

AIfA (Argelander-Institut für Astronomie, Universität Bonn)
MPIfR (Max-Planck Institut für Radioastronomie, Bonn)
AIRUB (Astronomisches Institut der Ruhr-Universität Bochum)

Catherine Vlahakis, add other authors
AIfA, Bonn, Germany
27. 07. 2006



Argelander-
Institut
für
Astronomie



Max-Planck-Institut
für
Radioastronomie

BoA – The Bolometer Data Analysis Project

v1.8

User's and reference manual

Abstract

BoA is a new software package for the reduction of data obtained with the bolometer arrays at the Atacama Pathfinder Experiment (**APEX**) telescope...

[CV: Need UPTODATE ABSTRACT]

Contents

I	User's Manual	4
1	Introduction	5
1.1	Philosophy and basic structure	6
2	Installing BoA	8
2.1	Prerequisites	8
2.2	Obtaining the installation script and packages	8
2.3	Installation using the <i>install.sh</i> script	9
2.4	Resuming an incomplete installation	10
2.5	Installation FAQ	10
2.6	Setting up environment variables	11
2.7	Optimizing your BoA installation	11
2.8	Updating BoA	11
3	Overview of BoA structure	12
3.1	Input data	12
3.2	Internal data handling	12
4	Quick User Guide	14
4.1	Starting up BoA	14
4.2	Some useful BoA commands	14
5	Detailed User Guide	16
5.1	Overview of how to use BoA	16
5.2	User methods for data reduction and map making	18
5.3	User methods for file reading	19
5.4	User methods for controlling graphics display devices	19
5.5	User methods for displaying data	20
5.6	MB-Fits to FITS file conversion	23
5.7	Scripts	23

5.8	Commands in alphabetical order	25
5.9	Commands in functional order	28
5.10	Abbreviations	31
6	BoGLi : the BoA Graphic Library	33
6.1	Introduction	33
6.2	Command handling	33
6.3	Device handling	34
6.4	Plotting graphics	36
6.5	Keywords	44
II	Reference Manual	46
7	Data Organisation	47
7.1	Data input: the MB-FITS format	47
7.2	BoA Data objects	48
7.3	Data output	50
8	Development	51
8.1	Basic programming rules	51
8.2	Adding classes	51
8.3	Adding methods	51
8.4	Adding Fortran90 code	51
8.5	Interfacing	54
III	All BoA classes and functions	59
A	BoA Module Index	60
A.1	BoA Modules	60
B	BoA Hierarchical Index	61
B.1	BoA Class Hierarchy	61
C	BoA Class Index	62
C.1	BoA Class List	62
D	BoA Module Documentation	63
D.1	Fortran subroutines	63
D.2	Utilities	65

D.3 DeviceHandler	69
D.4 Interface	70
D.5 MultiPlot	71
D.6 Plot	73
E BoA Class Documentation	77
E.1 boa::BoaError::BoaError Class Reference	77
E.2 boa::BoaDataEntity::BolometerArray Class Reference	78
E.3 boa::BoaDataAnalyser::DataAna Class Reference	81
E.4 boa::BoaDataEntity::DataEntity Class Reference	88
E.5 boa::Bogli::Interface::Fenetre Class Reference	92
E.6 boa::BoaDataAnalyser::FilterFFT Class Reference	93
E.7 boa::BoaFocus::Focus Class Reference	95
E.8 boa::BoaMapping::Image Class Reference	96
E.9 boa::BoaMapping::Kernel Class Reference	98
E.10 boa::BoaMessageHandler::Logger Class Reference	99
E.11 boa::MamboMBFits::MamboMBFits Class Reference	100
E.12 boa::BoaMapping::Map Class Reference	103
E.13 boa::BoaMBFitsReader::MBFitsReader Class Reference	106
E.14 boa::BoaMessageHandler::MessHand Class Reference	107
E.15 boa::BoaPointing::Point Class Reference	110
E.16 boa::BoaMessageHandler::printLogger Class Reference	113
E.17 boa::BoaDataEntity::ScanParameter Class Reference	114
E.18 boa::BoaDataEntity::Telescope Class Reference	117
E.19 boa::Utilities::Timing Class Reference	118

ToDo List

This is the latest organisational structure of the manual, and listed under each "chapter" are the jobs to be done:

- 0) ABSTRACT - write
- PART I: USER'S MANUAL
 - 1) INTRODUCTION
 - write opening short para
 - check content of existing text
 - build on content of existing text if necessary
 - 2) INSTALLATION
 - complete list of prerequisites, including any differences for different OS
 - description of where to get the install.sh script
 - description of any other installation method (other than using install.sh), if there is to be any
 - full installation instructions, including any differences for different OS
 - list of known installation problems for each OS
 - section on how to update existing BoA installation
 - 3) OVERVIEW OF BoA STRUCTURE
 - I think this chapter should give just a basic overview of the structure of BoA, perhaps referring the the structure charts/lists given in the new appendices, so that this chapter is accessible to EVERY reader.
 - I've left this section as it was previously, so that means that a lot of the content should no longer go in this chapter, but instead I suggest moving it to chapter 6 - chapter 6 will then include more in-depth/technical information as well as development information, which will be of interest to more advanced users but not for everyone.
 - nonetheless, make sure this existing material is up-to-date before shifting it around
 - 4) and 5) QUICK AND DETAILED USER GUIDES
 - make sure is clear, ordered in an easy to follow way, and up-to-date
 - these, and some of the following, chapter should make up the "cookbook"
 - 6) BoGLi
 - make sure is clear, ordered in an easy-to-follow way, and up-to-date
- PART 2: REFERENCE MANUAL
 - 7) DATA ORGANISATION
 - I think this chapter should include all the "too-technical" stuff chucked out from Ch.3

- MBFITS / DATA STORAGE / I/O etc. now included in this chapter
- add here more detailed info on how MBFITS works, how we store the data, input/output etc etc.
- a brief and simple description should also be included in Chapter 5 (Detailed User Guide)
- 8) DEVELOPMENT
 - re-write/make up-to-date
- PART 3: ALL BoA CLASSES AND FUNCTIONS
 - the appendices, generated by Doxygen
 - document any remaining undocumented items (there are quite a few)

Who should do what?

This is the current plan:

- Abstract & Intro (0 & 1) — FRANK
- Installation (2) — DIEGO & REINHOLD
- BoA structure (3) — FREDERIC & ALEXANDRE
- User guides (4, 5 & 6) — CAT
- Data org (7) — FREDERIC, ALEXANDRE, REINHOLD, FRANK
- Development (8) — FREDERIC, ALEXANDRE, REINHOLD, FRANK
- Part III — CAT
- HTML version — CAT

Status so far

- 0 & 1 need writing
- 2 Diego has updated the doc to match the info that Reinhold put on the Wiki. I made some further changes. Diego and Reinhold: can you get together and see what else you might be able to add there?
- 4,5 & 6 I need to do final testing of examples with up-to-date BoA
- 3 & 7 Frederic has been working on these chapters and made some re-arrangemtns to the structure of Part 2. Frederic, Alex, Reinhold and Frank: can you figure out between you what else needs to go into these chapters?
- 8 needs writing. Frederic, Alex, Reinhold and Frank: as above.

Notes:-

- It'd be really helpful if we try to keep the Wiki updates in synch with the manual. So, next time you write something on the Wiki, have a quick think whether it's something we could use in the manual, and if so, please just copy and paste it in to an appropriate place in the manual!
- Some of the chapters could be more or less written in final format right away (e.g. abstract, intro), while others will need to be added to over the next few weeks/months (e.g. installation), so please bear this in mind - whoever is working on a particular chapter should be responsible for making sure it is updated as soon as something new comes along!
- Either way, we should aim to get something of each chapter written up as soon as possible.
- Please use the definitions **BoA** and **BoGLi** if you want to write BoA or Bogli.
- Note that this version is latex-only (I've removed the previous html-only stuff and will re-introduce it later). I will deal with producing the html version, but in the meantime if you have something you desperately want to include only in the html version then let me know.
- Finally, note that I've cut out a lot of stuff from the "pre-doxygen" version of the documentation, but some of those basic ideas may still be necessary...so when you're working on your chapter please have a look through this "old" version (the one you'll find called `boa_master_doc.pdf` in your current BoA installation) and check there's nothing important I've cut out. Also note that if I cut something out in the first place I did so for a reason, so if you're re-including something please make sure it's relevant, up-to-date, and re-worded in a clear and concise way!

Part I

User's Manual

1. INTRODUCTION

The **Atacama Pathfinder Experiment (APEX)**¹ is a 12-meter radio telescope at the best accessible site for submillimeter observations, Llano de Chajnantor in Chile's Atacama desert.

[CV: ADD MORE HERE]

[CV: CHECK CONTENT OF THIS SECTION]

[CV: ADD NEW SECTIONS IF NECESSARY]



Figure 1.0.1: The APEX telescope at Chajnantor, Images from the APEX handover, Nov. 2003

¹<http://www.mpifr-bonn.mpg.de/div/mm/apex/>

1.1 Philosophy and basic structure

1.1.1 Philosophy

BoA is designed with essentially two goals in mind: a) to provide a comprehensive tool for the reduction and analysis of data from the new generation of bolometer arrays and b) to facilitate the extension and modification of the software by any user. **BoA** is intended to combine simple and intuitive usage with the coverage of all aspects of data reduction from raw data to final images.

The natural choice for the creation of **BoA** is object oriented programming.

[CV: Could add more here, as in previous version, but needs to be more or less comprehensible to any reader!]

1.1.2 Programming language: Python

The programming language used is Python. Python is an interpreted, interactive and object oriented language, though it does not adhere to all concepts of object orientation as strictly as e.g. C++ does. The resulting shortcomings have to be kept in mind when extending **BoA**. They can be overcome by sticking to some basic programming rules. Python is a scripting language and as such allows **BoA** to be quickly and easily extended by the user. It also facilitates the wrapping of code written in C/C++ or Fortran90.

1.1.3 Basic structure

BoA consists of a set of classes, most of which are defined in dedicated modules (files). In addition, a few functions are defined in separate modules. A detailed description of all classes and methods can be found in Sect. [CV: sort link to this ref] The subdivision was chosen to reach a high modularity and an obvious compilation of assortative functionalities within one class. [CV: above sentence unclear] Two kinds of classes may be distinguished:

- Data classes: The `DataEntity` class defines the data structure which is used within **BoA**. Objects of this class contain the raw and reduced data and all relevant parameters of a single scan. This class also defines methods to fill the data object from an MBFITS file. Then, the `DataAna` class inherits from `DataEntity`: it contains all data related methods, plus some methods for data analysis (e.g. flagging, baseline). Then, the `Map` class inherits from `DataAna`: it contains all methods defines in `DataEntity` and `DataAna`, plus specific methods for map processing and display. Finally, classes dedicated to various observing modes inherit from the `Map` class: they contain additional methods specific to a given type of observation. Table 1.1 lists **BoA** data classes, with module names and short descriptions of their responsibilities. [CV: should prob also ref the final chapter (where the detailed descriptions of classes are) here...the class structure is shown there too]
- Peripheral classes: All other classes provide methods which either are used by data objects (e.g. `Image` is used within `Map` objects), or provide functionalities on the **BoA** level (e.g. `MessHand`). These classes are summarized in Table 1.2.

[CV: Still need to check the content of these tables.]

Finally, a few functions are defined in separate modules (listed in Table 1.3), which do not define any class. Thus, these functions can easily be imported and run from any level. In particular, the **BoA** Graphic Library (**BoGLi**) is defined in a collection of modules, which can be imported at the python level and do not require **BoA**. A description of **BoGLi** is given in Sect. 6.

In addition, a number of utility and computing routines are written in Fortran modules. These routines are used within Python methods, and should in principle not be called directly by a **BoA** user.

Table 1.1: **BoA** data classes

class name	module	purpose
DataEntity	BoaDataEntity.py	data and parameters storage
DataAna	BoaDataAnalyser.py	general data analysis methods
Map	BoaMapping.py	map reduction
Focus	BoaFocus.py	focus reduction
Point	BoaPointing.py	pointing reduction
Sky	BoaSkydip.py	skydip reduction

Table 1.2: Other **BoA** classes

class name	module	purpose
Image	BoaMapping.py	image and axis description
Error	BoaError.py	
Help	BoaHelp.py	online help
MessHand	BoaMessageHandler.py	message handling
MamboMBFits	MamboMBFits.py	MAMBO to/from MB-Fits conversion
Timing	Utilities.py	benchmarking utilites

Table 1.3: Other **BoA** modules

module name	purpose
BoGLi (see Sect. 6)	Graphic library
Utilities.py (see Sect. ??)	collection of utilities
BoaConfig.py (see Sect. ??)	global parameters definitions
BoaSimulation.py	LABOCA data simulator

2. INSTALLING **BoA**

This section describes how to install **BoA** and all required additional software packages, as well as how to update an existing **BoA** version.

2.1 Prerequisites

So far, the installation was tested on the following LINUX distributions:

- SuSE 10.0
- Scientific Linux 4.2

In order to install **BoA** you first need to make sure you have the following prerequisite software packages installed on your system:

- a GNU C-compiler: gcc and gcc-c++ (tested for versions 3.4.4 (Scientific Linux) and 4.0.2 (SuSE))
- compat-g77 (tested for versions 3.3.5(SuSE) and 3.4.4 (Scientific Linux))
- readline-devel
- the libpng library: libpng-devel
- xorg-x11-devel
- findutils-locate
- cvs

All these packages are already included with a standard installation of Scientific Linux. With SuSE, use `yast` to check if they are present and to install them if necessary.

2.2 Obtaining the installation script and packages

Make sure that the shell variables `CVSROOT` and `CVS_RSH` are set to

```
CVSROOT:ext:[yourUserNameOn_aibn28]@aibn28.astro.uni-bonn.de:/var/lib/cvs
CVS_RSH /usr/bin/ssh
```

The command `cvs co boa-install` will download the external packages and the installation script *install.sh* (written by Alexandre Beelen, Thomas Jürges and Frederic Schuller) to the directory *boa-install* in your current directory.

Make the *install.sh* script executable: `chmod u+x boa-install/install.sh`

You are now ready to start the installation. (The **BoA** software itself is not downloaded at this stage. It will be downloaded from the CVS server during the installation.)

2.3 Installation using the *install.sh* script

2.3.1 Running the *install.sh* script

Before running the *install.sh* installation script make sure that you have fulfilled the prerequisites described in Sect. 2.1!

1. Go to the directory where you have downloaded the openboa cvs directory and files. Change into the directory *openboa/install/* where the installation script *install.sh* is stored.
2. Run the *install.sh* script by typing:

```
./install.sh
```

This begins the process of installing **BoA**.

The script will prompt you for some paths (reasonable defaults are offered). If you don't want to use the default path, then please enter your chosen path, e.g. */home/smueller/BoA*, when prompted. Don't forget to first create your chosen directory if not already present!

You will also be prompted to enter yes (y) or no (n) for the installation of each software package. For a fresh installation, you should install every package included (even if, say Python, is already present on your system). Skipping installation of packages is useful if you resume an aborted installation (see below). If you wish to, you can try to see whether **BoA** works with your preinstalled versions of software; however, that is at your own risk!

The script will create in this installation directory six sub-directories, *bin*, *BoA*, *include*, *lib*, *man* and *tmp* where all necessary files will be installed. The required disk space is about 220 MB.

3. After installation is complete, type `source ./boarc.sh` or `source ./boarc.csh` (located in your home directory).
4. You can now run **BoA** by typing `boa` at the prompt!

2.3.2 Details of the installation process

The installation consists of three stages, all of which are performed by the installation script:

- Installation of the external packages necessary for BoA
- Installation of BoA itself, including documentation and example FITS files
- Installation of BoA's initialization files *.boarc.sh* and *.boarc.csh*

After the installation, You will find a installation log in *boa-install/build.stat*. If the installation fails, the install script will tell you that something went wrong and give you a place where you can find information related with the failure. In addition, you can consult *boa-install/build.stat* for information about the earlier steps of the installation.

Installation of external packages

The installation script prompts you for the location of the external packages. The default should always be correct.

You are then prompted for the directory where BoA is to be installed. If this directory already exists, you must confirm that choice. (This case is necessary to resume an aborted installation or to update the BoA software itself. In all other cases, install to a new directory!)

The script then installs all external packages into this directory.

For some packages, (e.g. `scipy_obsutils`) you are prompted whether you want the package to be updated via CVS. This may not be necessary, so you can safely answer `n`. If you do update, be aware that the CVS server may be slow or even down. If this is the case, you are prompted after a timeout of 2 min whether you want to proceed without the CVS update. If you are nervous, cancel the installation with `Ctrl-C` and resume the installation (see below).

Installation of BoA

When the installation of the external packages is complete, the BoA software itself is installed. Since it is not included in the `boa-install` download, it is downloaded from the CVS server now. As an alternative, you may use a BoA tar-ball.

The script prompts you for a directory, where BoA is to be installed. You can choose any accessible directory.

After the installation of the BoA software, the documentation and example FITS files are installed. Again you are prompted whether and where you want these features to be installed.

Installation of BoA's initialization files

As last step, the script installs the initialization files `.boarc.sh` and `.boarc.csh` to your home directory. These scripts define a runtime environment for BoA (setting shell variables, paths, and aliases) for `bash` (`.boarc.sh`) and `csh` (`.boarc.csh`). Before running BoA, type `source ~/.boarc.sh` or `source ~/.boarc.csh`. You may want to add this to your shell's startup script.

2.4 Resuming an incomplete installation

To resume an incomplete installation, run `boa-install/install.sh` again. When prompted for the directory to which BoA is to be installed, specify the same directory as in the aborted installation. (Do this even if you will not install a single external package; the information is needed for the initialization files!)

You can then safely skip all installation steps, that were performed successfully in the last installation run.

Please be aware that you are prompted for the variable `PGPLOT_DIR` after skipping the installation of `pgplot`. A reasonable default is offered.

2.5 Installation FAQ

2.5.1 BoA fails to start

- `textttImportError: No module named fUtilities`

the fortran modules have not been compiled. Go to the fortran directory and type `make`

- `textttImportError: libifport.so.x`

you dont have the fortran librarie in you `$LD_LIBRARY_PATH`, please source the `boarc.xx` file or check your installation.

2.5.2 I can't open a Graphical Device

- check the `pgplot` and `p_pgplot` installation
- if trying to output `png` files, make sure that `libpng` was present when compiling `pgplot`.

2.5.3 Reading a MBFits file fails

- check the `cfitsio` and `pcfitsio` installation
- check that the version of `MBFits.xml`/`$MBFITSEXML` you are using match the file you are trying to read

2.6 Setting up environment variables

[CV: SECTION TO EXPLAIN WHAT ENVIRONMENT VARIABLES ETC. NEED TO BE DEFINED BEFORE BOA CAN BE RUN. BY THE END OF THIS SECTION USER SHOULD HAVE BEEN GIVEN EVERY PIECE OF INFO NEEDED TO GET BOA WORKING.]

Before running **BoA** you have to define an environment variable `MBFITSEXML`, whose value is the path to the file `MBFits.xml`, including the file name itself, i.e.

```
setenv MBFITSEXML ./MBFits.xml
```

2.7 Optimizing your BoA installation

[CV: SECTION ON HOW TO OPTIMIZE BOA BY PUTTING EVERYTHING IN YOUR `.CSHRC`, INCLUDING ALIASES AND ENVIRONMENT VARIABLES ETC.]

2.8 Updating BoA

3. OVERVIEW OF **BoA** STRUCTURE

[CV: NEED OPENING PARA TO EXPLAIN THAT WHAT THE INFORMATION IN THIS CHAPTER IS FOR]

[CV: I THINK THIS CHAPTER SHOULD GIVE JUST A BASIC OVERVIEW OF THE STRUCTURE OF **BoA**, PERHAPS REFERRING TO SOME OF THE STRUCTURE CHARTS/LISTS GIVEN IN THE APPENDICES, SO THAT THIS CHAPTER IS ACCESSIBLE TO EVERY READER. THEN I THINK WE SHOULD INCLUDE ADDITIONAL MORE IN-DEPTH/TECHNICAL INFORMATION IN THE DEVELOPMENT CHAPTER (CH.6)]

[CV: NEED TO MAKE SURE THIS CHAPTER UO-TO-DATE]

3.1 Input data

The data acquired at the APEX telescope are stored in a new file format, known as the MB-Fits format (for Multi-Beam FITS format, see Hatchell et al.) These files contain:

- the raw data as provided by the Frontend-Backend in use at the telescope
- data associated parameters: time of the observations, positions on the sky...
- a description of the complete Scan (eg. for a map: number of lines, steps between lines...)
- parameters of the receiver channels in the array: relative positions, relative gains

A more complete description of the input data format is given in Sect. [7.1](#).

3.2 Internal data handling

Taking full advantage of the object-oriented nature of Python, **BoA** handles data by means of objects of various classes. The primary class for data storage and manipulation is called `DataEntity` (see also Section ??). This class allows to store the raw data and associated parameters, and it provides methods relevant for any kind of observations (e.g. reading data from an MB-FITS file, plotting the signal as time series, plotting the telescope pattern). The most important attributes of this class are:

- `BolometerArray`: here, the relative positions and gains of the receiver channels are stored, as well as generic informations about the instrument and telescope (name, diameter, coordinates...)
- `ScanParam`: this contains the data associated parameters: coordinates of each point in several systems, timestamps (in LST and MJD), subscans related informations
- `Data`: this is a 2D array (time \times bolometer) which contains the current version of the data. At time of reading, the raw data are stored there; the content of this array is then altered by any processing step

- DataFlags, DataWeights: 2D arrays, with same size as Data, where flagging values and relative weights are stored for each individual data point

For processing different types of observations, **BoA** then provides several classes which inherits from DataEntity. Inheritance allows to define a class which contains all attributes and methods of the parent class, plus some specific attributes/methods. The inheritance scheme in **BoA** is as follows:

```
DataEntity < DataAna < Map < Point < Focus
```

When **BoA** is started, one object of class *Focus* is created with name *data*; this is the current data object, on which all reduction procedures can be applied. Additional objects of any data class can be created by the user within one **BoA** session. Then, applying processing methods to a data object with a different name than *data* requires to enter the full syntax (see Chapter ...), including the full name of the method, as opposed to the shortcuts described in Chapters 4 and 5.

Note: Python ensures no real difference between private and public attributes. There are only hidden attributes but this hiding can be overcome easily. Therefore the user might set any attribute directly and call any method. This is not advisable and may easily corrupt the whole **BoA** session. It is more recommendable to just use those methods for which the start script *BoaStart.py* provides abbreviations.

4. QUICK USER GUIDE

This section describes how to start up **BoA** for the first time and lists a small set of **BoA** commands needed when starting **BoA** for the first time. Detailed information on these and many more **BoA** commands can be found in Chapter 5.

4.1 Starting up BoA

You can invoke **BoA** in the following ways:

- call python in interactive mode (*-i*) with the file *BoaStart.py*

```
python -i BoaStart.py
```

- as above but using an alias you have set up in your *.cshrc* (see section 2.7)
- from an already running python session it is possible to import the **BoA** functionalities and commands by typing

```
>>> execfile('BoaStart.py')
```

at the python prompt.

BoA then prints a welcome message providing version information and changes the prompt. Nevertheless, you are still in the interactive python layer. The start script *BoaStart.py* imports a set of modules, instantiates the most essential objects and makes the respective methods available.

4.2 Some useful BoA commands

In this section we list some useful **BoA** commands, classified in terms of their function. Just enter them at the **boa>** prompt (note that the parentheses are mandatory).

Note, these commands are abbreviations for the full user method names, as is described in Chapter 4.

4.2.1 Setting up

- `indir()` Change the input directory
- `proj()` Define the APEX project ID and simplify the I/O
- `ils()` List the content of the input directory
- `find()` Reset the above input directory list

4.2.2 Display

- `open()` Open a device
- `close()` Close the current device
- `device()` Select a particular device

4.2.3 Reading in & plotting data

- `read()` Load a given fits file to BoA
- `signal()` Plot the signal against time
- `azelloff()` Plot the telescope pattern on the sky in azimuth/elevation offsets coordinates
- `chan()` Select a subset of channel
- `select()` Select scans depending on the given criteria

4.2.4 Flagging data

- `flagLST()` Flag data against time
- `flagCh()` Flag a given channel

4.2.5 Basic data analysis

- `base()` Remove a baseline
- `stat()` Compute basic statistic on the data
- `plotcor()` Correlation plot

4.2.6 Mapping

- `chanMap()` Produce a channel map
- `fastMap()` Project the data in the sky plane

4.2.7 Getting Help

You can get help on a **BoA** `command()` at any time by typing

```
print command.__doc__
```

at the prompt.

5. DETAILED USER GUIDE

In this chapter you will find detailed descriptions of user methods, their arguments, output and abbreviations and some examples of the different tasks possible to execute in **BoA**. As many user methods have an abbreviated form, these are listed in Section [5.10](#).

5.1 Overview of how to use BoA

5.1.1 Methods

BoA tasks are accessed by directly calling the appropriate methods from the interactive python layer. This ensures the full availability of all python and ppgplot facilities. As the method names to be called from the python layer may be rather long, the start script *BoaStart.py* provides a set of convenient abbreviations for those methods which are meant to be called directly by the user (“public” methods). We will therefore refer to these as user methods.

Example:

The name of the method to open a new graphic device is *DeviceHandler.openDev* and it can be called by

```
DeviceHandler.openDev()
```

or more conveniently by the abbreviations (user methods)

```
open() or op()
```

(note that the parentheses are always mandatory).

5.1.2 Arguments

Nearly all user methods require arguments to be passed. Nevertheless, the methods provide default arguments which thus may be omitted. In this case many methods just supply status information.

Example:

The user method `indir()` sets the desired input directory and requires the directory name as its argument:

```
indir('/home/user/data/')
```

The directory name is a string argument and has to be passed embedded in double or single quotes. Note that for consistency, in the examples throughout this manual we always use single quotes, but these can of course be substituted for double quotes.

Omitting the argument does not change the input directory but instead results in the supply of the current directory name:

```
indir()
```

In case an argument has to be typed more often a python variable can be used:

```
a='/home/user/data/'
indir(a)
```

Some methods require a list as argument. In python a list is embedded in square brackets with a comma as separator. Python provides a variety of functionalities to manipulate lists.

Example:

The user method `signal()` plots the time series of the data (flux density or counts versus time). It allows the user to define the list of channels plotted:

```
signal([18,19,20])
```

To create a list you can use the python function `range()`:

```
mylist=range(1,163)
signal(mylist)
```

or:

```
signal(range(1,163))
```

Even if the list contains only one element the square brackets are mandatory:

```
signal([5])
```

User methods can also be called using keyword arguments of the form *keyword = value*.

Example:

By default, the user method `signal()` plots the signal versus time connecting the datapoints with lines:

```
signal()
```

However, if you prefer, for example, to see the individual datapoints without lines, you can modify the value of the *style* argument:

```
signal(style='p')
```

A description of plotting related arguments such as *style* is given in Section ??.

5.1.3 Output

Most user methods supply status information as screen output when being called. The amount of information displayed can be restricted using the message handler associated with the main *data* object:

```
data.MessHand.setMaxWeight(4)
```

where the argument is an integer value between 1 and 5, with the following meaning:

- 1: errors, queries
- 2: warnings
- 3: short info
- 4: extended info
- 5: debug

5.2 User methods for data reduction and map making

5.2.1 Pointing

Processing a Pointing scan requires an object of the class *Point*. Since the default *data* object is of class *Map*, it has to be redefined before reading the file. Then the method to process the data is called *solvePointing*. Optionally, the method *showPointing* can be called to show the results on a map:

```
data = BoaPointing.Point() # instantiate a Point object
read('APEX-600')           # fill it with data
solvePointing()            # compute pointing offsets
showPointing()             # display map and fitted 2D-Gaussian
```

5.2.2 Focus

The recommended way to conduct Laboca focus observations is to perform a series of $n \times 3$ short, symmetric on-offs, e.g. 3 or 6*(4*5sec). For this simply the onoff has to be reduced and then the results can be fitted by a parabola.

```
solveFocus() # compute the optimal focus position
```

5.2.3 Skydip

[CV: Anything to add here yet?]

5.2.4 OnOff

[CV: Anything to add here yet?]

5.2.5 Mapping

Several methods are provided to construct a map, taking into account the relative positions of the bolometers in the instrument. The `slowMap()` method computes exact positions and loops over the pixels of the resulting map to calculate the contributions to the flux at a given position from all bolometers. This is a *very* slow method.

The `fastMap()` method loops over the signal series in each bolometer, and dumps fluxes at the nearest pixel on the final map. Then the maps produced from each bolometer are coadded. This method makes use of operations on arrays, and is thus very fast.

```
read('lissajou')
open()           # open an XWindow device
fastMap()        # reconstruct a map with the fast method
```

5.2.6 Beam maps

[CV: Anything to add here yet?]

5.3 User methods for file reading

5.3.1 Reading a FITS file

Reading a FITS file into **BoA** is done with the `read()` command. You may want to define the input directory first:

```
indir('../fits/')      # set the input directory
read('APEX-600')       # read file APEX-600.fits
```

The data are then stored in the default *data* object. It is possible to use several data objects, and to store the content of a file to a user defined object requires the following syntax:

```
data2 = BoaMapping.Map() # define a second data object
                        # of class Map
data2.read('APEX-600')
```

5.4 User methods for controlling graphics display devices

In order to display your data in various ways using the **BoA** plotting methods described in Section 5.5 below, you first need to open a graphics display device (e.g. Xwindows). Graphics display in **BoA** is controlled by a software package called **BoGLi** (the **BoA** Graphic Library), which is described in Chapter 6. A few basic **BoGLi** commands which are needed in order to carry out the **BoA** plotting methods described in section 5.5 are thus described in this section.

5.4.1 Opening a plot window

Opening a graphic device is done with the `open()` command:


```
open()      # open a device, default: XWindow
op()        # alternatively, use the abbreviated command
```

The default is to open an XWindow. You can use

```
op('?')
```

to get a list of all recognized devices. Alternatively, if you know which device you want you can enter it directly, for example

```
op('/ps')
```

You can also open a named PostScript file, here a colour PostScript file named *signal.ps*, with

```
op('signal.ps/CPS')
```

5.4.2 Clearing a plot window

Clearing a plotting window is done with the `clear()` command:

```
clear()      # clear the active device
```

However, any plot command will first clear the active device before plotting a new graph, unless the *overplot=1* keyword is supplied.

5.4.3 Closing a plot window

Closing a graphic device is done with the `close()` command:

```
close()      # open a device, default: XWindow
```

5.5 User methods for displaying data

5.5.1 Displaying channel maps

If you want to display channel maps you can do this with the command `chanmap()`. The default is to plot channel maps for all available channels. You can also specify a list of channels to be plotted.

Example:

```
read('3543')      # read in a file
op()              # open an XWindow device
chanmap()         # produce channel maps for all channels
chanmap(range(26)) # channel maps for the first 25 channels
chanmap([1,4,20,55]) # channel maps for a selection of channels
```

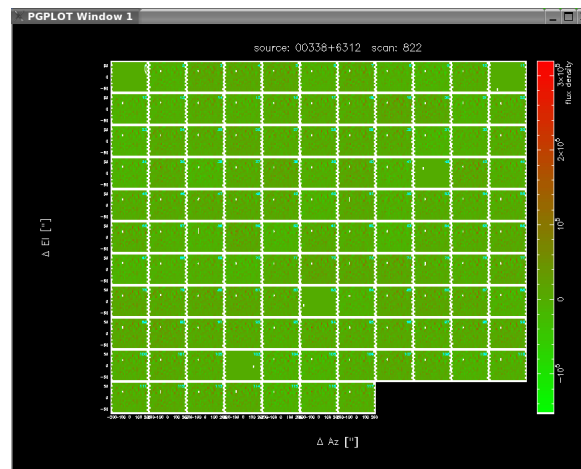


Figure 5.5.1: Default graphical outputs of a channel map of the source 00388+6312, including a wedge.

5.5.2 Plotting azimuth versus LST

DESCRIPTION: Plot the time series of the azimuth, i.e. azimuth versus LST.

USAGE: `azimuth (optional arguments)`

OPTIONAL ARGUMENTS:

<i>flag</i>	flag to be used (default = 0: all valid data; -1: plot all)
<i>limitsX</i>	range of X values to be plotted (comma separated values, in square brackets)
<i>limitsY</i>	range of Y values to be plotted (comma separated values, in square brackets)
<i>style</i>	linestyle to be used ('p' or 'l', for points and solid line respectively)
<i>ci</i>	colour index to be used (integer values)
<i>overplot</i>	
<i>aspect</i>	

A more detailed description of plotting related arguments can be found in Section ??.

Example:

```
azimuth(style='p', ci=2, limitsY=[-14,-13])
```

Plot azimuth versus LST but show individual plotted points (rather than lines), make plotted points red, and only plot azimuth (y axis) from -14 to -13 degrees.

5.5.3 Plotting elevation versus LST

DESCRIPTION: Plot the time series of the elevation i.e. elevation versus LST.

USAGE: `elevation (optional arguments)`

OPTIONAL ARGUMENTS:

flag flag to be used (default = 0: all valid data; -1: plot all)
limitsX range of X values to be plotted (comma separated values, in square brackets)
limitsY range of Y values to be plotted (comma separated values, in square brackets)
style linestyle to be used ('p' or 'l', for points and solid line respectively)
ci colour index to be used (integer values)
overplot

A more detailed description of plotting related arguments can be found in Section ??.

Example:

as for `azimuth()`, above.

5.5.4 Plotting elevation versus azimuth

DESCRIPTION: Plot elevation versus azimuth.

USAGE: `azel` (*optional arguments*)

OPTIONAL ARGUMENTS:

flag flag to be used (default = 0: all valid data; -1: plot all)
limitsX range of X values to be plotted (comma separated values, in square brackets)
limitsY range of Y values to be plotted (comma separated values, in square brackets)
style linestyle to be used ('p' or 'l', for points and solid line respectively)
ci colour index to be used (integer values)
overplot

A more detailed description of the plotting related arguments can be found in Section ??.

Example:

as for `azimuth()`, above.

5.5.5 Selecting channels

DESCRIPTION: Select a channel or a list of channels to be plotted. The list is automatically sorted.

USAGE: `channels` (*optional argument*)

OPTIONAL ARGUMENTS:

chanList: list of channel numbers, of the form: [1,2,3]
'all' ... 'al' ... 'a'
'?'

Example:

<code>channels([1,2,3])</code>	list of channels to be plotted
<code>channels(chanList=[1,2,3])</code>	list of channels to be plotted
<code>channels('all')</code>	set current list to all possible channels
<code>channels('?')</code>	get current list of channels (the default if no argument is specified)

5.5.6 Plotting flux density versus LST

DESCRIPTION: Plot the time series of the flux density i.e. flux density versus LST.

USAGE: `signal (optional argument)`

OPTIONAL ARGUMENTS:

<i>chanList</i>	list of channels, of the form [1,2,3]
<i>flag</i>	flag to be used
<i>mjd</i>	if set, use mjd instead of lst
<i>limitsX</i>	range of X values to be plotted (comma separated values, in square brackets)
<i>limitsY</i>	range of Y values to be plotted (comma separated values, in square brackets)
<i>style</i>	linestyle to be used ('p' or 'l', for points and solid line respectively)
<i>ci</i>	colour index to be used (integer values)
<i>overplot</i>	

A more detailed description of the plotting related arguments can be found in Section ??.

Example:

```
signal(chanList=[18,19,20], mjd=1, style='p', ci=2)
signal([18,19,20], mjd=1, style='p', ci=2)
```

5.5.7 Plotting the FFT of the signal

A Fast Fourier Transform (FFT) of the signal can be plotted using the *fft* method:

```
read('spiral1')
op() # open an XWindow device
data.fft(range(10)) # plot FFT for the first 9 channels
```

5.6 MB-Fits to FITS file conversion

To convert an MB-Fits file to a FITS file in the MAMBO format you can use the command `mambo()`. The current version does NOT use the data contained in the data object in BoA, but reads the input file (with default name = `BoaB.currData.FileName`) and converts it to the Mambo format. Therefore, this procedure is somewhat decoupled from BoA.

5.7 Scripts

As BoA provides the full functionality of python this allows the use of scripts. Scripts can be run with the `execfile()` function where the name of the file has to be given as string argument. The suffix of the file is arbitrary.

Example:

If you want to have a look at the time series of channels 10 to 30 succesively, create the following script with your preferred editor. Note that in python the contents of the for loop (like if blocks, method definitions, etc.) have to be indented.

```
# testBoa.py
indir('../Fits/')      # set the input directory
read('3543')           # read file 3543.fits
op()                   # open graphic display
for i in range(10,31): # start a for loop, the indentation in
                        # the following lines is mandatory
    sig([i])           # plot time series
    raw_input()         # wait for <Return>
```

To run the script type:

```
execfile('testBoa.py')
```

5.8 Commands in alphabetical order

arrayParameters	determine the array parameters from the data
basePoly	fit and subtract baseline from individual scans or subscans
basePolySubscan	subtract baseline subscan by subscan
beamMap	build a beam map in (Az,El) coordinates
blankAmplitude	blank the amplitude below and/or after a certain frequency
checkChanList	Return a list of valid channels
checkFits	check for MBFits name structure
clear	clear the active plot window
closeDev	close one device
computeBeamSize	Compute the beam size in arcsec
computeChanSep	Compute separation between pixels (in arcsec)
computeChanSepValid	Compute separation between VALID (i.e. not flagged -1) pixels (in arcsec)
computeCorMatrix	compute correlation matrix
computeOnOff	determine ON-OFF pairs from content of WobblerSta, and fill OnOffPairs attribute with pairs of integration numbers. The result is a 2 x Nb_Integ. array of integers.
computeSN	compute correlated noise, run after computeCorMatrix, computeWeight, correlate
computeWeight	compute weight matrix of the used channels, run after computeCorMatrix
correlate	compute correlation relative to a reference channel
despike	Flag yet unflagged data below 'below'*rms and above 'above'*rms
doFFT	perform the FFT
dumpData	save data object to a file
fastChanMap	plot channel maps (quick method)
fastmap	reconstruct a map in (Az,El) coordinates combining bolometers
findInDir	???
findSubscan	compute subscan indices from steps in az, el
flag	flag data at more than n*rms
flagChannels	flag a list of channels
flagLST	flag data by LST interval
flagLon	flag data by Az offset interval
flagPosition	flag a position in the sky within a given radius
flagRms	Flag channels with rms below or above respective given values
flagSubscan	flag certain subscans
getChanData	get data for one channel
getChanIndex	convert from physical channel number to index in UsedChannel
getChanListData	get data for a list of channels
getChanSep	return the channel separation in both direction from the reference channel

getPixel	allow user to get pixel values using mouse
invFFT	perform the inverse FFT
iterMap	reconstruct a map in (Az,El) coordinates combining bolometers and using varying scale to zoom on signal
listInDir	list the input directory
mambo	convert MB-Fits file to MAMBO format
medianBaseline	baseline: Remove median value per channel and per subscan
medianFilter	median filtering: remove median values computed over sliding window
open	open a graphic device
plotArray	plot the receiver parameters
plotAzEl	plot elevation versus azimuth
plotAzElOffset	plot elevation offset versus azimuth offset
plotAzimuth	plot azimuth versus LST
plotAzimuthOffset	plot azimuth offset versus LST
plotCorMatrix	plot the correlation matrix
plotCorrel	plot signal vs. reference channel
plotElevation	plot elevation versus LST
plotElevationOffset	plot elevation offset versus LST
plotFFT	plot FFT of signal
plotGain	plot the gain of the Array
plotMean	plot mean flux values vs. subscan numbers
plotMeanChan	plot mean value for each subscan vs. chan. number
plotRms	plot rms flux values vs. subscan numbers
plotRmsChan	plot rms value for each subscan vs. chan. number
plotSubscan	generate a plot showing starting and ending times of subscans
plotSubscanOffsets	Use four colours to show subscans on the Az, El pattern
read	read in a file
readAsciiRcp	update receiver channel offsets from a simple ascii file, channel-Number AzOffset(arcsec) ElOffset(arcsec)
readRCPfile	update Receiver Channel Parameters (attributes Offsets, Gain and ChannelSep) from the content of a file
reduce	Process a Pointing scan - this method is called by the apexCalibrator
resiz	resize the plot, after resizing window with mouse
resetCurrentList	reset the CurrentList to the complete list
restoreData	restore a previously stored BoA *.sav file
rotateArray	rotate array offsets according to elevation
saveMambo	convert an MB-Fits file to the MAMBO FITS format, readable by MOPSIC
selectDev	select an open device
selectInDir	make a selection in the current list

setCurrChanList	select list of channels
setInDir	set the input directory
setInFile	set the input file name
setMess	display a message
setOutDir	set the output directory
setOutFile	set the output file name
setProjectID	set the project ID
showMap	show the reconstructed map in (Az,El)
showPointing	???
signal	plot the time series of the data (flux density versus LST)
slowMap	reconstruct a map in (Az,El) coordinates combining bolometers
smoothBy	smooth the image with a 2D gaussian of gived FWHM
smoothWith	smooth the image with the given kernel
snf	compute and subtract skynoise
solveFocus	compute the optimal focus position
solvePointing	compute the pointing offset
solvePointingOnMap	compute the offset on the data.Map object
statistics	prints the statistics
unflag	unflag data
unflagChannels	unflag a list of channels
updateArrayParameters	Update the Parameters Offsets with the computed values
writeMBfits	write the data (and parameters) contained in the current data out to a FITS file in MB-Fits format
writeRCPfile	store current Receiver Channel Parameters (Offsets, Gain) to a file with mopsi like format

5.9 Commands in functional order

5.9.1 Plotting

plotArray	plot the receiver parameters
plotAzEl	plot elevation versus azimuth
plotAzElOffset	plot elevation offset versus azimuth offset
plotAzimuth	plot azimuth versus LST
plotAzimuthOffset	plot azimuth offset versus LST
plotCorMatrix	plot the correlation matrix
plotCorrel	plot signal vs. reference channel
plotElevation	plot elevation versus LST
plotElevationOffset	plot elevation offset versus LST
plotFFT	plot FFT of signal
plotGain	plot the gain of the Array
plotMean	plot mean flux values vs. subscan numbers
plotMeanChan	plot mean value for each subscan vs. chan. number
plotRms	plot rms flux values vs. subscan numbers
plotRmsChan	plot rms value for each subscan vs. chan. number
plotSubscan	generate a plot showing starting and ending times of subscans
plotSubscanOffsets	Use four colours to show subscans on the Az, El pattern
signal	plot the time series of the data (flux density versus LST)
slowMap	reconstruct a map in (Az,El) coordinates combining bolometers

5.9.2 Device handling

clear	clear the active plot window
closeDev	close one device
open	open a graphic device
resiz	resize the plot, after resizing window with mouse
selectDev	select an open device

5.9.3 Pointing and focus

reduce	Process a Pointing scan - this method is called by the apexCalibrator
showMap	show the reconstructed map in (Az,El)
showPointing	???
solveFocus	compute the optimal focus position
solvePointing	compute the pointing offset
solvePointingOnMap	compute the offset on the data.Map object

5.9.4 Flagging and despiking data

blankAmplitude	blank the amplitude below and/or after a certain frequency
despike	Flag yet unflagged data below 'below'*rms and above 'above'*rms
flag	flag data at more than n*rms
flagChannels	flag a list of channels
flagLST	flag data by LST interval
flagLon	flag data by Az offset interval
flagPosition	flag a position in the sky within a given radius
flagRms	Flag channels with rms below or above respective given values
flagSubscan	flag certain subscans
unflag	unflag data
unflagChannels	unflag a list of channels

5.9.5 Map making

beamMap	build a beam map in (Az,El) coordinates
fastChanMap	plot channel maps (quick method)
fastmap	reconstruct a map in (Az,El) coordinates combining bolometers
iterMap	reconstruct a map in (Az,El) coordinates combining bolometers and using varying scale to zoom on signal

5.9.6 Baseline subtraction, sky removal and statistics

basePoly	fit and subtract baseline from individual scans or subscans
basePolySubscan	subtract baseline subscan by subscan
computeCorMatrix	compute correlation matrix
computeSN	compute correlated noise, run after computeCorMatrix, computeWeight, correlate
computeWeight	compute weight matrix of the used channels, run after computeCorMatrix
correlate	compute correlation relative to a reference channel
doFFT	perform the FFT
invFFT	perform the inverse FFT
medianBaseline	baseline: Remove median value per channel and per subscan
medianFilter	median filtering: remove median values computed over sliding window
smoothBy	smooth the image with a 2D gaussian of given FWHM
smoothWith	smooth the image with the given kernel
snf	compute and subtract skynoise
statistics	prints the statistics

5.9.7 File handling

checkFits	check for MBFits name structure
dumpData	save data object to a file
mambo	convert MB-Fits file to MAMBO format
read	read in a file
restoreData	restore a previously stored BoA *.sav file
saveMambo	convert an MB-Fits file to the MAMBO FITS format, readable by MOPSIC
writeMBfits	write the data (and parameters) contained in the current data out to a FITS file in MB-Fits format

5.9.8 Data handling

arrayParameters	determine the array parameters from the data
checkChanList	Return a list of valid channels
computeOnOff	determine ON-OFF pairs from content of WobblerSta, and fill OnOffPairs attribute with pairs of integration numbers. The result is a 2 x Nb_Integ. array of integers.
findSubscan	compute subscan indices from steps in az, el
getChanData	get data for one channel
getChanIndex	convert from physical channel number to index in UsedChannel
getChanListData	get data for a list of channels
getChanSep	return the channel separation in both direction from the reference channel
getPixel	allow user to get pixel values using mouse

5.9.9 Selecting files and directories

findInDir	???
listInDir	list the input directory
resetCurrentList	reset the CurrentList to the complete list
selectInDir	make a selection in the current list
setCurrChanList	select list of channels
setInDir	set the input directory
setInFile	set the input file name
setOutDir	set the output directory
setOutFile	set the output file name
setProjectID	set the project ID

5.9.10 Misc.

computeBeamSize	Compute the beam size in arcsec
computeChanSep	Compute separation between pixels (in arcsec)
computeChanSepValid	Compute separation between VALID (i.e. not flagged -1) pixels (in arcsec)
readAsciiRcp	update receiver channel offsets from a simple ascii file, channel-Number AzOffset(arcsec) ElOffset(arcsec)
readRCPfile	update Receiver Channel Parameters (attributes Offsets, Gain and ChannelSep) from the content of a file
rotateArray	rotate array offsets according to elevation
setMess	display a message
updateArrayParameters	Update the Parameters Offsets with the computed values
writeRCPfile	store current Receiver Channel Parameters (Offsets, Gain) to a file with mopsi like format

5.10 Abbreviations

As we have noted already, user methods are abbreviations of the full methods. For example, the method `DeviceHandler.openDev()` can be called by the user method `open()`. For further convenience, most user methods can also be called by even shorter abbreviations of the user methods (in this example `op()` is all that is needed). A list of user methods and their abbreviations is given in Table 5.1.

Command	Abbreviations
basePoly	baseline ... base
basePolySubscan	basesub
clear	clea ... cle ... cl
closeDev	close ... clos ... clo
computeCorMatrix	cormatrix ... cmatrix
correlate	cor
dumpData	dumpDat ... dumpD ... dump
fastChanMap2	chanmap ... ChanMap ... chanMap
fastmap2	mapping ... fastMapping ...fastMap
findInDir	find ... fd
flagChannels	flagCh ... flagC ... fCh
listInDir	indirls ... ils
setMess	mess
open	ope ... op
plotAzEl	azel
plotAzElOffset	azeloff ... azelo
plotAzimuth	azimuth ... azim ... az
plotAzimuthOffset	azimuthOffset ... azimoff ... azo
plotCorrel	plotcorrel ... plotcor ... plotCor
plotElevation	elevation ... elev ... el
plotElevationOffset	elevationOffset ... eleoff ... elo
plotMean	plotmean ... plotMean
plotMeanChan	plotmeanchan ... plotMeanChan
plotRms	plotrms ... plotRms
plotRmsChan	plotrmschan ... plotRmsChan
readRCPfile	readRCP ... rcp
resiz	resi
restoreData	restoreD ... restore ... restor
saveMambo	mambo
selectDev	device ... devic ... devi ... dev
selectInDir	select ... slt
setCurrChanList	channels ... channel ... chan
setInDir	indir ... indi ... ind
setInFile	infile ... infil ... infi ... inf
setOutDir	outdir ... outdi ... outd
setOutFile	outfile ... outfil ... outfi ... outf
setProjectID	setproj ...proj
signal	signa ... sign ... sig
statistics	stat
unflagChannels	unflagCh ... unflagC ...ufCh

Table 5.1: List of user methods with abbreviations. Don't forget to add the round brackets () at the end of the commands.

6. BoGLi : THE BoA GRAPHIC LIBRARY

6.1 Introduction

The **BoA** Graphic Library (**BoGLi**) is an object-oriented software package for the graphical display of data. It is written in Python and uses **pgplot**, the python binding to **pgplot**. The main parts (classes) of the software are self-consistent and may independently be used from any python programme. Nevertheless, **BoGLi** comes with features which especially customise its use for the display of astronomical data from multi-channel receivers. Its main goal is to provide a graphic tool tailored for the use with **BoA** for the display of data from LaBoCa, Simba and Mambo.

6.2 Command handling

BoGLi has its own command handler. Nevertheless, anytime the **BoA** command handler encounters a graphic command this is automatically passed to the **BoGLi** command handler. Therefore, the user does not have to care about the separation between **BoA** and **BoGLi** commands. Table 6.1 gives an overview of some of the available commands.

BoGLi provides a variety of attributes that may be changed by the user. The attribute name is then used as command followed by the desired value as argument (see Sect. ?? for details.)

Table 6.1: List of useful BoGLi commands.

<code>DeviceHandler.openDev</code>	open a device
<code>DeviceHandler.closeDev</code>	close a device
<code>Plot.clear</code>	clear the active plot window
<code>DeviceHandler.selectDev</code>	select a device
<code>DeviceHandler.resizeDev</code>	resize the plotting area, after plot window resized using mouse
<code>Plot.plot</code>	make a single plot
<code>MultiPlot.plot</code>	plot multiple plots
<code>Plot.draw</code>	draw on an image
<code>MultiPlot.draw</code>	draw on plots of multiple channels

6.3 Device handling

BoGLi is based on `pgplot` and as a consequence the number and type of available devices depends on the actual configuration. A list of supported devices is given at <http://www.astro.caltech.edu/~tjp/pgplot/devices.html>. During installation the device drivers have to be selected by editing the file `drivers.list`. As many device drivers are available on selected operating systems only, you should ensure that drivers you do not want are commented out (place `!` in column 1) to avoid installation failures. A version of `drivers.list` used for a Linux PC can be found in Sect ??.

The command handler of BoGLi provides a set of commands to manage output devices. A detailed description of these commands is given below.

6.3.1 Opening a plot window

DESCRIPTION: Open a graphics device for `pgplot` output and make it the current device. The default, when no argument is provided, is to open an XWindow.

USAGE: `DeviceHandler.openDev (optional argument)`

The relevant abbreviations can also be used (see Table 5.1).

OPTIONAL ARGUMENT: *pgplot device type*

If the device is opened successfully, it becomes the selected device to which graphics output is directed until another device is selected (see 6.3.4) or the device is closed (see 6.3.2). If no device argument is specified PGPLOT will open the default graphics device (an XWINDOW). Alternatively, the graphics device may be selected using any of the following as arguments:

- (1) A complete device specification of the form `'device/type'` or `'file/type'`, where `/type` is one of the allowed PGPLOT device types (installation-dependent, e.g. `/xwindow`) and `'device'` or `'file'` is the name of a graphics device or disk file appropriate for this type. The `'device'` or `'file'` may contain `'/'` characters; the final `'/'` delimits the `'type'`. If necessary to avoid ambiguity, the `'device'` part of the string may be enclosed in double quotation marks.

Example: `'plot.ps/ps'`, `'dir/plot.ps/ps'`, `'"dir/plot.ps"/ps'`,
`'user:[tjp.plots]plot.ps/PS'`

- (2) A device specification of the form `'/type'`, where `/type` is one of the allowed PGPLOT device types, e.g. `/xwindow`. PGPLOT supplies a default file or device name appropriate for this device type.

Example: `'/ps'` (PGPLOT interprets this as `'pgplot.ps/ps'`)

- (3) A device specification with `'/type'` omitted; in this case the type is taken from the environment variable `PGPLOT_TYPE`, if defined (e.g., `setenv PGPLOT_TYPE PS`). Because of possible confusion with `'/'` in file-names, omitting the device type in this way is not recommended.

Example: `'plot.ps'` (if `PGPLOT_TYPE` is defined as `'ps'`, PGPLOT interprets this as `'plot.ps/ps'`)

- (4) A blank string (`' '`); in this case, `PGOPEN` will use the value of environment variable `PGPLOT_DEV` as the device specification, or `'NULL'` if the environment variable is undefined.

Example: `' '` (if `PGPLOT_DEV` is defined)

- (5) A single question mark, with optional trailing spaces, i.e. (`' ? '`). In this case, PGPLOT will prompt the user to supply the device specification, with a prompt string of the form `'Graphics device/type (? to see list, default XXX):'` where `'XXX'` is the default (value of environment variable `PGPLOT_DEV`).

Example: `' ? '`

- (6) A non-blank string in which the first character is a question mark (e.g. '?Device: '); in this case, PGPLOT will prompt the user to supply the device specification, using the supplied string as the prompt (without the leading question mark but including any trailing spaces).

Example: '?Device specification for PGPLOT: '

In cases (5) and (6), the device specification is read from the standard input. The user should respond to the prompt with a device specification of the form (1), (2), or (3). If the user types a question-mark in response to the prompt, a list of available device types is displayed and the prompt is re-issued. If the user supplies an invalid device specification, the prompt is re-issued. If the user responds with an end-of-file character, e.g., ctrl-D in UNIX, program execution is aborted; this avoids the possibility of an infinite prompting loop. A programmer should avoid use of PGPLOT-prompting if this behavior is not desirable.

The device type is case-insensitive (e.g., '/ps' and '/PS' are equivalent). The device or file name may be case-sensitive in some operating systems.

6.3.2 Closing a plot window

DESCRIPTION: Close a plotting device. The default, where no argument is supplied, is to close the current device.

USAGE: `DeviceHandler.closeDev (optional argument)`

OPTIONAL ARGUMENT:

device number (integer)
'all'
'current'...'curre'...'cur'

Example:

<code>DeviceHandler.closeDev(2)</code>	Close the device with identifier 2
<code>DeviceHandler.closeDev('all')</code>	close all devices
<code>DeviceHandler.closeDev('current')</code>	close current device (the default if no argument specified)

6.3.3 Clearing a plot window

DESCRIPTION: Clear the output of the current device. To clear the output of a different device change to that device first (see 6.3.4).

USAGE: `Plot.clear()`

6.3.4 Selecting a device

DESCRIPTION: Select an open device for graphical output. The selected device has to be previously opened with *open* (see 6.3.1).

USAGE: `DeviceHandler.selectDev (argument)`

ARGUMENT: *device number* (integer)

Example:

<code>DeviceHandler.selectDev(2)</code>	Make device number 2 the current device for graphical output
---	--

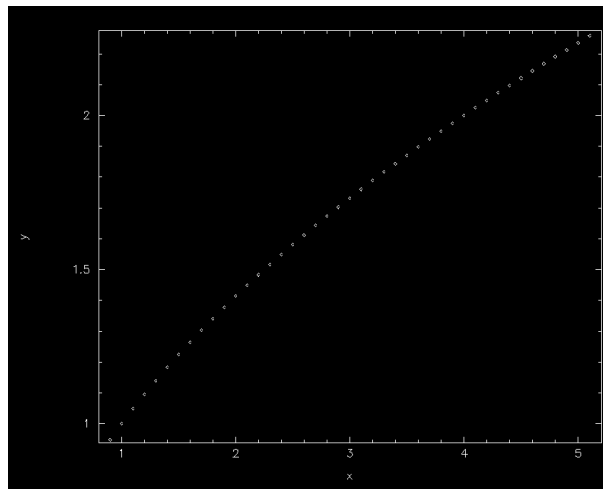


Figure 6.4.1: Example 1 of graphics produced using `Plot.plot`

6.3.5 Resizing a device

DESCRIPTION: Resize the plotting area after resizing of the graphics display window using the mouse. This is applicable to some interactive devices (e.g. `/xwindow`).

USAGE: `DeviceHandler.resizeDev()`

6.4 Plotting graphics

This section lists some of the graphics plotting capabilities of **BoGLi**.

6.4.1 Plotting single plots

DESCRIPTION: Make a single plot of `x` versus (optional) `y`.

USAGE: `Plot.plot(dataX, [dataY, limitsX, limitsY, labelX, labelY, caption, style, ci, width, overplot, aspect, logX, logY, noData])`

ARGUMENTS:

dataX values to plot along X

dataY values to plot along Y (optional - default: plot *dataX* vs. running number)

OPTIONAL ARGUMENTS:

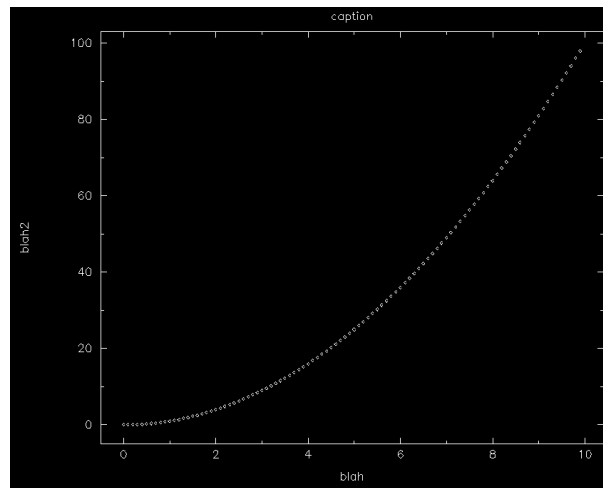


Figure 6.4.2: Example 2 of graphics produced using Plot.plot

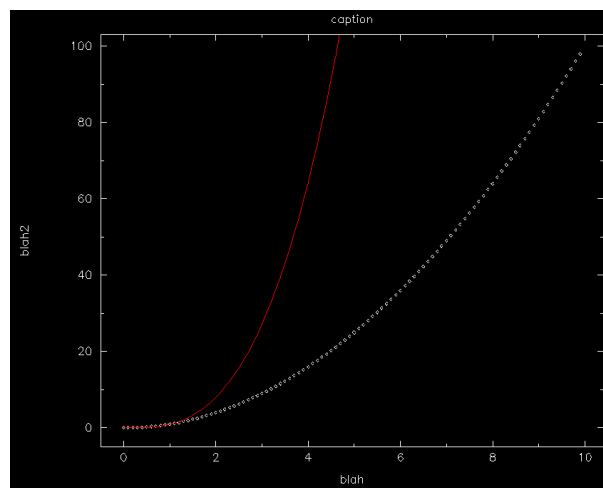


Figure 6.4.3: Example 3 of graphics produced using Plot.plot

<i>limitsX</i>	limits to use in X for the plot
<i>limitsY</i>	limits to use in Y for the plot
<i>labelX</i>	x label (default 'x')
<i>labelY</i>	y label (default 'y')
<i>caption</i>	the caption of the plot (default '')
<i>style</i>	the style used for the plot ('l': line, 'p': point (default), 'b': histogram)
<i>ci</i>	color index (default 1)
<i>width</i>	linewidth (default 0 = use previous)
<i>aspect</i>	keep the aspect ratio in 'physical' unit
<i>overplot</i>	set overplot=1 to overplot (default no)
<i>logX</i>	set logX=1 to use a log scale (default no)
<i>logY</i>	set logY=1 to use a log scale (default no)

These are also described in Section ???. Note *dataY* is also optional – if no *dataY* is supplied the default is to plot *dataX* versus running number.

Example:

```
x = Numeric.array(range(100),Numeric.Float)/10

Plot.plot(x,Numeric.sqrt(x),limitsX=[1,5])
```

Note that Y limits are then computed according to this X range.

The graphic output produced in this case is shown in Figure 6.4.1.

Example:

```
Plot.plot(x,x*x,labelX='blah',labelY='blah2',caption='caption')
```

Note that plot clear the screen first, you need to use the new 'overplot' keyword (see below).

The graphic output produced in this case is shown in Figure 6.4.2.

Example:

```
Plot.plot(x,x*x*x,overplot=1,ci=2,style='l')
```

The graphic output produced in this case is shown in Figure 6.4.3.

6.4.2 Plotting multiple channels

DESCRIPTION: Make a plot of x versus (optional) y for several channels simultaneously.

USAGE: `MultiPlot.plot(chanList, dataX, dataY, [limitsX, limitsY, labelX, labelY, caption, style, ci, overplot, logX, logY, nan])`

ARGUMENTS:

<i>chanList</i>	list of channels, of the form [1,2,3]
<i>dataX</i>	values to plot along X
<i>dataY</i>	values to plot along Y

OPTIONAL ARGUMENTS:

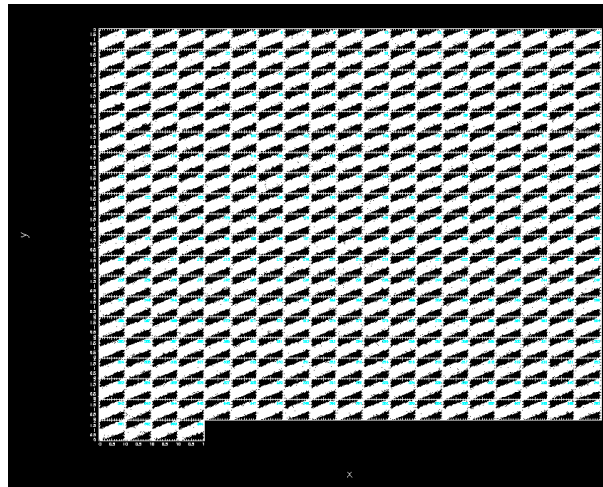


Figure 6.4.4: Example of graphics produced using MultiPlot.plot

<i>limitsX</i>	limits to use in X for the plot
<i>limitsY</i>	limits to use in Y for the plot
<i>labelX</i>	x label (default 'x')
<i>labelY</i>	y label (default 'y')
<i>caption</i>	the caption of the plot (default '')
<i>style</i>	the style used for the plot ('l': line, 'p': point (default), 'b': histogram)
<i>ci</i>	color index (default 1)
<i>overplot</i>	set overplot=1 to overplot (default no)
<i>logX</i>	set logX=1 to use a log scale (default no)
<i>logY</i>	set logY=1 to use a log scale (default no)

These are also described in Section ??.

Example:

```
n_point = 365
chanlist=range(n_point)

x2 = RandomArray.random([n_point,n_point])
y2 = RandomArray.random([n_point,n_point])

MultiPlot.plot(chanlist,x2,y2+x2,style='p')
```

The graphic output produced in this case is shown in Figure 6.4.4.

6.4.3 Drawing on an image

DESCRIPTION: Draw on an image

USAGE: `Plot.draw(map_array, [sizeX, sizeY, WCS, limitsX, limitsY, limitsZ, nan, labelX, labelY, caption, style, contrast, brightness, wedge, overplot, aspect, doContour, levels, labelContour])`

ARGUMENTS:

map_array map to display

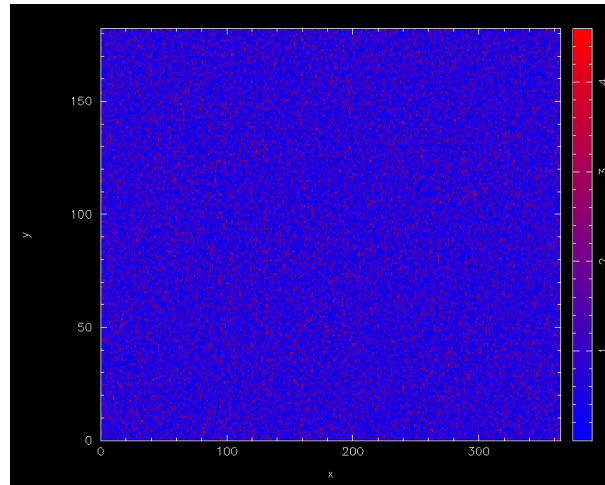


Figure 6.4.5: Example 1 of graphics produced using Plot.draw

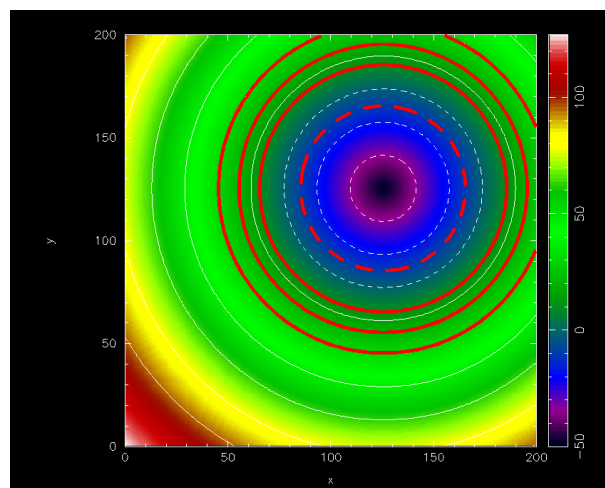


Figure 6.4.6: Example 2 of graphics produced using Plot.draw: drawing contours

OPTIONAL ARGUMENTS:

<i>sizeX</i>	the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels
<i>sizeY</i>	the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels
<i>limitsX</i>	limits to use in X for the plot
<i>limitsY</i>	limits to use in Y for the plot
<i>nan</i>	set =1 if NaN are present in the array
<i>labelX</i>	x label (default 'x')
<i>labelY</i>	y label (default 'y')
<i>caption</i>	the caption of the plot (default '')
<i>style</i>	the color used for the plot (default 'g2r', see <code>Plot.Plot.setImaCol()</code>)
<i>wedge</i>	set <code>wedge=1</code> to draw a wedge (default no)
<i>aspect</i>	keep the aspect ratio in 'physical' unit
<i>overplot</i>	set <code>overplot=1</code> to overplot (default no)
<i>doContour</i>	set =1 to draw contour instead of map (default no)
<i>levels</i>	the levels for the contours (default <code>nContour</code> , within <code>plotLimitsZ</code>)
<i>labelContour</i>	set =1 to label the contours (default no)

These arguments are also described in Section ??.

Example:

```
n_point = 365

mapping = Numeric.absolute(RandomArray.standard_normal([n_point,n_point/2]))

Plot.draw(mapping,style='b2r',wedge=1)

# You can also define 'physical' unit for your plot and still use
# limitsX/Y and aspect:

Plot.draw(mapping,sizeX=[-1,1],sizeY=[-2,2],limitsY=[-1,1],aspect=1, wedge=1)

The graphic output produced in this case is shown in Figure 6.4.5.
```

Example:

You can also use `Plot.draw()` to plot contours.

```
def dist(x,y):
    return (x-125)**2+(y-125)**2

image = Numeric.sqrt(Numeric.fromfunction(dist,(200,200)))-50

Plot.draw(image,wedge=1,aspect=1,style='rainbow') # display an image
Plot.draw(image,doContour=1,overplot=1)          # overlay some contours
Plot.contour['color'] = 2                         # change the colour and
Plot.contour['linewidth'] = 10                   # linewidth attributes

Plot.draw(image,doContour=1,overplot=1,levels=[-10,10,20,30]) # plot some
    # more contours with the new attributes
```

The graphic output produced in this case is shown in Figure 6.4.6.

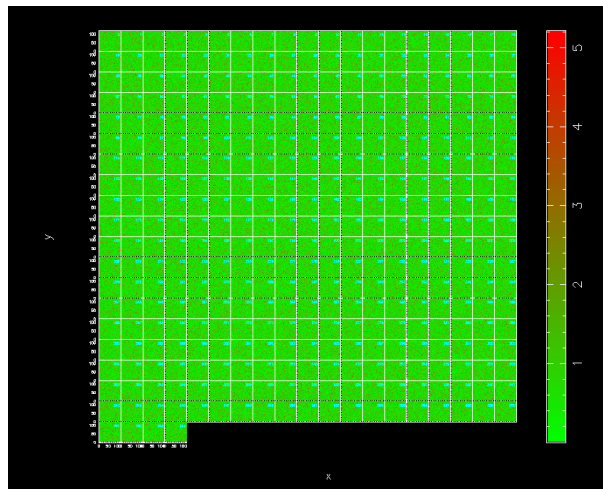


Figure 6.4.7: Example of graphics produced using MultiPlot.draw

6.4.4 Drawing on plots of multiple channels

DESCRIPTION: Draw on a multi-channel image

USAGE: `MultiPlot.plot.draw(chanList, map_arrays, [sizeX, sizeY, WCS, limitsX, limitsY, limitsZ, nan, labelX, labelY, caption, style, contrast, brightness, wedge, overplot])`

ARGUMENTS:

chanList list of channels
map_arrays lists of map to display

[CV: Re *map_arrays* above: this is what is written in MultiPlot.py, but I don't know what it is supposed to say... "lists of maps to display"????]
]

OPTIONAL ARGUMENTS:

sizeX the 'physical' size of the array (default pixel numbers)
sizeY the 'physical' size of the array (default pixel numbers)
limitsX limits to use in X for the plot
limitsY limits to use in Y for the plot
labelX x label (default 'x')
labelY y label (default 'y')
caption the caption of the plot (default ' ')
style the color used for the plot (default 'g2r', see Plot.Plot.setImaCol())
wedge set wedge=1 to draw a wedge (default no)
overplot set overplot=1 to overplot (default no)

These are also described in Section ??.

Example:

```
mapping_array = []
n_map = 365
```

```
for i in range(n_map):  
    mapping_array.append(Numeric.absolute(RandomArray.standard_normal([120,120])))
```

```
MultiPlot.draw(range(n_map), mapping_array, wedge=1)
```

The graphic output produced in this case is shown in [Figure 6.4.7](#).

6.5 Keywords

BoGLi provides a variety of parameters which allow the graphical output to be customised, as regards primitives such as colours, linestyles, character sizes, as well as text output and general appearance.

ci *colour index*

The colour index is an integer in the range 0 to a device-dependent maximum. The default colour index is 1, usually white on a black background for monitor displays or black on a white background for printed hardcopies. Colour index 0 corresponds to the background colour. If the requested color index is not available on the selected device, colour index 1 will be used.

ls *line style*

The line style is an integer in the range 1 to 5 with the following codes:

- 1: full line
- 2: dashed
- 3: dot-dash-dot-dash
- 4: dotted
- 5: dash-dot-dot-dot

The line style does not affect graph markers, text, or area fill.

lw *line width*

The line width is specified in units of 1/200 (0.005) inch (about 0.13 mm) and must be an integer in the range 1-201. This parameter affects lines, graph markers and text.

limitsX *limits to use in X for the plot*

limitsY *limits to use in Y for the plot*

labelX *x label*
(default 'x')

labelY *y label (default 'y')*

caption *caption label*
(default ' ')

style *linestyle*
(**'l'**: line, **'p'**: point (default), **'b'**: histogram)

width *linewidth*
(default 0 = use previous)

aspect	<i>aspect ratio</i> keep the aspect ratio in 'physical' unit
overplot	<i>allow/prohibit overplotting</i> set overplot=1 to overplot (default no)
logX	<i>logarithmic scale</i> set logX=1 to use a log scale (default no)
logY	<i>logarithmic scale</i> set logY=1 to use a log scale (default no)
sizeX	<i>set the 'physical' size of the array</i> the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels
sizeY	<i>set the 'physical' size of the array</i> the 'physical' size of the array (default pixel numbers), defined by the center of the two extreme pixels
nan	set =1 if NaN are present in the array
wedge	set wedge=1 to draw a wedge (default no)
doContour	<i>draw contours</i> set =1 to draw contour instead of map (default no)
levels	<i>set the levels for the contours</i> the levels for the contours (default nContour, within plotLimitsZ)
labelContour	<i>label the contours</i> set =1 to label the contours (default no)

Part II

Reference Manual

7. DATA ORGANISATION

7.1 Data input: the MB-FITS format

A complete description of the Multi-Beam FITS Raw Data Format is given in the reference document APEX-MPI-IFD-0002. In this section, we only give a brief description of this file format.

7.1.1 The hierarchy for a full scan

For a given observing sequence, corresponding to one scan, a set of tables are generated and stored in a hierarchical way in the MB-FITS format. Three tables are created on top of this hierarchy, where informations related to the full scan are gathered:

- **Primary header:** here, some general informations are stored, such as telescope name, project ID, date of observation start, versions of MB-FITS format and FitsWriter software
- **SCAN-MBFITS:** the header of this table contains a description of the scan pattern (type, geometry, line length in case of a raster map...), the source name and coordinates, together with a description of the referential used, and some generic informations about the telescope (coordinates, pointing coefficients). In addition, a binary table lists the names of frontend-backend (hereafter FEBE) combinations in use for this observation.
- **FEBEPAR-MBFITS:** one such table is created for each FEBE in use (in general, only one FEBE is active for bolometer observing). It contains the FEBE name and the number of available channels for this FEBE in its header. The associated binary table gives all relevant information about the instrument: relative gains, positions, gain/attenuation factors, polarisation angles...

7.1.2 Tables for each subscan

For each subscan within a scan, three tables are generated:

- **MONITOR-MBFITS:** this table gathers all the monitoring information sent by the control system during the observation. Each datapoint has an associated timestamp in MJD. In particular, this monitor stream contains commanded and actual telescope positions sampled every 48 ms. It also contains data related to the weather conditions, the subreflector angle and position, and the LST values.
- **DATAPAR-MBFITS:** this table also contains the telescope positions, subreflector angles and positions, and LST values, but interpolated to the timestamps corresponding to the data stream. It also contains a PHASE column, which can for example contains a succession of “ON” and “OFF” for a wobbler-switching observation.
- **ARRAYDATA-MBFITS:** here the raw data are stored. While some basic informations are stored in the header (e.g. central frequency of the observation), the binary table only contains two columns:

the timestamps (in MJD), and a vector with length equal to the number of channels in use containing the raw data for each integration.

Note: in case several FEBE are in use at the same time, then a DATAPAR table and an ARRAYDATA table are generated for each subscan and for each FEBE.

7.2 BoA Data objects

The manipulation of data within BoA is done with data objects of one class that inherits from the DataEntity class (Sect. 3.2; see also Section ??).

[FS: What else should we write in this Chapter?... the text below is far from being up-to-date, and it certainly goes too much into details to be of any relevance]

The DataEntity objects are defined such that the raw data and the reduced data can be stored, as well as associated parameters, as described below.

7.2.1 Raw data

When a file in the MB-Fits format is read into BoA, the data from a complete scan are stored in the following attributes of the current DataEntity object:

- **FITS_Header:** this is a dictionary, which contains some general descriptive parameters for the scan. This includes: telescope name and location, scan type and global parameters, wobbler mode and parameters, source name, coordinates and date of observation, frontend-backend combination in use.
- **Data_Obs:** the raw bolometer data (or counts) are stored in a 2-dimensional array (integrations \times bolometers).
- **Array parameters:** the receiver channel parameters are stored in two attributes: Array_Gain (relative gains), and Array_Geo (offsets with respect to the reference bolometer).
- **LST:** local apparent sidereal time at integration midpoint.
- **Az, El:** azimuth and elevation actually observed, including pointing corrections and wobbler offsets.
- **Lon, Lat:** longitude and latitude offsets from the source in user native frame (intermediate coordinates).
- **Wobbler_Sta:** indicates the wobbler status for each integration point. In case of wobbler switching observations, this is a succession of 'ON' and 'OFF'.
- **Subscan_Num, Subscan_Epo, Subscan_Time:** when a scan is composed of several observations (or subscans), these arrays contain, respectively: an increasing sequence of integers, the calendar date (floating value in years) of the observation start, and the local sidereal time (in seconds) at observation start.
- **Track_Az, Track_El:** tracking errors in Azimuth and Elevation (*not implemented yet*).

7.2.2 Derived data

As soon as the raw data are read, some additional quantities are derived and stored in the following attributes:

- Channel_Sep: separation between bolometers, computed from Array_Geo
- UT: universal time at each integration (*not implemented yet*).
- phase differences: for observations performed with wobbler switching, pairs of ON–OFF integrations are extracted from the Wobbler_Sta attribute, and the phase differences can be computed. The results are stored in the 3-dimensional arrays Data_Obs_p, Data_Bac_p, Data_Red_p, Data_Noi_p, Data_Flag_p, where the 1st dimension corresponds to integrations, the 2nd dimension means: 0 = phase diff, 1 = phase1, 2 = phase2, and the 3rd dimension corresponds to pixels. The following parameters are also computed, where the values associated with the phase difference is computed as the mean value of phase1 and phase2: LST_p, Az_p, El_p, Lon_p, Lat_p, UT_p.

7.2.3 Processed data

- Data_Red: this 2D array contains the reduced data after each processing step. It is first initialised with a copy of the raw data.
- Data_Bac: after every processing step, the data previously contained in Data_Red are copied to this backup array, to enable a quick 'undo' procedure.
- Data_Flag: used to flag individual data points.
- Data_Noi: rms noise associated with individual data points.
- Channel_Flag: used to flag channels.
- FFCF_CN: correlated noise.
- FFCF_Gain: computed flat field.
- Channel_rms: rms noise, per channel.
- Channel_mean: mean value, per channel.
- Results: depending on the type of observation, this can contain one or several numbers. It can also contain the results of several scans of the same type (e.g. when one wants to combine the results of several successive pointings).
- Weights: to be applied to the results (see previous item).
- Command_History: this is used to store the commands that are typed into BoA, in string format. This can then be saved into a script file, that can later be edited and imported again into BoA.

7.2.4 BoA Status

In addition to the raw, derived, processed data and associated parameters, a dictionary called Status_Dic is used to store some parameters related with the reduction steps that have been performed with BoA. This dictionary contains the following keys:

- Gain_Ele_Cor_Done
- Baseline_Cor_Ord
- Noi_Cor_Done
- Opa_Cor_Done
- Flux_Cal_Done
- Pha_Dif_Done

7.2.5 Storing a data object

At any time during a **BoA** session, the user can dump the content of the current data object to a file. It can later be loaded again into **BoA**, in order to continue with the data reduction. This is done with:

```
boa> dump()  
boa< I: current data successfully written to BoaData.sav
```

or:

```
boa> dump('myMap.data')  
boa< I: current data successfully written to myMap.data
```

to give another filename than the default `BoaData.sav`. Then to reload the data object, one has to do:

```
boa> dd = newRestoreData()
```

Note: it is not possible in its present state to apply this restore method to the default *data* object. Therefore, after reloading a data object to a new variable (*dd* in the above example), one has to use the extended syntax (Chapter ...) instead of the abbreviations defined in `BoaStart.py`.

7.3 Data output

7.3.1 Converting the raw data

BoA provides a procedure to convert an MB-FITS file to a FITS file with the same format as for MAMBO-ABBA data. The aim of this procedure is to be able to compare the results of a data reduction performed with **BoA** with those obtained with existing packages (e.g. NIC, MOPSI). *This procedure has not been extensively tested recently... it may well fail with the latest version of the MB-FITS format.*

7.3.2 Saving a map

Once a mapping observation has been read in and processed with **BoA**, the user can store the results, i.e. a map in sky coordinates, in a standard 2D FITS image, including a header with World Coordinate System (WCS) informations (*not implemented yet!!*)

8. DEVELOPMENT

[CV: NEED SOMEONE TO RE-WRITE/MAKE UP-TO-DATE THIS CHAPTER]

8.1 Basic programming rules

8.2 Adding classes

8.3 Adding methods

8.4 Adding Fortran90 code

FB040510

General

We are using Fortran 90/95 subroutines, wrapped to be called from python using the f2py package. This is because f90 code executes much faster than python scripts. There are some subtleties to pay attention to when wrapping fortran code, else you will add large overheads from the py-f90 interface, as arrays are copied and reindexed. For an introduction to F90/95 (only minor differences between the two), I recommend the compact and rather comprehensive (and free!) “Fortran 90 course notes”¹ by AC Marshall from the University of Liverpool. It contains all you probably need to know. I wrote a simple fortran method in BoA/fortran/BoaTest1.f90 to illustrate some basic features and give you a chance to test the wrapper without BoA. Look at its header for details. For an online F90/95 language reference² the best I found is at the NCSA resources page, describing IBM’s XL Fortran for AIX 8.1 – which is close to the Intel compiler.

F90 in BoA

For BoA our general idea is to have one f90.so extension module, which includes all the f90 methods (called subroutines and functions in fortran). This is necessitated by that the f90.data module, which contains much of a scans data, is connected (through an “use data”) to the other f90 program modules, and therefore they all need to be linked together.

The f90 methods may be split into different modules (classes) for convenience. We now have the first operational modules BoaF1.f90, BoaChannelAnalyser.f90, BoaBaseLine.f90, and the data module BoaData.f90. Each module may include any number of subroutines or functions. The data module BoaData.f90 is like a common block that contains all the data which does not change during data reduction. All data which does change is passed to the fortran subroutines as call arguments.

The BoaData.f90 (f90.data from python) module is filled in BoaDataEntity.FillF90. It must be refilled if

¹ <http://math.nist.gov/WMitchell/f90course/CourseNotes.pdf>

² http://www.ncsa.uiuc.edu/UserInfo/Resources/Hardware/IBMp690/IBM/usr/share/man/info/en_US/xlf/html/tr02.HTM#CONTENT

you change data object, else the fortran methods will work on a different scan. This re-filling must be implemented still. Currently the f90.data is only filled upon read of a new data file.

The CVS directory BoA/fortran contains the fortran source code. You will need to wrap/compile the BoA modules on your local system (see below), since it links to local libraries that have no standard address. This will create the extension module f90.so which you import to BoA. From the CVS directory BoA start BoA, then

```
>>> from fortran import f90
```

This is how to import any module from a subdirectory, which for this needs to include an empty file `__init__.py`

The python script fortran/ftest.py contains a series of calls to the fortran subroutines. To run it:

```
>>> read() # read in some scan
>>> op()   # open plot device
[enter]
>>> execfile('ftest.py') # start the script
```

which is followed with lots of output. To illustrate the use of new python methods that use fortran, you find BoA/TestFB.py, which you run like ftest.py. It goes through a number of data reduction steps and plots the data.

Wrapping F90 code with f2py

To wrap the f90 modules to produce f90.so:

```
ifc -c -w svd.f90
f2py -c -m f90 BoaData.f90 BoaF1.f90 BoaChannelAnalyser.f90 BoaBaseLine.f90 svd.o
    or on some installations alternatively:
f2py -c --fcompiler=intel -m f90 BoaData.f90 BoaF1.f90 BoaChannelAnalyser.f90 BoaBa
```

The first command recompiles the svd.o. On the f2py line there are some diagnostic options you may add if you debug your code:

```
-DF2PY_REPORT_ATEXIT : gives time statistics upon exit from python.
-DF2PY_REPORT_ON_ARRAY_COPY=1000 : reports when the f2py interface copies an array.
-DNUMARRAY : must be used for numarray support. Default is Numeric.
```

If the wrapping fails, one of the following may be wrong:

1. You have not initiated the ifc compiler properly. In your shell initialization file (e.g. `.cshrc` for `tcsh`) you need

```
if (-e /opt/intel/compiler60/ia32/bin/ifcvars.sh) then
    source /opt/intel/compiler60/ia32/bin/ifcvars.csh
endif
```

or something equivalent.

2. Your python path does not include the intel fortran compiler:

```
setenv PYTHONPATH " ./opt/intel/compiler60/ia32/lib/:
                  /usr/local/lib/python2.3:
                  /usr/local/lib/python2.3/site-packages:
```

```

/home/bertoldi/bin:
/opt:
/usr/lib"

```

3. You use an old version of f2py.

```

<fortran> f2py -version
2.39.235_1644

```

Once you have successfully imported f90 in BoA, you can inquire about the use of a given method by typing

```
print f90.f1.NAME.__doc__
```

Fortran attributes are called f90.data.name_of_attribute. To inquire which ones are available:

```

boa> print f90.data.__doc__
el - 'f'-array(218)
track_el - 'f'-array(218)
ffcf_gain - 'f'-array(120)
subscan_time - 'f'-array(4)
az_p - 'f'-array(109,3)
lst - 'f'-array(218)
lon_p - 'f'-array(109,3)
track_az - 'f'-array(218)
lat - 'f'-array(218)
az - 'f'-array(218)
lat_p - 'f'-array(109,3)
lst_p - 'f'-array(109,3)
array_gain - 'f'-array(120)
lon - 'f'-array(218)
ffcf_cn - 'f'-array(120)
ut_p - 'f'-array(109,3)
nodding_sta - 'i'-array(218)
subscan_index - 'i'-array(4)
subscan_num - 'i'-array(4)
weights - 'f'-array(0), not allocated
el_p - 'f'-array(109,3)
ut - 'f'-array(218)
wobbler_pos - 'f'-array(218)

```

They are filled in in BoaBusiness.py: BoaB.FillF90

Use f90 methods in BoA

To call a fortran method, here an example:

```
compressed_array, nmax = f90.f1.compress(array, flag_array, 0)
```

Two objects are returned as a tuple, an array and an integer. They both are not in the call argument list, they are hidden to python, but are listed in the f90 code call argument list – have a look at the source code.

Limitations

This particular example illustrates one of the limitations of wrapping f90 code: you cannot return an array with a length that is determined upon execution. The wrapper needs to specify the size of an array

somehow. It does not have to be fixed, but specified through the size of an input attribute at least. In this example we try to return an array that is a compression of the input array, determined by the condition that the corresponding flag is 0. The trick to still do this here is to return a compressed_array with the same size as array, plus an integer telling the size of the compressed array, so that the final answer is compressed_array[0:nmax].

Fortran vs. C-contiguous

If a Numeric array is proper-contiguous and has a proper type then it is directly passed to the wrapped Fortran function. Otherwise, an element-wise copy of an input array is made and the copy, being proper-contiguous and with proper type, is used as an array argument. There are two types of proper-contiguous Numeric arrays: Fortran-contiguous arrays when data is stored column-wise, i.e. indexing of data as stored in memory starts from the lowest dimension; C-contiguous when data is stored row-wise, i.e. indexing of data as stored in memory starts from the highest dimension. For one-dimensional arrays these notions coincide. To transform input arrays to column major storage order before passing them to Fortran routines, one may use the function `as_column_major_storage(<array>)` that is provided by all F2PY generated extension modules, such as the BoA f90. If you call a fortran method repeatedly with the same input array, you should convert the array first to avoid conversion by the wrapper interface on each call – which could dominate the execution time here. If you add the option `-DF2PY_REPORT_ON_ARRAY_COPY=1000` when wrapping, you will be informed on each copy that the wrapper interface performs. The option `-DF2PY_REPORT_ATEXIT` gives an execution time summary upon exit that splits up the time used in fortran and in the interface. If the interface time is large or comparable to the fortran execution time, your code is not efficient because it copies arrays too often. Look at examples in `BoaBaseLine.py`, e.g.:

```
Data = f90.as_column_major_storage(self.Data.Data_Red_p)
Flag = f90.as_column_major_storage(self.Data.Data_Flag_p)
...
for i_ch in ch_range: # loop over channels and phases
    for i_ph in ph_range:
        Data = f90.baseline.addpoly(Data, Poly, Mean, Rms, i_ph, i_ch)
```

The input arrays are copied once into fortran-contiguous arrays before the loop, so in the loop there is no overhead from copying. Note also the general scheme of calling a fortran method here: Data is in- and output argument.

8.5 Interfacing

8.5.1 ScientificPython-2.4.5

[CV: SOMEONE NEEDS TO CHECK THIS SECTION TOO]

ScientificPython is a collection of Python modules that are useful for scientific computing. Almost all modules make extensive use of Numerical Python (NumPy, Numeric), which must be installed prior to Scientific Python. Scientific consist of about one dozen modules, which contain methods written in Python that may come handy, but may be slow. The following lists a number of them.

`stat()` `statistics()` command calculates the statistics for all the channels in the range. Using `plotmean()` `plotrms()` we can plot mean and RMS values of each channels. The examples are as shows below:

You need to import Numeric for Scientific. You can access the methods by importing the class or all methods:

```
>>> from Numeric import *
>>> import Scientific.Statistics
```

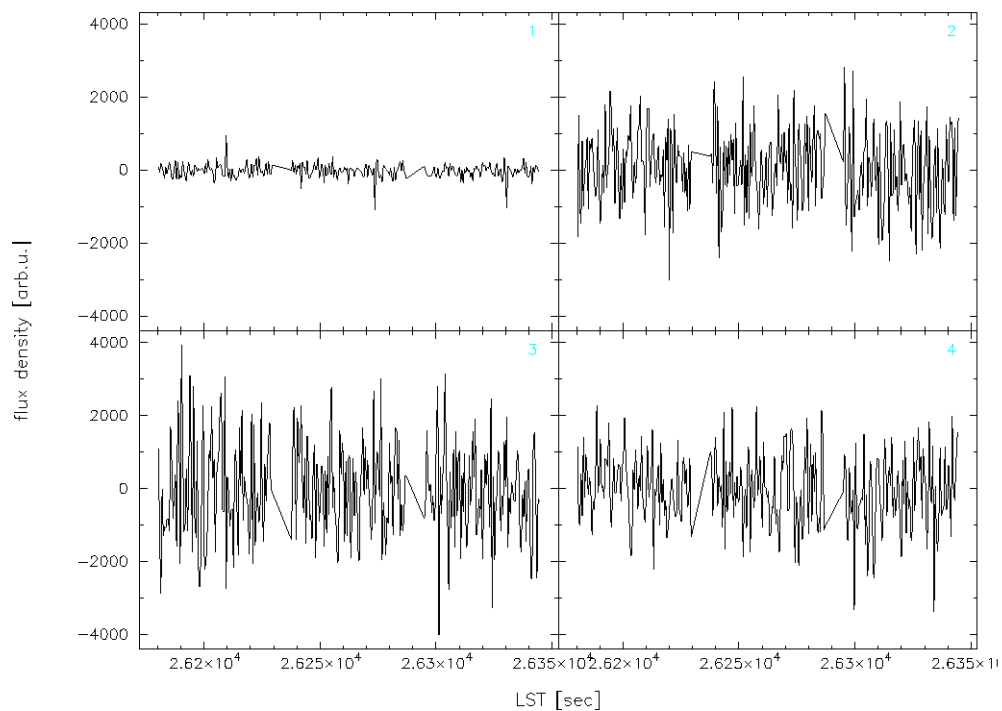


Figure 8.5.1: Plotting the Signal for channels in the range.

```
>>> Scientific.Statistics.median([1,2,3,5,6])
3.0
```

or alternatively

```
>>> from Scientific.Statistics import *
>>> median([1,2,3,5,6])
3.0
```

Available method in class Scientific.Statistics:

```
moment(data, order, about=None, theoretical=1)
mean(data)
weightedMean(data, sigma)
variance(data)
standardDeviation(data)
median(data)
mode(data)
normalizedMoment(data, order)
skewness(data)
kurtosis(data)
correlation(data1, data2)
```

There are also two classes for histograms:

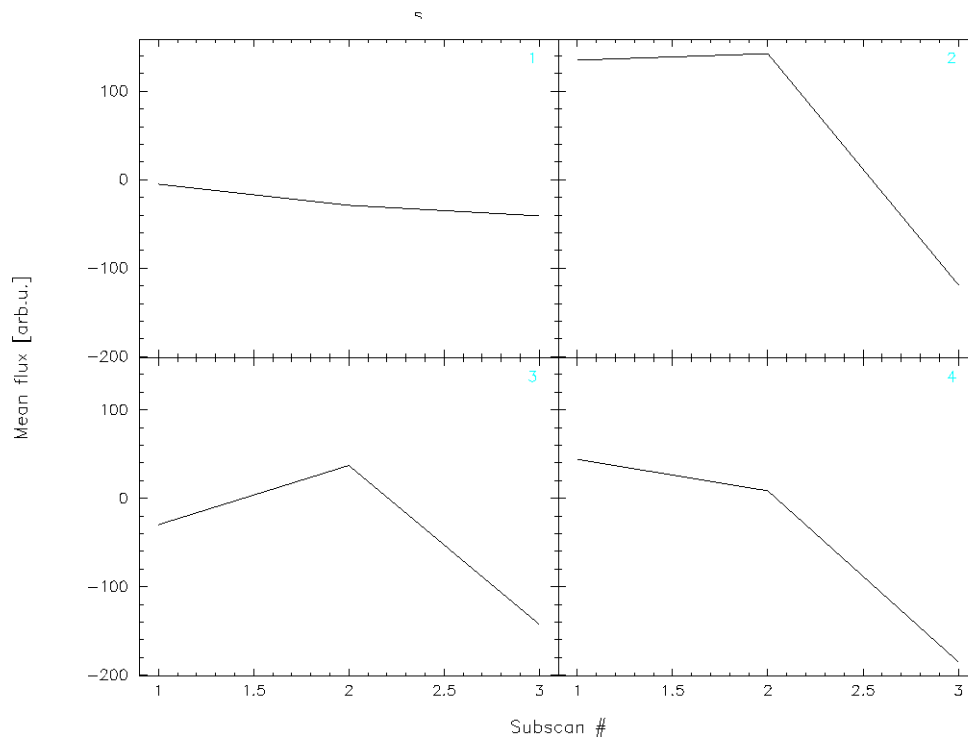


Figure 8.5.2: Plotting the Mean values of signal.

```
Histogram
WeightedHistogram(Histogram)
```

The following explains only those Scientific methods which are useful for Boa. Consult the scripts or the (very sparse) documentation for more info.

Scientific.Statistics.median

Description: Computes the median of a 1-d array.

Example:

```
>>> median([1,2,3,5,6])
3.0
```

Scientific.Statistics.mean

Description: Returns the mean (average value) of a 1-d array.

Example:

```
>>> mean([1,2,3,5,6])
```

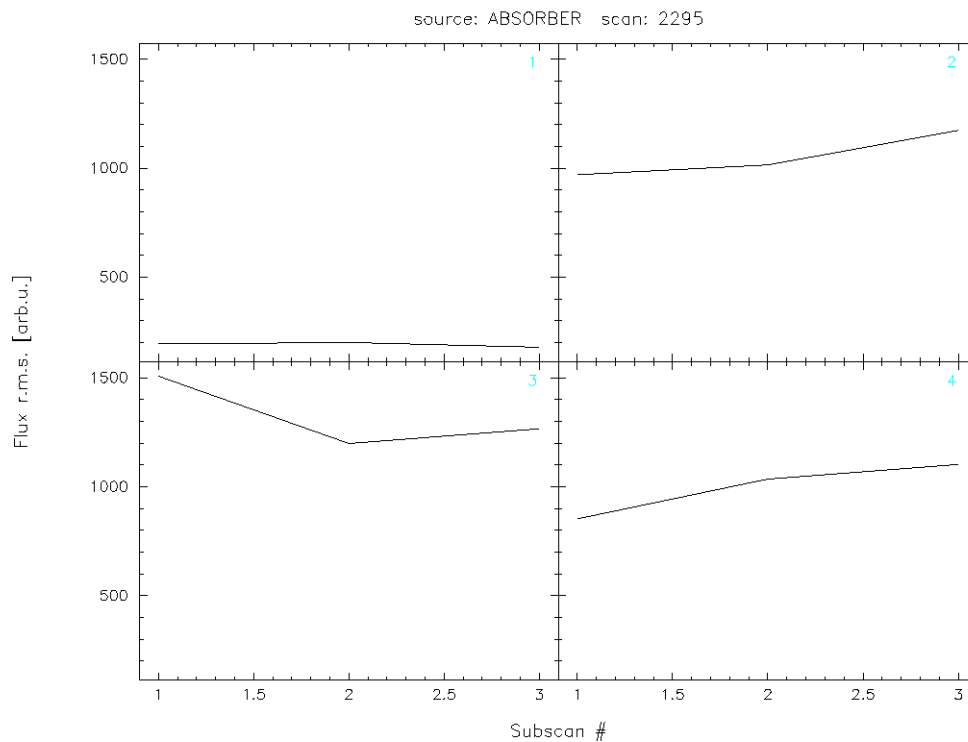


Figure 8.5.3: Plotting the RMS values of signal.

```
3.3999999999999999
```

Scientific.Statistics.correlation

Description: Computes the correlation coefficient between two 1-dim arrays a and b according to

$$c_{ab} = \frac{\langle (a - \bar{a})(b - \bar{b}) \rangle}{\langle (a - \bar{a})^2 \rangle^{1/2} \langle (b - \bar{b})^2 \rangle^{1/2}} \quad (8.5.1)$$

Example:

```
>>> correlation([1,2,3,4,5],[1,2,3,4,5])
1.0
>>> correlation([1,2,3,4,5],[1,2,3,5,5])
0.96476382123773219
>>> correlation([1,2,3,4,5],[5,4,3,2,1])
-1.0
```

Scientific.Functions.LeastSquares

Description: General non-linear least-squares fit using the Levenberg-Marquardt algorithm and automatic derivatives. The parameter model specifies the function to be fitted. It will be called with two parameters:

the first is a tuple containing all fit parameters, and the second is the first element of a data point (see below). The return value must be a number. Since automatic differentiation is used to obtain the derivatives with respect to the parameters, the function may only use the mathematical functions known to the module `FirstDerivatives`. The parameter `parameter` is a tuple of initial values for the fit parameters. The parameter `data` is a list of data points to which the model is to be fitted. Each data point is a tuple of length two or three. Its first element specifies the independent variables of the model. It is passed to the model function as its first parameter, but not used in any other way. The second element of each data point tuple is the number that the return value of the model function is supposed to match as well as possible. The third element (which defaults to 1.) is the statistical variance of the data point, i.e. the inverse of its statistical weight in the fitting procedure. The function returns a list containing the optimal parameter values and the chi-squared value describing the quality of the fit.

Example:

```
>>> from Numeric import exp
>>> def f(param, t):
...     return param[0]*exp(-param[1]/t)
...
>>> data = [(100, 4.999e-8), (200, 5.307e+2),
...          (300, 1.289e+6), (400, 6.559e+7)]
>>> print leastSquaresFit(f, (1e13, 4700), data)
([8641551709749.7666, 4715.4677901570467], 1080.2526437958597)
```

Part III

All BoA classes and functions

A. BOA MODULE INDEX

A.1 BoA Modules

Here is a list of all modules:

Utilities	65
DeviceHandler	69
Interface	70
MultiPlot	71
Plot	73

B. BOA HIERARCHICAL INDEX

B.1 BoA Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

boa::BoaError::BoaError	77
boa::BoaDataEntity::BolometerArray	78
boa::BoaDataEntity::DataEntity	88
boa::BoaDataAnalyser::DataAna	81
boa::BoaMapping::Map	103
boa::BoaPointing::Point	110
boa::BoaFocus::Focus	95
boa::Bogli::Interface::Fenetre	92
boa::BoaDataAnalyser::FilterFFT	93
boa::BoaMapping::Image	96
boa::BoaMapping::Kernel	98
boa::BoaMessageHandler::Logger	99
boa::MamboMBFits::MamboMBFits	100
boa::BoaMBFitsReader::MBFitsReader	106
boa::BoaMessageHandler::MessHand	107
boa::BoaMessageHandler::printLogger	113
boa::BoaDataEntity::ScanParameter	114
boa::BoaDataEntity::Telescope	117
boa::Utilities::Timing	118

C. BOA CLASS INDEX

C.1 BoA Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

boa::BoaError::BoaError	77
boa::BoaDataEntity::BolometerArray	78
boa::BoaDataAnalyser::DataAna	81
boa::BoaDataEntity::DataEntity	88
boa::Bogli::Interface::Fenetre	92
boa::BoaDataAnalyser::FilterFFT	93
boa::BoaFocus::Focus	95
boa::BoaMapping::Image	96
boa::BoaMapping::Kernel	98
boa::BoaMessageHandler::Logger	99
boa::MamboMBFits::MamboMBFits	100
boa::BoaMapping::Map	103
boa::BoaMBFitsReader::MBFitsReader	106
boa::BoaMessageHandler::MessHand	107
boa::BoaPointing::Point	110
boa::BoaMessageHandler::printLogger	113
boa::BoaDataEntity::ScanParameter	114
boa::BoaDataEntity::Telescope	117
boa::Utilities::Timing	118

D. BOA MODULE DOCUMENTATION

D.1 Fortran subroutines

D.1.1 Detailed Description

List of Fortran subroutines

Modules

- [Fortran subroutines](#)

safeExp(x,y,n)

NAM: safeExp (subroutine)
DES: Attempt to create a fast exponential for python

modelBaseEllipticalGaussian(p,n_p,position,n_position,returned_result)

NAM: modelBase2dgauss (subroutine)
DES: Compute a model of a 2D gaussian + gradient
(see corresponding python routine)

compress(data, flag, flag_value, dataOut, nData, nOut)

NAM: compress (subroutine)
DES: compress array based on mask
INP: data_input (f): 1-D array with data to be returned where flag = flag_value
flag (i): 1-D array, must be same size as data_input
flag_value (i): select data where flag=flag_value
OUT: tuple (x,nData): 1-D array of same size as data_input where first n elements are
those of data_input where flag = flag_value

USE:
Unfortunately it is not possible to pass from fortran to python an output array the size of which is computed in fortran. Therefore we must truncate the returned array at the size of the selected true mask elements.

Example:
input_array = array(range(5), 'f')
flag_array = array([0,1,0,1,0], 'i')
compress_array, nmax = f90.f1.compress(input_array, flag_array, 1)
compress_array = compress_array[0:nmax]

icompress(data, flag, flag_value, dataOut, nData, nOut)

NAM: compress (subroutine)
 DES: compress array based on mask
 INP: data_input (i): 1-D array with data to be returned where flag = flag_value
 flag (i): 1-D array, must be same size as data_input
 flag_value (i): select data where flag=flag_value
 OUT: tuple (x,nData): 1-D array of same size as data_input where first n elements are
 those of data_input where flag = flag_value
 USE:
 Unfortunately it is not possible to pass from fortran to python an output array the size of which is computed in fortran. Therefore we must truncate the returned array at the size of the selected true mask elements.

Example:

```
input_array = array(range(5),'f')
flag_array  = array([0,1,0,1,0],'i')
compress_array,nmax = f90.f1.compress(input_array,flag_array,1)
compress_array = compress_array[0:nmax]
```

ncompress(data, flag, flag_value, dataOut, nData, nOut)

NAM: compress (subroutine)
 DES: compress array based on mask
 INP: data_input (f): 1-D array with data to be returned where flag != flag_value
 flag (i): 1-D array, must be same size as data_input
 flag_value (i): select data where flag=flag_value
 OUT: tuple (x,nData): 1-D array of same size as data_input where first n elements are
 those of data_input where flag = flag_value
 USE:
 Unfortunately it is not possible to pass from fortran to python an output array the size of which is computed in fortran. Therefore we must truncate the returned array at the size of the selected true mask elements.

Example:

```
input_array = array(range(5),'f')
flag_array  = array([0,1,0,1,0],'i')
compress_array,nmax = f90.f1.compress(input_array,flag_array,1)
compress_array = compress_array[0:nmax]
```

minmax(array,nx,extrema)

NAM: fMin
 DES: return minimum of a 1D array

replaceNaN(array,nx,ny)

NAM:
 DES: replace the NaN in a 2D array by a value below
 the minimum value of the array

MreplaceNaN(array,nx,ny,nz)

NAM:
 DES: replace the NaN in a 3D array by a value below
 the minimum value of the array

D.2 Utilities

Classes

- class `boa::Utilities::Timing`

Functions

- def `boa::Utilities::array2list`
- def `boa::Utilities::as_column_major_storage`
- def `boa::Utilities::attrStr`
- def `boa::Utilities::baseCircularGaussian`
- def `boa::Utilities::baseEllipticalGaussian`
- def `boa::Utilities::compress2d`
- def `boa::Utilities::compressNan`
- def `boa::Utilities::Cp2r`
- def `boa::Utilities::Cr2p`
- def `boa::Utilities::cropped_circular_gaussian`
- def `boa::Utilities::detStartParaParabola`
- def `boa::Utilities::distsq`
- def `boa::Utilities::fitBaseEllipticalGaussian`
- def `boa::Utilities::fitParabola`
- def `boa::Utilities::gaussian`
- def `boa::Utilities::lCompressNan`
- def `boa::Utilities::max2D`
- def `boa::Utilities::min2D`
- def `boa::Utilities::modelBaseEllipticalGaussian`
- def `boa::Utilities::modelparabola`
- def `boa::Utilities::parabola`
- def `boa::Utilities::prettyPrintList`
- def `boa::Utilities::safeExp`
- def `boa::Utilities::solvePoly`

D.2.1 Function Documentation

def `boa::Utilities::array2list` (a)

NAM: `array2list` (function)
 DES: convert a list of 1D arrays to a single 1D array

def `boa::Utilities::as_column_major_storage` (classIn)

DES: save all the attribute as column major to avoid copy in fortran

def `boa::Utilities::attrStr` (object, badAttributes = [])

DES: return a string representing the attributes of the object
 OPT: (str list) `badAttributes` : list of attributes to remove from the output

def boa::Utilities::baseCircularGaussian (p, fjac = None, x = None, y = None, err = None)

NAM: baseCircularGaussian
 DES: function used by mpfit to fit a Circular gaussian+base
 (5 elmts array) p : parameters of the gaussian
 (2d array) x : position of the pixels on the map
 "x" = x[0] and "y" = x[1]
 (2d array) y : the map to fit
 y.shape should be (len(x[0]),len(x[1]))

def boa::Utilities::baseEllipticalGaussian (p, fjac = None, x = None, y = None, err = None)

NAM: baseEllipticalGaussian
 DES: function used by mpfit to fit a 2D gaussian+base
 (5 elmts array) p : parameters of the gaussian (see modelBase2Dgauss)
 (2d array) x : position of the pixels on the map
 "x" = x[0] and "y" = x[1]
 (2d array) y : the map to fit
 y.shape should be (len(x[0]),len(x[1]))

def boa::Utilities::compress2d (array, indexes)

DES: return a 2D sub array based on indexes
 INP: (f) array : square input array
 (i) indexes : the indexes to take from the array

def boa::Utilities::compressNan (array)

DES: return an array without nan
 INP: (array) array : input array
 OUT: (1D array) values of the previous array without Nan

def boa::Utilities::Cp2r (amp, phase)

NAM: Cp2r (function)
 DES: convert complex numbers in polar form to rectangular form (real,imag)
 IN: [float,float] : module and phase
 OUT: (complex) c : complex number or array

def boa::Utilities::Cr2p (c)

NAM: Cr2p (function)
 DES: convert complex numbers in rectangular form to polar (mod,arg) form
 INP: (complex) c : complex number or array
 OUT: [float,float] : module and phase

def boa::Utilities::cropped_circular_gaussian (p, position, threshold = 3)

NAM: cropped_circular_gaussian
 DES: compute a cropped circular gaussian with intensity=1
 defined by the parameter p within the position an a given threshold given in n*'sigma'
 position should be a list of 2 arrays of the same dimension defining the map

def boa::Utilities::detStartParaParabola (x, y)

NAM: detStartParaParabola (method)
 DES: define the proper start parameter to fit a parabola
 INP: (float) x = x data
 (float) y = y data

def boa::Utilities::distsq (x1, y1, x2, y2)

NAM: distsq (function)
 DES: returns distance squared between two points
 INP: (float) x1,y1,x2,y2: coordinates of the two points
 OUT: (float) distance^2

def boa::Utilities::fitBaseEllipticalGaussian (mapArray, x, y, err = 1.0, fwlm = 11.0, gradient = 1, circular = 0, Xpos = 0., Ypos = 0., fixedPos = 0)

NAM: fitBaseEllipticalGaussian (method)
 DES: fits a 2D Gaussian + 1st order base surface
 INP: (arrays) (x,y,mapArray,err) : the data to fit (arrays of same dimension(s))
 (2els arrays) sizeX/Y : alternative to the x/y array, this limit the size of the
 mapArray given as regular gridding between the center
 of the two extreme pixels
 (float) fwlm : the first guess for the fwlm
 (logical) gradient : should we also fit a gradient in the map (default no) ?
 (logical) circular : fit a circular gaussian instead of a elliptical gaussian
 (float) Xpos,Ypos : source position if using fixed position
 (logical) fixedPos : if set, don't fit position, but use Xpos, Ypos
 OUT: a dictionary containning the results of the fit
 check 'status' and 'errmsg' to see if the fit was done correctly
 then for each parameters (see the parname variable below) you have the
 'value' 'error' and 'limits' for the fit

def boa::Utilities::fitParabola (x, y, err)

NAM: fitParabola (method)
 DES: fits parabola to the data using mpfit
 INP: (float) x = x data
 (float) y = y data

def boa::Utilities::gaussian (r2, sig2)

DES: Compute value of a Gaussian function
 INP: r2 = _array_ of distances^2, sig2 = sigma^2, related to Gaussian width

def boa::Utilities::lCompressNan (array, listArray)

DES: remove the Nan of an array in a list of array
 INP: array : test array for the Nan
 (1 array): the list of array to compress

def boa::Utilities::max2D (a)

NAM: max2D (function)

DES: return the maximum value from a list of 1D Numeric arrays

def boa::Utilities::min2D (a)

NAM: min2D (function)

DES: return the minimum value from a list of 1D Numeric arrays

def boa::Utilities::modelBaseEllipticalGaussian (p, position)

NAM: model2Dgauss

DES: compute a 2D gaussian defined by the parameter p wihtin the position
position should be a list of 2 arrays of same dimensions defining the map

def boa::Utilities::modelparabola (p, x)

NAM: modelparabola

DES: compute a model parabola at position x for a given set of parameters p

def boa::Utilities::parabola (p, fjac = None, x = None, y = None, err = None)

NAM: parabola

DES: function used by mpfit to fit a parabola

def boa::Utilities::prettyPrintList (inputList)

DES: Pretty print a list avoiding useless entries

INP: (l) inputList : the input list, does not need to be sorted

OUT: (s) outputString : the resulting string

def boa::Utilities::safeExp (x)

NAM: safeExp (function)

DES: correct a bug in Numerical that raise an exception when
computing exponential of small numbers, this take a lot of time !
but faster thant converting to nummarray compute the exp and back to numeric !!

def boa::Utilities::solvePoly (order, dataX, dataY)

NAM: solvePoly (function)

DES: perform polyomial interpolation: solve linear system
dataY = P_n(dataX)

INP: (int) order : polynomial degree

(flt arrays) dataX/Y : system to solve

OUT: (flt array) coeff : polynomial coefficients

D.3 DeviceHandler

Functions

- def `boa::Bogli::DeviceHandler::closeDev`
- def `boa::Bogli::DeviceHandler::openDev`
- def `boa::Bogli::DeviceHandler::resizeDev`
- def `boa::Bogli::DeviceHandler::selectDev`

D.3.1 Function Documentation

def `boa::Bogli::DeviceHandler::closeDev` (devID = 'current')

DES: close selected device
INP: (int) device ID, 'all', 'current' (default)

def `boa::Bogli::DeviceHandler::openDev` (type = '/XWINDOW')

DES: open a device, return the device id
INP: (string) type = pgplot device type

def `boa::Bogli::DeviceHandler::resizeDev` ()

DES: resize plot area after resizing window with mouse
ABB: resize

def `boa::Bogli::DeviceHandler::selectDev` (devID = "")

DES: select an open device
INP: (int) device ID

D.4 Interface

Classes

- class `boa::Bogli::Interface::Fenetre`

Functions

- def `boa::Bogli::Interface::fenetreInteractive`
- def `boa::Bogli::Interface::pgrstr`

D.4.1 Function Documentation

def `boa::Bogli::Interface::fenetreInteractive` (**x0** = 600., **x1** = 675., **y0** = 400., **y1** = 480,
prevText = "", **bgCol** = 0)

```
method fenetreInteractive():  
INP: (float) x0,x1,y0,y1: box coordinates  
      (str) prevText: previous field value  
      (int) bgCol: background color  
  
OUT: (str) value: user input
```

def `boa::Bogli::Interface::pgrstr` (**X**, **Y**, **ANGLE**, **FJUST**, **TEXT**, **LSTR**, **BCI**)

```
method pgrstr(X, Y, ANGLE, FJUST, TEXT, LSTR, BCI)  
Lit une chaine de caractere dans la fenetre courante  
INP: X, Y, ANGLE, FJUST = position angle et justification du texte  
      BCI = couleur de la fenetre  
INP/OUT: TEXT, LSTR = texte et longueur de la chaine TEXT
```

D.5 MultiPlot

Functions

- def `boa::Bogli::MultiPlot::detSubDivView`
- def `boa::Bogli::MultiPlot::draw`
- def `boa::Bogli::MultiPlot::drawChanNum`
- def `boa::Bogli::MultiPlot::gloLabelling`
- def `boa::Bogli::MultiPlot::plot`
- def `boa::Bogli::MultiPlot::plotBox`
- def `boa::Bogli::MultiPlot::setLimits`
- def `boa::Bogli::MultiPlot::setMultiViewPoint`

D.5.1 Function Documentation

def `boa::Bogli::MultiPlot::detSubDivView` (numPlot, wedge = 0)

DES: Determine sub-division of plot page into viewpoints/boxes.
 INP: (int) numPlot : number of panels in the plot
 OPT: (logical) wedge : if a wedge is present (no by default)
 OUT: (ints) (numPlotWinX,numPlotWinY) the number of viewport in both directions

**def `boa::Bogli::MultiPlot::draw` (chanList, map_arrays, sizeX = [], sizeY = [], WCS = [],
limitsX = [], limitsY = [], limitsZ = [], nan = 0, labelX = 'x', labelY = 'y', caption = ' ', style
 = 'g2r', contrast = 1.0, brightness = 0.5, wedge = 0, overplot = 0)**

DES: do a multi channel image drawing
 INP: (int list) chanList = list of channels
 (map_arrays) lits of map to display
 OPT: (2elts arrays) sizeX/Y = the 'physical' size of the array (default pixel numbers)
 (2elts arrays) limitsX/Y = limits to use in X/Y for the plot
 (string) labelX = x label (default 'x')
 (string) labelY = y label (default 'y')
 (string) caption = the caption of the plot (default ' ')
 (char) style = the color used for the plot (default 'g2r'
 see Plot.Plot.setImaCol())
 (logical) wedge = shall we draw a wedge ? (default no)
 (logical) overplot = are we overplotting ? (default no)

def `boa::Bogli::MultiPlot::drawChanNum` (c)

DES: Draw channel number.
 INT: (int) c : the channel number

def `boa::Bogli::MultiPlot::gloLabelling` (wedge = 0)

DES: Label x, y, caption and channel number.
 OPT: (logical) wedge : if a wedge is present

```
def boa::Bogli::MultiPlot::plot ( chanList, dataX, dataY, limitsX = [], limitsY = [], labelX = 'x',
labelY = 'y', caption = ' ', style = 'p', ci = 1, overplot = 0, logX = 0, logY = 0, nan = 0)
```

```
DES: do a multi channel plot
INP: (int list) chanList = list of channels
      (array list) dataX = values to plot along X
      (array list) dataY = values to plot along Y
OPT: (2elts array) limitsX/Y = limits to use in X/Y for the plot
      (string) labelX = x label (default 'x')
      (string) labelY = y label (default 'y')
      (string) caption = the caption of the plot (default ' ')
      (char) style = the style used for the plot ('l': line,
        'p': point (default), 'b': histogram)
      (int) ci = color index (default 1)
      (logical) overplot = are we overplotting ? (default no)
      (logical) logX/Y = do we use log scale ? (default no)
```

```
def boa::Bogli::MultiPlot::plotBox ( numPlot, x, y)
```

```
DES: Draw box and labels.
INP: (int) numPlot : number of panels in the plot
      (int) x,y : the indices of the panel
```

```
def boa::Bogli::MultiPlot::setLimits ( dataX, dataY, limitsX = [], limitsY = [])
```

```
DES: compute and/or set the limits for the multiplot
INP: (list arrays) dataX/Y : the array to be plotted
OPT: (2elts array) limitsX/Y : limits to use in X/Y for the plot
```

```
def boa::Bogli::MultiPlot::setMultiViewPoint ( x = 0, y = 0, wedge = 0)
```

```
DES: Determine and set view points.
INP: (ints) x/y : position of the viewpoint from 0 to numPlotWinX/Y
```

D.6 Plot

Functions

- def `boa::Bogli::Plot::clear`
- def `boa::Bogli::Plot::draw`
- def `boa::Bogli::Plot::drawLabel`
- def `boa::Bogli::Plot::drawWedge`
- def `boa::Bogli::Plot::erase`
- def `boa::Bogli::Plot::getPixel`
- def `boa::Bogli::Plot::labelling`
- def `boa::Bogli::Plot::plot`
- def `boa::Bogli::Plot::plotBox`
- def `boa::Bogli::Plot::plotDataXY`
- def `boa::Bogli::Plot::readLut`
- def `boa::Bogli::Plot::removeNan`
- def `boa::Bogli::Plot::setImaCol`
- def `boa::Bogli::Plot::setLabels`
- def `boa::Bogli::Plot::setLimits`
- def `boa::Bogli::Plot::setMapLimits`
- def `boa::Bogli::Plot::setMapTransformation`
- def `boa::Bogli::Plot::setViewPoint`
- def `boa::Bogli::Plot::xyout`

D.6.1 Function Documentation

```
def boa::Bogli::Plot::clear ()
```

DES: clear plot

```
def boa::Bogli::Plot::draw ( map_array, sizeX = [], sizeY = [], WCS = [], limitsX = [], limitsY =
[], limitsZ = [], nan = 0, labelX = 'x', labelY = 'y', caption = "", style = 'g2r', contrast = 1.0,
brightness = 0.5, wedge = 0, overplot = 0, aspect = 0, doContour = 0, levels = [], labelContour = 0)
```

DES: do a image drawing

```
INP: (map_array) map to display
```

OPT: (2elts arrays) sizeX/Y = the 'physical' size of the array (default pixel numbers)
 defined by the center of the two extreme pixels !

(2elts arrays) limitsX/Y = limits to use in X/Y for the plot

```
(logical)      nan = set if NaN are present in the array
```

```
(string) labelX = x label (default 'x')
```

```
(string) labelY = y label (default 'y')
```

```
(string)          caption = the caption of the plot (default ' ')
```

```
(char)      style    = the color used for the plot (default 'g2r'
                      see BoqgliPlot.Plot.setImaCol())
```

```
(logical)      wedge    = shall we draw a wedge ? (default no)
```

```
(logical)      aspect = shall we draw a wedge ? (should we,
(logical)      aspect = keep the aspect ratio in 'physical' unit
```

```
(logical)      aspect = keep one aspect factor in plot (default no)
```

```
(logical)    doContour = draw contour instead of map (default no)
```

```

(drawgrid)      = draw contour instead of map (default no)
(array)         = the levels for the contours (default nContour
                  withing plotLimitsZ)

```

```
(logical) labelContour = label the contours (default no)
```

def boa::Bogli::Plot::drawLabel (label)

DES: generic function to draw labels/captions
 INP: (dict) label : a 'label' attribute like labelX

def boa::Bogli::Plot::drawWedge ()

DES: Draw a wedge.

def boa::Bogli::Plot::erase ()

DES: Erase any existing graphics.

def boa::Bogli::Plot::getPixel (order = 0)

DES: allow user to get pixel values using mouse
 INP: (int) order = for polynomial interpolation

def boa::Bogli::Plot::labelling (wedge = 0)

DES: Label x, y, caption and channel number.
 OPT: (logical) wedge : should we draw a wedge ? (default no)

def boa::Bogli::Plot::plot (dataX, dataY = [], limitsX = [], limitsY = [], labelX = 'x', labelY = 'y', caption = "", style = 'p', ci = 1, width = 0, overplot = 0, aspect = 0, logX = 0, logY = 0, nodata = 0)

DES: do a plot
 INP: (array) dataX = values to plot along X
 (array) dataY = values to plot along Y (optional - default:
 plot dataX vs. running number)
 OPT: (2elts array) limitsX/Y = limits to use in X/Y for the plot
 (string) labelX = x label (default 'x')
 (string) labelY = y label (default 'y')
 (string) caption = the caption of the plot (default ' ')
 (char) style = the style used for the plot ('l': line,
 'p': point (default), 'b': histogram)
 (int) ci = color index (default 1)
 (int) width = linewidth (default 0 = use previous)
 (logical) aspect = keep the aspect ratio in 'physical' unit
 (logical) overplot = are we overplotting ? (default no)
 (logical) logX/Y = do we use log scale ? (default no)
 (logical) nodata = do not plot the data

def boa::Bogli::Plot::plotBox ()

DES: Draw box and labels.

def boa::Bogli::Plot::plotDataXY (dataX, dataY, style = 'p', ci = 1, width = 0)

DES: Plot x y data.
 INP: (array) dataX/Y : the array to be plotted (same dimension)
 OPT: (string) style : the style used for the plot ('l': line,
 'p': point (default), 'b': histogram)
 (int) ci : the color index (default 1)
 (int) width : linewidth (default 0 = use previous)

def boa::Bogli::Plot::readLut (lutFile)

NAM: readLut (method)
 DES: read a LUT file
 INP: (string) lutFile : the name of the input lut file

def boa::Bogli::Plot::removeNan (array, value = 0)

DES: replace the Nan value by value
 INP: (array) array : input array
 (float) value : the value to replace

def boa::Bogli::Plot::setImaCol (style = 'g2r', contrast = 1, brightness = 0.5, transferFunction = 0, nan = 0)

NAM: setImaCol (method)
 DES: Set image colours.
 OPT: (string) style : the style (default 'g2r' : green to red)
 also defined 'r2g' red to green, 'b2r' blue to red,
 'r2b' red to blue, 'blue' or any lut file define in
 the BogliConfig.lutDir variable
 (float) contrast : the contrast of the plot (default 1)
 (float) brightness : the brightness of the plot (default 0.5)
 (int) transferFunction : the transfer function (linear/log/sqrt)

def boa::Bogli::Plot::setLabels (labelX, labelY, caption)

DES: check and set labels
 INP: (strings) labelX, labelY, caption : the X and Y labels and the caption

def boa::Bogli::Plot::setLimits (dataX, dataY, limitsX = [], limitsY = [])

DES: compute and/or set the limits for the graph
 INP: (arrays) dataX/Y : the array to be plotted
 OPT: (2elts array) limitsX/Y : limits to use in X/Y for the plot

def boa::Bogli::Plot::setMapLimits (map_array, limitsX = [], limitsY = [], limitsZ = [])

DES: compute and/or set the limits for the map
 INP: (2D array) map_array : the map array
 OPT: (2elts arrays) sizeX/Y : the 'physical' size of the array (default pixel numbers)
 this define the center of the pixels !!


```
(2elts array) limitsX/Y : limits to use in X/Y for the plot  
(2elts array) limitsZ   : the plotted color range in unit of the map_arrays
```

def boa::Bogli::Plot::setMapTransformation (map_array, sizeX = [0 ., sizeY = [0 ., WCS = [])

DES: compute transformation matrix

def boa::Bogli::Plot::setViewPoint (wedge = 0, aspect = 0)

DES: set the view point

OPT: (logical) wedge : if wedge is present (default no)
(logical) aspect : should we keep the aspect ratio (in 'physical' unit)
of the graph (default no)

def boa::Bogli::Plot::xyout (X, Y, text)

DES: generic function to overplot text

INP: (dict) text : a 'text' dictionary like below

E. BOA CLASS DOCUMENTATION

E.1 `boa::BoaError::BoaError` Class Reference

E.1.1 Detailed Description

A class used to generate exceptions related to Boa modules

Public Member Functions

- `def __init__`
- `def __str__`

Public Attributes

- `msg`
- `value`

E.2 boa::BoaDataEntity::BolometerArray Class Reference

E.2.1 Detailed Description

NAM: BolometerArray (class)

DES: Define all the useful parameters of a bolometer array

Public Member Functions

- def [__init__](#)
- def [__str__](#)
- def [checkChanList](#)
- def [computeBeamSize](#)
- def [computeChanSep](#)
- def [computeChanSepValid](#)
- def [fillFromMBFits](#)
- def [get](#)
- def [getChanIndex](#)
- def [getChanSep](#)
- def [plotArray](#)
- def [plotGain](#)
- def [printCurrChanList](#)
- def [readAsciiRcp](#)
- def [readRCPfile](#)
- def [rotateArray](#)
- def [setChannelsNumbers](#)
- def [setCurrChanList](#)
- def [writeRCPfile](#)

E.2.2 Member Function Documentation

def boa::BoaDataEntity::BolometerArray::__init__ (self)

DES: Instanciation of a BolometerArray object

def boa::BoaDataEntity::BolometerArray::__str__ (self)

DES: Defines a string which is shown when the print instruction is used.

def boa::BoaDataEntity::BolometerArray::checkChanList (self, inList)

DES: Return a list of valid channels

INP: (int list/string) inList: list of channel numbers to get, or empty list to get the complete list of unflagged channels, or 'all' or 'al' or 'a' to get the complete list of channels

OUT: (int list) list of channel numbers

def boa::BoaDataEntity::BolometerArray::computeBeamSize (self)

DES: Compute the beam size in arcsec

def boa::BoaDataEntity::BolometerArray::computeChanSep (self)

DES: Compute separation between pixels (in arcsec)

def boa::BoaDataEntity::BolometerArray::computeChanSepValid (self)

DES: Compute separation between VALID (i.e. not flagged -1) pixels (in arcsec)

def boa::BoaDataEntity::BolometerArray::fillFromMBFits (self, reader, febe, baseband, subscan)

NAM: fillFromMBFits()

DES: fill a BolometerArray object using the MBFitsReader object reader.

Calling sequence: DataEntity.fillFromMBFits(obsEntity)

INP: obsEntity: *LIST* of objects of the Entities.ObsEntity class

update (logical) if true, do not reset previous entity object

def boa::BoaDataEntity::BolometerArray::get (self, dataType, flag = 0, inverse = 0)

DES: get bolometers offsets or gain according to flag

INP: (string) dataType : type of data

(int) flag : retrieve data flagged with flag

(default 0 : good data, 'None' for all)

(log) inverse : retrieve all data without given flag set instead (default no)

OUT: (float array) : the requested data

def boa::BoaDataEntity::BolometerArray::getChanIndex (self, chanList = [])

DES: convert from physical channel number to index in UsedChannel

INP: (i list) chanList : the physical channel number

OUT: (i list) : the corresponding index (-1 if failed)

def boa::BoaDataEntity::BolometerArray::getChanSep (self, chanList = [])

DES: return the channel separation in both direction from the reference channel

def boa::BoaDataEntity::BolometerArray::plotArray (self, overplot = 0, num = 0, limitsX = [], limitsY = [], ci = 3)

DES: plot the receiver parameters

INP: (optional) overplot (logical) = overplot?

(optional) num (logical) = indicate chan numbers?

def boa::BoaDataEntity::BolometerArray::plotGain (self, style = 'g2r')

DES: plot the gain of the Array
 INP: (str) style : the style to be used (default g2r)
 WAR: the bolometer without know offsets should be flagged

def boa::BoaDataEntity::BolometerArray::printCurrChanList (self)

DES: print the current channel list in somehow "clever" way
 OUT: a string representing the current channel list

def boa::BoaDataEntity::BolometerArray::readAsciiRcp (self, filename)

DES: update receiver channel offsets from a simple ascii file
 channelNumber AzOffset(arcsec) ElOffset(arcsec)
 INP: (string) filename: the filename to read in

def boa::BoaDataEntity::BolometerArray::readRCPfile (self, rcpFile)

NAM: readRCPfile (method)
 DES: update Receiver Channel Parameters (attributes Offsets,
 Gain and ChannelSep) from the content of a file
 INP: (string) rcpFile: complete name of file to read in

def boa::BoaDataEntity::BolometerArray::rotateArray (self, elevation)

DES: rotate array offsets according to elevation
 INP: (float) elevation (in degree)

def boa::BoaDataEntity::BolometerArray::setChannelsNumbers (self, nChannels, UsedChannels = [])

DES: set the different array to default value
 INP: (int) nChannels : the number of channels

def boa::BoaDataEntity::BolometerArray::setCurrChanList (self, chanList = ' ? ')

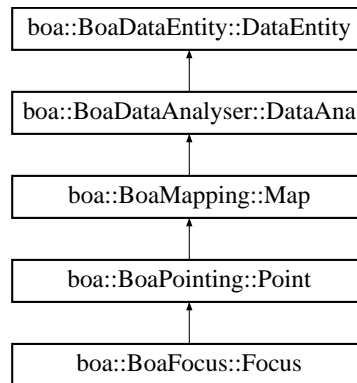
DES: set list of channels to be treated
 INP: (int list/string) chanList = list of channels, or string '?'
 to get current list of channels, or string 'a' or 'al' or 'all'
 to set current list to all possible channels. Default: '?'

def boa::BoaDataEntity::BolometerArray::writeRCPfile (self, rcpFile = 'rcpBoa.rcp')

NAM: writeRCPfile (method)
 DES: store current Receiver Channel Parameters (Offsets,
 Gain) to a file with mopsi like format
 INP: (string) rcpFile: complete name of output file

E.3 boa::BoaDataAnalyser::DataAna Class Reference

Inheritance diagram for boa::BoaDataAnalyser::DataAna::



E.3.1 Detailed Description

DES: An object of this class is responsible for the flagging of individual channels, i.e. it sets the values in the Channel_Flag array of the corresponding DataEntity object. It provides methods to derive the rms of each channel and to automatically search for bad or noisy channels. Channels might be flagged according to a given input file. This object provides methods to derive the correlation matrix.

Public Member Functions

- def [__init__](#)
- def [basePoly](#)
- def [basePolySubscan](#)
- def [computeCorMatrix](#)
- def [computeSN](#)
- def [computeWeight](#)
- def [correlate](#)
- def [correlate2](#)
- def [despike](#)
- def [flag](#)
- def [flagChannels](#)
- def [flagInTime](#)
- def [flagLon](#)
- def [flagLST](#)
- def [flagPosition](#)
- def [flagRms](#)
- def [flagSubscan](#)
- def [medianBaseline](#)
- def [medianFilter](#)
- def [plotCorMatrix](#)
- def [plotFFT](#)

- def [plotMean](#)
- def [plotMeanChan](#)
- def [plotRms](#)
- def [plotRmsChan](#)
- def [slidingRms](#)
- def [statistics](#)
- def [unflag](#)
- def [unflagChannels](#)

E.3.2 Member Function Documentation

def [boa::BoaDataAnalyser::DataAna::__init__](#) ([self](#))

DES: initialise an instance

Reimplemented from [boa::BoaDataEntity::DataEntity](#).

Reimplemented in [boa::BoaFocus::Focus](#), [boa::BoaMapping::Map](#), and [boa::BoaPointing::Point](#).

def [boa::BoaDataAnalyser::DataAna::basePoly](#) ([self](#), [chanList](#) = [], [order](#) = 0, [subscan](#) = 0, [plot](#) = 0, [subtract](#) = 1)

DES: polynomial baseline method for single scans or subscans

INP: (i list) channel : list of channel to flag (default: all; [] : current list)

- (i) order : polynomial order, >0
- (l) subscan : compute baseline per subscan (default: no)
- (l) plot : plot the signal and the fitted polynomials (default: no)
- (l) subtract : subtract the polynomial from the data (default: yes)

def [boa::BoaDataAnalyser::DataAna::basePolySubscan](#) ([self](#), [chanList](#) = [], [order](#) = 0, [plot](#) = 0, [subtract](#) = 1)

DES: polynomial baseline method, treat each subscan.

INP: (i list) channel : list of channel to flag (default: all; [] : current list)

- (i) order : polynomial order, >0
- (l) plot : plot the signal and the fitted polynomials (default: no)
- (l) subtract : subtract the polynomial from the data (default: yes)

def [boa::BoaDataAnalyser::DataAna::computeCorMatrix](#) ([self](#))

DES: compute correlation matrix. CM=1 for identical signals, and small symmetric around 0 for uncorrelated noise.

CM_nm = <D_n*D_m> / [rms(D_n)*rms(D_m)]

CM has dimension number_of_valid_channels^2

def [boa::BoaDataAnalyser::DataAna::computeSN](#) ([self](#), [subtract](#) = 0)

DES: compute correlated noise, run after computeCorMatrix, computeWeight, correlate

INP: (i) subtract=0 : subtract correlated noise from Data

```
def boa::BoaDataAnalyser::DataAna::computeWeight ( self, minCorr = 0., wa = 0.95, wb = 1.0,
core = 100., beta = 1.)
```

DES: compute weight matrix of the used channels, run after computeCorMatrix
weight is non-linear rescaling of correlation coefficient

```
weight_nm = ( CM_nm - wa * min_m( CM_nm ) )**wb
```

an additionnal weighting factor is applied with channel separation

```
weight_nm = weight_nm * 1.0 / ( 1 + ( dist_nm / core )**beta )
```

INP: (f) minCorr = minimum correlation coefficient (default:0, should be positiv)
(f) wa = parameter for weights, usually = 0.90-0.98
(f) wb = parameter for weights, usually = 1
(f) core = core radius in arcmin for radial weighting (weight = 0.5)
(f) beta = beta for beta profile for radial weighting

```
def boa::BoaDataAnalyser::DataAna::correlate ( self, channel = 97, chanList = [], skynoise = 0,
plot = 0, minSlope = 0.1, maxSlope = 10.0)
```

DES: compute correlation factor relative to given reference channel

INP: (i) channel=114 : reference channel against which to correlate and/or plot
(l i) chanList=[] : list of channel to correlate (default current list)
(l) skynoise=0 : correlate skynoise[channel] not signal[channel] to signal
(l) plot=0 : plot the correlation and the fit
(f) minSlope=0.1 : limit slope of least squares fit
(f) maxSlope=10 : limit slope of least squares fit

```
def boa::BoaDataAnalyser::DataAna::correlate2 ( self, channel = 97, chanList = [], skynoise = 0,
plot = 0, minSlope = 0.1, maxSlope = 10.0)
```

DES: compute correlation factor relative to given reference channel

INP: (i) channel=114 : reference channel against which to correlate and/or plot
(l i) chanList=[] : list of channel to correlate (default current list)
(l) skynoise=0 : correlate skynoise[channel] not signal[channel] to signal
(l) plot=0 : plot the correlation and the fit
(f) minSlope=0.1 : limit slope of least squares fit
(f) maxSlope=10 : limit slope of least squares fit

```
def boa::BoaDataAnalyser::DataAna::despike ( self, chanList = [], below = -5, above = 5, flag = 2)
```

DES: Flag yet unflagged data below 'below'*rms and above 'above'*rms.

INP: (i list) chanList : list of channel to flag (default: current list)
(f) below : flag data with value < 'below'*rms
(f) above : flag data with value > 'above'*rms

```
def boa::BoaDataAnalyser::DataAna::flag ( self, dataType = "", channel = 'all', below = '??',
above = '??', flag = 3)
```

DES: flag data based on dataType, general flagging routine, may be slow

INP: (s) dataType : flag based on this dataType
(i list) channel : list of channel to flag (default: all)
(f) below : flag dataType < below (default max; or 5*RMS)
(f) above : flag dataType > above (default min; or -5*RMS)

(i) flag : flag value (default 1 - 0 to unflag)

below and above should be in unit of the flagged data,
except for 'Lon' and 'Lat' where they should be in arcsec

def boa::BoaDataAnalyser::DataAna::flagChannels (self, chanList = [], flag = 1)

DES: assign flags to a list of channels
To unflag a channel simply flag with flag=0
INP: (i list) chanList : list of channels to be flagged (default current list)
(i) flag : flag value

**def boa::BoaDataAnalyser::DataAna::flagInTime (self, dataType = 'LST', channel = 'all',
below = '??', above = '??', flag = 1)**

DES: Flag data in time interval
INP: (int list) channel = list of channel to flag (default: 'all')
(float) below = flag data below this value (default end of the scan)
(float) above = flag data above this value (default start of the scan)
(int) flag = flag to be set (default 1)

**def boa::BoaDataAnalyser::DataAna::flagLon (self, channel = 'all', below = '??', above = '??',
flag = 1)**

NAM: flagLon (method)
DES: Flag data in Longitude interval
INP: (int list) channel = list of channel to flag (default: all)
(float) below = flag data below this value (default end of the scan)
(float) above = flag data above this value (default start of the scan)
(int) flag = flag to be set (default 1)

**def boa::BoaDataAnalyser::DataAna::flagLST (self, channel = 'all', below = '??', above = '??',
flag = 1)**

DES: Flag data in time interval
INP: (int list) channel = list of channel to flag (default: 'all')
(float) below = flag data below this value (default end of the scan)
(float) above = flag data above this value (default start of the scan)
(int) flag = flag to be set (default 1)

**def boa::BoaDataAnalyser::DataAna::flagPosition (self, channel = 'all', Az = 0, El = 0, radius =
0, flag = 5, offset = 1)**

DES: flag a position in the sky within a given radius
INP: (int list) channel : list of channel to flag (default: 'all')
(float) Az/El : the horizontal reference position (arcsec for offsets, deg for absolute)
(float) radius : aperture to flag in unit of the reference position
(int) flag : flag to be set (default 5)
(logical) offset : flag on the offsets (default yes,)

```
def boa::BoaDataAnalyser::DataAna::flagRms ( self, chanList = [], below = 0, above = 1e10, flag = 1)
```

```
DES: Flag channels with rms below 'below' or above 'above'
INP: (i list) chanList : list of channel to flag (default: current list)
      (f)      below      :flag channels with rms < 'below'
      (f)      above      :flag channels with rms > 'above'
```

```
def boa::BoaDataAnalyser::DataAna::flagSubscan ( self, subList, flag = 1)
```

```
DES: flag subscans
INP: (int list) subList = list of subscan numbers (or single number)
      to be flagged
      (int) flag      = value of flags to set
```

```
def boa::BoaDataAnalyser::DataAna::medianBaseline ( self, chanList = [], subscan = 1)
```

```
DES: baseline: Remove median value per channel and per subscan
INP: (i list) channel : list of channels to process (default: [] = current list)
      (l) subscan      : compute baseline per subscan (default: yes)
```

```
def boa::BoaDataAnalyser::DataAna::medianFilter ( self, chanList = [], window = 20, subtract = 1)
```

```
DES: median filtering: remove median values computed over sliding window
INP: (i list) channel: list of channels to process (default: [] = current list)
      (i)      window: number of samples to compute median
      (l)      subtract: subtract from data? (default: yes)
```

```
def boa::BoaDataAnalyser::DataAna::plotCorMatrix ( self, chanList = [], check = 1, distance = 0, weights = 0, xLabel = 'Channels', style = 'g2r')
```

```
DES: plot the correlation matrix
INP: (i list) chanList : the list of channel to plot
      (l)      check : check the chanList first ( default : yes)
      (l)      distance : sort the second dimension by distance (default : no)
      (l)      weights : plot weights instead of correlation matrix (default: no)
```

```
def boa::BoaDataAnalyser::DataAna::plotFFT ( self, chanList = [], flag = 0, optimize = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, window = 0)
```

```
DES: plot FFT of signal
INP: (i list) chanList : list of channels
      (i)      flag : flag to be used (default: 0, noflagged data)
      (l)      optimize : optimize or not (default) see FilerFFT.doFFT
      limits, style, ci...: plot parameters (see MultiPlot.plot)
      (i)      window : windowing function to apply (see FilerFFT.windowing)
```

```
def boa::BoaDataAnalyser::DataAna::plotMean ( self, chanList = [], flag = 0, limitsX = [], limitsY
= [], style = 'l', ci = 1, overplot = 0, map = 0)
```

DES: plot mean flux value vs. subscan number
 TODO: flag handling not implemented yet
 INP: (int list) chanList = list of channels
 (int) flag = flag to be used
 (logical) map = plot as a 2D map?

```
def boa::BoaDataAnalyser::DataAna::plotMeanChan ( self, chanList = [], flag = 0, limitsX = [],
limitsY = [], style = 'p', ci = 1, overplot = 0)
```

DES: PLOtting the MEAN value for each subscan against channel number.

```
def boa::BoaDataAnalyser::DataAna::plotRms ( self, chanList = [], flag = 0, limitsX = [], limitsY
= [], style = 'l', ci = 1, overplot = 0, map = 0)
```

DES: plot flux r.m.s. vs. subscan number
 TODO: flag handling not implemented yet
 INP: (int list) chanList = list of channels
 (int) flag = flag to be used
 (logical) map = plot as a 2D map?

```
def boa::BoaDataAnalyser::DataAna::plotRmsChan ( self, chanList = [], flag = 0, limitsX = [],
limitsY = [], style = 'p', ci = 1, overplot = 0, subscan = 0)
```

DES: PLOtting the RMS value for each subscan against channel number.
 INP: (logical) subscan: if 0, plot rms of the complete scan, if 1,
 plot for each subscan and each channel

```
def boa::BoaDataAnalyser::DataAna::slidingRms ( self, nbInteg = 10, channel = [], flag = 0)
```

NAM: slidingRms (method)
 DES: compute rms in a sliding window
 INP: (int) nbInteg : number of elements on which one rms is computed (= window size)
 (i list) channel : list of channel to flag (default: all; [] : current list)
 (int) flag : data flag to be used
 OUT: (array) the rms are returned

```
def boa::BoaDataAnalyser::DataAna::statistics ( self)
```

NAM: statistics (method)
 DES: computes mean, median, rms for all scans and subsamples for all used channels

```
def boa::BoaDataAnalyser::DataAna::unflag ( self, channel = [], flag = 1)
```

NAM: unflag (method)
 DES: Unflag data, i.e. reset to 0.
 INP: (i list) channel : list of channel to flag (default: current list)
 (i) flag : unflag only this value (default 1)

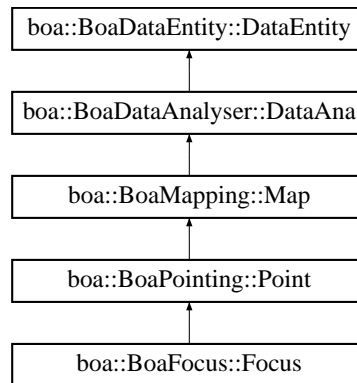
def boa::BoaDataAnalyser::DataAna::unflagChannels (self, chanList = [])

DES: unflags a list of channels

INP: (i list) chanList : list of channels to be unflagged (default current list)

E.4 boa::BoaDataEntity::DataEntity Class Reference

Inheritance diagram for boa::BoaDataEntity::DataEntity::



E.4.1 Detailed Description

NAM: DataEntity (class)

DES: Objects of this class store the data and associated parameters of a scan, which can contain several observations (or subscans).

They also contain additional arrays in which the current results of the data reduction are stored.

This class also provides the interface between the MB-FITS files and BoA, by the means of the `fillFromMBFits()` method.

Public Member Functions

- def `__init__`
- def `__str__`
- def `backup`
- def `computeSubIndex`
- def `dumpData`
- def `existData`
- def `fillFromMBFits`
- def `getChanData`
- def `getChanListData`
- def `phaseDiff`
- def `plotCorrel`
- def `read`
- def `reset`
- def `restore`
- def `restoreData`
- def `saveMambo`
- def `selectPhase`
- def `signal`
- def `writeMBfits`

E.4.2 Member Function Documentation

def boa::BoaDataEntity::DataEntity::__init__ (self)

DES: Instanciation of a new DataEntity object.
All attributes are defined and set to default values.

Reimplemented in [boa::BoaDataAnalyser::DataAna](#), [boa::BoaFocus::Focus](#), [boa::BoaMapping::Map](#), and [boa::BoaPointing::Point](#).

def boa::BoaDataEntity::DataEntity::__str__ (self)

DES: Defines a string which is shown when the print instruction is used. It contains the sizes and typecodes of all attributes.

def boa::BoaDataEntity::DataEntity::backup (self)

DES: backup the data

def boa::BoaDataEntity::DataEntity::computeSubIndex (self)

NAM: computeSubIndex (method)
DES: Compute start and end indices per subscan

def boa::BoaDataEntity::DataEntity::dumpData (self, fileName = 'BoaData.sav')

DES: save the current DataEntity object to a file
INP: (string) fileName: name of the output file
optional - default value = 'BoaData.sav'

def boa::BoaDataEntity::DataEntity::existData (self)

DES: check if the DataEntity object has been filled with data
OUT: (int) result: 0 if no data, 1 otherwise

def boa::BoaDataEntity::DataEntity::fillFromMBFits (self, reader, febe, baseband, subscans, update)

NAM: fillFromMBFits()
DES: fill a DataEntity object using the MBFitsReader object reader.

Calling sequence: DataEntity.fillFromMBFits(obsEntity)
INP: obsEntity: *LIST* of objects of the Entities.ObsEntity class
update (logical) if true, do not reset previous entity object

```
def boa::BoaDataEntity::DataEntity::getChanData ( self, dataType = ' ', chan = 'None', flag = 0, subscans = [], inverse = 0, flag2 = None)
```

```
DES: get data for one channel
INP: (string)  dataType : type of data
      (int)      chan : channel number
      (int)      flag : data flag to be used
      (int list) subscans : list of wanted subscan (default all)
OPT: (log)      inverse : if set, return datapoints where flag <> value
      (int array) flag2 : second array of flags to check
OUT: (float)     array : data of one channel
```

```
def boa::BoaDataEntity::DataEntity::getChanListData ( self, type = ' ', chanList = [], flag = 0, flag2 = None)
```

```
DES: get data for list of channels
INP: (string) type = type of data
      (int list) chan = channel list
      (int)      flag = data flag to be used
      (int array) flag2 = second array of flags to check (optional)
OUT: (list of float arrays) = data of the input list of channels
```

```
def boa::BoaDataEntity::DataEntity::phaseDiff ( self)
```

```
NAM: phaseDiff (method)
DES: Compute phase differences: call ScanParam.phaseDiffParam for
      coordinates and times, and compute Data(ON) - Data(OFF)
```

```
def boa::BoaDataEntity::DataEntity::plotCorrel ( self, chanRef = 1, chanList = [], flag = 0, skynoise = 0, limitsX = [], limitsY = [], style = 'p', ci = 1, overplot = 0)
```

```
DES: plot flux density of a list of channels vs. flux density of a
      reference channel
INP: (int)      chanRef = reference channel number
      (int list) chanList = list of channels
      (int)      flag = flag to be used
      (l)        skynoise = plot against the skynoise of chanRef (default : no)
```

```
def boa::BoaDataEntity::DataEntity::read ( self, inFile = "", febe = "", baseband = 0, subscans = [], update = 0, dataset = None, phase = 0)
```

```
DES: fill a data entity object
INP: (int list) subscans : subscan numbers to read (default: all)
      (logical) update : if true, do not reset previous entity object
      (object) dataset : can be already opened (optional)
      (int) phase : phase to be stored (default: phase diff)
```

```
def boa::BoaDataEntity::DataEntity::reset ( self)
```

```
DES: Reset all attributes - useful before reading a new file
```

def boa::BoaDataEntity::DataEntity::restore (self)

DES: backup the data

def boa::BoaDataEntity::DataEntity::restoreData (self, fileName = 'BoaData.sav')

DES: restore a DataEntity object previously saved in a file, and
set it as the currData attribute of BoaB
INP: (string) fileName: name of the input file
optional - default value = 'BoaData.sav'

def boa::BoaDataEntity::DataEntity::saveMambo (self, inName = "", outName = "")

DES: convert an MB-Fits file to the MAMBO FITS format, readable
by MOPSIIC
INP: (str) inName: name of the MB-Fits file (optional)
(str) outName: name of the MAMBO output file (optional)

def boa::BoaDataEntity::DataEntity::selectPhase (self, phase)

NAM: selectPhase (method)
DES: Keep only Data(ON) or Data(OFF)
INP: (int) phase: phase to keep, 1=ON, 2=OFF

def boa::BoaDataEntity::DataEntity::signal (self, chanList = [], flag = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, plotMap = 0, mjd = 0)

DES: plot time series of flux density
INP: (int list) chanList = list of channels
(int) flag = flag to be used
(logical) mjd = if set, use mjd instead of lst

def boa::BoaDataEntity::DataEntity::writeMBfits (self, outName)

NAM: writeMBfits (method)
DES: write the data (and parameters) contained in the current data
out to a FITS file in MB-Fits format
INP: outName (str) = output file name (.fits extension may be omitted)

E.5 boa::Bogli::Interface::Fenetre Class Reference

E.5.1 Detailed Description

```

classe Fenetre - parametres et methodes pour les boites et boutons
attributs:
int forme      : 0=cercle 1=rectangle 2=rectangle transparent
list pos       : positions (X,Y) des centres dans la fenetre
list float/tuple size: rayon ou (largeur,hauteur)
list label     : messages a apparaitre (vecteur de string) dans ou pres d'une fenetre
tuple txtpos   : position des labels relativement a pos
int font       : taille des caracteres
int coltxt     : couleur du texte
int colfond    : couleur de fond des boutons
int family     : police de caracteres

```

Public Member Functions

- def `__init__`
- def `dessine`
- def `saisie`

Public Attributes

- `colfond`
- `coltxt`
- `family`
- `font`
- `forme`
- `label`
- `pos`
- `size`
- `txtpos`

E.5.2 Member Function Documentation

def `boa::Bogli::Interface::Fenetre::dessine (self, new = 0)`

method `dessine`

INP: `new` : efface la fenetre si non nul
 OUT: aucune

def `boa::Bogli::Interface::Fenetre::saisie (self)`

method `saisie`

INP: aucune
 OUT: `choix` : selection (numero du bouton)

E.6 boa::BoaDataAnalyser::FilterFFT Class Reference

E.6.1 Detailed Description

DES: To easily do FFT filtering

INF: make the assumption that the input signal is real, so do not care about negative frequencies...

Public Member Functions

- def `__init__`
- def `blankAmplitude`
- def `doFFT`
- def `invFFT`
- def `plotfft`
- def `rebin`
- def `unbin`
- def `windowing`

Public Attributes

- `amplitude`
- `freq`
- `interpX`
- `interpY`
- `N`
- `phase`
- `X`
- `Y`

E.6.2 Member Function Documentation

def `boa::BoaDataAnalyser::FilterFFT::blankAmplitude` (`self`, `below` = ' ? ' , `above` = ' ? ')

DES: blank the amplitude below and/or after a certain frequency

def `boa::BoaDataAnalyser::FilterFFT::doFFT` (`self`, `optimize` = 0)

DES: perform the FFT

INP: (i) `optimize` : 0, will use the full data set (default)
1, will zero-pad the data til the next power of 2

def `boa::BoaDataAnalyser::FilterFFT::invFFT` (`self`)

DES: perform the inverse FFT

```
def boa::BoaDataAnalyser::FilterFFT::plotfft ( self, plotPhase = 0, labelX = 'Frequency [Hz]',  
labelY = 'Amplitude (a.b.u/sqrt(Hz))', \                                limitsX=[], limitsY =  
[], logX = 0, logY = 0, overplot = 0, ci = 1)
```

DES: Plot the fft

INP: (str) labelX/Y : the X/Y label

(2d f) limitsX/Y : the plot limits for X/Y

(bol) plotPhase : plot phase instead of amplitude (default no)

```
def boa::BoaDataAnalyser::FilterFFT::rebin ( self, interval = 0)
```

DES: linearly interpolate the starting value for the FFT

INP: (f) interval : force the interval (default : median)

```
def boa::BoaDataAnalyser::FilterFFT::unbin ( self)
```

DES: Rebin the data to the initial grid

```
def boa::BoaDataAnalyser::FilterFFT::windowing ( self, type = 1, undo = 0)
```

DES: apply some window on the signal.

INP: (i) type : the type of desired windowing

1 - Barlett (default)

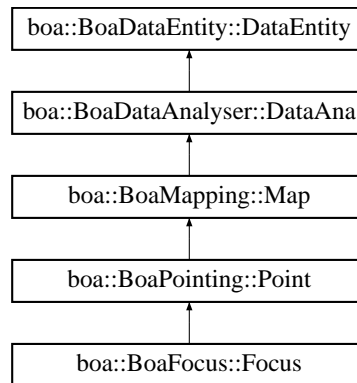
2 - Hann

3 - Welch

(l) undo : if set, divide by the window function (default: multiply)

E.7 boa::BoaFocus::Focus Class Reference

Inheritance diagram for `boa::BoaFocus::Focus`:



E.7.1 Detailed Description

NAM: `Focus` (class)

DES: An object of this class is responsible for the focus reduction of single or multiple scans and provides the offsets.

Public Member Functions

- `def __init__`
- `def reduce`
- `def solveFocus`

E.7.2 Member Function Documentation

def `boa::BoaFocus::Focus::__init__` (`self`)

DES: Initialise an instance

Reimplemented from `boa::BoaPointing::Point`.

def `boa::BoaFocus::Focus::reduce` (`self`, `FitsFile` = None, `obstoProc` = [])

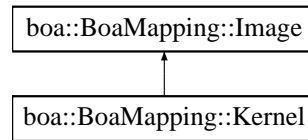
DES: Process a Focus scan - this method is called by the apexCalibrator
 INP: (mbfits.FitsFile) `FitsFile` : object with pointer to an open FITS file
 (i list) `obstoProc`: list of subscans to consider (default: all)

def `boa::BoaFocus::Focus::solveFocus` (`self`)

DES: compute the optimal focus position

E.8 boa::BoaMapping::Image Class Reference

Inheritance diagram for boa::BoaMapping::Image::



E.8.1 Detailed Description

NAM: Image (class)

DES: An object of this class describes an image and its axis

Public Member Functions

- def `__init__`
- def `computeWCS`
- def `display`
- def `physicalCoordinates`
- def `smoothBy`
- def `smoothWith`
- def `wcs2phy`
- def `wcs2pix`

Public Attributes

- `data`
- `rms`
- `WCS`
- `weight`

E.8.2 Member Function Documentation

def `boa::BoaMapping::Image::computeWCS (self, pixelSize, sizeX = [], sizeY = [], minmax = [])`

DES: fill main WCS keywords according to pixel size and map limits

INP: (int) pixelSize = size of pixel in arcsecond
 (float) sizeX = map limits in azimuth, in arcsecond
 (float) sizeY = map limits in elevation, in arcsecond
 (float) minmax = [minAzoff,maxAzoff,minEloff,maxEloff] in this order

def `boa::BoaMapping::Image::display (self, rms = 0, weight = 0, style = 'g2r', labelX = "\gD Az ["]", labelY = "\gD El ["]", caption = "", wedge = 1, aspect = 0, overplot = 0, doContour = 0, levels = [], labelContour = 0, limitsX = [], limitsY = [], limitsZ = [])`

DES: show the reconstructed maps in (Az,El)

INP: (boolean) rms,weight : plot the rms or weight map instead of signal map

```

(string) style           : the style used for the color (default g2r)
(string) labelX, labelY : the X and Y labels
(string) caption         : the caption of the plot (default '')
(flt array) limitsX/Y/Z : the limits in X/Y/intensity
(boolean) wedge          : draw a wedge ? (default : yes)
(boolean) aspect         : keep the aspect ratio (default : no)
(boolean) overplot       : should we overplot this image (default : no)
(boolean) doContour      : draw contour instead of map (default : no)
(float array) levels     : the levels of the contours (default : intensity progression)
(boolean) labelContour   : label the contour (default : no)

```

def boa::BoaMapping::Image::physicalCoordinates (self)

DES: return arrays with physical units corresponding to the map

def boa::BoaMapping::Image::smoothBy (self, beamSize)

DES: smooth the image with a 2D gaussian of gived FWHM
 INP: (float) beamSize : the FWHM of the smoothing gaussian

def boa::BoaMapping::Image::smoothWith (self, kernel)

DES: smooth the image with the given kernel
 INP: (kernel) : the kernel

def boa::BoaMapping::Image::wcs2phy (self, i, j)

DES: Convert from pixel coordinates to physical (world) coordinates
 INP: float (i,j) : the pixel coordinate to convert from
 OUT: float (X,Y) : the physical coordinate

We should switch to libwcs at some point

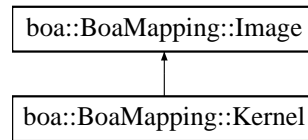
def boa::BoaMapping::Image::wcs2pix (self, X, Y)

DES: Convert from physical coordinate describe by self.WCS to pixel coordinate
 INP: float (X,Y) : the physical coordinate to convert from
 OUT: float (i,j) : the pixel coordinate(s)

We should switch to libwcs at some point

E.9 boa::BoaMapping::Kernel Class Reference

Inheritance diagram for boa::BoaMapping::Kernel::



E.9.1 Detailed Description

NAM: Kernel (class)
DES: define a kernel

Public Member Functions

- def [__init__](#)

E.9.2 Member Function Documentation

def `boa::BoaMapping::Kernel::__init__ (self, pixelSize, beamSize)`

DES: Initialise an instance of a Kernel class
INP: (float) pixelSize: the physical size of a pixel
 (float) beamSize : the beam FWHM in the same unit

E.10 boa::BoaMessageHandler::Logger Class Reference

E.10.1 Detailed Description

NAM: `Logger` (class)

DES: for compatiliby with the `CalibratorLog.Logger` class

Public Member Functions

- def [__init__](#)

E.10.2 Member Function Documentation

def `boa::BoaMessageHandler::Logger::__init__ (self, logType = 'ACS')`

DES: Initiabise an instance

E.11 boa::MamboMBFits::MamboMBFits Class Reference

E.11.1 Detailed Description

DES: Objects of this class contain two attributes of the FitsFile class. The 'Mambo' attribute stores the data as they appear in a MAMBO FITS file, and the 'MBfits' attribute contains the data in the MB-FITS format. Methods to do conversions in both directions are provided by this class.

Public Member Functions

- def [__init__](#)
- def [convertFebepar](#)
- def [convertMambo2MBFits](#)
- def [convertMamboPrimary](#)
- def [convertMB2MamboFits](#)
- def [ctype2sbas](#)
- def [fillFebepar](#)
- def [fillMamboData](#)
- def [fillMamboPrimary](#)
- def [getObsMode](#)
- def [getScanType](#)
- def [initMambo](#)
- def [initMB](#)
- def [processMamboData](#)
- def [readMambo](#)
- def [readMBfits](#)
- def [readRCP](#)
- def [sbas2ctype](#)

E.11.2 Member Function Documentation

def boa::MamboMBFits::MamboMBFits::__init__ (self, mamboName, mbName)

DES: Instanciation of a new MamboMBFits object.

INP: (str) mamboName = name of the Mambo file

(str) mbName = name of the MB-FITS file

in both cases, the .fits extension is appended if not present

def boa::MamboMBFits::MamboMBFits::convertFebepar (self)

DES: This method generates a RCP file, where the relative gains and offsets of pixels are stored, from the content of the FEBEPAR table. The output file name is <FE_name>.rcp.

def boa::MamboMBFits::MamboMBFits::convertMambo2MBFits (self)

DES: This function reads in the content of a Mambo FITS file, and writes out the data and associated parameters to a file conforming to the MB-FITS format.

def boa::MamboMBFits::MamboMBFits::convertMamboPrimary (self)

DES: This generates in the MB-FITS file all the header keywords that have one direct equivalent in the MAMBO primary header.

def boa::MamboMBFits::MamboMBFits::convertMB2MamboFits (self)

DES: This function reads in the content of an MB-FITS file, and writes out the data to a file in the Mambo-FITS format, and the associated parameters to a RCP file.

def boa::MamboMBFits::MamboMBFits::ctype2sbas (self)

DES: Converts the infos about the Astronomical basis frame from MB-FITS keywords (CTYPEj, EQUINOX...) to a SBAS value (+ epoch). The results are stored and returned in a dictionary.

def boa::MamboMBFits::MamboMBFits::fillFebepar (self)

DES: This method writes one row in the FEBEPAR-MBFITS table, by reading the content of a Mambo RCP file, specified by the total number of pixels (40 or 120).

def boa::MamboMBFits::MamboMBFits::fillMamboData (self)

DES: This methods fills the subscans and data tables in an output MAMBO file, using the data previously read in from an MB-FITS file.

def boa::MamboMBFits::MamboMBFits::fillMamboPrimary (self)

DES: Update the keyword values in the Mambo Primary header, using the equivalent keywords found in the MB-FITS file.

def boa::MamboMBFits::MamboMBFits::getObsMode (self)

DES: Convert the scanType (from MB-FITS) to OBSMODE + SRP1FLAG + SRP2FLAG as defined in the MAMBO format.

def boa::MamboMBFits::MamboMBFits::getScanType (self)

DES: Convert the informations about the observing type from Mambo format (contained in OBSMODE + SRP1FLAG + SRP2FLAG) to an MB-FITS SCANTYPE. Also return the SCANMODE, SCANGEOM, SWITCHMOD, WOBSW, OBSTYPE and SCANDIR infos. The results are stored in the Scantype attribute, which is a Dictionary.

def boa::MamboMBFits::MamboMBFits::initMambo (self)

DES: Create a file in the Mambo FITS format. This generates only the Primary header and subscan table, because the number of feeds must be written in the Primary header before the data table is created.

def boa::MamboMBFits::MamboMBFits::initMB (self)

DES: This generates the first three tables in the MB-FITS file (Primary header, SCAN-MBFITS and FEBEPAR-MBFITS tables).

def boa::MamboMBFits::MamboMBFits::processMamboData (self)

DES: This method generates ARRAYDATA, DATAPAR and MONITOR tables in the MB-FITS file for every subscan in the MAMBO file.

def boa::MamboMBFits::MamboMBFits::readMambo (self)

DES: This fills the TableList in the Mambo attribute with the content of the MAMBO FITS file.

def boa::MamboMBFits::MamboMBFits::readMBfits (self)

DES: This fills the TableList in the MBfits attribute with the content of the MB-FITS file.

def boa::MamboMBFits::MamboMBFits::readRCP (self)

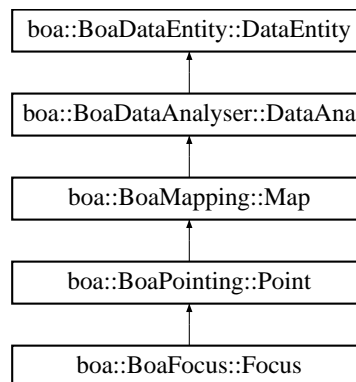
DES: Read the Receiver Channels Parameters from the file 'MRT_2002a_120.rcp' if the number of bolometers is 120, or 'MRT_2002s2_40.rcp' if it is 40. These files are supposed to be in the local directory. Returns a list of tuples, where each tuple is (Gain,X_off,Y_off).

def boa::MamboMBFits::MamboMBFits::sbas2ctype (self)

DES: Converts the SBAS value to the MB-FITS keywords that define the Astronomical basis frame: CTYPEj, WCSNAME, RADESYS, EQUINOX, and eventually MOVEFRAM. All these are stored in a dictionary.

E.12 boa::BoaMapping::Map Class Reference

Inheritance diagram for `boa::BoaMapping::Map`:



E.12.1 Detailed Description

NAM: `Map` (class)

DES: An object of this class is responsible for the restoration of mapping data of single or multiple files.

Public Member Functions

- `def __init__`
- `def beamMap`
- `def eq_to_ho`
- `def fastChanMap`
- `def fastChanMap2`
- `def fastMap`
- `def fastMap2`
- `def getPixel`
- `def showMap`
- `def slowMap`

E.12.2 Member Function Documentation

`def boa::BoaMapping::Map::__init__ (self)`

DES: Initialise an instance.

Reimplemented from `boa::BoaDataAnalyser::DataAna`.

Reimplemented in `boa::BoaFocus::Focus`, and `boa::BoaPointing::Point`.

`def boa::BoaMapping::Map::beamMap (self, chanList = [], flag = 0, oversamp = 2.0, sizeX = [], sizeY = [], style = 'g2r', noSmooth = 0, aspect = 0)`

DES: build a beam map in (Az,El) coordinates

```

INP: (int list) chanList = channels to consider
      (int) flag = flag values to consider
      (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
      (list float) sizeX = limits in Az of the map
      (list float) sizeY = limits in El of the map

```

def boa::BoaMapping::Map::eq_to_ho (self)

```

DES: convert EQ offsets to HO ones: assumes Lon and Lat attributes are
      offsets in RA, Dec, converts them to Az, El offsets and overwrites
      Lon and Lat - WARNING: not very accurate!

```

def boa::BoaMapping::Map::fastChanMap (self, chanList = [], flag = 0, oversamp = 2., sizeX = [], sizeY = [], noSmooth = 0, style = 'g2r', limitsZ = [], center = 0)

```

DES: plot channel maps (quick method)
INP: (int list) chanList = channels to consider
      (int) flag = flag values to consider
      (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
      (logical) center =0/1 if set to 1 it will shift each map by the bolometer offset
      from the fits header. Thereby it shifts the source to the center of
      each channel map

```

def boa::BoaMapping::Map::fastChanMap2 (self, chanList = [], flag = 0, oversamp = 2., sizeX = [], sizeY = [], noSmooth = 0, style = 'g2r', limitsZ = [], center = 0)

```

DES: plot channel maps (quick method)
INP: (int list) chanList = channels to consider
      (int) flag = flag values to consider
      (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
      (logical) center =0/1 if set to 1 it will shift each map by the bolometer offset
      from the fits header. Thereby it shifts the source to the center of
      each channel map

```

def boa::BoaMapping::Map::fastMap (self, chanList = [], flag = 0, oversamp = 2., beammap = 0, sizeX = [], sizeY = [], limitsZ = [], style = 'g2r', wedge = 1, noSmooth = 0, noPlot = 0, aspect = 0)

```

DES: reconstruct a map in (Az,El) coordinates combining bolometers
INP: (int list) chanList = channels to consider
      (int) flag = flag values to consider
      (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
      (list float) sizeX = limits in Az of the map
      (list float) sizeY = limits in El of the map
      (logical) noNan = remove NaN in self.Results?
      (str) style = color table to use in image
      (logical) noSmooth = do not smooth with beam?
      (logical) noPlot = do not plot the map?
      (str) caption = plot caption
      (logical) aspect = keep aspect ratio?

```

```
def boa::BoaMapping::Map::fastMap2 ( self, chanList = [], flag = 0, oversamp = 2.0, beammap = 0, sizeX = [], sizeY = [], limitsZ = [], style = 'g2r', wedge = 1, noSmooth = 0, noPlot = 0, aspect = 0)
```

DES: reconstruct a map in (Az,El) coordinates combining bolometers

INP: (int list) chanList = channels to consider
 (int) flag = flag values to consider
 (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
 (list float) sizeX = limits in Az of the map
 (list float) sizeY = limits in El of the map
 (logical) noNan = remove NaN in self.Results?
 (str) style = color table to use in image
 (logical) noSmooth = do not smooth with beam?
 (logical) noPlot = do not plot the map?
 (str) caption = plot caption
 (logical) aspect = keep aspect ratio?

```
def boa::BoaMapping::Map::getPixel ( self, nbPix = 3)
```

DES: allow user to get pixel values using mouse

INP: (int) nbPix : size of area to compute average

```
def boa::BoaMapping::Map::showMap ( self, style = 'g2r', labelX = "\gD Az ["]", labelY = "\gD El ["]", wedge = 1, limitsZ = [], aspect = 0, limitsX = [], limitsY = [])
```

DES: show the reconstructed map in (Az,El)

```
def boa::BoaMapping::Map::slowMap ( self, chanList = [], flag = 0, oversamp = 2.0, beammap = 0, sizeX = [], sizeY = [], offsets = [0.], style = 'g2r', wedge = 1, noSmooth = 0, noPlot = 0)
```

DES: reconstruct a map in (Az,El) coordinates combining bolometers

INP: (int list) chanList = channels to consider
 (int) flag = flag values to consider
 (float) oversamp = oversampling factor (beam fwhm / pixel size). Default=2.
 (list float) sizeX = limits in Az of the map
 (list float) sizeY = limits in El of the map
 (list float) offsets = Az,El to recenter a point source
 (logical) noNan = remove NaN in self.Results?
 (str) style = color table to use in image

E.13 boa::BoaMBFitsReader::MBFitsReader Class Reference

E.13.1 Detailed Description

Reader class for MBFITS 1.60 and earlier.

The only public method is read.

Public Member Functions

- def `__init__`
- def `getBlankFloat`
- def `getBlankInt`
- def `read`

E.13.2 Member Function Documentation

def `boa::BoaMBFitsReader::MBFitsReader::getBlankFloat (self)`

Blank value for floats as used in MBFITS

def `boa::BoaMBFitsReader::MBFitsReader::getBlankInt (self)`

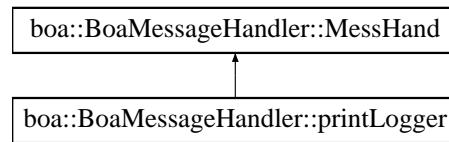
Blank value for integers as used in MBFITS

def `boa::BoaMBFitsReader::MBFitsReader::read (self, itemKey, kargs)`

Read item `itemKey` from the dataset, using the additional arguments in `**kargs`

E.14 boa::BoaMessageHandler::MessHand Class Reference

Inheritance diagram for boa::BoaMessageHandler::MessHand::



E.14.1 Detailed Description

NAM: MessHand (class)

DES: An object of this class is responsible for the management of output messages as well as the creation of message files.

Public Member Functions

- def `__del__`
- def `__init__`
- def `ask`
- def `closeMessFile`
- def `debug`
- def `error`
- def `info`
- def `initMessFile`
- def `longinfo`
- def `pause`
- def `setLogName`
- def `setMaxWeight`
- def `setMess`
- def `warning`
- def `Welcome`
- def `yesno`

E.14.2 Member Function Documentation

def `boa::BoaMessageHandler::MessHand::__init__ (self, logName = 'Unknown')`

DES: initialise an instance

def `boa::BoaMessageHandler::MessHand::ask (self, message = "")`

DES: ask the user

INP: (string) : the question

OUT: (string) : the answer

def boa::BoaMessageHandler::MessHand::closeMessFile (self)

DES: set self.__existMessFile to 0 and file name to ""

def boa::BoaMessageHandler::MessHand::debug (self, message = "")

DES: to print an debug message
INP: (string) message

def boa::BoaMessageHandler::MessHand::error (self, message = "")

DES: to print an error message
INP: (string) message

def boa::BoaMessageHandler::MessHand::info (self, message = "")

DES: to print an info message
INP: (string) message

def boa::BoaMessageHandler::MessHand::initMessFile (self, filename = "boa.mes")

DES: set & initialise new message file
OUT: screen output

def boa::BoaMessageHandler::MessHand::longinfo (self, message = "")

DES: to print an long info message
INP: (string) message

def boa::BoaMessageHandler::MessHand::pause (self, message = "")

DES: allow to make a pause in the program
OPT: (string) : a message to display

def boa::BoaMessageHandler::MessHand::setMaxWeight (self, weight = ' 2')

DES: Set the maximum weight of messages to be printed.
INP: (int) weight = maximum weight

1: errors, queries
2: warnings
3: short info
4: extended info
5: debug

def boa::BoaMessageHandler::MessHand::setMess (self, weight = 1, message = ' ')

DES: deposit messages for screen output and message files
INP: (int) weight = weight of transferred message (see setMaxWeight)
(string) message = message to be printed and added to message file

def boa::BoaMessageHandler::MessHand::warning (self, message = "")

DES: to print an warning message
INP: (string) message

def boa::BoaMessageHandler::MessHand::Welcome (self)

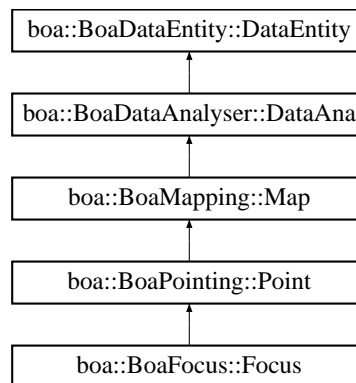
DES: print welcome message
OUT: screen output

def boa::BoaMessageHandler::MessHand::yesno (self, message = "")

DES: ask the user a question with yes/no answer type
INP: (string) : the question
OUT: (1) : : the answer

E.15 boa::BoaPointing::Point Class Reference

Inheritance diagram for boa::BoaPointing::Point::



E.15.1 Detailed Description

NAM: Point (class)

DES: An object of this class is responsible for the reduction of pointing scan(s)

Public Member Functions

- def [__init__](#)
- def [arrayParameters](#)
- def [iterMap](#)
- def [reduce](#)
- def [showPointing](#)
- def [solvePointing](#)
- def [solvePointingOnMap](#)
- def [updateArrayParameters](#)

E.15.2 Member Function Documentation

def `boa::BoaPointing::Point::__init__ (self)`

DES: Initialise an instance.

Reimplemented from [boa::BoaMapping::Map](#).

Reimplemented in [boa::BoaFocus::Focus](#).

def `boa::BoaPointing::Point::arrayParameters (self, chanList = [], gradient = 0, circular = 0, radius = 0, plot = 0)`

DES: determine the array parameters from the data

INP: (i list) `chanList` : the channel list to be used (default: current list)

(1) `gradient` : remove a background gradient in the data (default: no)

(1) `circular` : fit a cricular gaussian instead of an elliptical gaussian

```
def boa::BoaPointing::Point::iterMap ( self, chanList = [], phase = 0, flag = 0, sizeX = [], sizeY =
[])
```

DES: reconstruct a map in (Az,El) coordinates combining bolometers
and using varying scale to zoom on signal
INP: (int list) chanList = channels to consider
(int) phase = phase to plot
(int) flag = flag values to consider
(list float) sizeX = limits in Az of the map
(list float) sizeY = limits in El of the map

```
def boa::BoaPointing::Point::reduce ( self, FitsFile = None, obstoProc = [], radius = 100.)
```

DES: Process a Pointing scan - this method is called by the apexCalibrator
INP: (mbfits.FitsFile) FitsFile : object with pointer to an open FITS file
(i list) obstoProc: list of subscans to consider (default: all)
(float) radius: radius (in arcsec) to be used for fitting

```
def boa::BoaPointing::Point::showPointing ( self, plot = 1, display = 1, noMap = 0, caption = "",
aspect = 1, style = 'g2r')
```

DES: compute the offset
INP: (logical) plot : display the results on a map (default: no)
(logical) display : display the result on screen (default: yes)

```
def boa::BoaPointing::Point::solvePointing ( self, chanList = [], gradient = 1, circular = 0, radius =
-5, Xpos = 0., Ypos = 0., fixedPos = 0, plot = 0, display = 1, caption = "", aspect = 1)
```

DES: compute the offset
INP: (int list) chanList: list of channels to be used (default: all)
(boolean) gradient: shall we fit a gradient ? (default: yes)
(boolean) circular: fit a cricular gaussian instead of an elliptical gaussian
(float) radius : use only bolo inside this radius (negative means multiple of beam) (default 5 be
(float) Xpos,Ypos : source position if using fixed position
(boolean) fixedPos : if set, don't fit position, but use Xpos, Ypos
(boolean) plot : do we plot the results? (default: no)
(boolean) display : display the result of the fit (default: yes)
OUT: store in self.PoitingResult the results of the fit (i.e. all parameters
as computed by mpfit routine). If mpfit failed, then self.PoitingResult
is set to -1

```
def boa::BoaPointing::Point::solvePointingOnMap ( self, gradient = 1, circular = 0, radius = -5,
Xpos = 0., Ypos = 0., fixedPos = 0, plot = 0, display = 1, caption = "", aspect = 1, style = 'g2r')
```

DES: compute the offset on the data.Map object
INP: (boolean) gradient: shall we fit a gradient ? (default: yes)
(boolean) circular: fit a cricular gaussian instead of an elliptical gaussian
(float) radius : use only bolo inside this radius (negative means multiple of beam) (default 5 be
(float) Xpos,Ypos : source position if using fixed position
(boolean) fixedPos : if set, don't fit position, but use Xpos, Ypos
(boolean) plot : do we plot the results? (default: no)
(boolean) display : display the result of the fit (default: yes)
OUT: store in self.PointingResult the results of the fit (i.e. all parameters
as computed by mpfit routine). If mpfit failed, then self.PoitingResult
is set to -1

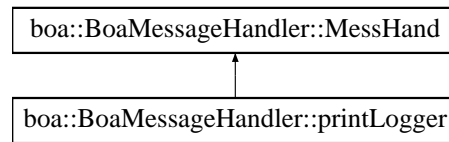
WARNING : No Smoothing should be applied to the map
before using this function, or the fitted fwhm will be
useless, use fine oversamp to make reasonable fit

def boa::BoaPointing::Point::updateArrayParameters (self)

DES: Update the Parameters Offsets with the computed values

E.16 boa::BoaMessageHandler::printLogger Class Reference

Inheritance diagram for boa::BoaMessageHandler::printLogger::



E.16.1 Detailed Description

NAM: `printLogger` (class)

DES: for compatibility with the `CalibratorLog.printLogger` class

Public Member Functions

- `def __init__`
- `def logDebug`
- `def logError`
- `def logInfo`
- `def logWarning`

E.17 boa::BoaDataEntity::ScanParameter Class Reference

E.17.1 Detailed Description

NAM: ScanParameter (class)

DES: Define all parameters (coordinates, time) for a scan

Public Member Functions

- def [__init__](#)
- def [__str__](#)
- def [caption](#)
- def [computeOnOff](#)
- def [fillFromMBFits](#)
- def [findSubscan](#)
- def [get](#)
- def [phaseDiffParam](#)
- def [plotAzEl](#)
- def [plotAzElOffset](#)
- def [plotAzimuth](#)
- def [plotAzimuthOffset](#)
- def [plotElevation](#)
- def [plotElevationOffset](#)
- def [plotSubscan](#)
- def [plotSubscanOffsets](#)
- def [selectPhase](#)

E.17.2 Member Function Documentation

def boa::BoaDataEntity::ScanParameter::__init__ ([self](#))

DES: Instanciation of a new ScanParameter object

def boa::BoaDataEntity::ScanParameter::__str__ ([self](#))

DES: Defines a string, shown when the print instruction is used.

def boa::BoaDataEntity::ScanParameter::caption ([self](#))

DES: Return a short caption of the scan

def boa::BoaDataEntity::ScanParameter::computeOnOff ([self](#))

DES: determine ON-OFF pairs from content of WobblerSta, and fill OnOffPairs attribute with pairs of integration numbers.
The result is a 2 x Nb_Integ. array of integers.

def boa::BoaDataEntity::ScanParameter::fillFromMBFits (self, reader, febe, baseband, subscans, update)

NAM: fillFromMBFits()

DES: fill a ScanParam object using the MBFitsReader object reader.

Calling sequence: DataEntity.fillFromMBFits(obsEntity)

INP: obsEntity: *LIST* of objects of the Entities.ObsEntity class
 update (logical) if true, do not reset previous entity object

def boa::BoaDataEntity::ScanParameter::findSubscan (self, threshold = 1 .)

DES: compute subscan indices from steps in az, el

INP: (float) threshold = value (in arcsec^2) of (d_az^2 + d_el^2) step
 used to detect turnovers / stationnary points

def boa::BoaDataEntity::ScanParameter::get (self, dataType = ' ', flag = 0, inverse = 0)

DES: get data of the ScanParam class

INP: (string) dataType : type of data

LST MJD Az El AzOff ElOff focX focY focZ

(int) flag : retrieve data flagged with flag

(default 0 : good data, 'None' for all)

(log) inverse : retrieve all data without given flag set instead (default no)

OUT: (float array) : the requested data

returned data are in the stored unit except for offsets which are
 converted to arcsec

def boa::BoaDataEntity::ScanParameter::phaseDiffParam (self)

NAM: phaseDiffParam (method)

DES: Compute the phase differences for data associated parameters.

Times are average of ON and OFF, coordinates are ON positions.

def boa::BoaDataEntity::ScanParameter::plotAzEl (self, flag = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 0)

DES: plot azimuth vs. elevation

INP: (int) flag : flag to be plot (default 0 : valid data, -1 plot all)

def boa::BoaDataEntity::ScanParameter::plotAzElOffset (self, flag = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 0)

DES: plot elevation offset versus azimuth offset

INP: (int) flag : flag to be plot (default 0 : valid data, -1 plot all)

def boa::BoaDataEntity::ScanParameter::plotAzimuth (self, flag = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 0)

DES: plot time series of azimuth

INP: (int) flag : flag to be plot (default 0 : valid data, -1 plot all)


```
def boa::BoaDataEntity::ScanParameter::plotAzimuthOffset ( self, flag = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 0)
```

DES: plot time series of elevation offset

INP: (int) flag : flag to be plot (default 0 : valid data, -1 plot all)

```
def boa::BoaDataEntity::ScanParameter::plotElevation ( self, flag = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 0)
```

DES: plot time series of elevation

INP: (int) flag : flag to be plot (default 0 : valid data, -1 plot all)

```
def boa::BoaDataEntity::ScanParameter::plotElevationOffset ( self, flag = 0, limitsX = [], limitsY = [], style = 'l', ci = 1, overplot = 0, aspect = 0)
```

DES: plot time series of azimuth offset

INP: (int) flag : flag to be plot (default 0 : valid data, -1 plot all)

```
def boa::BoaDataEntity::ScanParameter::plotSubscan ( self)
```

DES: generate a plot showing starting and ending times of subscans

```
def boa::BoaDataEntity::ScanParameter::plotSubscanOffsets ( self, overplot = 0)
```

DES: Use four colours to show subscans on the Az, El pattern

INP: (logical) overplot : if set, do not plot AzElOffset - assume these have been plotted already

```
def boa::BoaDataEntity::ScanParameter::selectPhase ( self, phase)
```

NAM: selectPhase (method)

DES: Keep only parameters (times, positions) associated with Data(ON) or Data(OFF)

INP: (int) phase: phase to keep, 1=ON, 2=OFF

E.18 boa::BoaDataEntity::Telescope Class Reference

E.18.1 Detailed Description

NAM: Telescope (class)

DES: Define all the useful parameters of a telescope

Public Member Functions

- def [__init__](#)
- def [__str__](#)
- def [set](#)

E.18.2 Member Function Documentation

def `boa::BoaDataEntity::Telescope::__init__ (self)`

DES: Instanciation of a Telescope object

def `boa::BoaDataEntity::Telescope::__str__ (self)`

DES: Defines a string which is shown when the print instruction is used.

def `boa::BoaDataEntity::Telescope::set (self, name = "", diameter = 0.0, longitude = 0.0, latitude = 0.0, elevation = 0.0)`

DES: set all the parameters

E.19 boa::Utilities::Timing Class Reference

E.19.1 Detailed Description

NAM: Timing (class)

DES: easily profile time computation in program

Public Member Functions

- def **__init__**
- def **__str__**
- def **getTime**
- def **resetTime**
- def **setIter**
- def **setTime**
- def **timeLeft**

Public Attributes

- **initTime**
- **lastTime**
- **nIter**

Index

- `__init__`
 - `boa::BoaDataAnalyser::DataAna`, 82
 - `boa::BoaDataEntity::BolometerArray`, 78
 - `boa::BoaDataEntity::DataEntity`, 89
 - `boa::BoaDataEntity::ScanParameter`, 114
 - `boa::BoaDataEntity::Telescope`, 117
 - `boa::BoaFocus::Focus`, 95
 - `boa::BoaMapping::Kernel`, 98
 - `boa::BoaMapping::Map`, 103
 - `boa::BoaMessageHandler::Logger`, 99
 - `boa::BoaMessageHandler::MessHand`, 107
 - `boa::BoaPointing::Point`, 110
 - `boa::MamboMBFits::MamboMBFits`, 100
- `__str__`
 - `boa::BoaDataEntity::BolometerArray`, 78
 - `boa::BoaDataEntity::DataEntity`, 89
 - `boa::BoaDataEntity::ScanParameter`, 114
 - `boa::BoaDataEntity::Telescope`, 117
- `array2list`
 - Utilities, 65
- `arrayParameters`
 - `boa::BoaPointing::Point`, 110
- `as_column_major_storage`
 - Utilities, 65
- `ask`
 - `boa::BoaMessageHandler::MessHand`, 107
- `attrStr`
 - Utilities, 65
- `backup`
 - `boa::BoaDataEntity::DataEntity`, 89
- `baseCircularGaussian`
 - Utilities, 65
- `baseEllipticalGaussian`
 - Utilities, 66
- `basePoly`
 - `boa::BoaDataAnalyser::DataAna`, 82
- `basePolySubscan`
 - `boa::BoaDataAnalyser::DataAna`, 82
- `beamMap`
 - `boa::BoaMapping::Map`, 103
- `blankAmplitude`
 - `boa::BoaDataAnalyser::FilterFFT`, 93
- `boa::BoaDataAnalyser::DataAna`, 81
- `boa::BoaDataAnalyser::DataAna`
 - `__init__`, 82
 - `basePoly`, 82
 - `basePolySubscan`, 82
 - `computeCorMatrix`, 82
 - `computeSN`, 82
 - `computeWeight`, 82
 - `correlate`, 83
 - `correlate2`, 83
 - `despike`, 83
 - `flag`, 83
 - `flagChannels`, 84
 - `flagInTime`, 84
 - `flagLon`, 84
 - `flagLST`, 84
 - `flagPosition`, 84
 - `flagRms`, 84
 - `flagSubscan`, 85
 - `medianBaseline`, 85
 - `medianFilter`, 85
 - `plotCorMatrix`, 85
 - `plotFFT`, 85
 - `plotMean`, 85
 - `plotMeanChan`, 86
 - `plotRms`, 86
 - `plotRmsChan`, 86
 - `slidingRms`, 86
 - `statistics`, 86
 - `unflag`, 86
 - `unflagChannels`, 86
- `boa::BoaDataAnalyser::FilterFFT`, 93
- `boa::BoaDataAnalyser::FilterFFT`
 - `blankAmplitude`, 93
 - `doFFT`, 93
 - `invFFT`, 93
 - `plotfft`, 93
 - `rebin`, 94
 - `unbin`, 94
 - `windowing`, 94
- `boa::BoaDataEntity::BolometerArray`, 78
- `boa::BoaDataEntity::BolometerArray`
 - `__init__`, 78
 - `__str__`, 78
 - `checkChanList`, 78
 - `computeBeamSize`, 78

- computeChanSep, 79
- computeChanSepValid, 79
- fillFromMBFits, 79
- get, 79
- getChanIndex, 79
- getChanSep, 79
- plotArray, 79
- plotGain, 79
- printCurrChanList, 80
- readAsciiRcp, 80
- readRCPfile, 80
- rotateArray, 80
- setChannelsNumbers, 80
- setCurrChanList, 80
- writeRCPfile, 80
- boa::BoaDataEntity::DataEntity, 88
- boa::BoaDataEntity::DataEntity
 - __init__, 89
 - __str__, 89
 - backup, 89
 - computeSubIndex, 89
 - dumpData, 89
 - existData, 89
 - fillFromMBFits, 89
 - getChanData, 89
 - getChanListData, 90
 - phaseDiff, 90
 - plotCorrel, 90
 - read, 90
 - reset, 90
 - restore, 90
 - restoreData, 91
 - saveMambo, 91
 - selectPhase, 91
 - signal, 91
 - writeMBfits, 91
- boa::BoaDataEntity::ScanParameter, 114
- boa::BoaDataEntity::ScanParameter
 - __init__, 114
 - __str__, 114
 - caption, 114
 - computeOnOff, 114
 - fillFromMBFits, 114
 - findSubscan, 115
 - get, 115
 - phaseDiffParam, 115
 - plotAzEl, 115
 - plotAzElOffset, 115
 - plotAzimuth, 115
 - plotAzimuthOffset, 115
 - plotElevation, 116
 - plotElevationOffset, 116
 - plotSubscan, 116
 - plotSubscanOffsets, 116
- selectPhase, 116
- boa::BoaDataEntity::Telescope, 117
- boa::BoaDataEntity::Telescope
 - __init__, 117
 - __str__, 117
 - set, 117
- boa::BoaError::BoaError, 77
- boa::BoaFocus::Focus, 95
- boa::BoaFocus::Focus
 - __init__, 95
 - reduce, 95
 - solveFocus, 95
- boa::BoaMapping::Image, 96
- boa::BoaMapping::Image
 - computeWCS, 96
 - display, 96
 - physicalCoordinates, 97
 - smoothBy, 97
 - smoothWith, 97
 - wcs2phy, 97
 - wcs2pix, 97
- boa::BoaMapping::Kernel, 98
- boa::BoaMapping::Kernel
 - __init__, 98
- boa::BoaMapping::Map, 103
- boa::BoaMapping::Map
 - __init__, 103
 - beamMap, 103
 - eq_to_ho, 104
 - fastChanMap, 104
 - fastChanMap2, 104
 - fastMap, 104
 - fastMap2, 104
 - getPixel, 105
 - showMap, 105
 - slowMap, 105
- boa::BoaMBFitsReader::MBFitsReader, 106
- boa::BoaMBFitsReader::MBFitsReader
 - getBlankFloat, 106
 - getBlankInt, 106
 - read, 106
- boa::BoaMessageHandler::Logger, 99
- boa::BoaMessageHandler::Logger
 - __init__, 99
- boa::BoaMessageHandler::MessHand, 107
- boa::BoaMessageHandler::MessHand
 - __init__, 107
 - ask, 107
 - closeMessFile, 107
 - debug, 108
 - error, 108
 - info, 108
 - initMessFile, 108
 - longinfo, 108

- pause, 108
 - setMaxWeight, 108
 - setMess, 108
 - warning, 109
 - Welcome, 109
 - yesno, 109
- boa::BoaMessageHandler::printLogger, 113
- boa::BoaPointing::Point, 110
- boa::BoaPointing::Point
 - __init__, 110
 - arrayParameters, 110
 - iterMap, 110
 - reduce, 111
 - showPointing, 111
 - solvePointing, 111
 - solvePointingOnMap, 111
 - updateArrayParameters, 112
- boa::Bogli::Interface::Fenetre, 92
 - dessine, 92
 - saisie, 92
- boa::MamboMBFits::MamboMBFits, 100
- boa::MamboMBFits::MamboMBFits
 - __init__, 100
 - convertFebepar, 100
 - convertMambo2MBFits, 100
 - convertMamboPrimary, 100
 - convertMB2MamboFits, 101
 - ctype2sbas, 101
 - fillFebepar, 101
 - fillMamboData, 101
 - fillMamboPrimary, 101
 - getObsMode, 101
 - getScanType, 101
 - initMambo, 101
 - initMB, 102
 - processMamboData, 102
 - readMambo, 102
 - readMBfits, 102
 - readRCP, 102
 - sbas2ctype, 102
- boa::Utilities::Timing, 118
- caption
 - boa::BoaDataEntity::ScanParameter, 114
- checkChanList
 - boa::BoaDataEntity::BolometerArray, 78
- clear
 - Plot, 73
- closeDev
 - DeviceHandler, 69
- closeMessFile
 - boa::BoaMessageHandler::MessHand, 107
- compress2d
 - Utilities, 66
- compressNan
 - Utilities, 66
- computeBeamSize
 - boa::BoaDataEntity::BolometerArray, 78
- computeChanSep
 - boa::BoaDataEntity::BolometerArray, 79
- computeChanSepValid
 - boa::BoaDataEntity::BolometerArray, 79
- computeCorMatrix
 - boa::BoaDataAnalyser::DataAna, 82
- computeOnOff
 - boa::BoaDataEntity::ScanParameter, 114
- computeSN
 - boa::BoaDataAnalyser::DataAna, 82
- computeSubIndex
 - boa::BoaDataEntity::DataEntity, 89
- computeWCS
 - boa::BoaMapping::Image, 96
- computeWeight
 - boa::BoaDataAnalyser::DataAna, 82
- convertFebepar
 - boa::MamboMBFits::MamboMBFits, 100
- convertMambo2MBFits
 - boa::MamboMBFits::MamboMBFits, 100
- convertMamboPrimary
 - boa::MamboMBFits::MamboMBFits, 100
- convertMB2MamboFits
 - boa::MamboMBFits::MamboMBFits, 101
- correlate
 - boa::BoaDataAnalyser::DataAna, 83
- correlate2
 - boa::BoaDataAnalyser::DataAna, 83
- Cp2r
 - Utilities, 66
- Cr2p
 - Utilities, 66
- cropped_circular_gaussian
 - Utilities, 66
- ctype2sbas
 - boa::MamboMBFits::MamboMBFits, 101
- debug
 - boa::BoaMessageHandler::MessHand, 108
- despike
 - boa::BoaDataAnalyser::DataAna, 83
- dessine
 - boa::Bogli::Interface::Fenetre, 92
- detStartParaParabola
 - Utilities, 66
- detSubDivView
 - MultiPlot, 71
- DeviceHandler, 69
- DeviceHandler
 - closeDev, 69

- openDev, 69
- resizeDev, 69
- selectDev, 69
- display
 - boa::BoaMapping::Image, 96
- distsq
 - Utilities, 67
- doFFT
 - boa::BoaDataAnalyser::FilterFFT, 93
- draw
 - MultiPlot, 71
 - Plot, 73
- drawChanNum
 - MultiPlot, 71
- drawLabel
 - Plot, 73
- drawWedge
 - Plot, 74
- dumpData
 - boa::BoaDataEntity::DataEntity, 89
- eq_to_ho
 - boa::BoaMapping::Map, 104
- erase
 - Plot, 74
- error
 - boa::BoaMessageHandler::MessHand, 108
- existData
 - boa::BoaDataEntity::DataEntity, 89
- fastChanMap
 - boa::BoaMapping::Map, 104
- fastChanMap2
 - boa::BoaMapping::Map, 104
- fastMap
 - boa::BoaMapping::Map, 104
- fastMap2
 - boa::BoaMapping::Map, 104
- fenetreInteractive
 - Interface, 70
- fillFebepar
 - boa::MamboMBFits::MamboMBFits, 101
- fillFromMBFits
 - boa::BoaDataEntity::BolometerArray, 79
 - boa::BoaDataEntity::DataEntity, 89
 - boa::BoaDataEntity::ScanParameter, 114
- fillMamboData
 - boa::MamboMBFits::MamboMBFits, 101
- fillMamboPrimary
 - boa::MamboMBFits::MamboMBFits, 101
- findSubscan
 - boa::BoaDataEntity::ScanParameter, 115
- fitBaseEllipticalGaussian
 - Utilities, 67
- fitParabola
 - Utilities, 67
- flag
 - boa::BoaDataAnalyser::DataAna, 83
- flagChannels
 - boa::BoaDataAnalyser::DataAna, 84
- flagInTime
 - boa::BoaDataAnalyser::DataAna, 84
- flagLon
 - boa::BoaDataAnalyser::DataAna, 84
- flagLST
 - boa::BoaDataAnalyser::DataAna, 84
- flagPosition
 - boa::BoaDataAnalyser::DataAna, 84
- flagRms
 - boa::BoaDataAnalyser::DataAna, 84
- flagSubscan
 - boa::BoaDataAnalyser::DataAna, 85
- Fortran subroutines, 63
- gaussian
 - Utilities, 67
- get
 - boa::BoaDataEntity::BolometerArray, 79
 - boa::BoaDataEntity::ScanParameter, 115
- getBlankFloat
 - boa::BoaMBFitsReader::MBFitsReader, 106
- getBlankInt
 - boa::BoaMBFitsReader::MBFitsReader, 106
- getChanData
 - boa::BoaDataEntity::DataEntity, 89
- getChanIndex
 - boa::BoaDataEntity::BolometerArray, 79
- getChanListData
 - boa::BoaDataEntity::DataEntity, 90
- getChanSep
 - boa::BoaDataEntity::BolometerArray, 79
- getObsMode
 - boa::MamboMBFits::MamboMBFits, 101
- getPixel
 - boa::BoaMapping::Map, 105
 - Plot, 74
- getScanType
 - boa::MamboMBFits::MamboMBFits, 101
- gloLabelling
 - MultiPlot, 71
- info
 - boa::BoaMessageHandler::MessHand, 108
- initMambo
 - boa::MamboMBFits::MamboMBFits, 101
- initMB
 - boa::MamboMBFits::MamboMBFits, 102
- initMessFile

- boa::BoaMessageHandler::MessHand, 108
- Interface, 70
 - fenetreInteractive, 70
 - pgrstr, 70
- invFFT
 - boa::BoaDataAnalyser::FilterFFT, 93
- iterMap
 - boa::BoaPointing::Point, 110
- labelling
 - Plot, 74
- lCompressNan
 - Utilities, 67
- longinfo
 - boa::BoaMessageHandler::MessHand, 108
- max2D
 - Utilities, 67
- medianBaseline
 - boa::BoaDataAnalyser::DataAna, 85
- medianFilter
 - boa::BoaDataAnalyser::DataAna, 85
- min2D
 - Utilities, 68
- modelBaseEllipticalGaussian
 - Utilities, 68
- modelparabola
 - Utilities, 68
- MultiPlot, 71
- MultiPlot
 - detSubDivView, 71
 - draw, 71
 - drawChanNum, 71
 - gloLabelling, 71
 - plot, 71
 - plotBox, 72
 - setLimits, 72
 - setMultiViewPoint, 72
- openDev
 - DeviceHandler, 69
- parabola
 - Utilities, 68
- pause
 - boa::BoaMessageHandler::MessHand, 108
- pgrstr
 - Interface, 70
- phaseDiff
 - boa::BoaDataEntity::DataEntity, 90
- phaseDiffParam
 - boa::BoaDataEntity::ScanParameter, 115
- physicalCoordinates
 - boa::BoaMapping::Image, 97
- Plot, 73
 - clear, 73
 - draw, 73
 - drawLabel, 73
 - drawWedge, 74
 - erase, 74
 - getPixel, 74
 - labelling, 74
 - plot, 74
 - plotBox, 74
 - plotDataXY, 74
 - readLut, 75
 - removeNan, 75
 - setImaCol, 75
 - setLabels, 75
 - setLimits, 75
 - setMapLimits, 75
 - setMapTransformation, 76
 - setViewPoint, 76
 - xyout, 76
- plot
 - MultiPlot, 71
 - Plot, 74
- plotArray
 - boa::BoaDataEntity::BolometerArray, 79
- plotAzEl
 - boa::BoaDataEntity::ScanParameter, 115
- plotAzElOffset
 - boa::BoaDataEntity::ScanParameter, 115
- plotAzimuth
 - boa::BoaDataEntity::ScanParameter, 115
- plotAzimuthOffset
 - boa::BoaDataEntity::ScanParameter, 115
- plotBox
 - MultiPlot, 72
 - Plot, 74
- plotCorMatrix
 - boa::BoaDataAnalyser::DataAna, 85
- plotCorrel
 - boa::BoaDataEntity::DataEntity, 90
- plotDataXY
 - Plot, 74
- plotElevation
 - boa::BoaDataEntity::ScanParameter, 116
- plotElevationOffset
 - boa::BoaDataEntity::ScanParameter, 116
- plotFFT
 - boa::BoaDataAnalyser::DataAna, 85
- plotfft
 - boa::BoaDataAnalyser::FilterFFT, 93
- plotGain
 - boa::BoaDataEntity::BolometerArray, 79
- plotMean
 - boa::BoaDataAnalyser::DataAna, 85

- plotMeanChan
 - boa::BoaDataAnalyser::DataAna, 86
- plotRms
 - boa::BoaDataAnalyser::DataAna, 86
- plotRmsChan
 - boa::BoaDataAnalyser::DataAna, 86
- plotSubscan
 - boa::BoaDataEntity::ScanParameter, 116
- plotSubscanOffsets
 - boa::BoaDataEntity::ScanParameter, 116
- prettyPrintList
 - Utilities, 68
- printCurrChanList
 - boa::BoaDataEntity::BolometerArray, 80
- processMamboData
 - boa::MamboMBFits::MamboMBFits, 102
- read
 - boa::BoaDataEntity::DataEntity, 90
 - boa::BoaMBFitsReader::MBFitsReader, 106
- readAsciiRcp
 - boa::BoaDataEntity::BolometerArray, 80
- readLut
 - Plot, 75
- readMambo
 - boa::MamboMBFits::MamboMBFits, 102
- readMBfits
 - boa::MamboMBFits::MamboMBFits, 102
- readRCP
 - boa::MamboMBFits::MamboMBFits, 102
- readRCPfile
 - boa::BoaDataEntity::BolometerArray, 80
- rebin
 - boa::BoaDataAnalyser::FilterFFT, 94
- reduce
 - boa::BoaFocus::Focus, 95
 - boa::BoaPointing::Point, 111
- removeNan
 - Plot, 75
- reset
 - boa::BoaDataEntity::DataEntity, 90
- resizeDev
 - DeviceHandler, 69
- restore
 - boa::BoaDataEntity::DataEntity, 90
- restoreData
 - boa::BoaDataEntity::DataEntity, 91
- rotateArray
 - boa::BoaDataEntity::BolometerArray, 80
- safeExp
 - Utilities, 68
- saisie
 - boa::Bogli::Interface::Fenetre, 92
- saveMambo
 - boa::BoaDataEntity::DataEntity, 91
- sbas2ctype
 - boa::MamboMBFits::MamboMBFits, 102
- selectDev
 - DeviceHandler, 69
- selectPhase
 - boa::BoaDataEntity::DataEntity, 91
 - boa::BoaDataEntity::ScanParameter, 116
- set
 - boa::BoaDataEntity::Telescope, 117
- setChannelsNumbers
 - boa::BoaDataEntity::BolometerArray, 80
- setCurrChanList
 - boa::BoaDataEntity::BolometerArray, 80
- setImaCol
 - Plot, 75
- setLabels
 - Plot, 75
- setLimits
 - MultiPlot, 72
 - Plot, 75
- setMapLimits
 - Plot, 75
- setMapTransformation
 - Plot, 76
- setMaxWeight
 - boa::BoaMessageHandler::MessHand, 108
- setMess
 - boa::BoaMessageHandler::MessHand, 108
- setMultiViewPoint
 - MultiPlot, 72
- setViewPoint
 - Plot, 76
- showMap
 - boa::BoaMapping::Map, 105
- showPointing
 - boa::BoaPointing::Point, 111
- signal
 - boa::BoaDataEntity::DataEntity, 91
- slidingRms
 - boa::BoaDataAnalyser::DataAna, 86
- slowMap
 - boa::BoaMapping::Map, 105
- smoothBy
 - boa::BoaMapping::Image, 97
- smoothWith
 - boa::BoaMapping::Image, 97
- solveFocus
 - boa::BoaFocus::Focus, 95
- solvePointing
 - boa::BoaPointing::Point, 111
- solvePointingOnMap
 - boa::BoaPointing::Point, 111

- solvePoly
 - Utilities, [68](#)
- statistics
 - boa::BoaDataAnalyser::DataAna, [86](#)
- unbin
 - boa::BoaDataAnalyser::FilterFFT, [94](#)
- unflag
 - boa::BoaDataAnalyser::DataAna, [86](#)
- unflagChannels
 - boa::BoaDataAnalyser::DataAna, [86](#)
- updateArrayParameters
 - boa::BoaPointing::Point, [112](#)
- Utilities, [65](#)
 - array2list, [65](#)
 - as_column_major_storage, [65](#)
 - attrStr, [65](#)
 - baseCircularGaussian, [65](#)
 - baseEllipticalGaussian, [66](#)
 - compress2d, [66](#)
 - compressNan, [66](#)
 - Cp2r, [66](#)
 - Cr2p, [66](#)
 - cropped_circular_gaussian, [66](#)
 - detStartParaParabola, [66](#)
 - distsq, [67](#)
 - fitBaseEllipticalGaussian, [67](#)
 - fitParabola, [67](#)
 - gaussian, [67](#)
 - lCompressNan, [67](#)
 - max2D, [67](#)
 - min2D, [68](#)
 - modelBaseEllipticalGaussian, [68](#)
 - modelparabola, [68](#)
 - parabola, [68](#)
 - prettyPrintList, [68](#)
 - safeExp, [68](#)
 - solvePoly, [68](#)
- warning
 - boa::BoaMessageHandler::MessHand, [109](#)
- wcs2phy
 - boa::BoaMapping::Image, [97](#)
- wcs2pix
 - boa::BoaMapping::Image, [97](#)
- Welcome
 - boa::BoaMessageHandler::MessHand, [109](#)
- windowing
 - boa::BoaDataAnalyser::FilterFFT, [94](#)
- writeMBfits
 - boa::BoaDataEntity::DataEntity, [91](#)
- writeRCPfile
 - boa::BoaDataEntity::BolometerArray, [80](#)
- xyout
 - Plot, [76](#)
- yesno
 - boa::BoaMessageHandler::MessHand, [109](#)