32 bit or 64 bit). You will then be able to open a console in the sub-directory `\examples\python` of your installation root and run:
  
  `\> python example.py`

  If the Python executable is not referenced in your PATH environment variable, you must invoke python with the full path; so, if Python is installed in C:\Python34\, the command will be:

  `\> C:\Python34\python example.py`

  Check `example.py` and the annex concerning the Python API to find more details about setting up the proper environment for running Python code that interfaces with the BRATHL library.

### 3.3.2. Installing from source

Generally, installation from source will be necessary if:

- You want to use the MATLAB interface to BRAT for a version that is incompatible with the pre-compiled interface in BRAT.

- You want to use the Fortran interface.

The BRAT source distribution can be found in the file:

`brat-4.2.1.tar.gz`

After unpacking this archive in a suitable location, instructions for configuring, compiling and installing BRAT for Windows can be found in the top-level file `INSTALL`.

### 3.3.3. Uninstalling

Open the 'Add/Remove Programs' or 'Programs and Features' control panel, and select the BRAT-<version>-<architecture> entry. Everything created during installation will then be removed. Files added after installation will remain, for the user to check if they can be safely deleted. Alternatively, choose the 'Uninstall BRAT' menu item from Start > Programs > BRAT<version>-<architecture> – this will have the same result.

These uninstall methods only work for BRAT installations created through the binary installers. For BRAT installations from source, you will need to remove the various files and directories manually.

### 3.4. Linux

#### 3.4.1. Installing the binary distribution

BRAT is developed on platforms running the Debian GNU/Linux 8.x operating systems, 32 bit (PAE) and 64 bit. The following dependencies are required for BRAT to run:
• Libcurl
• Libgdal
• libspatialindex
• rsync
• libgeos-c1
• libxerces-c3.1
• libproj0
• libegl1-mesa

Other Linux distributions could work equally well, depending on their compatibility with the Debian 8 distribution. Install the BRAT binary distribution and simply see if it works or not (if it does not, you can always try to compile BRAT from source – see below for details).

The binary distribution contains pre-built versions of the full toolbox as well as all the BRAT documentation, examples, C, Fortran and Python interfaces. Because of inherent library versioning and path issues on the Linux platform, no MATLAB interface is included in the binary installation. If desired, it can be created by compiling from source for your specific installed version of MATLAB. For the IDL interface, BRAT version 3.1.0 should be used.

The BRAT Linux binary installers are found in the files:

- brat-4.2.1-i386-installer.run (32 bit)
- brat-4.2.1-x86_64-installer.run (64 bit)

In order to install BRAT, double-click on the installer file from a desktop manager window (or execute it from a command-line shell) and follow the instructions. (If you downloaded the installer via a network it may have been given the wrong file permissions and not be recognised by the system as executable. You should run the command `chmod +x brat-4.2.1-x-installer.run`, replacing “x” by “i386” or “x86_64” as appropriate, in order to make it executable.)

By default, BRAT will be installed in `$HOME/brat-4.2.1-<architecture>/` (where `$HOME` stands for the user’s home directory and `<architecture>` is i386 or x86_64). It is also possible to specify a custom install location during the installation process.

After installation, the BRAT Console and GUI applications are immediately ready for use. A shortcut to the BRAT application will have been placed on the desktop. In order to use the Console Applications, open a command-line shell and call the applications directly from their installed location (`$HOME/BRAT-4.2.1-<architecture>/bin` or else wherever you instructed the installer to install BRAT).

There are a number of optional software prerequisites to using BRAT after installation:

- If you plan on using the C interface, you should have the GNU C or C++ compiler installed on your system. The C interface has been verified to work with GNU C/C++ 4.7.2.

- If you plan on using the Fortran interface, you should have a FORTRAN 77 or Fortran 90 compiler installed on your system. The Fortran interface has been verified to work with GNU Fortran 4.7.2.

- If you plan on using the Python interface you need to have installed a version of Python 3.x matching the installed BRAT architecture (32 bit or 64 bit). You will then be able to open a console in the subdirectory `/examples/python` of your installation directory and run

  ```
  $ python3 example.py
  ```

Please see the "Known Issues" section of the README file if you are using the Gnome desktop and notice missing buttons in the windows title bars (close, minimize and maximize buttons); this is a known issue that affects all GUI applications in Gnome.

### 3.4.2. Installing from source

Generally, installation from source on Linux will only be necessary if:

- You want to use the MATLAB interface to BRAT.
• You are on a system not compatible with the one used to create the BRAT Linux binary distribution (in which case BRAT will fail to run if installed as a binary).

The BRAT source distribution can be found in the file:

`brat-4.2.1.tar.gz`

After unpacking this archive in a suitable location, instructions for configuring, compiling and installing BRAT on Linux (or other Unix-based systems) can be found in the top-level file INSTALL.

3.4.3. Uninstalling

In the installation folder (the default one or the one chosen), there is an executable called Uninstall-brat-4.2.1-i386 or Uninstall-brat-4.2.1-x86_64 which can be executed to remove everything created during the installation. Any files created by BRAT or the user after the installation process will remain, so that the user can check if they should be kept or can be safely deleted.

There is also a shortcut, called ‘Uninstall BRAT-4.2.1-<architecture>’, which can be double-clicked from within your desktop manager (if you use the KDE or GNOME desktop environment) to get the same result.

3.5. Mac OS X

3.5.1. Installing the binary distribution

BRAT 4.2.1 is supported on Intel-based systems running Mac OS X versions 10.8 or later (32 and 64 bit kernels).

This binary distribution contains pre-built versions of the full toolbox as well as all the BRAT documentation, examples, C, Fortran and Python interfaces. Because of inherent library versioning issues on the Mac OS Unix-based platform, no MATLAB interface is included in the binary installation. If desired, these can be created by compiling from source for your specific installed version of MATLAB. For the IDL interface, BRAT version 3.1.0 should be used.

The BRAT Mac OS X binary installers can be found in the disk image files:

`brat-4.2.1-macosx-i386.dmg` (32 bit)

or:

`brat-4.2.1-macosx-x86_64.dmg` (64 bit)

In order to install BRAT, double-click on the image file to mount and open it. Copy “BRAT” and “SCHEDULER” application bundles (brat.app and scheduler.app) that are inside the disk image to your Applications folder.

In order to use the Console Applications, open the Terminal application in the MacOS sub-directory of the brat.app application bundle and run the applications directly from there.

To do a full installation, including the several documentation items (README, INSTALL, manuals, etc.), you can copy the mounted installation folder to Applications. This is also recommended if you have other versions installed, or if you plan to use both 32 and 64 bit versions on the same system. Each complete version will then be located in its own folder, properly identified, without overwriting any file previously installed, as would be the case if the separate items were dragged directly into Applications.

After installation, the BRAT Console and GUI applications are immediately ready for use. BRAT can be started by double-clicking the brat.app icon.

There are a number of optional software prerequisites to using BRAT after installation:

• If you plan on using the C interface, you should have the clang compiler installed on your system.

• If you plan on using the Fortran interface, you should have a FORTRAN 77 or Fortran 90 compiler installed on your system. The Fortran interface has been verified to work with GNU Fortran 5.1.0.

• If you plan on using the Python interface you need to have Python 3.x for Mac OS X installed on your system. You will then be able to open a console in the sub-directory /examples/python of the 'brat.app/Contents' folder and run
$ python3 example.py
Or, to use 32 bit Python,
$ arch --i386 python3 example.py

The Python version that you invoke must match the architecture (32 bit or 64 bit) of the BRAT installation where is located the example you are trying to run.

Check example.py and the annex concerning the Python API to find more details about setting up the proper environment for running Python code that interfaces with the BRATHL library.

3.5.2. Installing from source

Generally, installation from source on Mac OS X will only be necessary if:

- You want to use the MATLAB interface to BRAT.

The BRAT source distribution can be found in the file:

```
brat-4.2.1.tar.gz
```

After unpacking this archive in a suitable location, up-to-date instructions for configuring, compiling and installing BRAT on Mac OS X can be found in the top-level file INSTALL.

3.5.3. Uninstalling

To uninstall any version of BRAT, simply move to the trash any items you copied when installing that version.

3.6. The RADS Service

To use BRAT with RADS data, periodically synchronized with the RADS servers, you have to install the RadsService. This service or daemon starts when you login to the operating system, checking at specified time intervals if there is new data in RADS, and downloading it to a predefined location. You can define the period, in days, to check for new data, as well as specific missions, mission’s phases, and a storage location to save locally the downloaded files, organized in the same folder tree structure as in their origin.

3.6.1. Installing the RadsService

The RadsService installation is done from inside the BRAT GUI, using the Options dialog, or Preferences dialog in Mac OS X, which you can access in the “Tools” or the “brat” menu. Once in the dialog, click the RADS button to display the “RADS” configuration page. At the top you have the “Install service” button. All other widgets in the page will be disabled when the service is not installed.

3.6.1.1. Required Permissions

Installing a service requires permissions that the user may not have.

In Windows, installing and configuring the service requires running BRAT with administration permissions, otherwise the “RADS” configuration page will be disabled. If this happens, and your account is an administrator account, you should restart BRAT as administrator, by right-clicking the BRAT icon and selecting “Run as administrator”. If your account is a standard user account, or without administration privileges, it is recommended to login to Windows as administrator to install and configure the service; you can also try to run BRAT as administrator, but, depending on your operating system version and how it is configured, that may not be enough when logged in as a standard user.

In Mac OS X or Linux BRAT should be started without root privileges. The required permissions, if any, will be asked during the installation procedure. After installing, the daemon can also be configured without any special privileges.

In all systems, please make sure that any TCP traffic from your machine to port 873 on the RADS server is not being blocked by a firewall. This is usually not an issue, but it can happen if your system is accessing RADS behind a more restrictive network configuration.
3.6.1.2. **Installation and configuration procedures**

After accessing the “RADS” configuration page, click the “Install service” button. A dialog will pop-up showing the user name and asking a password. Do not change the user name and enter the respective password. After that, in Linux, the root password may also be requested to set up the daemon’s auto-start settings.

When the installation is complete, a notification is displayed, the button text will change to “Uninstall service”, and the “Service Settings” section will be activated. Use the widgets in “Service Settings” to specify the local download directory, the interval in days for data synchronization and the missions whose data you want to receive. The downloaded files will be stored under the “rads” sub-directory of the directory you specified; care should be taken to enter a directory where all the users that will run BRAT in the machine have read/write access.

In the RADS configuration page you can also find the “Start service” and the “Synchronize now” buttons. Using these buttons is optional, the RadsService will operate according to the settings you defined without the need for additional user intervention. However, the “Start service” button can be convenient if you want to start the service immediately after installation, without rebooting your machine, or if you want to start/stop it later for any reason. Also, the "Synchronize now" button will allow you to update the local data before the next scheduled synchronization, or to stop any current download. Note that this will not change the periodic synchronization schedule.

3.6.2. **Uninstalling the RadsService**

If you installed the RadsService, uninstalling it from the BRAT GUI is required before uninstalling the whole of BRAT. Otherwise you will have to manually uninstall it by using the tools and procedures provided by the respective operating system, which may not be as simple or convenient for most users.

You will need to run BRAT (as administrator, if in Windows, or as usual in the other systems) and, after accessing the “RADS” configuration page, click the "Uninstall service" button. If the service is running, you will have to stop it first. Uninstalling the service will not delete any data previously downloaded. Also, it will not affect any other component of the BRAT toolbox, which can be fully used independently of the RadsService.

3.7. **Troubleshooting OpenGL issues**

Some view types in the BRAT GUI use OpenGL to display 2D and 3D graphics. If your operating system or graphics card do not support all capabilities required by BRAT, some graphic issues may arise, in particular with the 3D plots and the globe. In some cases BRAT may even not be able to start at all.

However, BRAT OpenGL requirements are not very demanding, considering that the least OpenGL version supported is 1.5, dated 2003. Anyway, the OpenGL version and capabilities must be checked when the BRAT GUI is started, and a warning is displayed if any limitations are found or BRAT cannot start.

In older hardware, or in virtual machines, OpenGL issues are more likely to occur. If you are running your operating system in VirtualBox, you can try to test the "Enable 3D Acceleration" and the "Enable 2D Video Acceleration" options (in the "Display" tab of the VM settings). Note that disabling them can in some cases solve graphics problems.

In some systems, during the OpenGL check, segmentation faults can occur before any warning had the chance to be displayed (again, this more likely to occur in virtual machines). When that is the case, and all other options failed, you can disable the OpenGL verification in the configuration file, called BRAT.ini. The location of this file, in the several systems, is the following:

- Linux: in ~/.config/ESA
- Windows: in C:\Users\<user name>\AppData\Roaming\ESA
- macOS: in ~/.config/ESA

Open the file in any text editor and change the line

```
check_opengl_capabilities=true
```

to

```
check_opengl_capabilities=false
```
In systems where BRAT is run for the first time, it may happen that you cannot find BRAT.ini because the BRAT GUI could not write it before exiting. In that case, create the file, open it in a text editor, and paste the following, which will be the only file content:

```
[Common]
check_opengl_capabilities=false
```

Note that disabling the OpenGL verification is only a troubleshooting procedure, meant to identify possible causes of eventual BRAT GUI start-up problems. If you cannot start BRAT without resorting to it, it is probable that you will experience problems when trying to display the globe or 3D plots, problems that can only be fixed by upgrading OpenGL in your system.

### 3.8. Sample Build in Debian 8

Building BRAT and all its dependencies, or at least those that are not available as pre-compiled binaries for some intended build configuration, is a complex and time consuming task. In spite of BRAT support for all major desktop operating systems and most common processor architectures, providing installers for all of them as detailed above, the user may still want to try to build BRAT for some unsupported operating system or unusual configuration.

The INSTALL document, distributed with the BRAT source package, tries to cover the major guidelines and issues about this task, for all supported operating systems and major build tools, and to suggest some hints for the unsupported ones. But, given the big number of different compilation options, both for BRAT and for the packages that it depends on, it is not possible to cover all possible combinations, which are still increased by the possibly different operating system configurations, distributions, build tools, and so on. Because of this, the document is merely indicative in many respects, leaving some options for the user to decide. It is not an instruction set that, if accurately followed step by step, ensures a successful build of the whole BRAT package.

However, it is possible to have such an instruction set for a specific environment and build configuration. This sort of document can be useful either as a reference for similar contexts or to try to infer the necessary adjustments for different ones, if at all possible. The file “SampleBuild.pdf” consists precisely in a set of guidelines and detailed steps to build a specific BRAT configuration, in a freshly installed Debian 8, 64 bit. It can be found in the “doc” folder of BRAT sources, which can be downloaded from GitHub at https://github.com/BRAT-DEV/main. In the “COTS” sub-folder of the sources root there are also some preconfigured packages required to build BRAT according to the instructions in the build document.
4. BRAT GRAPHICAL USER INTERFACE (GUI)

4.1. 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).

The BRAT GUI includes:

- a "Workspace Elements" dock, with 4 tabs:
  - "Datasets"
  - "RADS Datasets"
  - "Filters"
  - "Operations"

- an "Output" dock, with 2 tabs:
  - "Logs"
  - "Processes"

- the main map

You can configure the position of both docks.

BRAT GUI basically creates parameter files (see Section 8, Using BRAT in 'command lines’ mode with parameter files), that are stored in the 'Operations' folder of the respective workspace. It also enables to save your preferences and work.

4.1.1. The Application Settings (Options)

Some aspects of BRAT appearance and usage can be configured through the Application Settings dialog. We have already seen how to use it to install and configure the RADS Service.

Besides the RADS section, the dialog includes 3 other sections:
- Paths: to define the default paths for BRAT to search workspaces and data.
- Start-up: to define which actions or settings to use when the application starts.
- Style: to define GUI themes.

In Figure 1: The start-up section of the BRAT settings dialog, the dialog is displayed with the Start-up section selected. Here you can configure the map layer types to use for the main map and the data views maps. You can also change the layer options for views in a view window, but in the main map case you will need to restart the application for the options to change.

Besides vector layers, you can also select raster layers with two display options:

1) A local raster file, such as the one provided in the BRAT installation folder, using the default GDAL provider.

2) A raster URI using the WMS provider.

The default raster URI used by BRAT can be explained as follows:

a) In the webpage
   http://www.gebco.net/data_and_products/gebco_web_services/web_map_service/
   it is described that the WMS access is done with the link
   https://www.gebco.net/data_and_products/gebco_web_services/web_map_service/mapserv?

b) One then requests the information of the WMS service via the link
   https://www.gebco.net/data_and_products/gebco_web_services/web_map_service/mapserv?req
uest=getCapabilities&service=wms
This returns an XML file with information about the WMS service.

c) The URI to use is then

http://www.gebco.net/data_and_products/gebco_web_services/web_map_service/mapserv?&service=wms&layers=GEBCO_LATEST&styles=&crs=EPSG:4326&format=image/jpeg

that is built by the link of the service

https://www.gebco.net/data_and_products/gebco_web_services/web_map_service/mapserv?&service=wms

and the respective parameters for the layer as retrieved from the XML file.

The mandatory parameters for the URI are the layers (layer name to use), styles (which can be empty as it assumes the default style), srs (the projection), format (the format of the image returned). These parameters have to match with the available values as described in the XML returned on point b)\(^5\).

The log window displays warning messages about raster display failures. You may also need to slightly adjust the zoom if you see no image in the map and the log window does not show any warnings.

\[Figure 1: The start-up section of the BRAT settings dialog\]

The next section of this manual (4.2, Starting with BRAT GUI) explains the basics of the interface. For more detailed information about all the functionalities, see section 4.3, BRAT GUI tabs description.

**4.2. Starting with BRAT GUI**

Using BRAT GUI is basically a 3-step process.

\(^5\) For more details on the WMS service standard please refer to http://www.opengeospatial.org/standards/wms/introduction
You have to:

1. create a new **Workspace** or open an existing one (see section 5);
2. define one or several ‘**Dataset(s)**’: the product data you want to work on (see section 4.2.2);
3. add one or more **Filters**: this step is optional and allows the creation of data filters (for your input datasets) using time or location criteria (see section 4.2.3); filters are independent of the workspace and can be reused in different workspaces;
4. create an **Operation** (quick or advanced): configure the data fields you want to visualize and respective process parameters that are used for generating the plots (see section 4.2.4).

The Datasets, RADS Datasets, Filters and Operations tabs are within the 'Workspace Elements' dock. Each tab corresponds to a different function, and to a different step in the process, so you'll have to use first the 'Datasets' or the 'RADS Datasets' tab, to define the input data, then the 'Filter' tab in case you want to filter inputs, and finally the 'Operations' tab to define a computation over the previously defined inputs, generating a view with the result.

This section gives the main information for a quick-start with BRAT GUI. For more complete information, see the relevant sections within the 4.3, BRAT GUI tabs description.

### 4.2.1. Create a workspace

When you open BRAT GUI, the software asks for the name and location of the 'Workspace’ you will be working in. A 'Workspace' is a way of saving your preferences, computations and generally the work done with BRAT GUI. Some or all elements of a workspace can be imported into another workspace. The “Workspace” menu (and also the main toolbar) allows the user to create, open, close, save, import, rename or delete a workspace.

**It is highly recommended to save your workspace** (ctrl+s, or ‘save’ in the “Workspace” menu) **while working**. You will be asked whether or not you wish to save the workspace when you quit BRAT GUI. Note that if you answer "no" and have not saved anything previously, none of your work can be recalled later.

If there are already one or more valid workspace(s), BRAT GUI recalls the last used Workspace by default. This behaviour is configurable in the Settings dialog.

![Workspace Dialog - Create](image)

*Figure 2: 'Create a new workspace' window. You can choose to save it wherever you want on your hard drive or local network, and name it as you prefer (preferably in such a way you will remember what's in it).*

### 4.2.2. Create a dataset

The first tab opened if you have never used BRAT is ‘**Datasets**’ (otherwise, the default tab is the one that was opened when you left BRAT GUI the last time you used it). This 'Datasets' tab is dedicated to the definition and selection of the data you want to use. You must define **at least one** dataset to be able to further use BRAT.

To create a dataset, click on the **new** button in the Datasets tab.
Default name for a new dataset is 'Dataset_1', with the number incrementing each time you create a dataset. You are strongly encouraged to rename it, so that you'll remember what's in it when using it later on. To rename it, simply double-click on dataset name, type in another one and press the Enter key.

When you have created your dataset and named it, you then have to add one or more data file(s), chosen from your hard drive, CD/DVD driver, local network or other medium. You can do so by using the 'Add Files' button. **At least one file is necessary.**

If you wish to add a long list of files, the 'Add Dir' button allows you to choose all of the files within a folder by simply choosing the directory in which they are stored. Please note that **only the data files recognized by BRAT are added** to the dataset, the remaining files in the folder (with an unknown format) are ignored after raising a warning message.

**Only coherent datasets are possible** (i.e. same format, same data product). BRAT’s own netCDF outputs can be used, even several of them, provided they have exactly the same variables, with the same names. In the current BRAT release, homogeneity of the Dataset files is performed automatically as the files are added.

Once you have added at least one data file, if you click on one file name in the list, you can see (at the bottom) information about the available fields within the data product, and (for netCDF files) about the file description below. The satellite tracks for the selected data file are also shown on the main map.
4.2.2.1. RADS datasets

A RADS dataset is created like a standard dataset, by using the 'New Dataset' button and naming the new dataset, but you won’t be able to select individual files or folders to add data to it. Instead, you select a single mission from the list of missions made available by the RADS server and assign it to the dataset. Each dataset can be associated with only one mission. All files locally downloaded by the RadsService for the respective mission and selected mission’s phases will be included in the dataset.

After naming it, the new RADS dataset is created empty, and you select the respective mission and phases from the list of missions displayed by clicking in the drop-down button of the dataset entry in the "Datasets" list.

Unlike with standard datasets, no satellite tracks are displayed when a RADS dataset is selected, because typically the large amount of data, even for the smallest phases samples, would cover the whole map, which besides inefficient is useless in terms of visual information.

4.2.3. Create a filter

The filter enables to select only the data relevant for your work, in order not to uselessly process data out of desired area and period (date or cycle and pass). This new feature “Filters” adds a graphical option to define the filters which can also be defined by hand in the ‘Define selection criteria’ feature as explained in Section 4.2.4.2.4. Note that, besides selecting relevant files, ‘Filters’ also allow extracting/selecting data from files. This new feature intends to visually define the data that the user wants to process and was designed with the selection of both geographical and time criteria in mind. Therefore, it is required to set both the area and time criteria to be able to use the filter feature in this tab.

To use this feature, click on “Create filter” button to create a new filter. The default name for a new filter is “Filter_1”, with the number incrementing each time you create a filter. Then follow the instructions below to define the selection criteria (spatial and temporal criteria).
Define spatial criteria (“Where” section):

Click on the “Selection Tool” button, located at the top left of the “New Area Selection” group box, to activate the selection tool, which will allow you to draw a selection rectangle over the map or write the area coordinates in the text boxes below the button.

Click on the “Create area” button, the top left button of the “Areas and Regions” section, to create a new area from the current map selection. If the area was defined by writing the coordinate values, the area selection will be created in the map; conversely, if the area was defined with the mouse, the text boxes will reflect the area coordinates. The default name for a new area is “UserArea_1”, with the number incrementing each time you create an area. You are strongly encouraged to rename it, in order to easy the identification of each area.

Once you have created all the required areas to define the spatial criteria, select the checkbox of each area to include it into the filter.

Note that:

- The “Create area” button is enabled only when the “Selection Tool” is active.
- If the selection is partially outside the map limits, BRAT creates a truncated area containing only the part within the map limits.
- To duplicate an existing area, select it on the areas list, enable the selection tool, and click on the “Create area” button.
- The areas can be grouped into different regions for easy handling of a long list of areas.

![Image of the filters tab in BRAT](image)

*Figure 5: The filters tab. The “New Area Selection” tool enables to draw a selection over the map or enter its values in the coordinates fields; the “Create area” button creates a new area from current map selection.*

Define temporal criteria (“When” section):

Edit the information on this section to define the desired time period. You can define the start and stop times using any of the following options:

- Absolute date and time;
- Cycle and Pass of the product mission;
- Relative time in days from a reference date and time.

**Restrictions on the usage of the “Filters” tab:**

The use of this filter feature does not invalidate the use of the manual selection criteria in the Operations tab, and in some cases, it may not be possible to use the filter defined in this tab. One such case is when the selected file(s) does not have a time variable. Since the filter defined in this tab requires the time criteria to be defined, if the data does not have a time variable the GUI will throw a warning saying that the filter cannot be applied.

In those cases, two options are available if the user wants to cut a geographical area of a dataset which does not have any time variable. The filter tab should not be used in those cases and the user can use the two following options:

1. The user defines the limits of the output grid in the Sampling configuration parameters at the definition of the operation (see Section 4.3.5.4.6). This is the easiest method that is suggested.
2. The user can also define manually the selection criteria where the lat and lon limits can be defined (see Section 4.2.4.2.4) as a logic expression. For example to define a box between 38° and 39° in latitude and 22° and 23° in longitude, the expression would be "lat < 39 && lat > 38 && lon > 22 && lon < 23”.

**4.2.3.1. Customizing filter application**

Filters are common across workspaces and independent of the operations (see 4.2.4), where they are used to select the pertinent data. So, the same filter can be applied to different operations, without requiring in each case the redefinition of data selection criteria. However, in some cases it is necessary to customize the filter for a specific operation. The customization details are then provided by the user, as explained next, and transparently attached to the operation (not to the filter, which always preserves its definition) until the BRAT session closes.

Filters are internally applied using aliases (see 5). The aliases mechanism allows BRAT to identify in data files of different products the respective variable names for the parameters longitude, latitude and time, which are required to define the space and time boundaries of the data to filter in those files. However, some files have more than one variable referring the same parameter, e.g., variables with different dimensions. For instance, there can be a variable called “lat_01” with dimension time_01=1458 and also a variable “lat_20_ku” with dimension time_20_ku=29274, and still other, all referring latitude. In this example, if the user wishes to include lat_20_ku in the operation, but the default latitude variable (as defined by the respective alias) is lat_01, an error will occur and a message is displayed after the operation fails to execute, notifying about the dimensions discrepancy problem (time_01 is automatically inserted by the filter in the operation, but so is time_20_ku, dimension of lat_20_ku).

In this case, the user must specify which variables should be used by the filter for the parameters longitude, latitude and time. When the "Selection criteria" node is selected in the "Data Expressions" tree widget (see 4.3.5.3), a button in the frame below it, identified by the filter icon, becomes active and triggers a dialog box where the default variable names are displayed and can be modified by the user. These names will be used by the filter, instead of the default ones, in later operation executions. In our example, lat_20_ku should be specified as the latitude name instead of lat_01; then, the variable names for longitude and time should also be changed to the names of the respective variables having the same dimension as lat_20_ku (e.g., lon_20_ku and time_20_ku).

**4.2.4. Create an operation**

Once you have defined which data you want to work on (and eventually the filters that will be used for data selection), you have to define which data fields you want to process and visualize. This is done in the ‘Operations’ tab. You can choose between creating a quick or an advanced operation.

**Create a quick operation:**

To quickly execute an operation, select a dataset, check a field of interest in the Fields list, and click the Map or Plot buttons to see the result. A new name, that you can change later, will automatically be assigned to the new operation.
Note that you can convert the quick operation into an advanced operation one by duplicating it in "Advanced" mode. This is necessary if you want to keep the results of a quick operation, because the next quick operation will replace all results and definitions of the previous one.

Create an advanced operation:
If none exist, you **have to create an Operation**. Click on the 'create operation' button.

Default name for a new Operation is 'Operations_1', with the number incrementing each time you create an operation. You are strongly encouraged to re-name it, so that you'll remember what's in it when using it later on. To rename it, simply click on "Rename operation" button, type in another one and press the Enter key.

![](image)

Figure 6: The 'Operations' tab in the advanced mode. The "Create operation" button enables to create a new operation.

Otherwise, you may work with a previously saved operation. The 'Operation' dropdown list contains all the already defined operations within the workspace, which can be selected, renamed, modified, deleted, duplicated, and other options.

4.2.4.1. Select source data

The information about the source data is in the topmost part of the Operations tab.

You first have to choose the dataset you want to work with from the 'Dataset' dropdown list. Then, within this dataset, the whole list of available fields is proposed, organised as a tree. If the data are split in different records, click on the '+' to expand the tree, '-' to flatten it.

The description of each field is given in the top (right) text box information available.
4.2.4.2. Define expressions

4.2.4.2.1. Generalities

An operation consists mainly in the definition of 'Expressions'.

An expression can be simple (one data field), or complex (with the use of arithmetic combinations, functions applied on several fields, etc.).

In the second row box of the 'Operations' tab ('Data Expressions'), you can see four categories of Expressions:

- 'X' or 'Lon' (case 'Plot' or 'Map' option is selected, respectively)
- 'Y' or 'Lat' (case 'Plot' or 'Map' option is selected, respectively)
- Data
- Selection criteria (optional)

At least one expression as 'X', and one as 'Data' must be defined for an Operation to be valid.

These expressions can be filled by several means, the quickest being by drag & drop: drag a field from the top (left) list and drop it in either of the Data Expressions nodes (you can also use contextual menus by right-clicking either on the data fields or on the expressions, or use the 'Insert expression' and/or 'Insert field' button, or type in an empty expression the field names and functions you want to apply).

A 'brat_index_data' can be listed within the available data field. This is the index of the data (i.e. Measurement number in the file and/or record) ordered along the time within a given file. This means that it is not available for (e.g.) longitude-latitude grids, or for some data where the time is not provided explicitly.

If using this index with several different files in the same dataset, note that the order of the files as appears within the dataset will be kept (thus, if the files are not ordered chronologically, the net result will not be chronological either).

Note that only one expression can be defined as X, and (optionally) one as Y, whereas more than twenty can be defined as Data.

An Expression can be:

- only one field in a dataset (typically, for a map, longitude as X-axis, latitude as Y-axis, and e.g. significant wave height as Data, etc.)
- a combination of fields, either +,-,* and /, or by using the available Functions (see 4.3.3.4.2, Functions).
• a pre-set combination of fields among the ones you will find in the ‘Formulas’ (see 4.3.3.4.3, Formulas), e.g. SSH computation.

To check if your expression is well formulated, you can click on the ‘Check syntax’ button (green tick on the right of the Data box). Note, however, that this won’t provide you with a validation of the relevance of your expression from the point of view of physics.

The ‘Show Info’ button provides information about the original units (the ones defined in the data products) and the units used during computation or selection.

The ‘Show Aliases’ button provides information about the aliases available for the chosen dataset. Aliases are equivalents that you can use instead of the fields’ name. E.g. a %{swh} alias exists, that works for all GDR data for the Ku-band significant wave height. Note, however, that since not all the fields exists for all the data, you may encounter warnings if you try some aliases on all the altimetry data.

If you want to go back on your work later on, or to save an expression as formula choose the ”Save as formula” option button and fill in the requested fields.

4.2.4.2.2. X, Y and data expressions

You can change the name of any X, Y or Data Expression, by double-clicking on their name, or by using the contextual menu available by right-click. This will then be the default name on the plots, on the axis or near to the scale if you have not given a title to your Expression (in the title/comment).

You can change the unit as it appears in the Data Expressions tree area.

BRAT is able to understand all SI units and their sub-units as defined in the International System, i.e. case sensitive (e.g. "ms" means milliseconds, whereas "Ms" would mean megaseconds). There are also "count" for data without dimension, and “dB” (see section 4.3.3.4.1, Units). 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 you used was defined in mm). Note that you have to validate your change of unit by typing “enter” or clicking on the box below.

If you choose a pre-saved formula, a default unit will appear as the unit. If you select one field in the dataset list and insert it, it will automatically be filled with the correct unit (but if you finally write your own formula, beware that the final unit might be different). If the unit you defined does not fit the unit of the data as defined, an error message will be generated (again, this does not work for complex expressions).

On any X, Y or Data Expression, you can apply ‘data computation’ (see 4.3.3.4.5, Data computation), to:

• compute statistics at each point (same X, optionally same Y): MEAN, STDDEV (standard deviation), COUNT.

• do some arithmetic operations between files within a dataset: adding, subtracting or multiplying: SUM, SUBTRACTION, PRODUCT)

• it 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.

Note that to compute the statistics for the Data Expressions as a whole (Number of valid data, Mean, Standard deviation, Minimum, Maximum), you can use the ‘Compute Statistics’ (capital epsilon on the top right) button.

There are two main kinds of Operations:

• one – or several – Data expression(s) with respect to another one (X), leading to a “curve” plot

• one – or more – Data expression(s) with respect to two others, leading to a “map” plot or 3D plot.

In the first case, you’ll fill only the “X” expression; in the second, you'll fill both X and Y expressions. Note that X and Y can be Longitude and Latitude, but can also be any other two fields or combination of fields within the dataset.

If you fill both X and Y, you have to define a resolution (or sampling). For Longitude, Latitude a default resolution (1/3 of a degree for both axis), minimum and maximum are proposed. For other X and/or Y, a step of 1 is proposed, but no minimum and maximum. You can define a step, minimum and maximum
values, or use the minimum and maximum value of your expression by clicking on the 'Get min/max expression values' button in the 'Sampling' section (on the bottom). The number of intervals is automatically computed from those elements, and cannot be directly changed.

Note that

- you cannot choose different resolutions for different data expressions within the same operation (they all share the same X and Y!).
- by choosing a step, you may sub-sample your source data.
- Changing the Min/Max can be used to extract a smaller X-Y area (as well as the selection criteria).
- And, of course, the smaller the steps, the higher the computation time! (and the heavier the output file)

You can also choose to smooth and/or extrapolate the data by means of a Loess filter so as to obtain a fully coloured plot (and not individual tracks or points on a map). In that case, you will have to fill in the corresponding information for X and Y, too (see section 4.3.3.4.7, Smoothing).

4.2.4.2.3. Constraints related to field dimensions

The BRAT core functions cannot handle properly the case where X, Y and data expressions have different dimensions, e.g., if Data Expression has three dimensions while X and Y have only two dimensions. If an operation is defined like this, it cannot be executed. A check is done just before the operation is executed and the user is warned if the dimensions of all involved formulas are not homogeneous in what concerns their dimensions.

If the number of dimensions is the same for all fields used in the operation expressions, the operation starts executing, but additional checks are done where the dimensions of all required fields are also compared by name. There is a good reason for this, because it is necessary to distinguish the case of different dimensions that, in spite of having the same size, are applicable to different variables. This will be a safeguard to prevent the user from running unsupported types of operations.

4.2.4.2.4. Selection criteria expression

The Selection criteria expression is used to select data e.g. by date and/or boundaries, etc. and/or for editing it using flag values, thresholds, etc. Logical, relational functions can be used, separated by && ('and'), || ('or') or with !(not'). Only the data fulfilling the whole set of conditions, and not equal to default values, are selected.

The Selection criteria expression can be filled the same way than X, Y and Data expression. There can be only one Selection criteria expression. It is optional; when it is filled the 'Selection criteria' title is bold.

All the fields or combination of fields of the source data can be used. To use a combination of fields, it can be clearer to use a formula (see section 4.3.3.4.3, Formulas).

Note that the selection criteria expression is working only with the basic SI units (i.e. when defining thresholds, you have to put values in e.g. meters, even if the data source field is in mm).

4.2.4.3. Output

To process the defined operation on the whole selected dataset, you have to click on the 'Execute' button. The Logs tab then opens (see section 4.3.5, Logs tab), and you can see the current task(s) being executed (both operations and views), comments during execution (verbose mode) and errors.

The "Schedule Execution" button enables to launch the Operation or an export (see next section) at a scheduled time. The "Launch scheduler" button launches the SCHEDULER, which have to be running in order to have the task executed (NB. the Brat SCHEDULER interface icon gives access to the same interface – see chapter 7 for more details).

Executing an operation builds an output netCDF. The name of this netCDF file is predefined using the name you gave to your operation, and cannot be changed within the GUI. It is stored in the Operation folder within your workspace.
BRAT output netCDF files can be used as source data in a new dataset or used with any other tool reading netCDF.

4.2.4.4. Export

You can choose to export the output data by clicking on the 'Export' button. Several formats are available:

- NetCDF (the same than the automatic one, but you can choose where you want it, and how it is named);
- Ascii - The Ascii export can also be seen (once saved) through a built-in text viewer ('Edit Ascii export' button);
- GeoTiff (if the axis of the operation are longitude and latitude), which also provides a Google Earth KML export format.

The KML/KMZ file contains the following information:

- The GeoTiff image overlaid,
- Along track points coloured as the GeoTiff. In the description of each point you will find:
  - Latitude and Longitude information
  - The Data variable chosen to be exported. In case the data exported is a distance measurement, the different along track points would be also placed with the elevation of the exported data. (Only one variable can be exported each time. If there are more variables placed on the Data folder, the export will not work).
  - Acquisition time, dataset/filetype information.
  - Colour bar relating the values of the variable exported and the corresponding colour.
  - Brat logo overlaid.

4.2.5. Create a view

When you have executed your operation, you may want to have a look at the results in a graphical way. This is usually automatic, after having computed a certain operation. If you want to view a certain operation you had previously computed, you can do so by accessing "Window>Workspace Views" or "Tools>Operation Views" menu, which opens all views of current workspace or operation, respectively. In this way, you will not have to recompute every operation you may want to view. The option at "Window>List" list all view windows that are opened or active.

Figure 8: Dialog shown when "Window>Workspace views“ menu is triggered that allows the selection and visualization of a certain operations
In the current version of Brat you can only visualize data expression(s) from the same Operation previously computed. Therefore it will only make sense to visualize data from a given Operation. You can select the operation you want to visualize in the "Operation" combo box.

The type of available display data has three main categories:

- \( Y=F(X) \), which are basically curve plots
- \( Z=F(X,Y) \), which are the representation of a value (in colours/contours) with respect to two others
- \( Z=F(Lon,Lat) \), i.e. maps

Different view types, will generate different view windows.

![Figure 9: A 'Display' window with one view created. Note the list of available views](image)

Notice that all the data expressions/fields are already made available for you to use under the "Data Options" tab. Here you can customize how to view your data or which data expressions to render.

You can select more than one data expression to be displayed. To handle animated plots configuration change the settings under "Animation" tab. All plot properties are to be changed within the visualisation window (see visualization interface).

### 4.2.5.1. Export

A view can be exported into an image file, either by using the appropriate toolbar button of the view window, or directly from the command line without the need to start a BRAT GUI session. In fact, the BRAT executable can be called with this purpose as a command line utility, by direct invocation in a prompt or by using a script.

For this to be achieved, open a terminal and make cd to the directory where BRAT is installed. Then, call BRAT as in the following example, taken from a Unix console:

```
./brat /home/brat/s3-altb/project/user-data/workspaces/RCCC WaveHeight_lat_2D_J2_226 /home/brat/images/WaveHeight_lat_2D_J2_226-3D.jpg 3
```

The first argument is the path to the root directory of the workspace that contains the operation. The second argument is the name of the display (view) you wish to export as image.
The third argument is the path of the file to export the image to, including the file extension (which will
be used by BRAT to infer the intended image format).
An optional fourth argument can be specified to require a 3D image, if supported by the operation. When
used, this fourth argument should always be the number 3.

4.3. BRAT GUI tabs description

4.3.1. Workspace menu

A 'workspace' is a way of saving your preferences, computations and generally the work done with BRAT.
A workspace contains definitions of:

- **Datasets**, that define the collections of files of the same kind you want to use,
- **Operations**, for reading and/or processing and/or selecting data within a dataset. An operation
  produces an intermediate file (netCDF) and a parameter file. Alternatively, data can be exported,
  in netCDF, Ascii or GeoTiff and KML.
- **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

All these are stored within a folder named from the workspace, with a sub-folder for each part: Datasets,
Displays, Formulas and Operations. The Operations folder includes parameter files (.par), which define
the Operations done, and the later also include the netCDF intermediate files produced by the tool.
Workspace folders can be copied and exchanged. Results saved within a workspace can be accessed even
if the source data are not available (but warning messages will be emitted when opening the workspace if
some source data are not available).
Workspaces in BRAT GUI are managed by the menu the further to the left. It contains the following
items:

- '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 views
- 'Import': imports a previously saved workspace within the current one (Datasets, Operations,
  Formulas and/or Views). Formulas can be imported separately, but otherwise, Views need the
  Operations and Operations need Datasets, so that you can't import Views without Operations and
  Datasets, nor Operations without Datasets.
- 'Rename': renames the current workspace (note that it is not a copy, but a change of name)
- 'Delete': deletes the current workspace
- 'Recent workspaces': lists the most recently used workspaces

4.3.2. Datasets tab

This tab is dedicated to the choice of the source data product files.
In this tab window:

- The selected files’ names are on the left; as well as the tools to select them.
- The bottom display lists all fields defined for this kind of data and, in the middle there is a more
detailed description of the selected field (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 8.3).

4.3.2.1. Creation of a dataset

The 'Dataset Name' is available in the “File Datasets” text box. This 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 “File Datasets” text box, and press the Enter key it renames your dataset.
- The 'New Dataset' button creates a new dataset, with a name like 'Datasets_2'
- The 'Delete...' button enables to delete an existing dataset, if your dataset is not used in an Operation.

4.3.2.2. Management of the data files list

The 'File Datasets' textbox list is organized as a tree like structure that will list all the Datasets included in the current workspace and, when expanded, a certain dataset will list all the files of the dataset. Note that only coherent datasets are possible (i.e. same format, same data product).

- The 'Clear' button will remove the whole list included in selected dataset.
- You can delete the selected file by using the 'Remove...' button.

File names don't have to be the original ones. However, files within a dataset have to be of the same data product (no mixing of e.g. Envisat and Jason-1 GDR data).
• The 'Add files...' button (at the top right 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...', or proceed in several steps. Otherwise, some files names could be truncated, thus leading to an error.

4.3.2.3. Data file information

On the middle left part of the Datasets tab, you can see information about the fields within the source data product. Among the listed dataset properties, the following are available:

• 'Full name': the fully described name in the file structure hierarchy and related to the record.
• 'Name': the short field name
• 'Unit': the unit of the field
• 'Dim': Dimension of the field (number of values in arrays, if the data is stored in an array)

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

Left, under the file list is a 'File description' box, which give the information about the file for netCDF products.

4.3.3. RADS Datasets tab

Apart from its creation, detailed in 4.2.2.1, the RADS dataset concept in BRAT is similar to the concept of the standard datasets in the 'Datasets' tab. This similarity is reflected also in the two dataset tabs. But, because no individual files or folders can be associated with a RADS dataset, this tab does not present the buttons to manage files and folders.

With the exception of these buttons, and of the behaviour of the button to create new datasets, both tabs present the same widgets, in the same places, with the same functions. The only remarkable differences are that no satellite tracks will be displayed when a RADS dataset is selected (otherwise the map would be covered for the most part with superimposed, indistinguishable satellite tracks, which besides inefficient would not be helpful), and that the product description showed in the 'RADS Datasets' tab is the common description of all files of the respective dataset’s mission, not of a single file.

4.3.4. Filters tab

For filtering data files, you can use the Filters tab. When you apply a certain filter to an operation, you will filter out all data that do not match the filtering criteria. A filter is a set of spatial (areas) and/or temporal criteria (absolute or relative time or cycle and pass constraints). The top four buttons allow you to perform basic Filters operations, namely:

• Create a new filter by pressing the "Create Filter" button.
• Rename an existing filter by pressing the "Rename Filter" button.
• Delete an existing filter by pressing the "Delete Filter" button.
• Saving an existing filter by pressing the "Save Filter" button.
To define which areas this filter will contain, you must select/unselect the checkbox of each one. Areas can be organised into regions – regions are mainly area containers and only exist to provide the user with better areas handling (i.e. the lake region contains only lake areas). To edit the region settings, click under “Region Settings” button and a new dialog entitled “Region Settings...” should pop-up. There are three buttons available for use:

- **“Create a Region”** - creates a new region
- **“Rename a Region”** – renames the current region
- **“Delete a Region”** – deletes the current region

The purpose of this Dialog is simply to create a new region – the user can do this simply by selecting the areas to be included in the current (new) region and simply click “close” when the current region is defined.

The list of all available regions is listed within the drop-down menu near the “show all” button. The text fields of the “New Area Selection” display the bounding area for each area within the selected region. The “Show All” button simply lists all the available areas and the user should select only the ones that are to be included into the current filter. There are several ways to define a new area:

- One can simply select a new area by first using the “Selection Tool” button in the “New Area Selection” and then clicking on the “Create Area” button. The newly created area will represent the selected area created with the “Selection Tool” (see section 4.2.3).
- From a KML file using the “Add Area from KML file” button. All layers defined in the KML with non-empty bounding boxes will be listed for the user to select one. Note, however, that only polygon geometry types (Polygon and MultiPolygon) are supported. Also, BRAT does not support rotations, so any values defined in the <rotation> tag of KML LatLonBoxes will be ignored.

The user can also rename a selected area by clicking on the “Rename Area” button. In a similar way, the user can also delete an area by clicking under the “Delete Area” button. Another filtering option is the Start Date/Stop Date – this will only keep the dataset files whose time range is contained within the range [Start Date, Stop Date]. Other filtering option is the Start/Stop Cycle and Pass that should only
keep the dataset files who’s Cycle and Pass is contained in the range [Start Cycle, Stop Cycle]. The relative Start/Stop dates works in a similar manner.

4.3.5. Operations tab
This tab is dedicated to the definition of what kind of computation(s) and/or selection(s) you want to apply on the data.

Building an operation in fact creates a 'parameter' files (.par), which keeps all the information and which is stored in the Workspace Operations folder. Executing an operation uses either the BRATCreateYFX or the BRATCreateZFXY program on this parameter file to generate the output of the operation. The whole process can however be done completely through the GUI.

In this Operations tab window:

- The management of the operations is at the top.
- The data source (datasets and fields available within) are on the left.
- The middle part shows the different Expressions within the current Operation
- The bottom part shows the content of the selected Expression.

You may define as many Operations as you wish.

Note that an Operation must contain at least one X expression, and one Data expression.

![Operations tab, with an operation being built.](image)

4.3.5.1. Manage Operations
Several functions are meant to 'manage' the operations.

- The 'operations' dropdown list contains all the defined operation names. To rename a certain operation you should use the "Rename Operation" button and allows you to select and rename an operation. When renaming an operation, take care that it does not copy it, but it replaces the old one.
The 'Create a new operation' button is used to create a new operation, with a name like 'Operations_2'.

The 'Duplicate selected operation' button enables you to copy an existing operation, and modify it (e.g. change the dataset for another one with the same kind of data at another date, change the selection criteria, etc.).

The 'Delete selected operation' button enables to delete an existing operation, if none of your operation's expression is used in a View.

The 'Execute' button executes the active operation.

The 'Export' button enables to save the BRAT GUI output on either another format (Ascii, GeoTiff and KML) or in netCDF, and under another name wherever you prefer it.

The 'Schedule Execution' button allows you to delay operation execution by scheduling it for a later time.

The 'Brat Scheduler' button launches the SCHEDULER application, which allows you to manage the delayed operations.

The 'Generate Statistics' button gives the global statistics for each Data expression and saves the result in file. You can thus retrieve:

- Number of valid data,
- Mean,
- Standard deviation,
- Minimum,
- Maximum,

If you want to apply the same operation to different datasets, and be able to compare their outputs, you will have to re-create it as many times as needed, using the 'Duplicate selected operation' button. You can also use the parameter file directly with a script (see section 8.3, Using the parameter files to process many datasets). Or, you can export the data in netCDF for future use (otherwise, the output file will be replaced by the new one).

4.3.5.2. Define source data

'Datasets' box lists the names of the datasets available within this workspace: you have to select one of them

'Fields' box shows the list of all fields available within the selected dataset, organised as a tree.

Right-click provides a contextual menu, with 'sort ascending' and 'sort descending' at the bottom, to sort the data field names in alphabetical order (or reverse).

To know some information about one field, hover the mouse pointer over it, and a tooltip will appear.
4.3.5.3. Define expressions

In the middle of the Operations tab is the tree with the expressions, including the selection applied. You have four kinds of expressions:

- **'X':** as axis (the data will be organised relative to the values within this field); only one X expression is possible, and one is necessary.

- **'Y (optional)':** to be used as second axis (e.g. X=longitude, Y=latitude); only one Y expression is possible.

- **'Data':** at least one Data expression is necessary, but you can add up to twenty of them.

- **'Selection criteria' (optional; the title is bold when it is filled):** it enables to select data e.g. by date and/or boundaries, etc. and/or for editing it using flag values, thresholds, etc. Logical, relational functions can be used, separated by && ('and'), || ('or') or with ! ('not'). All the fields, or combination of fields of the source data can be used. To use a combination of fields, it can be clearer to use a formula. Note that the selection criteria expression is working only with the basic SI units (i.e. when defining thresholds, you have to put values in e.g. meters, even if the data source field is in mm).

X and Y are used as axis: BRAT will read the source data and extract, for each X (optionally Y), the corresponding value of each Data expression fulfilling the conditions defined as 'selection criteria'.

All expressions can be filled the same ways.

The expressions can be filled by several means:

- The quickest is by **drag & drop:** drag a field from the leftmost list and drop it in either one of those, or in the 'Expression' box;
Figure 14: Example of menu that appears by right-click on a data expression (‘SLA’). Note that here one data field (‘equator_time’) is selected (left-click); if no data field is selected, this item is inactive.

‘Insert empty expression’ will add a new expression (in ‘Data’), or replace the active expression by an empty one (in X and Y). ‘Insert (field) into expression’ add the selected data field (if any) in the active expression. ‘Insert function’ enable to use the list of mathematical and logical functions, and ‘Insert formula’ insert one of the predefined expressions saved.

‘Sort’ enable to sort all the expressions (if there’s more than one) by their name in either ascending or descending (alphabetical) order.

- or use the ‘Insert field’ button (which will insert the selected data field in the active expression).
- or you can always use the ‘Insert expression’ (which will insert an empty expression, to be filled by one or several combined fields) and type in an empty expression the field names and functions you want to apply, using your keyboard

Since you can do more than insert one field, a set of functions is available, as well as

- The ‘Insert Function’ button opens the pop-up window with the list of available functions (see section 4.3.3.4.2, Functions) for the complete list and specifications)
- The ‘Insert Formula’ button opens the pop-up window with the list of available formulas. A set of those is pre-defined (see section Formulas); more can be saved using ‘Save as Formula’ button and re-used, in the same Workspace or imported in another one
- The ‘Insert Algorithm’ button opens the pop-up window with the list of available algorithms (see section Algorithms) for the complete list and specifications)

4.3.5.4. Expression information and parameters

When an expression is selected, several parameters can be filled/used.

- ’Unit’ of the expression: this text field is filled whenever you define a data field as expression, or use a predefined formula. Default is ‘count’ (meaning, without unit). See section 4.3.3.4.1, Units below for details about the units you can use. The unit of the Selection criteria expression is always ‘count, by default’, since it is a logical expression)

- The ‘Data box: this where the Data field of Data Expressions is defined
- ‘Data Computation’ rolling list
- ‘Check Syntax’ button
- ‘Show info’ button.
- ‘Show aliases’ button

Aliases have been added within BRAT to take into account the fact that the equivalent fields are not named similarly for all the datasets (names following the User documentation made by the data provider). The equivalent fields have been defined with the same alias(es) for all the altimetry data. If a given field is not available within the current dataset, a warning will be issued.
Note that there may be several aliases for a same field, in order to speed the typing (e.g. \%{mss}), or be more self-explaining (e.g. \%{mean_sea_surface}).

An alias can be a field or a combination of fields. They are stored in an “aliases.xml” file that can be edited (in BRAT program folder, data/ sub-folder). In the same folder, the aliases.xsd.html file gives the rules to define new aliases and/or modify the existing ones.

See section Aliases for more information.

![Figure 15: 'Show Aliases' pop-up window. Here for a Jason2 NetCDF file. Note the 'Syntax' column, where the alias syntax is given, while the 'Value' column gives the original field name (or combination).](image)

If you are in an expression (X, Y or Data expression, or Selection criteria) you can insert one or several alias(es) in your expression by checking the box(es) in the 'Select' rightmost column. If no expression is selected, this column won’t appear.

- **'Title/Comment' button**
  This feature is not available in current BRAT version.

- **'Resolution and filter information' and 'Set Resolution / Filter' feature from the older Brat** is now available at ‘Sampling’ section and ‘Smoothing’ button, respectively.

### 4.3.5.4.1. 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 × 1016 electrons/m²).

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 defined in mm)

Every Operation is computed using SI units even if a sub-unit is defined for the data source and for the Expression (e.g. metres instead of cm, mm or km). Thus you can put ‘km’ as unit even if the data source field is defined in mm and still end up with correct values.

4.3.5.4.2. Functions

The 'Insert Function' button provides a simple way of including (and knowing) the available functions and constants which can be used in expressions. The functions are organised by categories, but you can have a look at all of them. For each function, if selected, you will see a short explanation of what it does.

You can use those functions for, among others:

- compute geostrophic velocities modulus: sqrt(sqr(U) + sqr(V))
- a test on a flag: Surface_type == 0 will return only the ‘open ocean’ flagged Jason-1 GDR data
- boundaries: is_bounded(-130, alt_cog_ellip-ku_band_ocean_range ,100) (or: (alt_cog_ellip-ku_band_ocean_range >= -130) && (alt_cog_ellip-ku_band_ocean_range <= 100)) select only the data for which the uncorrected altimetric distance is between -130 and +100 metres

They are available for processing or selecting a data expression.

Basics functions (not listed below) are +, -, *, /, and ( and ); you can also use ‘^’ to indicate a number to the power of another number (or data field or data expression) e.g. ‘10^-6’ means ‘10-6’. Use the keyboard to insert them.

Table 3: BRAT functions

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Syntax</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>logical negation operator NOT</td>
<td>! expr1</td>
<td>Logical</td>
</tr>
<tr>
<td>!=</td>
<td>not-equal-to operator</td>
<td>expr1 != expr2</td>
<td>Relational</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>logical AND operator</td>
<td>expr1 &amp;&amp; expr2</td>
<td>Logical</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>logical OR operator</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>arithmetic expr1 &lt; arithmetic expr2</td>
<td>Logical</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>arithmetic expr1 &lt;= arithmetic expr2</td>
<td>Logical</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Syntax</td>
<td>Type</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------</td>
<td>---------------</td>
</tr>
<tr>
<td>==</td>
<td>equal-to operator</td>
<td>A == B is true (when there is no default in A or B) if abs(A-B) &lt; epsilon. The equal-to operator returns true (1) if both operands have the same value; otherwise, it returns false (0).</td>
<td>==</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>arithmetic expr1 &gt; arithmetic expr2</td>
<td>Relational</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>arithmetic expr1 &gt;= arithmetic expr2</td>
<td>Relational</td>
</tr>
<tr>
<td>~</td>
<td>bitwise not operator</td>
<td>~ expr1</td>
<td>Bitwise operator</td>
</tr>
<tr>
<td>&amp;</td>
<td>bitwise and operator</td>
<td>expr1 &amp; expr2</td>
<td>Bitwise operator</td>
</tr>
<tr>
<td>l</td>
<td>bitwise or operator</td>
<td>expr1</td>
<td>Bitwise operator</td>
</tr>
<tr>
<td>()</td>
<td>parenthesis operator</td>
<td>(expr1)</td>
<td></td>
</tr>
<tr>
<td>DV</td>
<td>Default value</td>
<td>DV</td>
<td>Constant</td>
</tr>
<tr>
<td>PI</td>
<td>PI value</td>
<td>PI</td>
<td>Constant</td>
</tr>
<tr>
<td>PI2</td>
<td>PI/2 value</td>
<td>PI2</td>
<td>Constant</td>
</tr>
<tr>
<td>PI4</td>
<td>PI/4 value</td>
<td>PI4</td>
<td>Constant</td>
</tr>
<tr>
<td>abs</td>
<td>absolute value</td>
<td>abs(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>ceil</td>
<td>ceiling of a value</td>
<td>ceil(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>cos</td>
<td>cosine (radian)</td>
<td>cos(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>cosd</td>
<td>cosine (degree)</td>
<td>cosd(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>deg2rad</td>
<td>Translates Degree to Radian.</td>
<td>deg2rad(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>deg_normalize</td>
<td>Normalizes longitude (degree)</td>
<td>deg_normalize(param1, param2)</td>
<td>geographical</td>
</tr>
<tr>
<td>dv (DV)</td>
<td>Default value</td>
<td>dv</td>
<td>Constant</td>
</tr>
<tr>
<td>exp</td>
<td>exponential</td>
<td>exp(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>floor</td>
<td>floor of a value</td>
<td>floor(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>frac</td>
<td>fractional parts</td>
<td>frac(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>iff</td>
<td>Inline if</td>
<td>iff(param1, param2, param3)</td>
<td>Logical</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Syntax</td>
<td>Type</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>return param2 else return param3 end if</td>
<td>Inline if with default value case 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: if (param1 is default value) return param4 else if (param1 is true) return param2 else return param3 end if</td>
<td>iif3(param1, param2, param3, param4)</td>
<td>Logical</td>
</tr>
<tr>
<td>int</td>
<td>integer parts</td>
<td>int(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>is_bounded</td>
<td>Checks whether a value x is included between two values (min/max).</td>
<td>is_bounded(param1, param2, param3)</td>
<td>Relational</td>
</tr>
<tr>
<td>is_bounded_strict</td>
<td>Checks whether a value x is strictly included between two values (min/max).</td>
<td>is_bounded_strict(param1, param2, param3)</td>
<td>Relational</td>
</tr>
<tr>
<td>is_default</td>
<td>Checks whether a value is a default value (1: yes, 0: no)</td>
<td>is_default(param1)</td>
<td>Logical</td>
</tr>
<tr>
<td>log</td>
<td>logarithm</td>
<td>log(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>log10</td>
<td>base-10 logarithm</td>
<td>log10(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>max</td>
<td>Maximum</td>
<td>max(param1, param2)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>min</td>
<td>Minimum</td>
<td>min(param1, param2)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>mod</td>
<td>floating-point remainder</td>
<td>mod(param1, param2)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>rad2deg</td>
<td>Translates Radian to Degree</td>
<td>rad2deg(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>round</td>
<td>rounded value</td>
<td>round(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>rnd</td>
<td>rounded value</td>
<td>Rnd(param1, param2)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>sign</td>
<td>Checks the sign of a value (-1: negative, 1: positive or zero)</td>
<td>sign(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>sin</td>
<td>sine (radian)</td>
<td>sin(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>sind</td>
<td>sine (degree)</td>
<td>sind(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>sqr</td>
<td>square</td>
<td>sqr(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>sqrt</td>
<td>square root</td>
<td>sqrt(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>tan</td>
<td>tangent (radian)</td>
<td>tan(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>tand</td>
<td>tangent (degree)</td>
<td>tand(param1)</td>
<td>Math&amp;Trigo</td>
</tr>
<tr>
<td>to_date</td>
<td>Date formats conversion</td>
<td>to_date(param1)</td>
<td>Date&amp;Time</td>
</tr>
</tbody>
</table>
### 4.3.5.4.3. Formulas

In the "Insert Formula" button, you will find pre-defined formulas (Sea Surface Height and Sea Level Anomaly formulas from the different satellites’ GDR fields, and also 'Ocean editing' formulas, to use as selection criteria to select only valid data over ocean). If you have saved as formula an expression in the current workspace (or imported one from another workspace), you will also find it here. Any expression, i.e. valid combination of data fields and functions can be saved as formula. You can insert a developed formula and modify it, or use a formula as part of an expression.

The formula will appear either by its name only (if you leave the 'as alias' checked), or complete (if you un-check 'as alias').
Figure 16: The 'Formulas' pop-up window, with the list of available formulas, top (sorted in alphabetical order).

Here, one of them (Ocean_data_editing_GFO_from_cycle_83) is selected, thus you can see the unit of the formula ('count', i.e. no unit, this is a selection formula), and the full formula in the box below. The check-box 'As alias' enables to insert the formula by its name only ('as alias') or, when unchecked, to insert in its full extent.

Figure 17: use of a pre-defined formula (Ocean_data_editing_GFO_from_cycle_83). Note the use in this particular expression of the formula as alias %{Ocean_data_editing_GFO_from_cycle_83}

To apply the inserted formula to the selected Data Expression in the tree item, click the button "Assign to the selected tree item".
4.3.5.4. Algorithms

Algorithms provide means of computing complex operations. They are pre-defined and compiled within BRAT. They include an algorithm name and a number of input parameters (depending on the algorithm) to be filled in by the user. The button "insert algorithm" enable to access the available algorithms with the relevant information provided.

Eleven algorithms are available at this time:

- computation of U (zonal) and of V (meridional) component of geostrophic velocities from gridded data
- computation of cross-track geostrophic velocities from along-track data.
- Filters to apply on along-track data (Gaussian, Median, Lanczos or Loess)
- Filters to apply on gridded data (Gaussian, Median, Lanczos or Loess)

Note that, as in the all of BRAT, computations are done in SI units. If the field(s) you are using have a unit defined, BRAT will take care of the conversion. However, beware if there is no unit really defined ("count"). BRAT will then consider the data as in S.I.

**Table 4: BRAT algorithms**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Input parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>BratAlgoGeosVelAtp</td>
<td>Geostrophic velocity computation for along-track data; result is the value of the geostrophic velocity perpendicular to the track. Input data must contain at least longitude, latitude and a field corresponding to height information.</td>
<td><strong>Latitude:</strong> to be replaced by the name of the latitude field within the data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Longitude:</strong> to be replaced by the name of the longitude field within the data.</td>
</tr>
<tr>
<td>BratAlgoGeosVelGridU</td>
<td>Geostrophic velocity computation for gridded data; result is the value of the geostrophic velocity zonal (North) component, U. Input data must contain at least longitude, latitude and a field corresponding to height information.</td>
<td><strong>Latitude:</strong> to be replaced by the name of the latitude field within the data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Longitude:</strong> to be replaced by the name of the longitude field within the data.</td>
</tr>
<tr>
<td>BratAlgoGeosVelGridV</td>
<td>Geostrophic velocity computation for gridded data; result is the value of the geostrophic velocity meridional (East) component, V. Input data must contain at least longitude, latitude and a field corresponding to height information.</td>
<td><strong>Latitude:</strong> to be replaced by the name of the latitude field within the data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Longitude:</strong> to be replaced by the name of the longitude field within the data.</td>
</tr>
<tr>
<td>BratAlgoFilterGaussianAtp</td>
<td>Gaussian Kernel filter for along-track data. A Gaussian filter is a linear weighted mean filter. Weights in the filter are calculated according to a Gaussian distribution.</td>
<td><strong>Expr:</strong> The input data (variable or Brat expression) on which the filter is applied. <strong>WindowLength:</strong> Window/region size (N). The value must be odd. 1: The standard deviation (sigma) of the distribution. Set by default to 1. The parameter must be a constant value. 3: The coefficient of spreading to the left and right of the distribution.* Set by default</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Input parameters</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BratAlgoFilterGaussianGrid</td>
<td>Gaussian Kernel filter for gridded data.</td>
<td><strong>Expr</strong>: The input data (variable or Brat expression) on which the filter is applied. <strong>WindowLength</strong>: Window/region size (N x N). The value must be odd. 1: The standard deviation (sigma) of the distribution. Set by default to 1. The parameter must be a constant value. 3: The coefficient of spreading to the left and right of the distribution. Set by default to 3. The parameter must be a strictly positive constant value. Usually in practice, the value used is 3 with sigma equals to 1. The part of Gaussian distribution utilized is the range ([-3 \times \text{sigma}, 3 \times \text{sigma})], the Gaussian distribution is truncated at points +/- (3 x sigma). When the range is ([-3 \times \text{sigma}, 3 \times \text{sigma})], the bell-shaped curve adjusts the corner values to 0.01. <strong>ValidPts</strong>: The minimum number of valid points below which the algorithm is not applied. 0: A flag to specify if the algorithm is applied when the current data is 'default value' (no value). 0: not applied, 1: applied.</td>
</tr>
<tr>
<td>BratAlgoFilterLanczosAtp</td>
<td>Lanczos kernel filter for along-track data.</td>
<td><strong>Expr</strong>: The input data (variable or Brat expression) on which the filter is applied. <strong>WindowLength</strong>: Window/region size (N). The value must be odd. <strong>CutOff</strong>: The value of the cut-off period (number of data points). The frequency ((1/\text{CutOff})) is the value at which the response passes from one to zero. <strong>ValidPts</strong>: The minimum number of valid points below which the algorithm is not applied. 0: A flag to specify if the algorithm is applied when the current data is 'default value' (no value). 0: not applied, 1: applied.</td>
</tr>
<tr>
<td>BratAlgoFilterLanczosGrid</td>
<td>Lanczos kernel filter for gridded data.</td>
<td><strong>Expr</strong>: The input data (variable or Brat expression) on which the filter is applied. <strong>WindowLength</strong>: Window/region size (N x N). The value must be odd. <strong>CutOff</strong>: The value of the cut-off period (number of data points). The frequency ((1/\text{CutOff})) is the value at which the response passes from one to zero. <strong>ValidPts</strong>: The minimum number of valid points below which the algorithm is not applied. 0: A flag to specify if the algorithm is applied when the current data is 'default value' (no value). 0: not applied, 1: applied.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td>Input parameters</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>BratAlgoFilterLoessAtp</strong></td>
<td>Loess kernel filter for along-track data.</td>
<td>Expr: The input data (variable or Brat expression) on which the filter is applied</td>
</tr>
<tr>
<td></td>
<td>A Loess filter is a low-pass filter mostly used for smoothing. It is based on</td>
<td>WindowLength: Window/region size. The value must be odd.</td>
</tr>
<tr>
<td></td>
<td>a local regression using weighted linear least squares and a 2nd degree</td>
<td>ValidPts: The minimum number of valid points below which the algorithm is not</td>
</tr>
<tr>
<td></td>
<td>polynomial model</td>
<td>applied.</td>
</tr>
<tr>
<td></td>
<td>X: The input data (X values) used to compute weights.</td>
<td>Extrapolate: A flag to specify if the algorithm is applied when the current data</td>
</tr>
<tr>
<td></td>
<td>WindowLength: Window/region size. The value must be odd.</td>
<td>is 'default value' (no value). 0: not applied, 1: applied.</td>
</tr>
<tr>
<td></td>
<td>ValidPts: The minimum number of valid points below which the algorithm is not</td>
<td>0: A flag to specify if the algorithm is applied when the current data is</td>
</tr>
<tr>
<td></td>
<td>applied.</td>
<td>'default value' (no value). 0: not applied, 1: applied.</td>
</tr>
<tr>
<td></td>
<td>Extrapolate: A flag to specify if the algorithm is applied when the current</td>
<td>0: not applied, 1: applied.</td>
</tr>
<tr>
<td></td>
<td>data is 'default value' (no value). 0: not applied, 1: applied.</td>
<td></td>
</tr>
<tr>
<td><strong>BratAlgoFilterLoessGrid</strong></td>
<td>Loess kernel filter for gridded data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>When used with X=longitude, Y=latitude, it is equivalent to the filter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>available in the 'set resolution/filter' box (but it can be applied here on</td>
<td></td>
</tr>
<tr>
<td></td>
<td>every X and Y)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Loess filter is a low-pass filter mostly used for smoothing. It is based on</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a local regression using weighted linear least squares and a 2nd degree</td>
<td></td>
</tr>
<tr>
<td></td>
<td>polynomial model</td>
<td></td>
</tr>
<tr>
<td><strong>BratAlgoFilterMedianAtp</strong></td>
<td>Median kernel filter for along-track data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Median filter is often used for speckle noise reduction. A median filter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is a non-linear filter which orders the elements within a window and pick</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the middle one.</td>
<td></td>
</tr>
<tr>
<td><strong>BratAlgoFilterMedianGrid</strong></td>
<td>Median kernel filter for gridded data.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>A Median filter is often used for speckle noise reduction. A median filter</td>
<td></td>
</tr>
<tr>
<td></td>
<td>is a non-linear filter which orders the elements within a window and pick</td>
<td></td>
</tr>
<tr>
<td></td>
<td>the middle one.</td>
<td></td>
</tr>
</tbody>
</table>
Figure 18: Insert Algorithm pop-up, with the BratAlgoGeosVelGridV selected.

A list of available algorithms is shown (top)

Description of the selected algorithm is available (just below) as well as the necessary input parameters (middle) and standard output unit (here m/s, bottom). Clicking on "OK" will insert the call to the algorithm within the current expression (it will appear as exec("BratAlgoGeosVelGridV", %{lat}, %{lon}, Height, 5) in the expression box.

You then have to change the four input parameters (or not; most of the time, only "Height" will have to be changed; Latitude and Longitude aliases are used, so they will work for any dataset) to fit your dataset and your needs.

Figure 19: Operation resulting from the insertion of algorithms.
4.3.5.4.5. Data computation

The data computation is used whenever 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

Possible computations 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
- 'STDDEV': 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
- 'TIME': for each grid cell, computes a weighted linear interpolation for distance (X,Y) and time. A dialog box is displayed for the user to specify the product's time field that should be used, an interpolation date and the weightings for distance and time.
Take care, however, that for along-track data, on a given ground track, longitudes or latitudes are scarcely ever exactly the same from one cycle to the next. So if you want to (e.g.) average data over several cycles for a given track with respect to only longitude or latitude, you will have to round the data in the X expression (see round or rnd functions).

4.3.5.4.6. Sampling (previous 'Resolution and filters')

When you fill both X and Y you 'grid' the data, and you then have to define the grid parameters, i.e. minimum, maximum and step, for the whole operation. Note that by choosing a step, you may sub-sample your source data, and that by changing the Min/Max you can extract a smaller X-Y area.

- for longitude/latitude, Minimum and Maximum are set by default to 0 – 360°, -90° – 90° (whatever the data source). For any other type of X and Y, Minimum and maximum have to be defined. The 'Get min/max expression' button is here to help you: if you don't have an idea of what the values of your field could reasonably be, this will provide you with the absolute minimum and maximum of your expression (note that if your dataset include a long list of files, it can take some time to be computed). The unit in which the minimum and maximum have to be defined are those defined in the corresponding expressions, and are recalled, top of each sub-part of the window.

- Pre-defined steps are proposed (1/3° for longitude and latitude, 1 for any other data), but may not fit your need. The number of intervals is automatically computed from those elements, and cannot be directly changed.

However, note that the higher the step, the smaller the resolution, and the longer the execution time for the operation.

![Figure 21: Configuration parameters for output grid data](image)

4.3.5.4.7. Smoothing

BRAT provides you with the possibility of “smoothing”, “binning”, or to extrapolate the data, using Loess filter

There are three different filters:

- 'Smooth': smooth's 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': smooth's and fills in the gap values (with some overlay on continents)

The choice depend on the result you want. '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.
If you select one of them, you have set the ‘Loess cut-off’ value for each axis (both X and Y), i.e. the number of grid points before the Loess filter becomes equal to zero (odd number).

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).

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 gives rather correct results, even if they still show a hint of ground tracks.

4.3.6. Logs tab

The ‘Logs’ tab displays the state of the BRAT GUI or external programs being run by it. Several operations and views can be executed at the same time. Errors can be detected using the messages from the Logs tab.

If things go well, you should have messages like:

‘===> Task 'DisplayDisplays_17' (pid 284) SUCCESSFULLY ENDED <===

======
5. ALIASES

Aliases are short names or unified names for data fields. Aliases have been added within BRAT to take into account the fact that the equivalent fields are not named similarly for all the datasets (names always follow the User documentation made by the data provider, in order that the user can refer to this documentation for more information).

Some are already defined. The equivalent fields have been defined with the same alias(es) for all the relevant altimetry data. If a given field is not available within the current dataset, a warning will be issued. However, you can either modify them, or create your own ones.

A few aliases are "universal" (pre-defined for all known datasets read by BRAT): %{lon}, %{lat}, %{time}

(NB. you may encounter NetCDF data read by BRAT but not pre-defined, for which this won't work, however, in this version we are currently developing ways to solve this problem)

Note that there may be several aliases for a same field, in order to either speed the typing (e.g. %{mss}), or be more self-explaining (e.g. %{mean_sea_surface}).

An alias can be a field or a combination of fields. They are stored in an "aliases.xml" file that can be edited (in BRAT program folder, data/ sub-folder). In the same folder, the aliases.xsd.html file gives the rules to define new aliases and/or modify the existing ones.

The following must be kept in mind:

- an alias always refer to a given data product.
- BRAT GUI call to aliases.xml for alias definition. If you modify this file, the aliases can change! (and, thus, if you used aliases previously, your Operations may not work anymore)

5.1. Using aliases

Aliases can be used as any field or combination of field, by using "%" before the name, and encompassing it between "{" and "}".

For example, a (nearly) universal SSH formula could be written as follow in the 'data expression' of an Operation:

```
%{alt} - %{range} - %{dry_tropo_corr} - %{dynamic_atmos_corr} - %{tides_all_corr} - %{ssb} -
%{iono_corr} - %{wet_tropo_corr}
```

(note that the fact that not all corrections are available for all satellites make it not absolutely universal!)

or, in a "selection criteria" expression, you could write:

```
is_bounded(40, %{lat}, 60)
```

to select data between 40°N and 60°N.

5.2. Structure

Here is an example of the structure of the xml file. For more information on this structure, please refer to aliases.xsd.html file.
Figure 22: Example of the definition of an alias. This example is for Envisat RA2 and MWR products, by default for data within the "ra2_mds" record. "ku_band_ocean_range" is the name given by default in the documentation and thus in BRAT. To keep it simpler, we call it here "range".

See Brat products format definitions in the doc/codadef/index.html file located in the Brat directory.

Products are classified in 'class' (product class) and 'type' (product type)

5.3. Modifying an alias

To modify an alias, edit the xml file in a text editor. And just change its name in <alias name="....">.

For example, you could replace:

alias name="range">ku_band_ocean_range</alias>

by

<alias name="THERANGE">ku_band_ocean_range</alias>

thus, afterwards, you would be using %{THERANGE} as alias. Note that, in this case, previous use of %{range} won't work anymore.

5.4. Creating an alias

5.4.1. For a field for which no alias exists

Find the product(s) for which you want the alias to work, and just add a line like:

<alias name="range">ku_band_ocean_range</alias>

defining the name you wish to use, and the given name of the field. You have to do it for any and every data product where you want to use this alias.

See Brat products format definitions in the doc/codadef/index.html file located in the Brat directory.

Products are classified in 'class' (product class) and 'type' (product type)

You may have to specify a record within the default record.

You will put this in a ProductType tag, like

aliases productType="RA2_MWS_2P" record="avg_waveforms_mds" ref="RA2_GDR_2P">

You can use combination of fields to define an alias. E.g. an alias including all tide-related corrections can be:

"<alias name="tides_all_corr">(ocean_tide_sol1+ solid_earth_tide + pole_tide)</alias> "

5.4.2. For a field for which an alias has already been defined

If you'd prefer something else than the predefined name, but do not want to erase it by modifying it, you can create alternate aliases.
For example, above we decided that when we will be using %{range}, it will be the field "ku_band_ocean_range". However, it can be misunderstood (there's a 'C-Band' range in Envisat data). So you may want to specify at least in some cases that you are using the Ku-band range (e.g. if you're using C-Band data close-by).

To do this, you would define:

```xml
<alias name="range">ku_band_ocean_range</alias>[as previously]
<alias name="range_ku" ref="range">[referring to the above alias]
```

You can then use either %{range} or %{range_ku} in an expression with the same results.
6. VISUALISATION INTERFACE

Within the “Operations tab” you also define which type of view you want to choose to display your data: you can choose between a Map type or a Plot type (simply by clicking on the button with the proper caption). However, to plot your data in a Map you need to have at least the field “Lon”, “Lat” and “Data” set. Otherwise (if you choose the “Plot” type) you are implicitly choosing 2D or 3D representation for your data. Once you do this you can view your data simply by clicking the “Execute” button.

The visualisation options are quite different for a ‘Y=F(X)’ (curve) than for a ‘Z=F(lon,lat)’ (map); the other plots (‘Z=F(X,Y)’) have functionalities from both types.

6.1. 2D Plots

![Figure 23: An example Y=F(X) visualisation with two curves](image)

Usually this type of Plot is used upon displaying a y=f(x) curve type.

BRAT support saving the plots as an image format. The Plot2D dialog is organised in several tabs, each one with a different functionality.
The first tab ‘General’ holds information about the current Operation View associated with this Plot. You can change the current Operation View by changing the contents of the dropdown menu in “Operation Views”. Title outputs the Plot title for the current plot. The Plot type text box allows you to change the visualization type for the current data, but only certain types of Views are available depending on the contents of your current data. The “Reset” button rebuilds the current view.

When a field is selected in the ‘Data Options’ tab, you have some options to choose the colour 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.

The tab, ‘Axis Options’, enables changing values for the axis labels, number of axis ticks and digits, or axis scales for 2D and 3D plots.

The label of each axis includes by default the name of the plotted field and its unit.

“2D scale range” allows you to see the currently selected range. You can also zoom in on a portion of curve using the wheel of your mouse. By pressing the wheel mouse button you can also move the Plot inside the view.

If the variable assigned to the X axis has multiple dimensions, it is possible to select which of those dimensions should primarily be used for that axis in the plot. Note that if the default dimension is changed, the X axis label is not updated; the label is a custom, convenience graphical item, to be updated by the user as is more informative in each particular context.
6.2. Map Plots

This type of plot is one of the three possibilities to display a plot of the type $Z = F(Lon, Lat)$ (the others are spectrograms and 3D Plots). The general tab view allows you to:

- Duplicate the current view by pressing the "Create a new view" buttons;
- Rename the current view by pressing the "Rename the selected view" button;
- Delete the current view by pressing the "delete selected view" button.

Similarly as before, the operation views drop-down menu allows you to choose a view for the currently selected operation (the current operation can be changed in the top drop-down menu "Operation"). The title text box displays the current plot title. The type text box shows you the current plot type. The "Reset" button rebuilds the entire view. The "Distance/Area" button allows you to measure distances or areas over the map:

**Distance**: use the left mouse button to draw points over the map in order to create a multi-line and click once on the right mouse button to stop drawing, or twice to clear the multi-line.

**Area**: use the left mouse button to draw points over the map in order to create a polygon and click once on the right mouse button to stop drawing, or twice to clear the polygon.

In both cases, a window appears showing the metrics of each feature. Click again the "Distance/Area" button to disable the tool return the mouse to its normal state.
Another tab available is the "Data Options" that shows all the current available data layers for the current view and, in addition, allows further visual details configuration.

Figure 27: Map plot type to display a simple $z=f(\text{lon},\text{lat})$ graph type.

Figure 28: The "Data Options" tab.
The contours feature allows the creation of map contours. You can choose the number of contour levels at "Number"; the line width at the "Width" text box and the contours precision at the precision text boxes. The contours detail improves for higher precision numbers; however the processing time also increases. For more information see the reference of the contour algorithm\(^6\).

The following is a globe view example:

![Figure 29: You can also trigger the Globe Plot for this type of data by clicking under the "3D" button.](image)

### 6.3. 3D Plots

The first tab ‘General’ holds information about the current Operation View associated with this Plot. The behaviour is similar to the same tab in other view types: you can change the current Operation View by changing the contents of the dropdown menu in "Operation Views". Title should output the Plot title for the current plot. The Plot type text box allows you to change the visualization type for the current data, but only certain types of Views are available depending on the contents of your current data. For 3D Plot types, one usually has two plotting possibilities: a spectrogram or a 3D graph. One can hide one or the other when clicking the “2D” or “3D” buttons.

Right-clicking on the 3D plot, a context menu pops up that allows the adjustment of some visualization details that can improve data visibility.

---

\(^6\) [http://www.codeproject.com/Articles/1727/A-C-implementation-of-an-improved-contour-plotting](http://www.codeproject.com/Articles/1727/A-C-implementation-of-an-improved-contour-plotting)
Figure 30 – Plotting a $z=f(x,y)$ graph.

Figure 31 – Same plot but with a hidden spectrogram plot by clicking under the 2D button.

The "Axis Options" tab works in a similar manner as for the 2D case. The ‘Animation’ tab should only address graphs that have animation, and implements features to stop and start the animation and define
the number of frames to be used. Currently, animations are only available for 2D plots, depending on the dimensions of the plotted variable.

6.4. Colour tables

Several pre-defined colour tables are available in the 'Data Options' tab for both maps and 3D plots. They can improve the visibility of the displayed data, or adjust the colour spectrum so that it contrasts better with the background. This option can be useful in particular when maps are using raster layers with colour patterns that blur the distinction of data and background.

Approximate data values corresponding to the sharper thresholds of the colour tables are also displayed under the colours spectrum widget. If you move the mouse over each value, a more precise value is displayed.

It is possible to select different colour tables for the different fields, as well as hide the data of the field currently selected in the 'Fields' table by clearing the 'Show' check box near the colour table.

6.5. Vector Plots

Vector plots are displayed when fields from the visualization tab are selected as East and North vector components. Both components have to be present, otherwise an error message will be issued.

East/North Component can be selected in 'Map View proprieties' button at 'Operations' tab. One expression has to be selected as north component and a different one for east component. Only one vector plot can be displayed at a time. Both expressions must be of the same data type.

Vectors are naturally visualized as arrows. The magnitude values available in the data are displayed when the user sets the mouse over an arrow. To improve the arrows visibility, you may need to adjust their scale by using the 'Vector Scale' field in the 'Data Options' tab, which is visible only when a Vector Plot is displayed: press the return key after entering a new value to see how it impacts the arrow dimensions in the map.
7. BRAT SCHEDULER INTERFACE

The BRAT SCHEDULER interface enables to postpone the execution of operations. Each delayed execution has to be configured through BRAT GUI (by clicking on the 'Schedule Execution' button).

The SCHEDULER application can be launched either from BRAT GUI or from the desktop icon. It will allow you the management and monitoring of the delayed operation executions. If it is not running, no scheduled task will be processed.
8. USING BRAT IN ‘COMMAND LINES’ MODE WITH PARAMETERS FILE

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

BratCreateYFX.exe create an output netCDF with one or several data field(s) with respect to a single field
BratCreateZFXY.exe create an output netCDF with one or several data field(s) with respect to two different fields (e.g. longitude, latitude)

brat.exe can be used as the older "BratDisplay.exe" from the previous versions of Brat in order to just display a *.par file. In order to activate this mode one only needs to execute “brat.exe <par_path>” where <par_path> is a simple directory that points to your .par file. (i.e. brat.exe C:\projects\workspaces\UserJohn\Displays\DisplayDisplays_25.par).

BratExportAscii.exe export an output to Ascii
BratExportGeoTiff.exe export gridded data from a netCDF product to GeoTiff (with optional GoogleEarth wrapper)
BratListFieldNames.exe
BratShowInternalFile.exe
BratStats.exe

8.1. 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.
```plaintext
#---- 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_lon_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
```

Figure 32: 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.

8.2. 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 type of data to be displayed (Y=F(X) ==> 0  Z=F(Lat,Lon) ==> 2  Z=F(X,Y) ==> 1)
- 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, colour table...)

```
#!/usr/bin/env BratCreateZFXY
#Type:Z=F(X,Y)
#----- DATASET ----- 
FILE=Createenvisat_cycle.nc
#----- GENERAL PROPERTIES ----- 
DISPLAY_TITLE=title of the plot
DISPLAY_PLOT_TYPE=1
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
```

*Figure 33: Example ‘display’ parameter file*

You open the visualisation tool by typing:

`'BratDisplay.exe command_file.par'`
8.3. 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;

```bash
#!/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_soll + ocean_tide_equil + load_tide_soll) - solid_earth_tide - pole_tide - sea_state_bias_ku - ionic_corr_alt_ku - rad_wet_tropo_corr - mss

X = longitude
X_TYPE = long
X_NAME = Longitude
X_UNIT = DV
X_MIN = DV
X_MAX = DV
X_INTERVALS = 1800

Y = latitude
Y_TYPE = latitude
Y_NAME = Latitude
Y_UNIT = DV
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}
```

*Figure 34: 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
BratCreateZFXY C:\BRAT\MyCmdPath\BratCreateZFXYJasonSLASample.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
BratCreateZFXY C:\BRAT\MyCmdPath\BratCreateZFXYJasonSLASample.par

Figure 35: An example script for DOS (to be inserted in a .bat file) to launch a parameter file over several cycles

#!/bin/bash
# BratCreateZFXYJasonSLASample.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
BratCreateZFXY BRAT/MyCmdPath/BratCreateZFXYJasonSLASample.par

# -------------------------------

# Set the cycle number
export CYCLE=110

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

# Launch 'BRAT create Z=F(X,Y)' process
BratCreateZFXY BRAT/MyCmdPath/BratCreateZFXYJasonSLASample.par

Figure 36: An example Shell script for Linux for launching a parameter file over several cycles
9. BRATHL APPLICATION PROGRAMMING INTERFACES (APIS)

Some functions of BRAT are not available through the GUI, but through C, Fortran, Python, IDL and MATLAB APIs. Note that for IDL and MATLAB under Linux and Mac OS you need to compile the API before being able to use them – they are not included in the binary distributions of BRAT.

9.1. 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/m3 (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.
9.2. 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
- for Python

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

9.3. Date conversion/computation function

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
• for Python

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
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 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

9.4. 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
- for Python

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

BRATHLDATEDSM        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 :BRATHLDATEDSM  
NBSEC seconds.milliseconds  

**BRATHL_DATEJULIAN**  
decimal Julian date structure  
REFDATE reference date - see :BRATHLDATEDSM  
JULIAN decimal Julian day
10. ANNEX A: LIST OF DATASETS READ BY BRAT

10.1. Cryosat product overview

Table 5: 10.1. Cryosat product overview

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIR_LRM_1B</td>
<td>SIRAL L1B LRM product</td>
</tr>
<tr>
<td>SIR_SAR_1B</td>
<td>SIRAL L1B SAR mode product</td>
</tr>
<tr>
<td>SIR_SIN_1B</td>
<td>SIRAL L1B SARin mode product</td>
</tr>
<tr>
<td>SIR1LRM_0M</td>
<td>SIRAL MON-LRM/TRK product (Rx1 channel)</td>
</tr>
<tr>
<td>SIR2LRM_0M</td>
<td>SIRAL MON-LRM/TRK product (Rx2 channel)</td>
</tr>
<tr>
<td>SIR1SAR_0M</td>
<td>SIRAL MON-SAR product (Rx1 channel)</td>
</tr>
<tr>
<td>SIR2SAR_0M</td>
<td>SIRAL MON-SAR product (Rx2 channel)</td>
</tr>
<tr>
<td>SIR_SIN_0M</td>
<td>SIRAL MON-SARin product</td>
</tr>
<tr>
<td>SIR_SIC40M</td>
<td>SIRAL MON-CAL4 product</td>
</tr>
<tr>
<td>SIR1LRC11B</td>
<td>SIRAL CAL1-LRM product (Rx1 channel)</td>
</tr>
<tr>
<td>SIR2LRC11B</td>
<td>SIRAL CAL1-LRM product (Rx2 channel)</td>
</tr>
<tr>
<td>SIR1SAC11B</td>
<td>SIRAL CAL1-SAR product (Rx1 channel)</td>
</tr>
<tr>
<td>SIR2SAC11B</td>
<td>SIRAL CAL1-SAR product (Rx2 channel)</td>
</tr>
<tr>
<td>SIR_SIC11B</td>
<td>SIRAL CAL1-SARin product</td>
</tr>
<tr>
<td>SIR_SICC1B</td>
<td>SIRAL complex CAL1-SARin product</td>
</tr>
<tr>
<td>SIR1SAC21B</td>
<td>SIRAL CAL2-SAR product (Rx1 channel)</td>
</tr>
<tr>
<td>SIR2SAC21B</td>
<td>SIRAL CAL2-SAR product (Rx2 channel)</td>
</tr>
<tr>
<td>SIR1SIC21B</td>
<td>SIRAL CAL2-SARin product (Rx1 channel)</td>
</tr>
<tr>
<td>SIR2SIC21B</td>
<td>SIRAL CAL2-SARin product (Rx2 channel)</td>
</tr>
<tr>
<td>SIR_LRM_2</td>
<td>SIRAL L2 product from LRM processing</td>
</tr>
<tr>
<td>SIR_FDM_2</td>
<td>SIRAL L2 product from fast delivery ocean processing</td>
</tr>
<tr>
<td>SIR_SIN_2</td>
<td>SIRAL L2 product from SARin processing</td>
</tr>
<tr>
<td>SIR_SID_2</td>
<td>SIRAL L2 product from SARin degraded processing</td>
</tr>
<tr>
<td>SIR_SAR_2A</td>
<td>SIRAL L2 product from SAR step 1 processing</td>
</tr>
<tr>
<td>SIR_SAR_2B</td>
<td>SIRAL L2 product from SAR step 2 processing</td>
</tr>
<tr>
<td>SIR_GDR_2A</td>
<td>SIRAL L2 consolidated product including SAR step 1 data (SIR_SAR_2A)</td>
</tr>
<tr>
<td>SIR_GDR_2B</td>
<td>SIRAL L2 consolidated product including SAR step 2 data (SIR_SAR_2B)</td>
</tr>
<tr>
<td>SIR_LRM2_</td>
<td>SIRAL intermediate L2 product from LRM processing</td>
</tr>
<tr>
<td>SIR_FDM2_</td>
<td>SIRAL intermediate L2 product from fast delivery ocean processing</td>
</tr>
<tr>
<td>SIR_SIN2_</td>
<td>SIRAL intermediate L2 product from SARin processing</td>
</tr>
<tr>
<td>SIR_SID2_</td>
<td>SIRAL intermediate L2 product from SARin degraded processing</td>
</tr>
<tr>
<td>SIR_SAR12A</td>
<td>SIRAL intermediate L2 product from SAR step 1 processing</td>
</tr>
<tr>
<td>SIR_SAR12B</td>
<td>SIRAL intermediate L2 product from SAR step 2 processing</td>
</tr>
</tbody>
</table>

10.2. Cryosat Ocean products overview

Table 6: Cryosat Ocean products overview

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIR_IOP_1B</td>
<td>Interim L1B Ocean Product</td>
</tr>
<tr>
<td>SIR_GOP_1B</td>
<td>Geophysical L1B Ocean Product</td>
</tr>
<tr>
<td>SIR_IOP_2</td>
<td>Interim L2 Ocean Product</td>
</tr>
</tbody>
</table>
10.3. Jason-2 product overview

Table 7: Jason-2 product overview

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA2_OPR_2P</td>
<td>The Operational Geophysical Data Record (OGDR), produced on a NRT basis</td>
</tr>
<tr>
<td>JA2_OPR_2P</td>
<td>The reduced Operational Geophysical Data Record (SSHA-OGDR), produced on a NRT basis</td>
</tr>
<tr>
<td>JA2_IPN_2P</td>
<td>The Interim Geophysical Data Record (IGDR)</td>
</tr>
<tr>
<td>JA2_IPR_2P</td>
<td>The reduced Interim Geophysical Data Record (SSHA-IGDR), produced on a NRT basis</td>
</tr>
<tr>
<td>JA2_IPS_2P</td>
<td>The Sensor Interim Geophysical Data Record (SIGDR)</td>
</tr>
<tr>
<td>JA2_GPN_2P</td>
<td>The Geophysical Data Record (GDR)</td>
</tr>
<tr>
<td>JA2_GPR_2P</td>
<td>The reduced Geophysical Data Record (SSHA-GDR), produced on a NRT basis</td>
</tr>
<tr>
<td>JA2_GPS_2P</td>
<td>The Sensor Geophysical Data Record (SGDR)</td>
</tr>
</tbody>
</table>

10.4. Envisat product overview

Table 8: Envisat product overview

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RA2_FGD_2P</td>
<td>RA-2 Fast Delivery Geophysical Data Record</td>
</tr>
<tr>
<td>RA2_GDR_2P</td>
<td>RA-2 Geophysical Data Record</td>
</tr>
<tr>
<td>RA2_MWS_2P</td>
<td>RA-2 Sensor Data Record</td>
</tr>
<tr>
<td>RA2_WWV_2P</td>
<td>RA-2 wind/wave product for Meteo Users</td>
</tr>
</tbody>
</table>

10.5. Jason-1 product overview

Table 9: Jason-1 product overview

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JA1_OSD_2P</td>
<td>The Operational Sensor Data Record (OSDR), produced on a NRT basis</td>
</tr>
<tr>
<td>JA1_IGD_2P</td>
<td>The Interim Geophysical Data Record (IGDR)</td>
</tr>
<tr>
<td>JA1_GDR_2P</td>
<td>The Geophysical Data Record (GDR)</td>
</tr>
<tr>
<td>JA1_SDR_2P</td>
<td>The Sensor Geophysical Data Record (SGDR)</td>
</tr>
</tbody>
</table>

10.6. Topex/Poseidon product overview

Table 10: Topex/Poseidon radar altimetry products

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MGDR_cycle_header_File</td>
<td>Merged GDR Topex/Poseidon cycle header file</td>
</tr>
<tr>
<td>MGDR_pass_file</td>
<td>Merged GDR Topex/Poseidon pass file</td>
</tr>
<tr>
<td>MGDR_crossover_point_file</td>
<td>Merged GDR Topex/Poseidon crossover point file (XNG)</td>
</tr>
<tr>
<td>SDR_pass_file</td>
<td>SDR Topex/Poseidon pass file</td>
</tr>
</tbody>
</table>
10.7. ERS-1 and 2 product overview

*Table 11: ERS-1 and ERS-2 radar altimetry products*

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPR_pass_file</td>
<td>Same as the off-line intermediate product but enhanced with all geophysical corrections and precise orbit altitude.</td>
</tr>
<tr>
<td>URA</td>
<td>Radar Altimeter Fast delivery</td>
</tr>
<tr>
<td>WAP</td>
<td>Radar Altimeter Waveform product</td>
</tr>
</tbody>
</table>

10.8. GFO product overview

*Table 12: GFO product overview*

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GDR</td>
<td>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.</td>
</tr>
</tbody>
</table>

10.9. PODAAC product overview

*Table 13: Physical Oceanography Distributed Active Archive Center radar altimetry products for Jason-1 and Topex/Poseidon*

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1SSHA_CYCLE_HEADER_FILE</td>
<td>The PODAAC JASON-1 SSHA cycle header file</td>
</tr>
<tr>
<td>TPSSHA_CYCLE_HEADER_FILE</td>
<td>The PODAAC TOPEX/POSEIDON SSHA cycle header file</td>
</tr>
<tr>
<td>J1SSHA_PASS_FILE</td>
<td>The PODAAC JASON-1 SSHA pass file</td>
</tr>
<tr>
<td>TPSSHA_PASS_FILE</td>
<td>The PODAAC TOPEX/POSEIDON SSHA pass file</td>
</tr>
<tr>
<td>J1SSHA_ATG_FILE</td>
<td>The PODAAC JASON-1 Along Track Gridded SSHA file</td>
</tr>
<tr>
<td>TPSSHA_ATG_FILE</td>
<td>The PODAAC TOPEX/POSEIDON Along Track Gridded SSHA file</td>
</tr>
</tbody>
</table>

10.10. River and Lake product overview

*Table 14: ENVISAT-ERS Exploitation River and Lake Products*

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLH</td>
<td>River/Lake Hydrology Product</td>
</tr>
<tr>
<td>RLA</td>
<td>River/Lake Altimetry Product</td>
</tr>
</tbody>
</table>

10.11. NetCDF products

NetCDF products are self-describing products. This means that when a netCDF 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 netCDF product format descriptions in this documentation). What BRAT will do is use the underlying netCDF library to retrieve the product format dynamically once a netCDF 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:

Access to netCDF attributes
Identify default/missing values (see _FillValue standard attribute)
Convert data to its actual value (not the value stored in file): see scale_factor and add_offset standard attributes.
Interpret the structure of file to compute actual values of data (and not solely returning the netCDF variables values ‘as is’).
Avoid making available variables belonging to data structure (which are not the data themselves)

### 10.11.1. Aviso Altimetry data in netCDF

**Table 15: Aviso Altimetry data in netCDF**

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRT- or DT-MSLA (h)</td>
<td>Ssalto/Duacs multimission Near real-time or Delayed time Maps of sea level anomalies (gridded)</td>
</tr>
<tr>
<td>NRT- or DT-MSLA (uv)</td>
<td>Ssalto/Duacs multimission Near real-time or Delayed time Geostrophic velocities associated to the Maps of sea level anomalies (gridded)</td>
</tr>
<tr>
<td>NRT- or DT-MSLA (err)</td>
<td>Ssalto/Duacs multimission Near real-time or Delayed time Maps of sea level anomalies Formal mapping error (gridded)</td>
</tr>
<tr>
<td>NRT- or DT-SLA</td>
<td>Ssalto/Duacs multimission Near real-time or Delayed time Sea level anomalies (along-track)</td>
</tr>
<tr>
<td>NRT- or DT-NRT- or DT-MADT (h)</td>
<td>Ssalto/Duacs multimission Near real-time or Delayed time Maps of absolute dynamic topography (gridded)</td>
</tr>
<tr>
<td>NRT- or DT-MADT (uv)</td>
<td>Ssalto/Duacs multimission Near real-time or Delayed time Geostrophic velocities associated to the Maps of absolute dynamic topography (gridded)</td>
</tr>
<tr>
<td>NRT- or DT-ADT</td>
<td>Ssalto/Duacs multimission Near real-time or Delayed time Absolute dynamic topography (along-track)</td>
</tr>
<tr>
<td>Monomission DT-SLA</td>
<td>Delayed time Sea level anomalies (along-track)</td>
</tr>
<tr>
<td>Monomission DT-CorSSH</td>
<td>Delayed time Corrected sea surface height (along-track)</td>
</tr>
<tr>
<td>NRT-MSWH</td>
<td>Near real-time Maps of Significant wave height (gridded)</td>
</tr>
<tr>
<td>NRT-MWind</td>
<td>Near real-time Maps of Wind speed modulus (gridded)</td>
</tr>
</tbody>
</table>

### 10.11.2. ERS REAPER data in netCDF

**Table 16: ERS REAPER data in netCDF**

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERS_ALT_2_</td>
<td>REAPER L2 GDR Product</td>
</tr>
<tr>
<td>ERS_ALT_2S</td>
<td>REAPER L2 SGDR Product (GDR with echo waveforms)</td>
</tr>
<tr>
<td>ERS_ALT_2M</td>
<td>REAPER L2 Meteo Product (reduced 1Hz meteo product)</td>
</tr>
</tbody>
</table>
10.11.3. Sentinel 3 data in netCDF

Table 17: Sentinel 3 data in netCDF

<table>
<thead>
<tr>
<th>product type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SR_1_SRA___</td>
<td>Echoes parameters for LRM, PLRM and SAR mode (resolution 20Hz)</td>
</tr>
<tr>
<td>SR_1_CAL___</td>
<td>Calibration parameters for LRM and SAR mode</td>
</tr>
<tr>
<td>SR_2_LAN___</td>
<td>1-Hz and 20-Hz Ku and C bands parameters (LRM/SAR/PLRM), waveforms. Over Land</td>
</tr>
<tr>
<td>SR_2_WAT___</td>
<td>1-Hz and 20-Hz Ku and C bands parameters (LRM/SAR/PLRM), waveforms. Over Water</td>
</tr>
</tbody>
</table>
11. ANNEX B: Y=F(X) PARAMETER FILE KEYS

NOTE: The following table of parameter file keyword help can be always be obtained by calling: “BratCreateYFX -k”.

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Type</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Str</td>
<td>[1-n]</td>
<td>Input file name.</td>
</tr>
<tr>
<td>RECORD</td>
<td>Str</td>
<td>1</td>
<td>Record set name to take into account for a file.</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Str</td>
<td>1</td>
<td>Name of created/modified file.</td>
</tr>
<tr>
<td>OUTPUT_TITLE</td>
<td>Str</td>
<td>[0-1]</td>
<td>Title of created/modified file (string describing the content and which should appear as a graphic title, for example). (Default=”&quot;&quot;)</td>
</tr>
<tr>
<td>SELECT</td>
<td>Expr</td>
<td>[0-n]</td>
<td>True for record values selected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Default=1)</td>
</tr>
<tr>
<td>FIELD</td>
<td>Expr</td>
<td>[1-20] X</td>
<td>Expression of fields of <em>RECORD</em> to take into account.</td>
</tr>
<tr>
<td>FIELD_NAME</td>
<td>Name</td>
<td>X</td>
<td>Name of the <em>FIELD</em> data</td>
</tr>
<tr>
<td>FIELD_TYPE</td>
<td>KW1</td>
<td>X</td>
<td>Type of <em>FIELD</em> data.</td>
</tr>
<tr>
<td>FIELD_UNIT</td>
<td>Unit</td>
<td>X</td>
<td>Unit of <em>FIELD</em> expression.</td>
</tr>
<tr>
<td>FIELD_TITLE</td>
<td>Str</td>
<td>X</td>
<td>Long name describing <em>FIELD</em>. The one which should appear in graphics on axis or legends, for example.</td>
</tr>
</tbody>
</table>
### 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
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Expr</strong></td>
<td>Combination of fields of the current record. An expression which can contain function calls like trigonometric, conversion, test...</td>
</tr>
<tr>
<td><strong>Str</strong></td>
<td>String. Leading and trailing blanks are ignored.</td>
</tr>
<tr>
<td><strong>Unit</strong></td>
<td>Unit string conforming to Udunits package and the special keyword 'DATE' which means that the data is a date.</td>
</tr>
<tr>
<td><strong>KW1</strong></td>
<td>Keywords: X/Y/Z/T/Latitude/Longitude/Data</td>
</tr>
<tr>
<td><strong>KW2</strong></td>
<td>Keywords: FIRST/LAST/MIN/MAX/Mean/STDDEV/COUNT</td>
</tr>
</tbody>
</table>
12. ANNEX C: Z=F(X,Y) PARAMETER FILE KEYS

NOTE: The following table of parameter file keyword help can be always be obtained by calling: “BratCreateZFXY -k”

<table>
<thead>
<tr>
<th>KEYWORD</th>
<th>Type</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Str</td>
<td>[1-n]</td>
<td>Input file name.</td>
</tr>
<tr>
<td>OUTPUT</td>
<td>Str</td>
<td>1</td>
<td>Name of created/modified file.</td>
</tr>
<tr>
<td>OUTPUT_TITLE</td>
<td>Str</td>
<td>[0-1]</td>
<td>Title of created/modified file (string describing the content and which should appear as a graphic title, for example). (Default=”&quot;)</td>
</tr>
<tr>
<td>SELECT</td>
<td>Expr</td>
<td>[0-n]</td>
<td>True for record values selected.</td>
</tr>
<tr>
<td>RECORD</td>
<td>Str</td>
<td>1</td>
<td>Record set name to take into account for a file.</td>
</tr>
<tr>
<td>DATA_MODE</td>
<td>KW2</td>
<td>[0-1]</td>
<td>Keyword to indicate how data are stored/computed.</td>
</tr>
<tr>
<td>POSITION_MODE</td>
<td>KW3</td>
<td>[0-1]</td>
<td>How position is computed.</td>
</tr>
<tr>
<td>OUTSIDE_MODE</td>
<td>KW4</td>
<td>[0-1]</td>
<td>How data outside limits are managed.</td>
</tr>
<tr>
<td>X</td>
<td>Expr</td>
<td>1</td>
<td>Expression of fields of <em>RECORD</em> to take into account.</td>
</tr>
<tr>
<td>X_NAME</td>
<td>Name</td>
<td>1</td>
<td>Name of the <em>X</em> data</td>
</tr>
</tbody>
</table>
**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 \(\%\{\text{NAME}\}\) construct. Names are case sensitive. If a name reference \(\%\{\text{XXX}\}\) 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==

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)
### 13. ANNEX D: DISPLAY PARAMETER FILE KEYS

NOTE: The following table of parameter file keyword help can be always be obtained by calling: “BratDisplay -k”.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Str</td>
<td>[1-n]</td>
<td>Input file name.</td>
</tr>
<tr>
<td>FIELD</td>
<td>Expr</td>
<td>[1-23]=X</td>
<td>Expression of fields of <em>RECORD</em> to take into account.</td>
</tr>
<tr>
<td>FIELD_GROUP</td>
<td>Int</td>
<td>X</td>
<td>Group id from where belongs <em>FIELD</em>. generally used to group many fields in one plot.</td>
</tr>
<tr>
<td>DISPLAY_PROPERTIES</td>
<td>Bool</td>
<td>[0-1]</td>
<td>Indicates if property panel is shown. (Default=No)</td>
</tr>
<tr>
<td>DISPLAY_TITLE</td>
<td>Str</td>
<td>[0-1]</td>
<td>Title of the plot to be displayed. (Default=&quot;&quot;)</td>
</tr>
<tr>
<td>DISPLAY_ANIMATIONBAR</td>
<td>Bool</td>
<td>[0-1]</td>
<td>Keyword to indicate if property panel is shown. (Default=No)</td>
</tr>
<tr>
<td>DISPLAY_COLORBAR</td>
<td>Bool</td>
<td>[0-1]</td>
<td>Keyword to indicate if colour bar (legend) is shown. (Default=Yes)</td>
</tr>
<tr>
<td>DISPLAY_CENTERLAT</td>
<td>Flt</td>
<td>[0-1]</td>
<td>Latitude of the projection’s centre point. (Default=0)</td>
</tr>
<tr>
<td>DISPLAY_CENTERLON</td>
<td>Flt</td>
<td>[0-1]</td>
<td>Longitude of the projection’s centre point. (Default=0)</td>
</tr>
</tbody>
</table>
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 colour value map image:  
1.0 colour is totally opaque  
0.0 is completely transparent.  
(Default=0.7)

DISPLAY_MINVALUE  
Type : Flt  Count : 0 or W  
Minimum colour table value to use in plot.  
(Default=min of data values)

DISPLAY_MAXVALUE  
Type : Flt  Count : 0 or W  
Maximum colour table value to use in plot.  
(Default=max of data values)
<table>
<thead>
<tr>
<th>Setting</th>
<th>Type</th>
<th>Count</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISPLAY_NUMCOLORLABELS</td>
<td>Int</td>
<td>0 or W</td>
<td>2</td>
<td>Number of labels shown on the plot’s colour bar. (Default=2)</td>
</tr>
<tr>
<td>DISPLAY_COLORTABLE</td>
<td>Str</td>
<td>0 or W</td>
<td>Aerosol</td>
<td>Name of a predefined colour table:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Blackbody</td>
<td>Aerosol</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>BlackToWhite</td>
<td>Blackbody</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cloud</td>
<td>BlackToWhite</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ozone</td>
<td>Cloud</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>GreenToRed</td>
<td>Ozone</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Rainbow</td>
<td>GreenToRed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RedToGreen</td>
<td>Rainbow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WhiteToBlack</td>
<td>RedToGreen</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>or name of a file containing the colour table definition (absolute or relative path). (Default=Aerosol)</td>
</tr>
<tr>
<td>DISPLAY_COLORCURVE</td>
<td>KW5</td>
<td>0 or W</td>
<td>Linear</td>
<td>Set the colour table on a specific curve. (Default=Linear)</td>
</tr>
<tr>
<td>DISPLAY_CONTOUR</td>
<td>Bool</td>
<td>0 or W</td>
<td>No</td>
<td>Indicates if the contour layer of the field is shown or not. (Default=No)</td>
</tr>
<tr>
<td>DISPLAY_CONTOUR_NUMBER</td>
<td>Int</td>
<td>0 or W</td>
<td>5</td>
<td>Number of contour lines to generate (equally spaced contour values between specified range See DISPLAY_CONTOUR_MINVALUE and DISPLAY_CONTOUR_MAXVALUE). (Default=5)</td>
</tr>
<tr>
<td>DISPLAY_CONTOUR_LABEL</td>
<td>Bool</td>
<td>0 or W</td>
<td>No</td>
<td>Indicate if the contour labels (value) are shown or not. (Default=No)</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Type</td>
<td>Count</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_CONTOUR_LABEL_NUMBER</td>
<td>Int</td>
<td>0 or W</td>
<td>Number of labels on each contour. Default = 1</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_CONTOUR_MINVALUE</td>
<td>Flt</td>
<td>0 or W</td>
<td>Minimum value to use to contour calculation. Default values are the same as the colour scale one.</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_CONTOUR_MAXVALUE</td>
<td>Flt</td>
<td>0 or W</td>
<td>Maximum value to use to contour calculation. Default values are the same as the colour scale one.</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_SOLID_COLOR</td>
<td>Bool</td>
<td>0 or W</td>
<td>Indicates if the colour layer of the field is shown or not. Default = Yes</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_EAST_COMPONENT</td>
<td>Bool</td>
<td>0 or W</td>
<td>Indicates if this field is the East component of a vector plot. Default = No</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_NORTH_COMPONENT</td>
<td>Bool</td>
<td>0 or W</td>
<td>Indicates if this field is the North component of a vector plot. Default = No</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_COLOR</td>
<td>KW7</td>
<td>0 or W</td>
<td>Colour name of the XY plot field. Default = rainbow colour</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_POINTS</td>
<td>Bool</td>
<td>0 or W</td>
<td>Indicates if points are displayed in a XY plot (for the field). Default = No</td>
<td></td>
</tr>
<tr>
<td>DISPLAY_LINES</td>
<td>Bool</td>
<td>0 or W</td>
<td>Indicates if line is displayed in a XY plot (for the field). Default = Yes</td>
<td></td>
</tr>
</tbody>
</table>
**DISPLAY_POINTSIZE**

Type : Flt  
Count : 0 or W  
Size of the points (XY plot, for the field).  
(Default=1.0)

**DISPLAY_LINEWIDTH**

Type : Flt  
Count : 0 or W  
Width of the line (XY plot, for the field).  
(Default=0.8)

**DISPLAY_STIPPLEPATTERN**

Type : KW10  
Count : 0 or W  
Stipple pattern for the line (field) (XY plot).  
(Default=Full)

**DISPLAY_POINTGLYPH**

Type : KW8  
Count : 0 or W  
Glyph of the points (field) (XY plot).  
(Default=Circle)

**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/OUJ/0/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
<table>
<thead>
<tr>
<th>Keyword</th>
<th>Keywords</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KW9</strong></td>
<td>3D, Azimuthal Equidistant, Lambert Cylindrical, Lambert Azimuthal, Mercator, Mollweide, Plate-Caree, Robinson</td>
</tr>
<tr>
<td><strong>KW10</strong></td>
<td>DASHTINY, DASH, DASHDOT, DOT, FULL</td>
</tr>
</tbody>
</table>
14. ANNEX E: BRATHL-MATLAB API

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

==strukturen ==

BRATHL_DATEYMDHMSM = 0
BRATHLDATEDSM = 1
BRATHL_DATESECOND = 2
BRATHL_DATEJULIAN = 3

To create a structure, use BRATHL_CREATESTRUCT (see description below)

BRATHL_DATEYMDHMSM structure

This structure represents an 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
BRATHLDATEDSM 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 dates of his choice (see BRATHL_SETREFUSER1 and BRATHL_SETREFUSER2 functions).

Example:

```c
MyDate = BRATHL_CREATESTRUCT(1)
```

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

BRATHLDATESECONDS structure
------------------------------------
This structure represents a decimal seconds date structure:

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

Example:

```c
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 :BRATHLDATEDSM
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_DIFYMDHMSM

BRATHL_DSM2JULIAN
BRATHL_DSM2SECONDS
BRATHL_DSM2YMDHMSM
BROADVIEW RADAR ALTIMETRY TOOLBOX

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 asci 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Cycle</th>
<th>Pass</th>
<th>Reference date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jason-1</td>
<td>99</td>
<td>230</td>
<td>19987.9081795</td>
</tr>
<tr>
<td>Topex/Poseidon</td>
<td>442</td>
<td>230</td>
<td>19987.9127535</td>
</tr>
<tr>
<td>ERS2</td>
<td>66</td>
<td>598</td>
<td>18831.768334</td>
</tr>
<tr>
<td>ERS1-A</td>
<td>15</td>
<td>1</td>
<td>15636.938955</td>
</tr>
</tbody>
</table>
BROADVIEW RADAR ALTIMETRY TOOLBOX

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

BRATHL_DIFFDASM(BRATHLDATEDSM date1, BRATHLDATEDSM 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={BRATHLDATEDSM}
d1.REFDATE=3
d1.DAYS=423
d1.SECONDS=5
d1.MUSECONDS=0

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

diff = 0.0D
r = BRATHLDIFFYMDHMSM(d1, d2, diff)
print, r, diff

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

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

BRATHL_DIFFYMDHMSM(BRATHLDIFFYMDHMSM 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 seconds, 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

\[ r = \text{BRATHL\_DSM2YMDHMSM}(dIn, dOut) \]
\[ \text{print, } r, \text{ dOut.YEAR, dOut.JULIAN, dOut.MONTH, dOut.DAY, dOut.HOUR, dOut.MINUTE, dOut.SECOND, dOut.MUSECOND} \]

\text{BRATHL\_JULIAN2DSM} 
------------------------------------
Converts a decimal Julian date into a days-seconds-microseconds date, according to refDate parameter

\text{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)

\text{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 \text{BRATHL\_DSM2JULIAN}

\text{BRATHL\_JULIAN2SECONDS} 
------------------------------------
Converts a decimal Julian date into seconds, according to refDate parameter

\text{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 \text{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 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
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

dOut.second, " MS ", dOut.muSecond
print, "mission ", mission, " cycle ", cycle, " pass ", pass
15. ANNEX F: 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 ODAYS
    INTEGER OSECONDS
    INTEGER OMUSECONDS

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

    INTEGER RESULT
    CHARACTER*128 ERRSTR
    CHARACTER*28 REFUSER
    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,
& OSECONDS, 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_YMDHMSM2DSM(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 asci 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 contain 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 :

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

brathl_Cycle2YMDHMSM
brathl_YMDHMSM2Cycle
Cycle/date conversion example:

```fortran
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

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,  
  NbResults,  
  Ignore,  
  Statistics,  
  Default)
print *, NbResults
print *, ReturnCode
END
16. ANNEX G: 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-latex 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;
    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("Choose Source Reference : \\
    "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' :
BROADVIEW RADAR ALTIMETRY TOOLBOX

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("Choose date of reference with the format YYYY MM DD hh:mm:ss:ms ");
    gets(Buff);
    strncpy(brathl_refDateUser1, Buff, BRATHL_REF_DATE_USER_LEN - 1);
    break;
default: refDate = REF19500101;
}

puts("Choose Destination Reference:
"1 --> 1950\n"2 --> 1958\n"3 --> 1990\n"4 --> 2000\n"5 --> user 1\n"X Exit\n"\n
c = 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("Choose the reference date with the format YYYY MM DD hh:mm:ss: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));
    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_DSMDSM2Julian(&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:mm:ss:ms :");
    gets (Buff);
    sscanf(Buff, "%4d%*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_YMDHMSMSM2DSM(&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
    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);
```c
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:mm:ss:ms :");
        gets (Buff);
        sscanf(Buff, "%4d%*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:mm:ss:ms :");  
gets (Buff);  
sscanf(Buff, "%4d%*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:mm:ss: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:mm:ss:ms :");
gets (Buff);
sscanf(Buff, "%4d%f%2d%f%2d%f%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;

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:mm:ss:ms :");
    gets (Buff);
    sscanf(Buff, "%4d%*c%2d%*c%2d%*c%6d",
"%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_Quantieme(&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 asci 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 contain 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Cycle</th>
<th>Pass</th>
<th>Reference date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jason-1</td>
<td>99</td>
<td>230</td>
<td>19987.9081795</td>
</tr>
<tr>
<td>Topex/Poseidon</td>
<td>442</td>
<td>230</td>
<td>19987.9127535</td>
</tr>
<tr>
<td>ERS2</td>
<td>66</td>
<td>598</td>
<td>18831.768334</td>
</tr>
<tr>
<td>ERS1-A</td>
<td>15</td>
<td>1</td>
<td>15636.93895</td>
</tr>
<tr>
<td>ERS1-B</td>
<td>42</td>
<td>108</td>
<td>16538.6732895</td>
</tr>
<tr>
<td>ENVISAT</td>
<td>30</td>
<td>579</td>
<td>19986.106016</td>
</tr>
</tbody>
</table>

brathl_Cycle2YMDHMSM
brathl_YMDHMSM2Cycle

Cycle/date conversion example

```c
#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("Choose the 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();
    
    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);
            PrintDateYMDHMSM(&dateYMDHMSM);
            break;
        case '2' : // Date YMDHMSM --> Cycle
            memset(&dateYMDHMSM, '0', sizeof(dateYMDHMSM));
cycle = pass = 0;

            puts ("YYYY MM DD hh:mm:ss: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;

} //--------------------------------------
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;
}
17. ANNEX H: BRATHL-PYTHON API

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

-------------------------------------
BRATHL-Python API: Structures
-------------------------------------

- brathl_DateYMDHMSM
- brathl_DateDSM
- brathl_DateSecond
- brathl_DateJulian

brathl_DateYMDHMSM data structure:
-----------------------------------
This structure represents an YYYY-MM-DD HH:MN:SS:MS date structure.

=> Example - Defining date 2000-01-01 12:25:20.1:
   MyDate = brathl_DateYMDHMSM (2000, 1, 1, 12, 25, 20, 100000)

=> Example - Retrieving information:
   MyDate.YEAR       : numbers of years
   MyDate.MONTH      : numbers of months
   MyDate.DAY        : numbers of days
   MyDate.HOUR       : numbers of hours
   MyDate.MINUTE     : numbers of minutes
   MyDate.SECOND     : numbers of seconds
   MyDate.MUSECOND   : numbers of microseconds

brathl_DateDSM data structure:
-------------------------------
This structure represents day/seconds/microseconds date structure.

=> Example - Defining date 1 day, 62 seconds and 100000 microseconds:
   MyDate = brathl_DateDSM(brathl_refDate.REF19500101, 1, 62, 100000)

=> Example - Retrieving information:
   MyDate.REFDATE    : date reference number
MyDate.DAY : numbers of days
MyDate.SECOND : numbers of seconds
MyDate.MUSECOND : numbers of microseconds

REFDATE is the reference date i.e.:
0: brathl_refDate.REF19500101 : reference to 1950-01-01 00:00:00.0
1: brathl_refDate.REF19580101 : reference to 1958-01-01 00:00:00.0
2: brathl_refDate.REF19850101 : reference to 1985-01-01 00:00:00.0
3: brathl_refDate.REF19900101 : reference to 1990-01-01 00:00:00.0
4: brathl_refDate.REF20000101 : reference to 2000-01-01 00:00:00.0
5: brathl_refDate.REFUSER1 : user reference 1
6: brathl_refDate.REFUSER2 : user reference 2

-> NOTE: REFUSER1 and REFUSER2 allow the user to set two specifics reference dates of his choice
(see brathl_SetRefDateUser1 and brathl_SetRefDateUser1 functions)

brathl_DateSecond data structure:
-------------------------------
This structure represents a decimal seconds date structure.

=> Example - Defining 86401.01 seconds (starting reference date: 1950-01-01 00:00:00.0):
   MyDate = brathl_DateSecond (brathl_refDate.REF19500101, 86401.01)

=> Example - Retrieving information:
   MyDate.REFDATE : date reference number
   MyDate.SECOND : numbers of seconds

brathl_DateJulian data structure:
-------------------------------
This structure represents a decimal Julian date structure.

=> Example - Defining 1.5 days (starting reference date: 2000-01-01 00:00:00.0):
   MyDate = brathl_DateJulian (brathl_refDate.REF20000101, 1.5)

=> Example - Retrieving information:
   MyDate.REFDATE : date reference number
   MyDate.JULIAN : decimal julian day
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_SetRefDateUser1
- brathl_SetRefDateUser2

Cycle/Date Conversion functions

- brathl_Cycle2YMDHMSM
- brathl_YMDHMSM2Cycle
Data Reading functions

- brathl_ReadData

  brathl_DayOfYear function:
  ---------------------------
  Retrieves the day of the year of a date.

  brathl_DayOfYear(date)

  [in] date : date object (Type: brathl_DateYMDHMSM)
  return dayOfYear : day of year (Type: Python integer)

  brathl_DiffDSM function:
  ------------------------
  Computes the difference between two dates (date1 - date2).

  brathl_DiffDSM(dateDSM1, dateDSM2)

  [in] dateDSM1 : date object (Type: brathl_DateDSM)
  [in] dateDSM2 : date object (Type: brathl_DateDSM)
  return diff : difference in seconds (Type: Python float)

  brathl_DiffJulian function:
  ---------------------------
  Computes the difference between two dates (date1 - date2).

  brathl_DiffJulian(dateJulian1, dateJulian2)

  [in] dateJulian1 : date object (Type: brathl_DateJulian)
  [in] dateJulian2 : date object (Type: brathl_DateJulian)
  return diff : difference in seconds (Type: Python float)

  brathl_DiffYMDHMSM function:
  ---------------------------
  Computes the difference, in seconds, between two dates (date1 - date2).

  brathl_DiffYMDHMSM(date1, date2)
brathl_DSM2Julian function:
---------------------------
Converts a days-seconds-microseconds date into a decimal julian date, according to refDate parameter.

brathl_DSM2Julian(dateDSM, refDate)

[in] dateDSM : date to convert (Type: brathl_DateDSM)
[in] refDate : date reference conversion (see REFDATE on brathl_DateDSM data structure example)
return dateJulian : result of the conversion (Type: brathl_DateJulian)

brathl_DSM2Seconds function:
---------------------------
Converts a days-seconds-microseconds date into seconds, according to refDate parameter.

brathl_DSM2Seconds(dateDSM, refDate)

[in] dateDSM : date to convert (Type: brathl_DateDSM)
[in] refDate : date reference conversion (see REFDATE on brathl_DateDSM data structure example)
return dateSeconds : result of the conversion (Type: brathl_DateSecond)

brathl_DSM2YMDHMSM function:
---------------------------
Converts a days-seconds-microseconds date into a year, month, day, hour, minute, second, microsecond date.

brathl_DSM2YMDHMSM(dateDSM)

[in] dateDSM : date to convert (Type: brathl_DateDSM)
return dateYMDHMSM : result of the conversion (Type: brathl_DateYMDHMSM)

brathl_Julian2DSM function:
---------------------------
Converts a decimal julian date into a days-seconds-microseconds date, according to refDate parameter.

brathl_Julian2DSM(dateJulian, refDate)
brathl_Julian2Seconds function:
-----------------------------------
Converts a decimal julian date into seconds, according to refDate parameter.

\[
\text{brathl\_Julian2Seconds}(\text{dateJulian}, \text{refDate})
\]

\[
\text{[in]} \quad \text{dateJulian} \quad : \text{date to convert (Type: \text{brathl\_DateJulian})}
\]
\[
\text{[in]} \quad \text{refDate} \quad : \text{date reference conversion (see \text{REFDATE on brathl\_DateDSM data structure example})}
\]
\[
\text{return} \quad \text{dateSeconds} \quad : \text{result of the conversion (Type: \text{brathl\_DateSecond})}
\]

brathl_Julian2YMDHMSM function:
-----------------------------------
Converts a decimal julian date into a year, month, day, hour, minute, second, microsecond date.

\[
\text{brathl\_Julian2YMDHMSM}(\text{dateJulian})
\]

\[
\text{[in]} \quad \text{dateJulian} \quad : \text{date to convert (Type: \text{brathl\_DateJulian})}
\]
\[
\text{return} \quad \text{dateYMDHMSM} \quad : \text{result of the conversion (Type: \text{brathl\_DateYMDHMSM})}
\]

brathl_Seconds2DSM function:
-------------------------------
Converts seconds into a days-seCONDS-microseconds date, according to refDate parameter.

\[
\text{brathl\_Seconds2DSM}(\text{dateSeconds}, \text{refDate})
\]

\[
\text{[in]} \quad \text{dateSeconds} \quad : \text{date to convert (Type: \text{brathl\_DateSecond})}
\]
\[
\text{[in]} \quad \text{refDate} \quad : \text{date reference conversion (see \text{REFDATE on brathl\_DateDSM data structure example})}
\]
\[
\text{return} \quad \text{dateDSM} \quad : \text{result of the conversion (Type: \text{brathl\_DateDSM})}
\]

brathl_Seconds2Julian function:
-------------------------------
Converts seconds into a decimal julian date, according to refDate parameter.

\[
\text{brathl\_Seconds2Julian}(\text{dateSeconds}, \text{refDate})
\]
[in] dateSeconds : date to convert (Type: brathl_DateSecond)
[in] refDate : date reference conversion (see REFDATE on brathl_DateDSM data structure example)
return dateJulian : result of the conversion (Type: brathl_DateJulian)

brathl_Seconds2YMDHMSM function:
---------------------------------
Converts seconds into a year, month, day, hour, minute, second, microsecond date.

brathl_Seconds2YMDHMSM(dateSeconds)

[in] dateSeconds : date to convert (Type: brathl_DateSecond)
return dateYMDHMSM : result of the conversion (Type: brathl_DateYMDHMSM)

brathl_NowYMDHMSM function:
----------------------------
Gets the current year, month, day, hour, minute, second, microsecond date.

brathl_NowYMDHMSM()
return dateYMDHMSM : current date/time (Type: brathl_DateYMDHMSM)

brathl_YMDHMSM2DSM function:
-----------------------------
Converts a year, month, day, hour, minute, second, microsecond date into a days-seCONDS-microseconds date.

brathl_YMDHMSM2DSM(dateYMDHMSM, refDate)

[in] dateYMDHMSM : date to convert (Type: brathl_DateYMDHMSM)
[in] refDate : date reference conversion (see REFDATE on brathl_DateDSM data structure example)
return dateDSM : result of the conversion (Type: brathl_DateDSM)

brathl_YMDHMSM2Julian function:
-------------------------------
Converts a year, month, day, hour, minute, second, microsecond date into a decimal julian date, according to refDate parameter.

brathl_YMDHMSM2Julian(dateYMDHMSM, refDate)

[in] dateYMDHMSM : date to convert (Type: brathl_DateYMDHMSM)
[in] refDate : date reference conversion (see REFDATE on brathl_DateDSM data structure example)

return dateJulian : result of the conversion (Type: brathl_DateJulian)

brathl_YMDHMSM2Seconds function:
-----------------------------
Converts a year, month, day, hour, minute, second, microsecond date into seconds, according to refDate parameter.

brathl_YMDHMSM2Seconds(dateYMDHMSM, refDate)

[in] dateYMDHMSM : date to convert (Type: brathl_DateYMDHMSM)
[in] refDate : date reference conversion (see REFDATE on brathl_DateDSM data structure example)

return dateSeconds : result of the conversion (Type: brathl_DateSecond)

brathl_SetRefDateUser1 function:
-------------------------------
Set first user defined reference date: REFUSER1.

brathl_SetRefDateUser1(dateRef)

[in] dateRef : date to set in format: YYYY MM DD HH:MN:SS.MS (Type: Python string).

brathl_SetRefDateUser2 function:
-------------------------------
Set first user defined reference date: REFUSER2.

brathl_SetRefDateUser2(dateRef)

[in] dateRef : date to set in format: YYYY MM DD HH:MN:SS.MS (Type: Python string).

brathl_Cycle2YMDHMSM function:
-------------------------------
Converts a cycle/pass into a date.

brathl_Cycle2YMDHMSM(mission, cycle, nbPass)

[in] mission : mission type (Type: brathl_mission)
[in] cycle : number of cycle to convert (Type: Python int/long)
[in] nbPass : number of pass in the cycle to convert (Type: Python int/long)
return dateYMDHMSM : date/time corresponding to the cycle/pass (Type: brathl_DateYMDHMSM)

'mission' is the Satellite/mission reference i.e.:
0: brathl_mission.TOPEX : Topex/Poseidon mission
1: brathl_mission.JASON2 : Jason-2 mission
2: brathl_mission.JASON1 : Jason-1 mission
3: brathl_mission.ERS2 : ERS2 mission
4: brathl_mission.ENVISAT : Envisat mission
5: brathl_mission.ERS1_A : ERS1-A mission
6: brathl_mission.ERS1_B : ERS1-B mission
7: brathl_mission.GFO : GFO mission

=> Example:
   cycle = 1
   nbPass = 2
   dateYMDHMSM = brathl_Cycle2YMDHMSM(brathl_mission.JASON1, cycle, nbPass)

brathl_YMDHMSM2Cycle function:
-----------------------------
Converts a date into a cycle/pass.

brathl_YMDHMSM2Cycle(mission, dateYMDHMSM)

[in] mission : mission type (Type: brathl_mission)
[in] dateYMDHMSM : date/time to convert (Type: brathl_DateYMDHMSM)
return cycle : number of cycle (Type: Python int/long)
return nbPass : number of pass in the cycle (Type: Python int/long)

=> Example:
   dateYMDHMSM = brathl_DateYMDHMSM (2002, 1, 15, 6, 35, 43, 261871)
   cycle, nbPass = brathl_YMDHMSM2Cycle(brathl_mission.JASON1, dateYMDHMSM)

brathl_ReadData function:
-------------------------
Reads data from a set of files.

brathl_ReadData(fileNames, recordName, selection, expressions, units, ignoreOutOfRange, statistics, defaultValue):

[in] fileNames : File name list. Empty strings are ignored (Type: Python list of strings).
[in] recordName : Name of the fields record. For netCDF files is 'data' (Type: Python string).
[in] selection : Expression for selecting data fields. If empty string, all data are selected
(Type: Python string).

[in] expressions : Expressions applied to data fields to build wanted value.
    If empty string, the returned data are always default values
    (Type: Python list of strings).

[in] units : Wanted unit for each expression. Must be None or of 'expressions' size.
    If None, no unit conversion is done. If an entry is None or an empty string,
    no unit conversion is applied to the data of the corresponding expression
    (Type: Python list of strings).

[in] ignoreOutOfRange : Skip excess data. If there are too many values to store they are ignored
    (case is set True).
    Must be False if statistics is True (Type: Python bool).

    The returned values for each expression are:
    - Count of valid data taken into account;
    - 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 (Type: Python float or int).

return dataResults : Data read. Must contain a number of entries to values to read
    equal to expressions size
    (Type: Python list).

=> Example:
    fileNames = ['example.nc']
    recordName = 'data'
    selection = ''
    expressions = ['lat_mwr_l1b', 'lon_mwr_l1b']
    units = ['radians', 'radians']
    ignoreOutOfRange = False
    statistics = False
    defaultValue = 0

    dataResults = brathl_ReadData(fileNames,
        recordName,
        selection,
        expressions,
        units,
        ignoreOutOfRange,
statistics,
defaultValue)

print ('------------------ Printing data values ------------------')
for i in range(len(dataResults)):
    print (expressions[i], '(', len(dataResults[i]), ' values) =', dataResults[i])
print ('-----------------------------------------------------------')
18. ANNEX I: BRAT-PYTHON ALGORITHMS

The user can also define new Brat algorithms using python scripts. As the other algorithms that are compiled within Brat, the python algorithms should follow a pre-defined structure. The algorithm name, number of input parameters, calculation steps and other properties must be set by the user.

The following instructions explain all the required steps to build a new Brat algorithm using a python script.

1- The name of the python script should be "BratAlgorithm-AlgorithmName.py", in which the "AlgorithmName" must be replaced by the name of your algorithm.

2- The python module "BratAlgorithmBase.py" contains the algorithm base that is loaded by Brat. Therefore, every Python algorithm must import this module. As you will see, the new algorithm will be an extension of the base algorithm (or derived from the base algorithm class). The first line in the script should contain:

   from BratAlgorithmBase import PyBratAlgoBase

3- Import all other modules that are required for your algorithm (this step is optional). If you pretend to use modules that are available in your Python installation, extend the list of search paths with the directories of your system installation:

   import sys
   sys.path.extend( ['/USER/PYTHON_DIR', '/USER/PYTHON_PACKAGES_DIR'] )

To get the full list of directories, write "import sys" and then "sys.path" in your python terminal.

4- Complete all class methods with the required information. The following example contains the full code needed to define an algorithm that calculates the SSH (Sea Surface Height) according with the SSH Jason2 formula. Each class method is explained (see the commented lines).

```
#!/usr/bin/python

from BratAlgorithmBase import PyBratAlgoBase

# NOTE: In this case the name of the script should be "BratAlgorithm-Example_SSHjason2.py"

class Example_SSHjason2(PyBratAlgoBase):
    # Initialize here the input parameters of the algorithm
    def __init__(self):
        self.m1_alt = float()
        self.m2_range_ku = float()
        self.m3_model_dry_tropo_corr = float()
        self.m4_hf_fluctuations_corr = float()
        self.m5_inv_bar_corr = float()
        self.m6_ocean_tide_sol1 = float()
        self.m7_solid_earth_tide = float()
```
self.m8_pole_tide = float()
self.m9_sea_state_bias_ku = float()
self.m10_iono_corr_alt_ku = float()
self.m11_rad_wet_tropo_corr = float()

# Define here all the calculation steps of the algorithm
def Run(self, PyAlgoParams):
    self.SetParamValues(PyAlgoParams)  # Sets the algorithm parameters values
    self.Dump()                        # Prints all the algorithm text during execution

    # Returns the result of the calculation
    return (self.m1_alt - self.m2_range_ku - self.m3_model_dry_tropo_corr
            - (self.m4_hf_fluctuations_corr + self.m5_inv_bar_corr)
            - self.m6_ocean_tide_sol1 - self.m7_solid_earth_tide
            - self.m8_pole_tide - self.m9_sea_state_bias_ku
            - self.m10_iono_corr_alt_ku - self.m11_rad_wet_tropo_corr)

# Insert here the name of the algorithm
def GetName(self):
    return "Example_SSHjason2"

# The algorithm description
def GetDescription(self):
    return "Example of an algorithm that calculates the SSH from Jason2 data."

# Insert the number of input parameters
def GetNumInputParam(self):
    return 11

# Define the name of each parameter
def GetInputParamName(self, indexParam):
    Param_dict = {0 : "%{alt}",
                  1 : "%{range_ku}",
                  2 : "model_dry_tropo_corr",
                  3 : "hf_fluctuations_corr",
                  4 : "inv_bar_corr",
                  5 : "ocean_tide_sol1",
                  6 : "solid_earth_tide",
                  7 : "pole_tide",
                  8 : "sea_state_bias_ku"}
def GetInputParamDesc(self, indexParam):
    Param_dict = {0 : "alt",
                  1 : "range_ku",
                  2 : "model_dry_tropo_corr",
                  3 : "hf_fluctuations_corr",
                  4 : "inv_bar_corr",
                  5 : "ocean_tide_sol1",
                  6 : "solid_earth_tide",
                  7 : "pole_tide",
                  8 : "sea_state_bias_ku",
                  9 : "iono_corr_alt_ku",
                 10 : "rad_wet_tropo_corr"}
    value = Param_dict.get(indexParam)
    return value

# Define each parameter data type
def GetInputParamFormat(self, indexParam):
    Param_dict = {0 : PyBratAlgoBase.Py_FLOAT,
                  1 : PyBratAlgoBase.Py_FLOAT,
                  2 : PyBratAlgoBase.Py_FLOAT,
                  3 : PyBratAlgoBase.Py_FLOAT,
                  4 : PyBratAlgoBase.Py_FLOAT,
                  5 : PyBratAlgoBase.Py_FLOAT,
                  6 : PyBratAlgoBase.Py_FLOAT,
                  7 : PyBratAlgoBase.Py_FLOAT,
                  8 : PyBratAlgoBase.Py_FLOAT,
                 9 : PyBratAlgoBase.Py_FLOAT,
                 10 : PyBratAlgoBase.Py_FLOAT,}
    value = Param_dict.get(indexParam)
    return value

# Define parameter units
def GetInputParamUnit(self, indexParam):

Param_dict = {0 : "m",
              1 : "m",
              2 : "m",
              3 : "m",
              4 : "m",
              5 : "m",
              6 : "m",
              7 : "m",
              8 : "m",
              9 : "m",
              10 : "m"}

value = Param_dict.get(indexParam)
return value

# Define the output unit of the algorithm
def GetOutputUnit(self):
    return "m"

# Sets the parameter values
def SetParamValues(self, PyAlgoParams):
    self.CheckInputParams(PyAlgoParams)

    self.m1_alt = float(PyAlgoParams[0])
    self.m2_range_ku = float(PyAlgoParams[1])
    self.m3_model_dry_tropo_corr = float(PyAlgoParams[2])
    self.m4_hf_fluctuations_corr = float(PyAlgoParams[3])
    self.m5_inv_bar_corr = float(PyAlgoParams[4])
    self.m6_ocean_tide_sol1 = float(PyAlgoParams[5])
    self.m7_solid_earth_tide = float(PyAlgoParams[6])
    self.m8_pole_tide = float(PyAlgoParams[7])
    self.m9_sea_state_bias_ku = float(PyAlgoParams[8])
    self.m10_iono_corr_alt_ku = float(PyAlgoParams[9])
    self.m11_rad_wet_tropo_corr = float(PyAlgoParams[10])

# Define here the text that is printed during the algorithm execution
def Dump(self):
    fOut = ""
    fOut += ("==>
              Dump a Example_SSHjason2 Object at " + str(hex(id(self))) + "\n")
    fOut += ("==>
              END Dump a Example_SSHjason2 Object at " + str(hex(id(self))) + "\n")
    print (fOut)
# This function is optional and can be used to test the algorithm outside Brat (e.g. test # the script consistence during the algorithm development).
def main():
    algo = Example_SSHjason2()

    print (" ")
    print ("Name:        " + str( algo.GetName() ))
    print ("Description: " + str( algo.GetDescription() ))
    print ("Nb of Input: " + str( algo.GetNumInputParam() ))
    print ("Par 1 name:  " + str( algo.GetInputParamDesc(1) ))
    print ("RESULT:      " + str( algo.Run([60, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
            + " " + algo.GetOutputUnit()            )

# This is the standard boilerplate that calls the main() function.
if __name__ == '__main__':
    main()

5 – After testing the algorithm outside the Brat application and checking that there are no errors, copy the algorithm script to the sub-directory “/bin/Python” of the installation directory, near the example algorithms already provided, or to any sub-directory you wish to create under “/bin/Python”. At start-up, Brat will always search this directory and all its sub-directories for python algorithms to load.

The button "insert algorithm" In the "Operations" tab enable to access the available algorithms with the relevant information provided.
19. ANNEX J: COMPILATION IN GPOD ENVIRONMENT

In this annex it is explained how to compile the main BRAT command line tools, version 4, in a CentOS 6, 64 bit operating system.

19.1. Dependencies

BRAT version 4.2.1 requires CMake to be installed. The least suitable version is 3.1.0, but any version not greater than 3.2.3 will also work. These versions are not provided by default in CentOs 6: the link https://cmake.org/files/ can be used to download the required installer for any Linux x86_64 system. The installation procedure consists simply in running the downloaded script. To run CMake form any directory, the full path to the bin/ sub-directory of the CMake install location must be present in the PATH environment variable.

BRAT version 4.2.1 also requires Python 3. The standard version being 3.2.3, versions 3.3.x are equally suitable. Also, these versions are not found by default in CentOst 6 and must be obtained from 3rd party providers, following the respective instructions.

Besides CMake, Python3, and the compilation tools and packages installed in a typical CentoOs 6 system with minimal development capabilities (e.g., with g++ and make toolchains), the following system packages must also be installed: libssh2-devel, expat-devel and qt-devel (optional, if other valid Qt 4.6.2 or above is installed, as explained below).

If the software previously installed in the machine does not include other necessary dependencies, the build system will complain about the missing packages. In any case, all these dependencies for the command line tools compilation can be found in the default operating system software package list, and can be easily installed with the default package manager.

19.2. Source and Build directories

An out-of-source compilation is intended, meaning that the build outputs go to a dedicated directory, outside the source directory.

With that purpose, unpack the BRAT source package in its own directory (the <source directory>). A directory brat-<brat version> will be created and contain the software source code.

Create an empty directory (the <build directory>), outside the source directory created in the previous step, and, in the command line prompt, change (cd) to the <build directory>.

19.3. Configure and make

Then, invoke cmake with the following options:

$ cmake -DBRAT_TARGET_PROCESSOR=x86_64 -DBRAT_BUILD_GUI:BOOL=OFF -DCMAKE_BUILD_TYPE:STRING=Release -DBUILD_TESTING:BOOL=OFF -DCMAKE_CXX_FLAGS:STRING=-m64 -DCMAKE_C_FLAGS:STRING=-m64 -DDHDF5_BUILD_FORTRAN:BOOL=OFF -DDHDF5_BUILD_EXAMPLES:BOOL=OFF -DENABLE_TESTS:BOOL=OFF -DENABLE_DAP:BOOL=ON -DPYTHON_LIBRARY=<full path to the shared python library, including filename and extension> -DPYTHON_INCLUDE_DIR=<full path to the python include directory> -DIS_CENTOS_SYSTEM:BOOL=ON <source directory>

To use an existing Qt 4 installation other than qt-devel (the minimum required version being 4.6.2), the following must also be added to the invocation above:

-DQT_QMAKE_EXECUTABLE=<qmake executable full path>

CMake warnings about modules such as QtWebKit not being found can be safely disregarded.

If the CMake configuration process runs without errors, it will display the message

"Build files have been written to: <build directory>"
Finally, call make with the target "command-line-tools":

```
$ make command-line-tools
```

The message "Built target command-line-tools" will be displayed when the compilation is complete. The executables can be found in the bin sub-directory of `<build directory>`.

The executables require the environment variable BRAT_DATA_DIR to be defined, pointing to the data sub-directory of `<source directory>`, or of any other location where you wish to copy it.

Some executables, like BratCreateYFX and BratCreateZFXY, require also the embedded subset of Python in a specific location, namely in `<build directory>/bin/Python/lib64`. Simply create the Python sub-directory in `<build directory>/bin` and copy there the `lib64` sub-directory of your Python installation.
End of Document