OYSTER

Optical Interferometry Script Data Reduction
Version 8 for IDL

Christian A. Hummel
European Southern Observatory, Karl-Schwarzschild-Str. 2, 85748 Garching, Germany

January 30, 2023


aisdr
File
Access
Reduce
Calibrate
Utilities
Data
Model
Plot
Fit
Astrometry

Contents

Part I

OYSTER

Chapter 1

General information

1.1 About OYSTER

OYSTER is the original interactive data reduction package developed for the Navy Precision Optical Interferometer at Lowell (NPOI). *OYSTER* is an endeavor to combine **all aspects** of data reduction, modeling, astrometry, and imaging, including a stellar database, with an interactive data language in one package. Procedures are often shared. It also tries to maximize ease of use by providing both command line procedures and widget routines.

- **HDS data base access**: A full set of functions to access NPOI data files written in the Hierarchical Data System format (developed by Starlink). Individual objects can be accessed directly.
- **Plotting and editing**: General plotting widgets, with interactive display options and editing routines. Flagtables are implemented for bookkeeping.
- **Astrometry**: Multi-color dispersion corrections, metrology calibration, astrometric functions including the Naval Observatory Vector Astrometry Subroutines, epoch conversions, and polar motion data. Solution for baseline and star coordinates.
- **Calibration**: General multi-dimensional visibility amplitude and phase calibration capability for several types of dependencies.
- **Global modeling**: Use of a hierarchical stellar systems format which enables fitting physical parameters directly to any combination of data from interferometry, spectroscopy, photometry, and astrometry.
- **Stellar spectra and limb darkening**: atmospheric models which are used in bandwidth smearing computations and modeling. Display of stellar spectra and line identifications.
- **Stellar data base**: procedures for star catalog access. Storage of data in a table, and procedures for estimation of physical parameters of stars, like diameters, masses, orbital elements.
- **Planning of observations**: A set of procedures for computing visibility and simulating data for scheduling. The stellar data base serves to aid in the selection of calibrator stars.

$\mathcal{OYSTER}$ was created when three collections of scripts written first in PV-WAVE Command Language (PV-WAVE is a trademark of Rogue Wave Software) and then also in the Interactive Data Language (IDL is a trademark of Exelis Visual Information Solutions) were merged into one package for the purpose of providing a comprehensive data analysis package. The PV-WAVE version of $\mathcal{OYSTER}$ is however no longer supported, while support for GDL (Gnu Data Language) has been added with version 8 of $\mathcal{OYSTER}$ (no GUI support).

The three original collections, CHAMELEON which reduces a single night of NPOI data, AMOEBA which fits models to the calibrated visibility data of several nights, and STARBASE which implements a stellar data base and provides catalog access, are described in this User's Guide in separate parts. Also introduced are two new collections of scripts, COBRA, which provides functions for the interactive analysis of raw interferometry data (NPOI only), and PEARL, which images interferometry data.

As of 2014, $\mathcal{OYSTER}$ has evolved into a package also supporting the reduction and analysis of data from interferometers other than NPOI. This includes the GUI front-ends MyMidiGui and MyAmberGui to the MIA+EWS (*http://www.strw.leidenuniv.nl/∼nevec/MIDI/index.html*) and amdlib (*http://www.jmmc.fr/data_processing_amber.htm*) data reduction softwares for MIDI and AMBER at VLTI, respectively. Also, observation planning and advanced data analysis is available in $\mathcal{OYSTER}$. Stand-alone reduction of data from other interferometers is not planned to be included in $\mathcal{OYSTER}$ since this involves obviously very instrument specific tasks.

Data of other interferometers can be read and modeled by $\mathcal{OYSTER}$ if they are stored using the OIFITS format (*http://www.mrao.cam.ac.uk/research/OAS/oi_data/oifits.html*).

1.2 About this guide

This is not only a manual on how to use the $\mathcal{OYSTER}$ software, but a reference as well on how the procedures work. Study of the **Programmer's reference** sections should enable you to write and add your own procedures to the software, or create modified copies of existing routines which replace the latter if they are compiled during a session.

1.3 Fonts used in the guide

In order to facilitate reading this manual, various fonts are used. Responses you are asked to type in at the command level appear in these fonts: `hds_open,`*filename*. In this example, a lower-case italic font indicates numeric or character parameters you have to supply. Procedure names in the text appear like this: `average`, whereas widget buttons are typed like this: OPEN or DispCorr, depending on whether or not they bring up a new widget or take immediate action, respectively. Note, that IDL is not case sensitive Messages from the $\mathcal{OYSTER}$ software or IDL are denoted in this style: `Averaging complete`.

Chapter 2

Installing OYSTER

The *OYSTER* installation consists of code, catalog, and stellar atmosphere data. Please see more detail at *http://www.eso.org/~chummel/oyster/download.html* and contact the author to obtain a copy of the code. The external C and FORTRAN library compiled for Linux 64-bit systems is included, but can be recompiled if necessary from the C code which can also be obtained from the author.

The file *oyster.tar.gz* is unpacked first, this will create the folder named *oyster*. Inside it, unpack *catalogs.tar.gz* and *atmospheres.tar.gz*. Two new directories are created, *atmospheres* and *catalogs*.

2.1 Compiling OYSTER

Compilation always includes the procedure code which is then saved in *oyster/oyster.cpr*. This does not apply to GDL installations, where code is compiled on the fly. Makefiles exists in the main *OYSTER* directory which simplify the installation. Use `make -f Makefile.idl` to compile the IDL installation, and `make -f Makefile.gdl` to prepare the GDL installation.

If you need to compile the C/FORTRAN external library due to incompatibility of your OS with the pre-compiled Linux 64-bit version, unpack the *src.tar.gz* file (to be obtained from the author) in the *oyster/source* directory. Make sure your OS is either Linux or Darwin, checking with the `uname` UNIX command. The Makefiles will use `uname` to determine which of the pre-installed Makefiles in the library directories are used for compilation. The only part of the Makefile which still has to be edited manually before running `make` is *Makefile.<uname>.in* in *source/c* which needs the path to the local IDL installation, as well as the location of the *libgfortran.a* library file. Then run the `make -f Makefile.src` command in the home directory.

2.2 Programmer's reference

OYSTER uses an external library (*oyster.so*) containing C and FORTRAN objects. This library must be sharable in order for IDL to access it (via the `call_external` command). Aside from the *OYSTER* specific code (written in C), this library contains five external libraries. These are:

- HDS (Hierarchical Data System, STARLINK)
- NOVAS (Naval Observatory Vector Astrometry Subroutines, USNO)

- FITSIO (The Flexible Image Transport System I/O package, HEASARC)
- WD (Wilson-Devinney binary light curve software, U Florida)
- ROCHE (Gravity-darkening in rotating stars, Deane Peterson, SUNY)
- SHADOW (VLTI shadowing computations)

After the standard installation procedure described above, manual re-compilation can be performed by going into the C source directory *source/c*, and editing *Makefile*. (Please note that the *Makefile* is the result of a concatenation of *Makefile.all.in* and *Makefile.<uname>.in* performed by the standard installation procedure.) Then run make clean and make to compile the library, which is named *oyster.so*. Copy this file into the sub-directory of the $\mathcal{OYSTER}$ home directory corresponding to your operating system using the provided shell scripts *release.Darwin* or *release.Linux*. into the $\mathcal{OYSTER}$ home directory. Please send a copy of the Makefile to the author if you had to modify it.

This procedure should work for LINUX and DARWIN. $\mathcal{OYSTER}$ will detect your OS, and then know the path to the shared library and the binary JPL ephemeris file (which differs between MAC and PC based work stations), residing in directories *linux* and *darwin* under the main $\mathcal{OYSTER}$ directory.

To recompile and install the IDL procedures, start up IDL (in the *idl* directory) with the command file *c.pro*, i.e. `idl c`, which will then compile the IDL code and save it into files *oyster.cmb* and *oyster.cpr*. The command file will then end the session. You should use $\mathcal{OYSTER}$ first to recompile the crossindex subdirectory of the catalog directory, if the code and catalog data distribution tape was not specifically assembled for your architecture. (This is because the crossindex files are binary files in XDR format.) To recompile the crossindices, type get_crossindex,1. To exit the session, type quit.

Finally, it might be necessary to re-create the binary ephemeris file used by NOVAS. To do this, go into the *jpl* subdirectory and follow the instructions in *README.1st*.

2.3 Earth orientation updates

Edit the file *oyster/usno/cronjob* to contain the proper path to the *update* shell script, and then use the command

crontab cronjob

to install a job which fetches every Thursday the new UT1 and polar motion data from a computer at the US Naval Observatory. Otherwise, the database *usno/mark3.dat* obtained with the download will not be updated and eventually will not provide Earth orientation data for the date observed. In this case, a warning message will be displayed, which, however, can usually be ignored in all cases but astrometry.

Chapter 3

Using OYSTER

3.1 Starting up

You invoke $\mathcal{OYSTER}$ at the UNIX prompt by typing (the actual path may be different on your system):

```
idl /home/chummel/oyster/oyster
```

or

```
gdl /home/chummel/oyster/oyster
```

Provided IDL (version 5.0 or higher) or GDL (version 0.9.3 or higher) is installed, the language will start up and read and execute the commands in *oyster/oyster.pro*. This procedure file will first determine the location of the *oyster*, *atmospheres*, and *catalogs* home directories, then load the compiled procedures in *oyster.cpr*, set the common blocks and call the procedure `setup_oyster` to finish the setup. At this point, $\mathcal{OYSTER}$ procedures and functions are ready to use at the command line or through invoking the main widget by typing oyster. If you are using remote login without the capability of displaying windows, edit the *oyster.pro* file to include the /notv option. Type quit to exit $\mathcal{OYSTER}$.

If this is the first time you run $\mathcal{OYSTER}$, run the commands `limbdata` and `kurudata,'ip00k2.pck19'` to create files in the *atmospheres* directory necessary for using model atmospheres.

3.2 What to do next?

More than a thousand procedures and functions available in $\mathcal{OYSTER}$ support three main tasks: reducing and calibrating one night's interferometric data, as well as scheduling and HDS file access (this collection is referred to as CHAMELEON), computing stellar models and fitting them to the data (AMOEBA), and creating and managing stellar data bases and catalogs (STARBASE). Please refer to that part of this guide which is appropriate for your task. Note that many of the scripts are interdependent and share resources.

3.3 Working with OYSTER

OYSTER procedures are called from within the IDL environment. This is done from either the IDL command line by typing in the appropriate commands, or by invoking a widget, which provides buttons to click on for specific action. The latter method is less flexible but has the advantage of not requiring you to memorize all the commands and their parameters. Widgets provide an easy way to set parameter values in common blocks, which is how most of the information is passed between functions. You may use both operation modes simultaneously. Note that widget buttons which bring up another widget are labeled in all caps, whereas the other type either displays a sub-menu (indicated by the arrow) or executes a procedure upon selection. At the command level, you manipulate data using IDL. This was one of the reasons to base *OYSTER* on a language like IDL. Note that neither language is case sensitive. An understanding of the language is very helpful when reducing data with

AMOEBA tries to place the widgets in convenient places on the screen so that there is not too much overlap and so that the plot windows do not cover too many of the widgets. The IDL version detects the dimensions of your screen and uses this information for the placements of the widgets. *OYSTER*.

Procedures usually confirm proper completion of the task with a message terminated with a period. In-progress messages are not terminated. Error messages usually contain the name of the procedure in which the error occurred. In case of a crash inside a procedure under IDL, click on the terminal window to make it active, then try to return to successively higher levels by typing .return until a .con command causes IDL to resume. In this case the widgets, if previously displayed, will be updated. Note that in a crash situation, IDL is in a debugging mode with operation stopped at the offending line in the code. You can enquire about variables using the info,*variable*,/*structure* command. Under IDL, just type retall and you will be returned to the main level.

3.4 In case of trouble

Unlike many programs compiled from C or FORTRAN code, the *OYSTER* code, being written in an interpreted language, almost never crashes by returning to the operating system prompt. What usually happens is that execution stops at the offending line in the script and an error message is printed. At this point, the fearless user can print values of variables, enter IDL commands, all within the current context, in order to investigate the causes of the crash. To resume the *OYSTER* session by backing out of the procedure, it is best to return right away to the main level by typing retall.

3.5 Getting on-line help

Click on the big *OYSTER* logo button to start a browser (Firefox) on *OYSTER*'s main HTML help page (*oyster/html/oyster.html*). These pages provide brief reminders of implemented features. (This page is under construction for the forseeable future.) Some of the widget routines provide HELP buttons. At the IDL prompt, type info for help, help at the IDL prompt. Type info for a list of currently defined IDL variables and available procedures and functions.

3.6 Finishing up

Before quitting *OYSTER*, you will need to decide whether to save any calibrated data, FITS files, observing lists, or stellar data base tables. *OYSTER* never automatically saves any data for reasons of not destroying existing files or creating files the user doesn't know about.

3.7 Using OYSTER at NPOI

At the US Naval Observatory, raw data from the interferometer is first reduced with CONSTRICTOR, which is a stand-alone program. It computes the visibilities and averages the data into 1 second intervals, the so-called point data. Averaged raw data from CONSTRICTOR can be found in */home/constrictor* on octans at the site. It is best to start up *OYSTER* in a directory owned by the user, and then to enter the full path to the data file when opening it. The path is stripped off by *OYSTER* when deriving output file names from the input file name so that all files are written into the local directory. Due to the size of the *.con* files, it is best not to make copies of them.

Averaged scan data from *OYSTER* is output into HDS files with the extension *.cha*. If you have a file with unknown history, the data might have been calibrated, or they might not. It is best to assume that they have to be calibrated, in which case one should remove any existing calibration before continuing. In order to find the right file for observations of a particular star, use the `obsdates` procedure (see section 19.8).

Chapter 4

OYSTER support for other interferometers

4.1 Motivation

Once interferometric data have been reduced and averaged using instrument-specific data reduction packages, they can, if they have been stored in the OIFITS format, be read by $\mathcal{OYSTER}$ for calibration (if necessary), and further analysis employing models and images. In this way, the vast number of data analysis procedures written for $\mathcal{OYSTER}$ can be used for any kind of interferometric data.

4.2 OIFITS reader

The primary access to $\mathcal{OYSTER}$ procedures is via the OIFITS reader (`get_oifits`), which stores the data internally in the *scans* and other auxilliary IDL structures. Alternatively, VLTI MIDI and AMBER data can be reduced using the MyMidiGui and MyAmberGui front ends (*http://www.eso.org/~chummel/midi/mymidigui/mymidigui.html*, *http://www.eso.org/~chummel/amber/myambergui/myambergui.html*). These provide easy to use GUIs for the externally developed data reduction softwares (MIA+EWS and amdlib), and automatically store data internally using $\mathcal{OYSTER}$'s data structures, as well as write data to disk in OIFITS format.

For data to be stored internally in $\mathcal{OYSTER}$ many requirements are to be met, mostly for historical reasons of a data package first released over 15 years ago. These procedures are handled in a mostly transparent way, but need to be kept in mind in case of adaptations of the code to specific applications. Here we point out relevant "pecularities" of $\mathcal{OYSTER}$.

- **Array information** such as station names and coordinates are not strictly necessary to be present in the OIFITS data file, and models can be computed by $\mathcal{OYSTER}$ for them using the *uv*-coordinates. But OYSTER considers the latter as secondary data and usually computes these from the apparent star positions, observation dates and times, as well as the station coordinates. This is due to one of the original applications of $\mathcal{OYSTER}$ to astrometry with NPOI. In practical terms, use of astrometric routines such as `calcastrom` should therefore be avoided.

- **Object coordinates** are also not strictly necessary for the computation of model data, and thus $\mathcal{OYSTER}$ will read OIFITS even without the OI_TARGET table. However, internally each target must have an ID composed of a three letter catalog name (e.g. HDN or HIP), and an index into this catalog (e.g. HDN123306). This is the so-called star ID. Again this has to do with the use of $\mathcal{OYSTER}$ as astrometric software for NPOI. The $\mathcal{OYSTER}$ installation includes several tables in the *starbase* directory (e.g. *vlti.hdn* which are used for looking up catalog IDs for a given target name. If no entry can be found, the default OBJ designation is used (OBJ+HHMMSSFFDDMMSSF). This is usually not a problem except that the STARBASE procedures will not work anymore, such as compiling stellar data for the observed targets from installed catalogs.

- **General configuration** items, stored in the *genconfig* structure, will often be incomplete when reading an OIFITS file.

4.3 Programmer's reference

Almost all interferometer specific information is encoded in *oyster.pro*.

Chapter 5

Plotting and editing

We describe here all *OYSTER* plotting and editing procedures. For calibrated visibility data, as well as other data from e.g. astrometry, the user will select plotting menus from the so-called AMOEBA (lower) section of the main *OYSTER* GUI. While AMOEBA is described later on in a dedicated part of this manual, it is the general data *analysis* (as compared to *reduction*) facility of *OYSTER*. The following plot menu is available.

PLOT |

INTERFEROMETRY	Plot visibility data
ASTROMETRY	Plot ρ/θ data
SPECTROSCOPY	Plot radial velocity data
PHOTOMETRY	Plot magnitude measurements

Each of the above buttons will display a widget designed for this class of plot. Plotting of interferometry data is handled by the same routines as used in CHAMELEON (see dedicated part on this NPOI data reduction facility).

5.1 Interferometry

Most of the plots of interferometric data in *OYSTER* are handled by one procedure (`plotdata`). Consequently, there is one widget handling the plot selection (`ww_plot`). Whenever a plot is requested, a selection widget designed for that class of plot (point, scan, astrom, etc.) is opened, replacing any currently displayed selection widgets from previous requests. A second widget is created for the selection of which indices to plot, if this input is required. Plots can be 2-dimensional or 3-dimensional. Therefore, three data streams (`'x'`, `'y'`, and `'z'`) are available.

5.1.1 The plot selection widget

Here you define what is plotted along the axes, for which stars, and other various options. If the choice for an axis required additional selection on what range of data indices to plot, another widget, the data selection widget will open (see below).

If more than one channel, baseline, or point is selected, multiple plots will appear in the plot widget for those indices not corresponding to the selected slice (the SLICE menu is optional in

classes scan and astrom; in class point the default slice is `pt`. In order to have all plots in a single graph, select the ALL IN 1 option. The PRESET menu implements a short-cut to setting the most common selection for plots of the visibility either for slice `pt` or `ch`.

Clicking on PLOT|SCREEN will display the plot. At this stage, you can fit functions to the data (FIT: the residuals are plotted and can be manually edited), enter a new plot range (RANGE: enter 0,0 for automatic scaling) or by defining a window (WINDOW: click left button to anchor upper left corner, click middle button fix size and shift box, click left button again to fix position or right button to select automatic scaling and exit), identify data points (IDENTIFY: click left button for repeated identifications, right button to exit), and edit data manually (EDIT).

PLOT |	
SCREEN	Open new plot window and display plot
FILE	Write PostScript output
EDIT |	
AUTO	Do automatic editing
ZERO	Do automatic zero value editing
UTIL |	(single plots only)
EDIT	Manual editing
WINDOW	Set new range by placing a box
RANGE	Enter new range
MEAN	Return average data value in a window
IDENTIFY	Identify data point
H-LINE	Display horizontal line
V-LINE	Display vertical line
FIT	Fit functions to data

As for editing, you have three options: automatic, zero, and manual editing. The first option will put the data through a median filter (point data) or edit outliers based on their deviation from a 3-rd order polynomial fit to the data (scan data). The second option will remove data points with value zero, and the last option (recommended) will let you define boxes to delete data inside or outside of a box. Note that the data selection applies to both plotting *and* editing.

OPTION |	
ERRORS	Display error bars
FLAGGED	Display flagged data too
LINES	Connect data points
ALL IN 1	Plot all graphs in one plot
3D	Open 3D plot widget

IMAGE	Plot 3rd dim in TV style
NOFIR	Fits to all data (FIR = Fit In Range)
NOSORT	Do not sort x-values
NOTRACE	'x' will not trace 'y' selection
ALL OBS	Do all spectrometers (uv-plots only)
ALL IBS	Do all IBs (not implemented)
MODEL	Plot model too (AMOEBA only)
COLOR	Select color printer
SMOOTH	Compute and plot model values also between observations
PAPER	Display camera-ready plot labels
CUSTOM	Vis. plot for NPOI: scale data to model

5.1.2 The data selection widget

A data selection widget is opened for a stream when the selected axis item requires further user input. However, if the NOTRACE option is de-selected (the default), such a selection widget will not be activated for the `'x'`-stream, which will be assigned the selection of the `'y'`-stream.

In the data selection widget, input beam, output beam, and triple have to be specified, depending on the selected axis item. The selection for channels, baselines, and points are for this input, output beam, or triple only. The ALL OB option is valid only for *uv*-coverage plots, where it implies that the data of all output beams, channels, and baselines shall be plotted.

A data selection for a specific index (e.g. channel) is made by combining a directive (ALL, NEXT, CURRENT, PREVIOUS, SELECTED) with a selection typed into the selection line. A selection line entry is a series of numbers separated by commas, and/or number ranges consisting of a lower and upper value separated by a dash. This entry is parsed into a vector of selected index values (e.g., '2,3,6-9' is parsed into '2,3,6,7,8,9'). *Please do not forget to hit <return> after typing in your selection line entry!*

NEXT and PREVIOUS return a selection of the next or previous n index values, where n is the *number* of selected index values in the selection line. Thus, if the selection line reads '1,2', NEXT will select '3,4' for the next plot, then '5,6' and so on. It is not recommended to select these directives for more than one axis at a time as they will act simultaneously. The CURRENT directive simply leaves the current selection unchanged (which is not necessarily identical to the displayed selection!).

It is not necessary to specify indices or alter the corresponding selection for axis items which do not contain those indices.

The pt and star selection address the same index. Therefore, the star selection is applied *after* the pt selection as displayed in the data selection widget. When plotting scan data, you can obtain information on which scans correspond to given stars from the UTILITIES|LIST|SCANS button.

5.1.3 Flagging data

Flagging data is allowed in only some plot classes (point, bg, and scan), and manual flagging only if the plot window contains a single plot (remember to use the ALL IN 1 option to plot

a selection of data into one plot). Automatic flagging is recommended only for point data. In the manual mode, you will define a box on the plot by first clicking the left mouse button on the upper left corner, then set the size/shape of the box by moving the cursor and clicking the middle button, then positioning the box. Click on the left button to place another box, or the right button to finish. Finally you will be presented with a list of options of what to do with the boxed data. Flagged data will be colored red.

5.1.4 Logical editing and flag tables

The scope of bad data flags can go beyond the currently selected item, if other data items exist which depend on the selected one. This means that a particular point of data is unlikely to be valid if the item on which it depends is flagged. Thus, for example, flagging a photon rate point will cause CHAMELEON to flag all visibilities where this photon rate point has a contribution.

Flagging information will be automatically entered into flag tables, with a label attached to every entry specifying the reason for the flag. (The reason currently is derived from the system date/time.) A list of these labels will be displayed if the UNFLAG button is selected. Each entry is identified by the exact time of observation, and the data selection. Flag tables can be saved to disk for later retrieval. The name of the disk file holding flag tables is made up of the body of the input data file name plus an extension *.flg*.

5.2 Astrometry

UTIL |

IDENTIFY — Identify data points. Program enters loop in which left mouse button click identifies, right button click identifies and exits the loop. If no plot window is displayed, a list of the measurements will be shown with the individual χ^2 contributions to identify bad data.

WINDOW — Change display area

RANGE — Enter new range of display

ELLIPSE — Interactively define an ellipse. Use left mouse button to set or move either semi-major axis end points, the middle button to set or change the ellipticity, and the right button to exit either mode, and the utility upon clicking twice.

ORBIT — Run Thiele-Innes method to obtain inital estimate of orbital parameters. The results are stored in a single orbit storage area.

DATE — Enter a date and display the position.

EDIT — Edit data. No flag tables used.

OPTIONS |

ERRORS — Display error ellipses

FLAGGED — Display flagged data too

ELLIPSE — Superpose apparent ellipse

ORBIT — Superpose apparent orbit

J2000	Precess observations to J2000 before plotting
Color	For hardcopies, use color
Submit	Submit hardcopy file to printer

5.3 Spectroscopy

Util |

Identify	Identify data points
Window	Change display area
Range	Enter new range of display
Edit	Edit data

Options |

Errors	Display error ellipses
Flagged	Display flagged data too
Phase	Plot as a function of phase instead of time. Use the period of that orbit where the selected component appears alone. Subtract the systemic velocity of that binary component.
Orbit	Superpose apparent orbit

5.4 Photometry

Util |

Identify	Identify data points
Window	Change display area
Range	Enter new range of display
Edit	Edit data

Options |

Errors	Display error ellipses
Flagged	Display flagged data too
Phase	Plot as a function of phase instead of time. Use the period of that orbit where the selected component appears alone. Subtract the systemic velocity of that binary component.
Orbit	Superpose apparent orbit

Chapter 6

Programmer's reference

6.1 Code

All source code resides in *oyster/source*, subdirectories *common*, *idl*, and *c*, as well as the directories for the HDS, NOVAS, FITSIO, and WD code libraries. A *Makefile* is used to compile the C code, whereas a command file, *c.pro*, is used to compile the scripts. Note that the compilation order in *c.pro* has some significance, as functions in general have to be compiled first, that is before they are used in any procedure. In the GUI source code directory *idl*, type idl c to start up IDL and have it compile all procedures and functions. The command file will also save the compiled code to *oyster.cpr* and the common block definitions to *oyster.cmb*. If you would like to change the way a procedure works in your $\mathcal{OYSTER}$ session, make a copy of the source code (call it *myfile.pro*, for example), modify it, and recompile it using .run *myfile.pro*. For your current session, this compiled new code will replace the previous one. It is a good idea to place modified routines into a file and compile them after startup.

6.1.1 Script library files

Here we list the files (in *italics*) which contain IDL code. The itemization is not complete, however, and only intended to provide an overview of some $50,000+$ lines of code. Note that except for the IDL specific widget libraries, all code is in the same area (*common*).

- $\mathcal{OYSTER}$
  - *oyster.pro*. Contains the setup script and common data functions.
  - *idlfunctions.pro*. Functions which exist in PV-WAVE CL but not IDL.
  - *wavefunctions.pro*. Functions which exist in IDL but not PV-WAVE CL.
  - *functions.pro*. Math and miscellaneous functions.
  - *time.pro*. Date and time conversion, Julian date, formatting and parsing of date strings.
  - *structure.pro*. Allocation of data structures.
  - *mainwidget.pro*. All scripts related to the top level widgets of CHAMELEON and AMOEBA.
  - *misc.pro*. Miscellaneous unsupported procedures.

- CHAMELEON
  - *chameleon.pro.* Basic point data reduction and scan calibration.
  - *access.pro.* All higher level HDS data file access scripts.
  - *hds.pro.* All HDS CL wrappers. These verify arguments and call the C-wrappers which in turn call HDS library functions.
  - *astrom.pro.* Basic astrometric procedures.
  - *plotwidget.pro.* All plot widget scripts for AMOEBA and CHAMELEON.
  - *plotting.pro.* All general plot scripts.
- AMOEBA
  - *ameeba.pro.* Basic data reading and model computation.
  - *model.pro.* Model data computation functions.
  - *fitwidget.pro.* Managing of fit widgets.
  - *fitting.pro.* Fitting of hierarchical models.
  - *math.pro.* Math routines in CL.
  - *limb.pro.* All limb darkening code.
  - *filter.pro.* Filter transmission functions.
- STARBASE
  - *starbase.pro.*Basic data base routines.
  - *catalogs.pro.* Access of auxilliary (secondary) catalogs (lists).
  - *stars.pro.* Estimation of stellar parameters.
  - *starplot.pro.* Miscellanous plot routines.
- COBRA
  - *cobra.pro.* Raw data manipulation.
  - *cobrawidget.pro.* Raw data viewing.
- PEARL
  - *pearl.pro.* Imaging.

6.1.2 C libraries

Each of the five directories containing external code have make files. Use the Makefile appropriate for your system to recompile them. Use make clean *first* to prepare the compilation. Please send copies of the Makefiles to the author if it was necessary to modify them.

OYSTER

- *chameleon.c.*
- *amoeba.c.*
- *cobra.c.* Raw packet data access.
- *hds.c.* HDS wrappers.
- *writefits.c.* The FITS writer.
- *posprop.c.* The Hipparcos epoch transformation.
- *sidfuncs.c.* NPOI siderostat model functions.
- *nrutil.c.* Numerical Recipes subroutines.

FITSIO

A library containing functions for FITS I/O. *OYSTER* uses this package to write out interferometric data in single source FITS format.

NOVAS

A library containing function of the Naval Observatory Vector Astrometry Subroutines.

- *novascon.c.*
- *novas.c.*
- *solsys2d.c.*
- *jpleph.f.*

6.1.3 Mixed and FORTRAN libraries

HDS

One of the libraries of the STARLINK (UK) astronomy software project. These subroutines implement the Hierarchical Data System used for NPOI data files. They can usually be downloaded from the STARLINK web site and therefore do not need to be compiled.

- *libhds.a.*

The HDS library, if not appropriate for your system, should be obtained directly from the STARLINK web site, *http://star-www.rl.ac.uk/*, following links to the Software Store, Subroutine Libraries, and HDS (*http://star-www.rl.ac.uk/cgi-store/storeform1?HDS*). Follow the instructions to unpack (*sh storeformhdr*), paying close attention to the target directories. Then, set environment variables STARLINK and INSTALL to point to the starlink installation directory. In each of the *ems*, *chr*, *cnf*, and *hds* sub-directories (*hds* must be processed last!), use the mk command with the targets, in this order, deinstall, clean, build, and then install to compile the *.a* library files, which are automatically placed in the *lib* subdirectory of the starlink home.

WD

This is the Wilson-Devinney code for the eclipsing binary model. They were modified for use in $\mathcal{OYSTER}$. This proprietary library is not included in the standard distribution.

- *fnl.f.*
- *imlc.f.*
- *imlight.f.*

ROCHE

This is Deane Peterson's code for the simulation of gravity-darkening in rotating stars. This proprietary library is not included in the standard distribution.

Part II

AMOEBA

Chapter 7

Introduction

7.1 About AMOEBA

AMOEBA procedures and functions are used to fit parameters of hierarchical stellar system models to data from interferometry, astrometry, spectroscopy, and photometry. The data from the latter three methods are read from formatted ASCII files and stored in structure arrays. Interferometric data, however, can only be stored in the scans structure one night at a time due to the complexity of the data and the varying configurations. Therefore, the different nights read by `load_interferometry` are buffered by a C-function (`nightbuffer`).

7.2 Data files

7.2.1 Interferometry

To model interferometric data, $\mathcal{OYSTER}$ has to compute model visibilities taking into account the measurement process and all its limitations.

The integration time, *scans.int_time*, is defined as the length of time over which the data were averaged to yield the given data point. For large sources, e.g. wide binaries, long integration times can significantly lower the averaged fringe contrasts. Taking the system variable !int_time to specify an upper limit for the integration time in seconds which will not require integrating the model visibilities, $\mathcal{OYSTER}$ will compare the actual integration times to this limit and determine the maximum number of computations required to cover the largest integration interval. The step size for each observation will be adjusted according to the actual integration time of the particular observation, given the number of computations.

7.2.2 Astrometry

Astrometry data file have to have the extension *.psn*. The following gives an example for Algol, where the columns give the hierarchical component designation, the Julian year of observation, separation (in mas) and position angle (in degrees from N over E), major and minor axis (in mas), and position angle of the major axis of the error ellipse (in degrees, from N over E).

```
AB-C 1991.7487 10.68   67.30   1.511  0.265  107.8
```

```
AB-C 1991.7545 15.70   83.18   0.589  0.091   86.8
AB-C 1991.8036 25.52  105.36   0.253  0.086   91.1
AB-C 1991.8336 33.11  111.73   0.186  0.061   92.6
AB-C 1992.6936 47.04  141.94   0.874  0.165   99.5
AB-C 1992.7756 19.16  160.03   0.283  0.090   95.6
AB-C 1992.7864 15.64  165.57   0.210  0.093  150.9
AB-C 1992.7921 14.27  174.39   0.714  0.133   87.8
AB-C 1992.8822 24.25  292.96   0.692  0.198   81.6
AB-C 1992.8930 28.01  295.10   0.956  0.208   92.9
AB-C 1992.9203 35.86  301.53   0.862  0.223   97.8
AB-C 1992.9450 43.11  305.27   1.106  0.237   91.1
```

7.2.3 Spectroscopy

Spectroscopy data files have the extension *.vel*. The following gives an example for Algol, where the columns are the component designation, the Julian date of observation, and value and error of the radial velocity (in km/s) measured. Optionally, plot symbol and color can be added to each record. Available symbols are + (0), x (1), * (2), triangle up (3), triangle down (4), square (5), diamond (6), pentagon (7), star (8) and circle (9). These are open symbol, unless 10 is added to the symbol number. The available colors are black (0), white (1), red (2), green (3), blue (4), and more. **New feature: composite component designations, e.g. AB, are allowed and the model prediction for such components will be computed from their center of mass velocity.**

```
A 2428890.9305 -16.8 1.8
A 2428918.6569 +54.2 4.4
B 2443477.748 +145.0 5.0
B 2443478.743 -199.0 5.0
B 2443497.646 +179.0 5.0
B 2443498.673 -169.0 5.0
B 2443501.689 -201.0 5.0
C 2422984.4 -10.5 1.6
C 2425125.5  +5.9 0.6
```

7.2.4 Photometry

Photometry data files have the extension *.mag*. The following gives an example for Algol, where the columns are the component designation, filter name, the Julian date of observation, and value and error of the magnitude measured. The filter names must be one of Hp, U, B, V, R, and I.

```
ABC V   2445249.299     -2.60900 0.001
ABC V   2445249.304     -2.60000 0.001
```

```
ABC V   2445249.313     -2.59200 0.001
```

7.3 The Hierarchical Stellar Systems Model Format

In a hierarchical stellar system, the separation between two neighboring stars is always much smaller than the separation of this pair from the next companion in the system, be it another pair or a single star. Hierarchical systems are dynamically stable, and therefore by far the most common type encountered. They are also numerically easier to handle than non-hierarchical systems, since each pair can be described with the standard orbital parameters, and each star with a small number of physical parameters. Because AMOEBA is designed to combine different data sets, the HSSMF eliminates parameters only specific to one dataset in favor of replacing them with physical parameters such as masses, luminosities, etc. It is important to not over determine the model, i.e. to allow one or more parameters to be functions of others. The following illustrates an example for a triple star, Algol.

```
; Global parameters:
starid          ='FKV0111'
wavelengths     =[0.550,0.800]
rv              =4.0
;
; Star parameters (for each star):
name(0)         ='A'
type(0)         =1
mass(0)         =3.67
diameter(0)     =0.98
omega(0)        =1.0
teff(0)         =0
gr(0)           =1.0
albedo(0)       =1.0
magnitudes(*,0) =[2.26,2.36]
;
name(1)         ='B'
type(1)         =1
mass(1)         =0.82
diameter(1)     =1.2
omega(1)        =1.0
teff(1)         =0
gr(1)           =0.3
albedo(1)       =0.5
magnitudes(*,1) =[5.23,4.26]
;
name(2)         ='C'
type(2)         =1
mass(2)         =1.88
```

```
diameter(2)     =0.58
magnitudes(*,2) =[5.1,4.8]
;
; Binary parameters (for each binary):
component(0)    ='A-B'
method(0)       =1
wdmode(0)       =5
semimajoraxis(0)=2.04
eccentricity(0) =0.0
inclination(0)  =97.69
periastron(0)   =91.86 ; of primary
apsidalmotion(0)=0.0
ascendingnode(0)=47.4
period(0)       =2.8673285
epoch(0)        =2441773.4894
; Fit to AB-C astrometry
component(1)    ='AB-C'
method(1)       =1
semimajoraxis(1)=94.6
eccentricity(1) =0.229
inclination(1)  =84.0
periastron(1)   =310.5
apsidalmotion(1)=0.0
ascendingnode(1)=312.3
period(1)       =679.9966
epoch(1)        =2453731.4d0
```

The syntax of the model format is identical to the language, e.g. IDL. The individual lines are actually commands which are executed by AMOEBA upon reading the model file.

Model parameters are defined in the following. Please note that Julian Day epochs (model parameters and data) are stored internally with 2440000 days subtracted.

- System
  - `Starid` Star identifier. Character string, CCCNNNN, or CCCNNNNNN
  - `RA` Right Ascension. Needed for precession computation.
  - `Dec` Declination. Needed for precession computation.
  - `Rv` Systemic radial velocity in km/s
  - `Wavelengths` The wavelengths corresponding to elements of the magnitudes array in the stellar parameter section. Any number of elements is allowed. In microns.
- Star
  - `Name/Component` Single character string, 'A', 'B', etc.
  - `WMC` WMC designation of component, e.g. Aa

  - `Type` Integer, described in the following section
  - `Model` Kurucz atmosphere model to use, e.g. ip00k2.pck19, or name of Aufdenberg atmosphere (without *.dat* extension). In case of images, the filename.
  - `SED` Name of XDR file which restores wavelength l and flux f (in equal wavelength bins) if to be used to define the SED of a component.
  - `Mass` In units of the solar mass
  - `Diameter` In milliarcseconds
  - `Ratio` Axial ratio minor/major axis (elliptical components or CLEAN beam
  - `PA` Position angle of major axis
  - `Hole` Disk hole in milliarcseconds
  - `Omega` Ratio of axial rotation rate to breakup rate (type 14) or ratio of axial rotation rate to orbital rate (WD)
  - `Tilt` Inclination of rotation axis, zero is pole-on
  - `Teff` Effective temperature in K
    * 0 Set intrinsic flux to 1
    * < 0 Use blackbody law
    * > 0 Use model atmosphere
  - `Logg` Logarithm of surface gravity
  - `gr` Exponent in gravity darkening law, convective envelopes 0.3, radiative 1.0
  - `albedo`
  - `alpha` Temperature exponent in passive disks
  - `magnitudes` Apparent visual magnitude. These are converted to flux factors and applied to the fluxes derived from the effective temperature parameter.

- Binary
  - `Component` Name of component, character string
  - `WDmode` WD mode
  - `Method`
    * `1` Use orbital elements
    * `2` Use (ρ,θ) parameters
    * `3` Use (ρ,θ) parameters with orbital motion
    * `4` Interacting binary, use WD code
  - `Semimajoraxis` in milliarcseconds
  - `Eccentricity`
  - `Inclination` in degrees
  - `Periastron` in degrees, of primary
  - `Apsidalmotion` in degrees/year
  - `Ascendingnode` in degrees

- `Period` in days
- `Epoch` Full Julian date, periastron passage
- `Rho` Separation, in milliarcseconds
- `Theta` Position angle, in degrees

The model is checked upon reading to make sure all components are defined. All wavelength dependent parameters are defined at a set of wavelengths given in the global parameter section, and therefore all have to have the same number of elements. (Polynomials are used to interpolate intermediate values.)

7.4 Stellar models

The `type` parameter allows to select from a range of stellar models, either analytical, or available as data files or FITS images. Please note that a new numbering scheme was introduced with Version 6.12 (2 Aug 2007)

For all types from zero to 11, the flux distribution (spectrum) is computed as flat (`teff=0`), a black body (`teff < 0`), or derived from a stellar atmosphere (`teff > 0`). For all types, specifying an arbitrary SED will override stellar spectra or fluxes derived from the images. The XDR file to read is specified in `sed`, and should restore variables l and f, which contain the wavelength in meters and the flux (per wavelength interval), respectively. Please note that if the effective of a component is non-zero, the flux will be scaled with the squared diameter! As mentioned above (`readmodel`), please keep in mind that an internal value of -5555 for an effective temperature is interpreted as zero.

Stellar spectra and limb-darking coefficients as a function of `log(g)` and `teff` are from Van Hamme (1993) and are stored in file *atmospheres/vanhamme/limbdata.vh.xdr*. For stellar temperatures less than 3500 K it is possible to use the NextGen models (*http://phoenix.ens-lyon.fr/Grids/*) by copying one of the files *atmospheres/vanhamme/limbdata.n?.xdr* to *limbdata.xdr*. Please note that the NextGen models do *not* provide the limb-darkening coefficients! Further information can be found in the file *Notes.txt*.

In many models, specifying `pa` and `ratio` is used to rotate and stretch the (u, v) coordinates. The only exceptions are the images if a positive non-zero diameter of the CLEAN beam have been specified.

7.4.1 Uncorrelated flux component

Type 0

- The correlated flux of this component is zero.

7.4.2 Uniform disks

Type 1

- The disk diameter is independent of wavelength.
- Parameters used: `diameter` (UD), `pa`, `ratio`

Type 2

- The disk diameter scales with wavelength, $d = \sqrt{(1 - 7\mu/15)/(1 - \mu/3)}$ where μ is the linear limb darkening coefficient.
- Parameters used: `diameter` (LD), `pa`, `ratio`, `teff`, `logg`

7.4.3 Limb-darkened disks

Type 3

- The linear law is used for an analytical result. The coefficients are from Van Hamme.
- Parameters used: `diameter` (LD), `pa`, `ratio`, `teff`, `logg`

Type 4

- The logarithmic law is used to produce maps which are then Fourier transformed.
- Parameters used: `diameter` (LD), `pa`, `ratio`, `teff`, `logg`

Type 5

- Linear law, with coefficients directly from Kurucz models.
- Parameters used: `diameter` (LD), `pa`, `ratio`, `teff`, `logg`

Type 6

- Hestroffer law, with coefficients directly from Kurucz models.
- Parameters used: `diameter` (LD), `pa`, `ratio`, `teff`, `logg`

Type 7

- Semi-analytical transformation using stellar profiles directly from Kurucz models. Only (`model`) atmosphere available so far is ip00k2.pck19.
- Parameters used: `diameter` (LD), `pa`, `ratio`, `model`, `teff`, `logg`

Type 8

- Semi-analytical transformation using stellar profiles directly from Aufdenberg models. Models are only available upon request.
- Parameters used: `diameter` (LD), `pa`, `ratio`, `model`, `teff`, `logg`

7.4.4 Miscellaneous stellar disks

Type 9

- Numerical transformation of elliptical uniform disk, for test purposes.
- Parameters used: `diameter`, `pa`, `ratio`, `teff`, `logg`

Type 10

- Elliptical Gaussian disk
- Parameters used: `diameter`, `pa`, `ratio`, `teff`, `logg`

Type 11

- Pearson disk. Produces exponentially decaying visibilities.
- Parameters used: `diameter`, `pa`, `ratio`, `teff`, `logg`

7.4.5 Images

OYSTER handles two types of images, one being the classical type FITS image or image cube, the other one being the PEARL image. The latter is a parametric image, assigning an effective temperature to each pixel.

Type 12

- FITS image cube. The filename is given in `model`. If a non-zero positive diameter is specified, its value, position angle and axis ratio are used as CLEAN beam parameters which is deconvolved from the image, i.e. the visibilities are divided by ther transform of the Gaussian CLEAN beam.
- Parameters used: `model`, `diameter`, `pa`, `ratio`

Type 13

- PEARL image. As described for type 12, visibilities can be normalized by a CLEAN beam.
- Parameters used: `model`, `diameter`, `pa`, `ratio`

7.4.6 Rotating stars

Type 14

- Includes Roche sphere computation and gravity darkening, using code library written by D. Peterson.
- Parameters used: `diameter`, `pa`, `omega`, `tilt`

7.4.7 Young stellar objects and disks

Type 15

- Hillenbrand passive disk, optically thick, with temperature profile.
- Parameters used: `diameter`, `fractional width`, `pa`, `alpha`, `ratio`. The diameter is the size of the inner hole, the outer radius is diameter*(1+width)/2.

Type 16

- DUSTY simulation. Reads the radial intensity map, scales the shell radii with the given diameter (mas), and computes the Fourier transform.

- Parameters used: `model`, `diameter`

7.5 Fit scenarios

7.5.1 Single-lined spectroscopic binary

Select the appropriate orbital elements, and the mass of the secondary. The reason for not choosing the mass of the primary is that with decreasing primary mass, the situation is asymptotically reached where the primary settles on a Keplerian orbit of a planet sized body around a massive secondary and its velocity amplitude would not not increase anymore. This would prevent proper fitting of primary velocity curves of larger amplitudes. If, after fitting, the corresponding velocity amplitude K_1 is desired, one uses the `modelk` function, e.g. PRINT, MODELK('A').

7.5.2 Double-lined spectroscopic binary

Add the primary mass as a fit parameter, but remember, as in the single-lined case, that these masses are not true masses since it is only the velocity amplitude as a function of several parameters including the masses that is effectively constrained by the radial velocity data. In other words, the mass ratio is constrained, but the scale.

7.5.3 Single-lined spectroscopic binary with astrometric orbit

Add the remaining astrometric orbital elements to the fit parameters, and vary the secondary mass only. If you have a measurement of the parallax, say from the Hipparcos catalogue, you can add this measurement using the `set_parallax` procedure. In this case, you may add also the primary mass to the fit parameters as the system is now constrained just like a double-lined spectroscopic binary.

7.5.4 Triple with velocity curves of close pair

Select all three masses as fit parameters. The mass ratio of the close pair will be constrained, but as far as the absolute masses, this system is identical to a single-lined binary.

7.5.5 Triple with velocity curves for all components

Fit all three masses, as well as the inclination of one of the pairs. That not any combination of orbital inclinations is allowed can be seen from the fact that the triple consists of two double-lined binaries and only one mass solution exists for a given inclination. Which inclination to vary depends on whether the adopted value of the other orbital inclination is larger or smaller than the true value.

Orbital phases in spectroscopy plots

Plots of the radial velocity curves versus orbital phase can be made for all three components using the widget PLOT|SPECTROSCOPY. When plotting the velocity curves for the individual components of a triple versus orbital phase the lowest hierarchical component is used to calculate the phase. For components A and B the A-B orbital parameters are used to calculate the phase. For component C the AB-C orbit parameters in the model are used to calculate the orbital phase. Plotted velocities of lower level components will have the contribution from higher components removed. In this example with velocity data for all three components the close binary orbit is used to calculate orbital phase for components A and B. The plotted velocities of components A and B will have the velocity of component C removed.

7.6 Pass band integrations and field of view

As any interferometer, even those with spectrometers, measure the visibilities integrated over band passes of non-negligible widths, $\mathcal{OYSTER}$ computes model visibilities on a grid of wavelengths. The results (visibilities, fluxes) are then integrated over wavelength, whereby special attention needs to be payed to the averaging corresponding to what the instrument actually does.

The grid chosen is usually appropriate for the selected interferometer; if the spectral resolution is high, the grid wavelengths correspond exactly to the channel center wavelengths. The choice of grid is made in function `calcmodel`, for each spectrometer.

Finally, the photometric field of view is also computed based on the diameters of the telescopes and the seeing during the observation. The seeing values are stored in *scans.r0* and default to 1".

Chapter 8

Basic widget procedures

This section describes the complete line of model computation and fitting procedures within the widget environment. To invoke the main widget, type the following command at the IDL prompt:

```
IDL> oyster
```

8.1 Load data

AMOEBA is designed to handle data input from interferometry, astrometry, spectroscopy, and photometry. (The data for the first type, including configuration information, is buffered, i.e. loaded one night at a time into the ScanData section from the buffer. This procedure was necessary since CHAMELEON procedures handling the data cannot handle more than one night at a time.)

The loading of the data is handled by widgets, which also allow to set flags and weights to be used in the subsequent data analysis. The files are selected from a list (option LIST) of files with the proper extension found in the current directory and displayed in the widget, or from a file giving the file names (option FILE). The DATA|SUMMARY button gives information about the currently loaded data sets and their relative weights. If weights need to be changed, this can be done at any time after loading the data by simple selection if the corresponding data loading widget is displayed, or if not, by redisplaying it.

DATA |

INTERFEROMETRY	Load visibility data
ASTROMETRY	Load ρ/θ data
SPECTROSCOPY	Load radial velocity data
PHOTOMETRY	Load magnitude measurements
SUMMARY	Print summary of currently loaded data

8.2 Manipulation of a model

The following menu bundles all routines related to reading and fitting of the hierarchical model. The fit routine should not be confused with specialized and customized fitting routines to be found elsewhere (in AMOEBA|FIT).

Model |

READ	Read and check a model file.
Info	CL info,/struct.
Calc	Compute model data.
FIT	General model fitting widget. Results are stored in the model.

The general model fitting widget displays all components of the model, and the parameters associated with each one of them. Whether or not all or a subset of parameters are constrained by the data is not determined by AMOEBA, but has to be decided by the user. However, since a SVD algorithm is used for the fitting, no crash will occur if too many fit parameters are selected and the design matrix becomes singular. Characteristics on control parameters of the non-linear iterative fit can be modified using the FIT|CONTROL widget.

Data selection for the model fit comprises all data sets with a non-zero weight. In addition, one can select single interferometry scans in the plot widget for the fit, while others are ignored until the scan selection is set back to All.

8.3 Fit routines

Under this menu (Fit) are listed specialized fit routines for the different data types. Results are not stored in the hierarchical model, but either in separate areas or written to file.

8.3.1 AstrometryFit

This widget handles initial single orbit fits to improve the apparent ellipse or orbit as determined by the Ellipse and Orbit utilities in the OrbitPlot widget. The procedure is as follows: the user defines an ellipse first, then fits the apparent ellipse to the binary positions using Astrometry|Ellipse. Then OrbitPlot|Util|Orbit is used to estimate the orbital elements, which are then improved by AstrometryFit|Orbit. If the fit was successful, AstrometryFit|SetModel will store the elements in the appropriate model component.

8.3.2 InterferometryFit

This procedure is used to fit (ρ, θ) pairs to individual nights. The list of nights is derived from the ones currently loaded, and they are re-loaded (`load_interferometry`) despite the overhead in order to allow data sets of the same night to be combined. Initial estimates for (ρ, θ) are obtained from the orbital elements. The results are written to file *fitnights.psn*, including fitted synthesized beam widths. This file is in the standard astrometry format and can be read back into AMOEBA.

8.3.3 Control

This menu contains variables which control the performance of the Marquardt-Levenberg (M-L) non-linear least squares fitting.

Fit options |

One	One iteration only
Float	Allow V^2 calibration to float
Opt.	Optimize step size for M-L.
Lambda	Gradient-Brute force parameter
Tolerance	Ratio of smallest over largest eigenvalue allowed
Convergence	Convergence criterion as ratio of new over previous χ^2

Choosing the Opt. option will cause a one-time delay before starting M-L, including the loading of the interferometric data a second time as the optimization of the step size overwrites the currently loaded visibilities. The step size data are written into a file *Z_h_steps.xdr*.

Chapter 9

Command line procedures

9.1 Interferometry data buffering

`storenight`,*mode*<_int>

Manipulate interferometric scan data buffer.

mode

0 Free allocated buffer

10 Store new night

11 Store/overwrite existing night in buffer

`loadnight`,*night*<_char>

Load night from buffer.

9.2 Model manipulation

`readmodel`,*file*<_char>

Read a model file. If the effective temperature of a component is zero, it is set to −5555 Kelvin for compatibility of the AMOBA and PEARL procedures. Within AMOBA, this value however is interpreted as Zero.

`calcmodel`,[*/pearlmv*][,*/pearlcv*]

Compute the model data for data sets with non-zero weight. The additional parameters select passing the computed complex visibilities to PEARL and/or subtracting them from the complex visibility data in PEARL. Please see section on PEARL for more details.

`modelchisq`()

Function. Return reduced χ^2 of currently loaded data wrt model.

`binarypos`([*epoch*<_real>[, *component*<_char>]][,*lambda=lambda*][,*com=com*][,*abs=abs*])

Function. Return ρ/θ relative position for binary model at JD epoch. Use today if no epoch is specified, and use the top hierarchical component if no component is specified. If a component is specified, so must be the epoch. If *com=1*, compute relative to center of mass, not center of brightness. In the latter case, one can specify *lambda* (in meters) for the wavelength to use. If *abs=1*, return individual component positions instead of relative position. In that case, the component index is in the first dimension, (ρ, θ) are in the second.

`modelk`(*component*<_char>)

Function. Return velocity semi-amplitude for component. Examples: PRINT,MODELK('A'), PRINT,MODELK('AB').

`modelpx`(*component*<_char>)

Function. Return parallax derived from parameters of the specified component. Note that for systems higher than binaries, the results could be inconsistent, indicating an inconsistent set of model parameters.

`modelvel`(*JD - 2400000*<_real>, *component*<_char>)

Function. Return predicted velocities derived from parameters of the specified hierarchical model components.

9.3 Auxilliary data

`set_parallax`,*value*, *error* [,*weight=weight*]

Set a value for the system parallax. Set weight to zero to ignore stored value, to positive value to re-instate (if neither value nor error are provided). Set value to -1 to enforce the same parallax for all binary components, without enforcing a specific value.

Chapter 10

Programmer's reference

10.1 Organization of data

AMOEBA data is exclusively contained in the ScanData common area, whereas the model data resides in the Model common block.

If some currently loaded data have been manipulated, they have to be stored in the buffer (`storenight`) for AMOEBA routines to load the new data. A general exception has been implemented which does not load data from the buffer if the buffer has only one night stored.

Part III

STARBASE

Chapter 11

Introduction

11.1 About STARBASE

STARBASE is a collection of scripts written in IDL with additional functions written in C callable from within IDL. The scripts are run interactively.

STARBASE provides a table structure which can be filled with astrometric and astrophysical information on a list of stars from catalogs, auxilliary data files, and by computation. The star list can be either an entire catalog or a list compiled by the user. The former option can be invoked using a single command for various catalogs (e.g. the Bright Star Catalog, `get_bsc`). The name of the data base variable is STARTABLE.

STARBASE distinguishes several types of external data files. Primary catalogs have their own unique identifier, e.g. HDN, and are accessed using C external functions. These catalogs are static and never changed. Secondary catalogs are either extensions of primary catalogs (as in the case of HDN, which is based on SkyCat 2000, only complete to $V = 8$, and does not contain many of the MIDI calibrators), or implement dynamical catalogs (as in the case of MIR, for mid-infrared targets not contained in the HD primary catalog). Finally, lists contain specific data for selected stars identified by a primary catalog ID, e.g. jhk.hdn contains JHK magnitude for selected HD stars.

11.2 The format of STARTABLE

Inside IDL, STARTABLE is in the table format. Thus for example, to access the NAME field you access the variable STARTABLE.NAME. In the following we list all valid fields.

- STAR (string) Identifier of form CCCNNNNNN/CCCNNNN
- NAME (string) Star name
- VAR (string) Star name in Variable Star Catalogs
- TOE (string) Common name (e.g. Capella)
- RA (double) Right Ascension [h]
- RAE (double) Error in Right Ascension [h]

- DEC (double) Declination [deg]
- DECE (double) Error in Declination [deg]
- BAT (short int) Number in Batten's Catalogue
- FKV (short int) Number in Fundamental Katalog No. 5
- BSC (short int) Number in Bright Star Catalogue
- FLN (short int) Number in Finding list for Interacting Stars
- ADS (short int) Number in Aitken Double Star Catalogue
- HDN (long int) Number in Henry Draper Catalogue
- HIC (long int) Number in HIPPARCOS Input Catalogue
- SAO (long int) Number in Smithsonian Astrophysical Catalogue
- WDS (long int) Number in Washington Double star catalog
- MV (float) Johnson V magnitude (for combined system if multiple)
- BV (float) $(B-V)$ color (for combined system if multiple)
- UB (float) $(U-B)$
- RI (float) $(R-I)$
- DMV (float) V-magnitude difference for binaries
- AMV (float) Absolute V-magnitude (for combined system if multiple)
- BY (float) $(b-y)$ in Strömgren system
- M1 (float) $m1$ of Strömgren system
- C1 (float) $c1$ of Strömgren system
- BETA (float) β index of Strömgren system
- FEH (float) [Fe/H]
- PMRA (float) Centennial proper motion in RA [s]
- PMDEC (float) Centennial proper motion in DEC ["]
- RV (float) Radial velocity [km/s]
- PX (float) Parallax ["]
- PXE (float) Error of parallax ["]
- D (float) Distance [pc]

- SPECTRUM (string) Spectrum
- TYPE1 (float) Floating point spectral type of component 1
- TYPE2 (float) Floating point spectral type of component 2
- CLASS1 (float) Floating point luminosity class of component 1
- CLASS2 (float) Floating point luminosity class of component 2
- TEFF1 (float) Effective temperature of component 1 [K]
- TEFF2 (float) Effective temperature of component 2 [K]
- LOGG1 (float) Surface gravity of component 1 [cgs]
- LOGG2 (float) Surface gravity of component 2 [cgs]
- P (float) Orbital period [d]
- T (double) JD epoch of periastron
- O (float) Argument of periastron [deg], of primary
- E (float) Orbital eccentricity
- I (float) Orbital inclination [deg]
- N (float) Argument of ascending node [deg]
- A (float) Semi-major axis ["]
- K1 (float) [km/s]
- K2 (float) [km/s]
- V0 (float) [km/s]
- MF (float) Mass function for single lined binaries
- M1SIN3I (float) [km]
- M2SIN3I (float) [km/s]
- A1SINI (float) [km]
- A2SINI (float) [km]
- DIAMETER (float) Stellar diameter [mas]
- ZEROSPACING (float) Zerospacing visibility
- A0 Linear limb darkening fit coefficient, absolut
- A1 Linear limb darkening fit coefficient, linear

- A2 Linear limb darkening fit coefficient, quadratic
- MASS1 (float) Mass of component 1 [$\mathcal{M}_\odot$]
- MASS2 (float) Mass of component 2 [$\mathcal{M}_\odot$]
- BFLAG (string) Binary flag: 'B', calibrator 'C', other: '.'
- HFLAG (string) Hipparcos status flag, e.g. 'C'=component sol.
- SFLAG (string) Status flag (as a result of computations), 'OK' or '!'
- MODEL (string) Model on which diameter is based, e.g. 'LD'
- REFERENCE (string) General diameter reference, e.g. '080.D-0514'

11.3 Available primary catalogs

- HIC Hipparcos Input Catalogue:
  - HIC, HDN
  - RA, DEC
  - PMRA, PMDEC
  - RV, PX
  - MV, BV
  - SPECTRUM,TYPE1,TYPE2,CLASS1,CLASS2
- SAO Smithsonian Astrophysical Observatory Star Catalogue (National Space Science Data Center 1990):
  - SAO, HDN
  - RA, DEC
  - PMRA, PMDEC
  - MV
  - SPECTRUM,TYPE1,TYPE2
- HDN Sky Catalogue 2000.0, Volume 1, Second Edition (Sky Publishing Corp. 1991):
  - NAME
  - HDN, SAO, ADS
  - RA, DEC
  - PMRA, PMDEC
  - RV, PX
  - MV, BV
  - SPECTRUM,TYPE1,TYPE2,CLASS1,CLASS2

- FKV Fundamental Katalog No. 5:
  - FKV, HDN
  - RA, DEC
  - PMRA, PMDEC
  - RV, PX
  - MV
  - SPECTRUM,TYPE1,TYPE2
- BSC Bright Star Catalogue 4th Edition (Hoffleit 1982, National Space Science Data Center 1982):
  - NAME, VAR
  - BSC, HDN,
  - RA, DEC
  - PMRA, PMDEC
  - RV, PX
  - MV, BV, UB, RI
  - SPECTRUM,TYPE1,TYPE2,CLASS1,CLASS2
- BAT Eighth Catalog of the Orbital Elements of Spectroscopic Binaries (Batten, Fletcher, & MacCarthy 1989):
  - NAME
  - BAT, HDN
  - MV
  - SPECTRUM,TYPE1,TYPE2,CLASS1,CLASS2
  - A, E, I, O, N, P, T, K1, K2, MF, M1SIN3I, M2SIN3I, A1SINI, A2SINI
- FLN A Finding List for Observers of Interacting Binary Stars (Wood *et al.* 1980)
  - FLN, HDN
  - MV
  - SPECTRUM,TYPE1,TYPE2,CLASS1,CLASS2
  - P, T
- WDS Fourth Catalog of Orbits of Visual Binary Stars (Worley & Heintz):
  - NAME
  - WDS, ADS
  - RA, DEC
  - MV, DMV
  - SPECTRUM
  - A, E, I, O, N, P, T

11.4 File usage

STARBASE accesses the following primary star catalogs in *catalogs*:

- *fk5/fk5.dat*
- *bsc4/bsc4.dat*
- *hipparcos/hip_main.dat*
- *hipparcos/input/hipp.dat*
- *batten/sb8.dat*
- *skycat/hdn.dat*
- *findlist/findlist.dat*
- *sao/sao.dat*
- *wds/wdsorb.dat*

STARBASE accesses the following secondary star catalogs in *catalogs*:

- *hdn/HDN.xdr*
- *mir/MIR.xdr*
- *cal/CAL.xdr*
- *bbc/BBC.xdr*

It also accesses lists in *catalogs/npoi*, namely:

- *diameters.bsc* Compiled from various sources
- *ubv.hdn* Mermilliod 1991
- *ubvri.hdn* Nagy & Hill 1980
- *uvbybeta.hdn* Hauck & Mermilliod 1990
- *jhk.hdn* 2MASS at IPAC
- *feh.hdn* Cayrel de Strobel *et al.* 1985
- *parallax.hdn* van Altena *et al.* 1991
- *toe.bsc* from Bright Star Catalogue
- *position.hdn* van Altena *et al.* 1991

Chapter 12

How to work with STARBASE

12.1 Introduction

At this time, STARBASE is used to perform two different tasks, namely reading primary and secondary catalogs, and computing astrophysical quantities from the collected information. The data is stored in different fields of a table, STARTABLE (see next section).

Primary catalogs reside in subdirectories of *catalogs* and are read by the C-function `catalog`. Stars in the primary catalogs have unique ID numbers. Entries are in the order of these numbers. Information found in the primary catalogs *replaces* existing information in STARTABLE. The available data for each primary catalog is listed below.

Secondary catalogs are lists of specific data on stars, e.g. UBV photometry. They were derived from various astronomical catalogs by extracting the ID number (usually the HDN) and the desired data. Secondary catalogs reside in *catalogs/npoi* and have an extension indicating the identifier for the ID numbers (e.g. HDN). They are read by PV-WAVE procedures (command `dc_read_free`), and are used to *replace* information in STARTABLE.

Any star in STARTABLE is defined through a string of format CCCNNNNNN or CCC-NNNN, where CCC indicates the primary catalog and NNNNNN/NNNN the ID number in that catalog. Valid catalog identifiers are FKV, BSC, FLN, BAT, SAO, HDN, and HIC. Please use 4 digits numbers NNNN for the small catalogs FKV, BSC, FLN, and BAT. Use 6 digit numbers NNNNNN for the large catalogs SAO, HDN, and HIC.

12.2 Managing a calibrator catalog

This section is dedicated to a topic not of common interest, and therefore just serves as a container for the procedure developed to manage calibrators for ESO.

The secondary catalog holding calibrators submitted by the PIs of ESO programmes is *CAL.xdr*. To add calibrators, use the `compile_cal` procedure, and then `update_cat` to add the entries to the catalog. Using `write_calvin`, a list of calibrators can be written in the format used by the CalVin database.

Chapter 13

Command line procedures

In this section we list some command line procedures.

13.1 Allocate/load STARTABLE

`create_startable`,*starids*	Allocate STARTABLE, initialize star IDs.
`read_catalogs`	Read catalogs corresponding to CCC identifier.
`get_startable`,*starids*	Compound procedure to allocate and read.

Example: get_startable,['FKV0193','BSC1412','HDN198217']. Remember that the catalog identifier (e.g. HDN) determines which catalog is to be accessed for that particular star. See next section for routines to rename stars and inquire cross indices.

`list_star`,*star_id*<_char>

> Print currently loaded information on star to screen. Star is identified by catalog and number, e.g. BSC1708, FKV0193. For more options, see part on STARBASE.

`get_startable`

> Create startable and fill with catalog information for stars listed in scantable. Note that in this mode (i.e. without parameter), the procedure will prepare the table specifically for use by CHAMELEON. This involves getting the stellar positions from the Hipparcos catalog, diameters from *diameters.bsc*, and colors from the bright star catalog. If a list of stars is specified, just the catalog data available in the specified catalogs is loaded.

`rename_starids`,*from-to*<_char>

> Rename stars, e.g. from FKV designation to BSC designation, in which case *from-to* would be 'fkv-bsc'.

13.2 Read entire primary catalogs

Note that your workstation has to have enough memory installed to hold the entire startable.

`get_bsc`	9096 entries.
`get_bat`	1469 entries.
`get_fkv`	4652 entries.
`get_fln`	3564 entries.
`get_sky`	49418 entries.
`get_wds`	1635 entries.

13.3 Print/list information

`list_star`,*star*

Print (some) information from startable on star to screen. For *star*, the standard combination of catalog acronym and identifier can be used, or the Flamsteed and Bayer names, or a proper name. Examples: list_star,'FKV0193'; list_star,'alp aur'; list_star,'Capella'. The second example works only if the NAME field was set.

`list_note`,*star*[,*note*]

Print notes to screen or save in *note* if specified. Notes must have been read previously with `read_notes`.

`write_stars`

Write startable to LATEX format output file.

`write_npoi`,*startable*<_char>,*file*<_char>

Write startable to file in NPOI embedded catalog file format.

13.4 Lists

Note that data only for those stars in the table is retrieved which have the right ID available. If, for example, there is no BSC number in the table (*startable.bsc*) for a specific star, no diameter will be read for it.

`get_diameter`	Diameters, BSC identifier.
`get_parallax`	Parallaxes, HDN identifier.
`get_position`	Coordinates, HDN identifier.
`get_ubvri`	UBVRI photometry, HDN identifier.
`get_ubv`	UBV photometry, HDN identifier.
`get_uvbybeta`	Strömgren photometry, HDN identifier.
`get_feh`	[Fe/H] metallicity, HDN identifier.
`get_toe`	Common names, BSC identifier.

13.5 Plot data from the WDS

Visual and speckle observations from the Washington Double Star catalog can be plotted if a startable with good epoch 2000.0 positions is loaded. The coordinates are used to locate a binary in the WDS. The data are placed into a file */tmp/wdsdat.psn*, which can be read by AMOEBA for orbit fitting.

`plot_wds`,*starid*<_char>

This command will print out the following (example here: Capella) if a star is found:

```
RA    Dec Disc.+#Comp._DafDal#OPAfPAlSepfSeplMagAMagBSpectrum_pmrapmdeDM######No
hhmmf ddmm             YYYYYY  degdeg  "   "  m   m            s    "
                       y+1000                                /century
05167+4600ANJ   1Aa    91999799    49      01 008    G3III    +076-425+45 1077NO
05167+4600BAR  25AB    898898 1 23 23 466 466 21 171
05167+4600BU 1392AC    878878 1318318 782 782 21 151
05167+4600BU 1392AD    878878 118318312621262 21 136
05167+4600BU 1392AE    878878 131631614321432 21 121
05167+4600HJ 2256AF    8519221014714416951415 21 111          +003-006        Np
05167+4600SHJ  51AG    821895 234834845424846 21 101
05167+4600FRH   1AH    895    1141   7233     21 117                          N
Component to select:
```

Enter the component name you wish to select (Aa, for example), and the available data will be plotted.

13.6 Stellar parameters from calibrations of spectral types

Files containing stellar parameter data reside in *oyster/starbase*. Extensive photometric data is found in files *spec_par.X*, where *X* signifies the luminosity classes for dwarfs, giants, and supergiants, *V, III, Iab*, respectively. For subgiants, brightgiants, and the subclasses of the supergiants, limited data is in files *spec_MV.IV*, *spec_MV.II*, *spec_MV.Ia*, *spec_MV.Iab*, *spec_MV.Ib*.

13.6.1 Absolute visual magnitudes

Absolute visual magnitudes for stars are derived from their spectral type and luminosity class. Calibrations are stored in files *spec_par.V, spec_MV.IV, spec_par.III, spec_MV.II, spec_MV.Ia, spec_MV.Iab, spec_MV.Ib* in *oyster/starbase*.

`amv_stars` Compute MV for all stars in startable.

`amv_star`(*spectrum*<_char>)

Function. Return MV for star with specified spectrum.

13.6.2 Masses

Masses for single stars are derived from their spectral type and luminosity class. In addition, for double lined binaries with the secondary component not classified, `mass_binaries` will compute the secondary mass from $m_{1,2} \sin i^3$. Calibrations are stored in files *dwarfs.masses, spec_mass.IV, spec_mass.III, spec_mass.II, spec_mass.I* in *oyster/starbase*.

`mass_stars` Compute masses for stars in startable.
`mass_binaries` Compute secondary mass.
`mass_star`(*spectrum*<_char>)

Function. Return mass for star with specified spectrum.

13.6.3 Gravities

Gravities for stars are derived from their spectral type and luminosity class. Calibrations are stored in files *spec_logg.V, spec_logg.IV, spec_logg.III, spec_logg.II, spec_logg.I* in *oyster/starbase*.

`logg_stars` Compute log(g) for all stars in startable.
`logg_star`(*spectrum*<_char>)

Function. Return log(g) for star with specified spectrum.

13.6.4 Effective temperatures

Temperatures of stars are derived from their spectral type and luminosity class. Calibrations are stored in files *spec_par.V, spec_par.IV, spec_par.III, spec_par.II, spec_par.I* in *oyster/starbase*.

`teff_stars` Compute effective temperature for all stars in startable.
`teff_star`(*spectrum*<_char>)

Function. Return effective temperature for star with specified spectrum.

13.6.5 Diameters

Compute stellar diameters, in units of the Solar diameter, from the spectral type.
`diam_stars` Compute diameter for all stars in startable.
`diam_star`(*spectrum*<_char>)

Function. Return diameter for star with specified spectrum.

13.7 Limb darkening

Linear limb darkening coefficients are taken from tables published by Van Hamme (1993, AJ 106, 2096).

`limb_stars` Compute limb darkening fit coefficients for all stars.
`limb_star`(*spectrum*<_char>,*filter*<_char>)

Function. Return linear limb darkening coefficient integrated over specified bandpass. Example: print, limb_star('G0I','V').

`limb_filter`(*teff*<_real>,*log(g)*<_real>,*filter*<_char>)

Function. Return linear limb darkening coefficient integrated over specified bandpass. Example: print, limbfilter(5500, 0.5, 'R').

`limbband`(*teff*<_real>,*log(g)*<_real>,*center*<_real>, *width*<_real>)

Function. Return linear limb darkening coefficient integrated over band pass with specified center and full width (in nm). Example: print, limbband(5000, 0.5, 700, 25).

`limbfactor`(*coeff*<_real>)

Function. Return factor to multiply the uniform disk diameter with in order to get a limb darkened diameter using the specified linear limb darkening coefficient. Based on the squared visibility matching at 0.3 and a formula derived by Hanbury Brown (1974, MNRAS 167, 475).

13.8 Derived parameters

13.8.1 Absolute magnitudes

Absolute magnitudes are derived from the spectral type or trigonometric parallax. For dwarfs, it can also be computed from the $(B-V)$ color index.

`a_stars` Derive absolute visual magnitude from apparent magnitude and parallax.

13.8.2 Distances

`d_stars` Compute distance D from parallax, absolute magnitude, or both.

13.8.3 Diameters

Diameters computed with the following routines *exclude* super giants and binaries! If you want to bypass the latter restriction, set the BFLAG field to '.' (it is set to 'B' for binaries). The routine computing the diameter from (B-V) also *excludes* bright giants and stars later than K.

`diameter_ri` Compute diameter from $(R-I)$ color index and V-magnitude.
`diameter_bv` Compute diameter from $(B-V)$ color index and V-magnitude.

13.8.4 Orbital inclinations, semi-major axes, and secondary masses

`i_binaries`	Estimate I from MASS1/2+MF or MASS1/2+M1/2SIN3I.
`a_binaries`	Estimate A from MASS1/2+P+D.
`k_binaries`	Estimate K1+K2.
`m_binaries`	Estimate secondary mass from primary of an SB2

13.9 Cross indices

A comprehensive array of crossindices has been installed in *catalogs/crossindex*. The files have the extension *.cri* and are read by IDL within the crossindex function `cri`. To obtain, for example, the BSC number for FKV0193 (Capella), enter

```
print,cri(193,'fkv-hdn').
```

This function can be nested, if a particular index does not exist and the request needs to be routed around it. The function will return "−1" if the ID number could not be found. If you know the Bayer or Flamsteed names of the star or its proper name, you can also use `cri` to obtain its ID in any catalog. Note that in the following example only the destination catalog is specified in the function call:

```
print,cri('capella','fkv'),
```

or

```
print,cri('alp aur','bsc').
```

To rename the entire STARID field of STARTABLE, use `rename_starids`, for example:

```
rename_starids,'fkv-hdn'.
```

In the last example, the command will cause STARBASE to access the crossindex for all entries with an FKV ID and try to produce HDN IDs. If an HDN ID is already available in the table, this will also cause the STARID field to be renamed. (Note: this behaviour might be changed in an upcoming release.)

Chapter 14

Widget procedures

There are currently only two widget routines installed.

14.1 Spectrum plotter

This widget allows to plot spectra read from a Library of Stellar Spectra, by Jacoby, Hunter, and Christian (1984), and to identify lines of various elements and molecules.

`plotspectrum` Invoke spectrum plotter widget.

14.2 Startable editor

Starts a GUI to edit the contents of a specific entry of the startable.

`update_startable`,*starid* Invoke update widget.

Part IV

STARWHEEL

Chapter 15

Introduction

STARWHEEL is a container name for utilities related to observing list preparations including simulations. The actual creation of the NPOI observing list is performed by the stand-alone Python based `obsprep` software.

15.1 Simulations

A set of procedures described in Section 15.2 of this manual is used to create a star table. Note that the `plotuptime` procedure which produces a plot with the rise and set times of the stars does not currently take the delay limits into account. In order to do this, run `fakedata` (with *dutycycle=0.1* and simulate a data set using the star table and selected configuration. Then use `plotuptime,1` to plot rise and set times derived from the data.

`fakedata`,*stations*<_char>,*starids*<_char> [,dutycycle=dutycycle]

> Simulate scan data for a full observation (scan every hour while the star is up) with NPOI using the stations given in the input character array.
> Example: fakedata,['N00','N05','N10','W06','W08','W10'],'FKV0621' If stars are not specified, the startable will be used instead. Date and SystemID will default to the current date and NPOI, unless set manually. The dutycycle specifies the interval (in hours) between successive observations of the same star.

`mockdata`,[,pearl=pearl][,poisson=poisson][triple=triple][,init=init]

> Replace observed visibilities and triple data with model values, in combination with `fakedata` or with real data sets. Poisson noise can be added, based on the photon rates of the data set.

To show the *uv*-tracks for the simulated data on top of an image of the visibility amplitude function, first plot the *uv*-coverage and read a model file with *readmodel*, and then use *uvimage,/uv* to underlay the grey scale amplitude image.

15.2 Observing list preparation

The $\mathcal{OYSTER}$ scheduling routines assist in creating an observing list (stored in the startable) from scratch or from another list stored on disk, and computing the visibility of the stars in a

specific night and array configuration. The date of observation is taken from the variable *date*, and the array name from the variable *systemid*. If these are not defined, defaults are used for date (the next night) and for array (NPOI).

As to the system configuration, a list of stations can be passed to `plotuptime` which will then create a system configuration for them.

`get_startable`,*stars*<_char>

Create startable for stars specified.

`read_obslist`,*file*<_char>

Read observation list from file and create startable. This procedure only reads the first, i.e. the star column of the file. It ignores lines containing an exclamation mark.

`selectup`[,*from_data*<_int>][,*mintime=min*]

Remove stars from startable which are not available. Either compute predicted availability (default), or derive from currently loaded data ($from\_data = 1$). Use keyword *mintime* (default 3 hours) to specify minimum availability after sunset and before sunrise.

`selectnew`[,*dates=nd*][,*scans=ns*]

Remove stars from startable which were observed before at least nd times with ns scans or more each. The parameters default to $nd = 2$ and $ns = 4$.

`selectdone`[,*dates=nd*][,*scans=ns*]

Remove stars from startable which have not been observed before at least nd times with ns scans or more each. The parameters default to $nd = 2$ and $ns = 4$.

`addstar`,*stars*<_char>

Add one or more stars to the startable.
Examples: addstar,'FKV0193', addstar,['FKV0621','FKV0641']. Also sorts the stars by ID.

`removestar`,*stars*<_char>

Remove one or more stars from the startable.

`addcal`,*star*<_char>[,*vlimit*<_real>]

Lookup a calibrator for given star. The nearest calibrator (based on classification in *diameter.bsc*) with a visual magnitude brighter than *vlimit* (default 4.5) is added to the startable.

`compileobslist`

Rename stars in startable to their FKV equivalents, if possible, read the catalogs, get auxilliary information, and re-order stars.

`plotuptime`[,*from_data*<_int>][,*stations=stations*] [,*stars=stars*]

Compute visibility for stars in startable and plot. If *from_data=1*, plot actual visibility for currently loaded scan data. A list of stars can be passed to this procedure, which is a shurtcut over allocation with `get_startable`. If *stations* are not defined, a four-station astrometric array is assumed.

`writeobslist`[,*from_data*<_int>][,*time=time*]

Compute visibility and write startable with auxilliary information to file. The name of the file is derived from the date, the extension is *.obs*. The RA and declination fields in the observing list now refer to a suitable background, i.e. blank sky location.

`compile_astrometry`

Compile astrometry observing list.

Part V

CHAMELEON

Chapter 16

Introduction

16.1 About CHAMELEON

The CHAMELEON collection of scripts as part of $\mathcal{OYSTER}$ is used to read, edit, average, and calibrate output ("point data") produced from raw fringe data by a standalone C program, CONSTRICTOR. The scripts are run interactively in the IDL environment, with the data stored in IDL variables and therefore available at the command level. Beginning with version 8, a pipeline has been implemented which produces averaged and calibrated data.

Data reduction and analysis are two steps separated by the `average` procedure. In the classic NPOI 3-way reduction, the squared visibility, delay, background and photonrate data are edited, then averaged into scans. For astrometry, the emphasis is on the dispersion correction of the delay point data before the averaging step. Calibration of the scan-averaged visibilities is based on unresolved or nearly unresolved calibrator stars, whereas the astrometric calibration is based on the application of constant term and pivot motion metrology data delivered by INCHWORM. A new feature of the 6-way data reduction is the photometric and bias calibrations derived from the averaged data, but used in a second averaging step in order for them to be applied.

Following the **Introduction**, a **Quick Guide** will lead you through a sample session. **Basic widget procedures** is a more comprehensive description of how to reduce data using the GUI (Graphical User Interface). **Command line procedures** lists some of the useful routines callable at the IDL prompt.

16.2 Data and auxilliary files

Interferometric **data** files handled by CHAMELEON use the HDS (Hierarchical Data System) format, and have an extension *.con* for point data files output by CONSTRICTOR and *.cha* for scan-averaged data output by CHAMELEON. Think of the HDS format as a UNIX directory/file tree contained within a single file. Thus, individual objects can be accessed and modified directly, without having to read through the file until a specific section. The *.con* files are found in */data/npoi/con* on gemini.

Output files produced by CHAMELEON are always stored in the current working directory. Aside from the data file, three types of auxilliary files can be written, files with extension *.flg* (XDR format) contain the flag tables for interferometric data, *.cal* (XDR format) contain the calibration table, and *.stn* (ASCII) contain fitted station coordinates. These files are only written

by request, never automatically.

As to auxilliary **input** files, all these reside in the following directories of the $\mathcal{OYSTER}$ home directory. The files used by STARBASE are listed in section 11.4. i

- *usno/ mark3.dat*, the UT1-UTC and polarmotion data from the EO department of the US Naval Observatory.
- *npoi/ stations.config*, *fdl.config*, *3way.config*, and *6way.config*, array geometry configuration files, and *biasbase.xdr*, *obsbase.xdr*, *linear.xdr*, and *skycoord.xdr*, the bias correction data base, observation data base, flux ratio file due to detector non-linearity, and star coordinate list, respectively.
- *pti/ stations.pti*.
- *keck/ stations.keck*.
- *vlti/ stations.vlti*.
- *mark3/ stations.mark3*.

16.3 Pipeline

16.3.1 Introduction and command syntax

The pipeline automatically performs all of the steps described in the interactive session below, with the exception of the astrometry. The full command is as follows:

```
npoipipe, confiles, calibrators=calstars,addcal=calstar,model=modelfile
```

`confiles` is a string file specifier given to IDL's `file_search` function to return a list of files for processing. If `confiles` is omitted, all CONSTRICTOR files in the current directory are processed.

A list of calibrators to be used can be specified as a string array for the `calibrators` keyword, otherwise `npoipipe` will determine which stars can be used as calibrators based on information found in the standard NPOI calibrator list (*diameter.bsc*). Calibrators can be added to the standard list using the `addcal` keyword. It is thus important to keep in mind that `npoipipe` will normally use all calibrators observed to calibrate the science targets. If not enough calibrators were observed, a warning message will be displayed, and processing will continue with the next data file.

Finally, the `model` keyword can be used to pass the name of a model file which will be used to flag data on non-tracking baselines. Non-tracking (reference) baselines are difficult to detect, and the method used by `npoipipe` is based on the delay jitter, which is compared to the median value derived from all calibrator observations (in the same data file). If, despite this initial step, the (calibrated) visibility measured on a baseline is less then half of what is predicted by a model, the baseline is considered to have been tracking noise, and all values are flagged.

16.3.2 Output files

All output files will be written into a directory called *npoipipe* so that no files in the current directory will be overwritten. However, existing files in *npoipipe* may be overwritten.

For each processed file, in addition to the CHAMELEON file, a log file (extension *.txt*), a plot file with results from the bias correction (*.ps*), and a flag table (*.flg*) will be written.

16.3.3 Flag tables

If, upon inspection of the calibrated data, some data points need to be flagged, this can be done manually, and the flag table saved in the current directory will be read by *npoipipe* if run again. It is thus important to keep only those flag table files in the current directory that shall be used.

16.3.4 Bias correction

As usual, bias correction is based in incoherent scans and is performed separately for each interferometer configuration. First, bias correction fit coefficients are computed combining all observations (for each configuration) to make sure that a basic correction is performed. Second, bias corrections are updated for each star with the measurement obtained for this star only.

16.3.5 Phase unwrapping

Two methods are used for phase unwrapping, one able to detect the bimodal distribution of wrapped phases, the other one using the *rewraptriple* procedure to remove phase drifts as a function of wavelength cause by excess dispersion in the interferometer.

16.3.6 Logging and quality control

Upon completion of the `npoipipe` command, the log files will be examined by a `npoipipeqc` which extracts the most pertinent information of the log files. Attention should be paid especially to the calibration errors.

Chapter 17

Quick Guide

Here we describe briefly a sample session with CHAMELEON. We include tips on how to get organized during the data reduction, as this is a time consuming process. We also assume that you are reducing output from CONSTRICTOR. (When re-calibrating data however, you will usually begin by reading the output of CHAMELEON.) After starting $\mathcal{OYSTER}$, type oyster at the IDL prompt. In general, we work our way from the top (FILE) to the bottom (CALIBRATE) in the displayed $\mathcal{OYSTER}$ main widget. Also note that button labels are all caps if they bring up another widget. All other buttons either open another button list (if an arrow is displayed), or immediately execute the associated procedure.

In order to restart CHAMELEON and to continue from where you left it or to get close to this point quickly, it can be important that you have saved flag and calibration tables, and (if applicable) the output file in the previous session. These files are *not* written automatically, for reasons of greater flexibility. Please also note that tables loaded are *not* automatically applied. Look at the tables as logs, containing information on flags and calibration entries. If you want to load point data and apply flags from a previous session, or want to undo specific actions on the scan data, these tables are useful. In order for them to be up-to-date, you should load any existing table *before* you do additional editing or calibration. If you want to start over with, e.g. the calibration, then you should also start a new calibration table by clearing any loaded table with UNDO|RESET in the calibration widget, which also initializes the calibrated data (for the selected variable) to be equal to the uncalibrated data.

17.1 Open the data file

FILE|OPEN: select a CONSTRICTOR file with a *.con* extension. Please do not double-click! (Bug/feature in the communication of IDL with X-windows. This can cause your session to crash!)

17.2 Read configuration information, create tables, and load data

ACCESS|LOAD|INTERFEROMETRY will load Date, SystemId, GenConfig, and GeoParms, create the ScanTable and BGTable, load the background scans, and display the point data loading widget (see below).

17.3 Print form sheets

UTILITIES|FORMS: click on SCANFORMS and SPECFORMS to get printouts of form sheets. They should be used to log information about the data file being worked on.

17.4 Reduce the background data

You should reduce this data of type "scan" before anything else. REDUCE|BG DATA: examine and edit the data with PLOT. Plot the background rate (four channels at a time) versus time. (Select channels 1−4 for the first panel, don't forget to hit <return> after entering your selection, and then the Next options in the data selection widget to plot all channels in groups of four.) Do this for all output beams (i.e. spectrometers). Remove scans taken on a star (instead of on blank sky) by identifying the scan number (or time) and then editing the data in a plot with ScanNo (or Time) along the Y-axis. Look out for the laser light in channel 11 (the default is to delete this channel). Remember that editing is only allowed in single plot windows (i.e. a window containing only a single plot). Save the BGFlagTable.

Use PROCESS BG if you want to replace edited scans with an average value from other valid scans in the same channel. Use EXPAND BG to create new background scans for non-existing ones by correlating the BGTable with the ScanTable. (Check the background form or use UTILITIES|LIST|BGSCANS to see missing background scans.) The reason for reducing the background data first is that `average` will not average the point data of a scan for which no background data exists.

17.5 Reduce point data

The point data loading widget already present on your desktop gives you the choice of loading the point data for all stars or a specific selection of them. Load all data if you have sufficient memory (this may take a few moments). If not all data can be loaded, select a group of stars by dragging the cursor over the star list in the widget. Load the data for the selected stars by clicking the LOAD button.

Use REDUCE|POINTDATA|IMAGING|PLOT to plot and edit the point data data in much the same way as you did with the background data. Start by plotting the DelayJitter versus Index, and use the AUTO editor (PL/E|AUTO) on it. Do the same for NATJitter (if data present). (The first one has an OutputBeam and Baseline index, the second one an InputBeam index.) The jitter data provide a good first check whether the visibility data is going to be useful or not. Make sure you look at all spectrometers and baselines. If more than one baseline is selected, they are plotted along the X-axis; channels are stacked vertically. Use plots vs channel if you have to flag several entire channels. Use the PLOT|AUTO option to automatically edit outliers in PhotonRate and VisSq (select all channels for speed). It is recommended to not flag the data in the laser channels, since VisSq will not be averaged anyway if the background data for these channels has been removed. This prevents the flag table from getting too large. Save the PointFlagTable.

Examine and edit the FDLDelay data on a star-by-star basis, using the Fit utility to detect outliers. You should plot FDLDelay vs Time for this. However, there are usually no outliers if

they have been removed from the DelayJitter data, so this step may be skipped and returned to if upon examination of the averaged delays outliers are found.

If reducing 6-way data and photometric and bias corrections are desired, set a default background before averaging (i.e. set all background scans to zero using the `defaultbg` procedure, or the REDUCE|BG DATA|DEFAULT button). After averaging all data, perform the calibrations by using CALIBRATE|CHANNELS, and then re-average the data (but do not forget to reload, and expand, the back ground data, and apply any flag tables).

17.6 Average

REDUCE|AVERAGE averages all the data currently contained in the point data arrays. The corresponding scans in the scans array (it will be automatically created when non-existent) will be updated. `average` will print out messages as to invalid (flagged) data encountered, and non-existing background scans. If you are reducing the data on a star-by-star basis, go back to load the next star and process it. If, for any reason, you want to postpone the following calibration of the scan data, you can write these data to a *.cha* file now (see below).

17.7 Astrometry

CALIBRATE|ASTROMETRY: this is the first calibration step (of the *uv*-coordinates, that is) you should take. Here we examine the averaged delay data with PLOT (select FDL_O-C), before attempting to fit station coordinates with SOLVE. Note that FDL_O-C for the reference station is identical to zero by definition. (To change the reference station use the AstrometrySolution widget. The reference station is station 1 by default in the *.con* file.) Do not solve for star positions. (When the solution widget is displayed, the default settings do not need to be changed for this step.) Check the O-C values again and save the StationTable if you are happy with the fit. If for any reason there are outliers, try to locate and flag them in the corresponding point data, then average again.

17.8 Calibration of visibility amplitudes and closure phases

CALIBRATE|VISIBILITY: Select PLOT, and flag first the channels $21 - 32$ since their SNR is usually very low (if you observed with the 3-way system; the 6-way system has only 16 channels). To do this, plot VisSq vs channel, selecting all channels and the All-in-1 option. Then, for calibration, look at the data from the calibrator stars (select them using the information in the StarForm), then select CALIBRATE to display the calibration widget. Make sure the star selection in the plot widget is set to the stars to which you want the calibration be applied before solving for a set of calibration coefficients. The calibration will be applied to the data automatically. Save the calibration table (entries) if you expect to restart $\mathcal{OYSTER}$ and want to redo some of the calibrations. For more information on calibration, see section 18.5.

17.9 Write data

ACCESS|WRITE|HDS will, after closing any data file currently in use, open the output *.cha* file in order to write the ScanData, uncalibrated and calibrated, along with the configuration

information. The file name is derived from the date, with an extension *.cha*. (If you restarted the session by reading the averaged data, the input file is closed first since it is identical with the output file and all files opened with the *File* menu are read-only.) FITS will ask for a star to be written in FITS format. You have to run CALIBRATE|VISIBILITY|SETCOMPLEX before that in order to set the baseline phases.

Chapter 18

Basic widget procedures

This section describes the complete line of data processing within the CHAMELEON widget environment. To invoke the main widget, type the following command at the IDL prompt:

$\mathcal{OYSTER}$ > oyster

18.1 Open a data file

Data files for CHAMELEON are in the HDS format. The File|Open button opens HDS files for read-only access. (Note that when writing files using widget commands, any file currently open will be closed and then the output file is opened in UPDATE mode. It will be closed after writing has finished.) The file selection menu will be displayed with an initial filter for files having extension *.c??*. CONSTRICTOR data files have extension *.con*, CHAMELEON output data files extension *.cha*. Only one data file can be open at a time; to close a data file, use the File|Close button. The Clrst button is used to clear the HDS status to zero if an HDS error occurred. This button will also close the file.

File |

OPEN	Open HDS file for read-only access
Close	Close HDS file
ClrSt	Close file and clear HDS status

18.2 Access a data file

18.2.1 Browse through an HDS file

Since an HDS-file is a binary file, no editor can be used to look at it or extract data. Special HDS-routines have to be used instead to access an HDS-file. CHAMELEON provides a set of procedures which are 'wrappers' for C-functions (contained in *hds.c*) which themselves call the appropriate HDS-functions. These procedures have names identical to the HDS routines, and are simple to use at the command level. Please refer to appendix A for a list of these procedures.

Access|Tree prints the HDS-file directory to the screen recursively starting at the current level. Click on the terminal window and hit a key to get the next page until the message `Listing`

`complete.` is displayed. Note that for array objects, the procedure will descend only through the first array element!

ACCESS|BROWSE allows you to navigate through an HDS-file and obtain information about the components in an object. At any level, you can jump up several levels just by closing a parent widget or clicking on an object displayed in a parent widget. ACCESS|BROWSE always starts at the current level. You have to close the top-level browse widget before you can click on ACCESS again.

ACCESS |

TREE	Print a 'directory' of an HDS file to screen
BROWSE	Expand different levels in an HDS file

18.2.2 Load configuration, tables, and logs

Before any data, the system configuration (consisting of the general configuration and the earth and geometric parameters) and the background and scan tables must be loaded. You do not have to load all these by individually selecting each one, just use the ACCESS|LOAD|INTERFEROMETRY button, which will load all items listed above it. (This button is also used to read *.cha* files, as it automatically detects the format to read the relevant data.) Please note, that in addition to this the INTERFEROMETRY button will load the startable for the observed stars and will compute (for scan data) the derived geometric quantities (*uv*-coverage, hour and zenith angles, etc.).

ACCESS | LOAD | *cont'd*

INTERFEROMETRY	All objects in the file, get auxilliary data
SYSCONFIG |	
GENCONFIG	General configuration
GEOPARMS	Longitude, latitude, etc.
TABLES |	
SCANTABLE	Table of scan information
BG TABLE	Table of background scan information
LOGS |	
OBSLOG	Observers log
CONLOG	CONSTRICTOR parameters
DATA |	
BG SCANS	Background scans
SCANS	Scan data
POINTS	Point data

18.2.3 Load point data

Loading point data when using the widget is based on stars or groups of stars. Note that loading all data at once might lead to excessive memory requirements. When loading the data for stars, you will be presented a widget listing all observed stars. Click on the one whose data you need. Under IDL, press the <*control*> key when selecting the stars with the left mouse button. This way, the previous selections will not be undone. Alternatively, you can drag the mouse over the list and highlight a group of stars. After selecting the stars, click on Load and the loading commences. The data will overwrite any point data currently loaded. The widget will stay active. The way to proceed is to fully reduce the data for one or more stars at a time (editing, dispersion correction, and averaging). The averaged data are stored in a structure which is allocated the first time you average data and which is sized so that it will eventually hold all the scan data of the night.

18.2.4 Load scan data

CHAMELEON can read its own output file. Use ACCESS|LOAD|INTERFEROMETRY if you need to re-start CHAMELEON to work on averaged data. This will read all relevant objects in a *.cha* file, get a startable for the observed stars and compute derived quantities (CALIBRATE|ASTROMETRY|CALC). Configuration items can also be read separately using buttons ACCESS|LOAD|SYSCONFIG.

18.2.5 Load metrology data

This button is used to load metrology data processed by INCHWORM. These files have the *.inch* extension.

ACCESS | LOAD |	*cont'd*
METROLOGY	Display widget for selection of metrology data to load. *.inch* file

18.2.6 Write data

After reducing and calibrating the data, scan data should be written to an output HDS file. The filename is derived from the input filename, except that the extension is now *.cha*. The input *.con* file is closed automatically before opening the ouput file. Note that if the file already exists, objects are replaced with the new ones.

Data can also be written to a multi-source FITS file. It is the *scans.complexvis* data being written; calculate it after the calibration using CALIBRATE|VISIBILITY|SETCOMPLEX.

ACCESS | WRITE |	*cont'd*
HDS	Write all objects to HDS file
FITS	Write multi-source FITS file
SYSCONFIG |	
GEOPARMS	Write earth parameters
GENCONFIG	Write general configuration

TABLES |
- SCANTABLE — Write table of visibility scans
- BG TABLE — Write table of background scans

LOGS |
- OBSLOG — Observers log
- CONLOG — Constrictor log

DATA |
- SCANS — Write scan data

18.3 Reduce visibility data

The goal of this step (and the next one concerning the reduction of the delay data for astrometry) is to prepare the point data for averaging. This involves editing the background and point data, and the dispersion correction of the delay data (if needed for astrometry). It is recommended to first edit the background data (which is always loaded for all stars due to the relatively small size of the corresponding object). Thus, when the point data for a specific star has been edited, the averaging will make used of background data that has been already processed! The averaged data will be stored internally in an array of structures (scans), which is allocated the first time `average` is called. This array will eventually hold all averaged scans. *Do not forget to average the point data before loading new point data, as the old one will be overwritten!*

REDUCE |

BG DATA |
- PLOT — Plotting and editing
- PROCESS BG — Fix flagged data
- EXPAND BG — Fill in non-existing scans
- DEFAULT BG — Set default BG data
- FLAGTABLE | — Background flag table
  - SAVE — Store to disk file *.flg*
  - LOAD — Read from disk file *.flg*
  - APPLY — Fully apply flags from table
  - CLEAR — Initialize flag table
  - UNFLAG — Specify flags to be removed
  - RESET — Unflag all data

POINTDATA|IMAGING |
- PLOT
- A/R TRIPLE — Compute a new triple
- UNWRAP TRIPLE — Unwrap the triple phase
- CLEAR DATA — Deallocate memory for point data
- FLAGTABLE | — Point data flag table
  - SAVE

LOAD
APPLY
CLEAR
UNFLAG
RESET

AVERAGE — Average currently loaded point data

18.4 Reduce delay data

For the reduction of astrometric delays, corrections from atmospheric dispersion and array metrology have to be computed. They are applied during averaging of the point data into scans. Metrology correction can also be applied to the scan data later, in which case one would initialize these data before averaging.

The procedure is as follows: for the metrology, open the appropriate INCHWORM output file, and load a motion group using ACCESS|LOAD|METROLOGY. (A motion group is a solution for the siderostat pivot motion for all or a subset of the laser readings.) As part of the motion group, two types of data are provided, one being the (x, y, z) coordinates of the pivot, the other one being the delay correction. In CHAMELEON, `metrocorr` uses the former to compute delay corrections, whereas `astrocorr` only does the interpolation of the corrections computed by INCHWORM to the observed epochs of the data points. The result is stored in only one variable, *metrodelay*, and can be plotted using the PointData plot widget.

The dispersion correction is computed from the observed variation of the visibility phase with wavelength. Due to the low SNR of the blue channels, REDUCE|DELAYS|PHASEEDIT can be used to flag these. Also, before computing the correction, the closure relations have to be enforced using REDUCE|DELAYS|PHASECLOSE, and the phases have to be unwrapped using REDUCE|DELAYS|PHASEWRAP. After completion of `dispcorr`, the corrections for the white light scans (if present), can be set to zero if these are too noisy and one would rather use the raw fdl delays.

REDUCE|ASTROMETRY |

DELAYS |

PLOT	Plotting and editing
INITALL	Initialize (zero) all stars
INITW	Initialize white light scans only (FKV0000)
PHASEEDIT	Flag channels 11 and $21-32$
PHASECLOSE	Enforce delay closure
PHASEWRAP	Unwrap the phases
DISPCORR	Compute dispersion corrected delays

METROLOGY |

PLOT	Plotting and editing
INIT	Initialize (zero) metrology corrections
METROCORR	Compute delay corrections from pivot point positions

AstroCorr — Obtain delay corrections from INCHWORM file
Flagtable | — Metro data flag table
- Save
- Load
- Apply
- Clear
- UNFLAG
- Reset

18.5 Calibrate data

The Calibrate menu unites all procedures working with scan data in order to calibrate it or obtain calibration information. The tasks range from calibrating the visibility amplitudes and closure phases to plotting of the channel sensitivities and fitting station coordinates.

Calibrate |

- STARS — *uv* based plotting
- SYSTEM — Channel sensitivities
- Visibility |
  - PLOT — Plot ScanData
  - CALIBRATE — Calibrate
  - SetErrors — Apply calibration error
  - SetComplex — Fill ComplexVis
  - Triple |
    - RewrapTriple — Unwrap triplephase
    - DewrapTriple — Unwrap triplephase
    - ManualUnwrap — Unwrap triplephase
  - Flagtable — Scan data flag table
    - Save
    - Load
    - Apply
    - Clear
    - UNFLAG
    - Reset

18.5.1 Scan data

The Calibrate|Visibility|CALIBRATE button is for the calibration of visibilities (squared amplitudes, triple amplitudes, and closure phases). You will be presented a widget in which to select the calibrator stars and system visibility indicators. As for the calibrator stars, there are three options: to select all (an unusual case), select according to a flag in the startable (derived from information contained in file *diameter.bsc*), or to select manually (recommended).

For each selected indicator, a list of base function will be displayed and you will select a minimum number of functions necessary to represent the dependence of the visibility on this indicator. The final fit will be multi-dimensional, with all functions fitted to the calibrator data simultaneously. (Exception: the selection of a smoothing function, see below!) The mathematical procedure used is the Singular-Value-Decomposition (SVD). In case of a design matrix singularity, SVD will edit the small eigenvalues and proceed to calculate a solution for the calibration.

The smoothing functions S_XX (the index gives the width in minutes) are available for hour angle and time calibrations. Selection of these disable any other selection since they cannot be used in the multi-dimensional fit. This is brought to the attention of the user by making all other base function widgets insensitive while a smoothing function has been selected.

Which indicators and functions are to be used in the calibration depends on atmospheric and instrumental circumstances and has not been fully investigated for the NPOI yet. Therefore, even though a wide array of indicators is available, only one or two might typically be used. Ideally, the visibility amplitude should depend only on the delay jitter, but instrumental effects might require an additional time dependent calibration, either depending on time itself, or on hour angle if the delay line position or the siderostat mirror pointing has played a role in the visibility degradation.

Before the actual calculation of a solution it is required to select the data which is to be used for the computation of the calibration coefficients and is to be calibrated. Clicking on Calibrate|Visibility|Plot will display the plot widget. Please note that the star selection in this widget will now indicate the stars for which the data is to be calibrated. *It should also include the selected calibrator stars!* Since several calibrations can be applied sequentially to the same data, appropriate selections are required here if calibrations differ from star to star. Also note that the scan selection is used just as in the case of plotting scan data. The calibration is only applied to valid visibilities. By selecting the Loop option, selected channels and baseline will be calibrated independently rather than combined in a simultaneous fit. This options was implemented to speed up calibration of an entire spectrometer.

Remember to work on all spectrometers and baselines. In the case of NPOI 6-way data, combine only scans of the same sub-array configuration. Unless the photometric calibration is very good, sub-arrays will differ in their system visibility.

Look at the normalized and calibrated visibilities to make sure they were not calibrated before if you want to start over with a new calibration. This is because, as mentioned above, more than one calibration can be applied sequentially, and calibrated data is part of the data saved in the output.

Click on Calibrate in the calibration widget to compute a solution based on the calibrated estimated visibilities (amplitudes or closure phases, depending on your selection) of the calibrator stars. (The calibrated visibilities are initialized as a copy of the uncalibrated visibility.) The results will be entered in the calibrations table (strictly speaking an array of calibration entry structures), and *automatically* applied to the data. The entire calibration table can be stored to disk using Entries|Save, and loaded from disk using Entries|Load. A list of all calibration entries will be printed to screen by Entries|List.

Here we summarize again the calibration widget functions:

Options |

Loop	Loop over channels and baselines

ENTRIES |

SAVE	Save all calibration entries to disk file
LOAD	Load entries from disk file, replacing current entries
APPLY	Calibrate data using all calibration entries
CLEAR	Clear the list of calibration entries
LIST	List the entries

UNDO |

UNDO	Remove entries and corresponding calibration
UNCAL	Completely remove calibration from the selected data only.
RESET	Reset calibration of all data *for the selected* variable (data type)

UNDO|UNDO will remove a specific calibration from the data as well as the corresponding entries in the calibration table. You will be presented with a list of calibrations for the selected variable, from which to choose the calibration to be removed. The data selection is provided by the information stored in the calibration table. (To remove a calibration only from the table, use the ENTRIES|EDIT function.) UNDO|UNCAL will remove all calibrations for the *selected data* and *all* stars/scans. UNDO|RESET resets the calibration table (for amplitudes or phases, depending on your selection) and remove *all calibrations* from *all* data.

18.5.2 Calibrating amplitude errors

The errors on the amplitudes of the visibility and the triple can be recomputed based on their formal uncertainties and the calibration error. The latter is derived from the variation of the calibrator visibilities and the comparison to their formal errors. Both are added in quadrature. This procedure requires the selection of calibrator and program stars.

18.5.3 Unwrapping triple phases

There are three unwrapping algorithms available under CALIBRATE|VISIBILITY|TRIPLE. For the first two, REWRAPTRIPLE and DEWRAPTRIPLE, you have to have calibrator stars selected first, since they unwrap only the data for these stars (for the time being). Rewrapping here shall mean the rotation of the complex visibility by a specified angle so that the entire range of phases fits into the interval $[-180, +180]$ degrees. The the phases are recomputed, and backrotation is applied to the complex quantities, but merely subtracted from the phases. If the phase is wrapping more rapidly, one should plot the (calibrated) triple phase, and fit a low order polynomial to a stretch which does not wrap (use FIT). The fit parameters are passed to the unwrapping algorithm, which removes it from the data and then removes also the resulting steps. The fit is then reapplied to the data. The manual unwrapping utility, MANUALUNWRAP, will allow the user to place a box into a plot of phase vs time or channel, and the data inside will be rotated $+360$ or -360 degrees, as specified. No calibrator stars need be specified for this operation. Note that in this case the calibrated phase has to be plotted, since this is the phase which is unwrapped!

18.5.4 Astrometry

The CALIBRATE|ASTROMETRY menu allows you to solve for station coordinates (and star positions) using the delay data. Use PLOT to plot the O-C data. (Select input beams higher than one, since delays are referenced to the delay in input beam one.)

Use SOLVE to display a selection widget for the astrometric solution. A solution will be computed for the selected stations and stars, using data only of the selected stars. If no star is selected, then no star position will be solved for, but data of all stars will be used for the station coordinate solution. As for the selected data, either uncorrected FDL data can be used, or the group delay and the dispersion corrected delays. (Remember that the latter can also include metrology corrections.) SAVE will write the stationtable to disk.

CALIBRATE |

ASTROMETRY |

PLOT	Plot residual delays.
CALC	Compute zenith angles, etc.
MCORR	Apply metrology and white light corrections to group-, dry-, and wet delays. (It is not applied to the FDL position data!) This is a toggle, i.e. if you click on this button a second time, the corrections will be removed again, and so forth.
SOLVE	Solve for new station coordinates.

18.5.5 Stars

Plots based on data for individual stars are accessed through this button, e.g. *uv*-coverage, visibility amplitude versus projected baseline length, etc.

18.5.6 System

Plots based on data for individual channels are accessed through this button, e.g. the sensitivity of the channels versus the $(B - V)$ color index of the stars, or visiblity amplitude biases. In addition, relative flux contributions from the stations can be calibrated with this widget for 6-way applications.

The calibration works as follows. For the selected type, any plots produced store the corresponding calibration and fit results in a field of the general configuration structure. These numbers are used in the `average` procedure to compensate visibility biases based on incoherent photons and fixed pattern noise. Remember that BIAS and RELFLUX plots have channel, baseline, and scan indices. For the latter is it advised to only use the listed sub-array configurations.

18.6 Utilities

A collection of various tasks supporting the data reduction in CHAMELEON.

UTILITIES |

OBSBASE	Invoke data base widget. Must own home directory.

.CAL− >.CHA	Convert Mark III *.cal* files to *.cha* format.
LIST |	
SUMMARY	Summary
SCANS	Stars and scans
BGSCANS	BG Scans
SCNLOG	Scan log (observer star log)
OBSLOG	Observer log
CONLOG	CONSTRICTOR parameters
STARS	StarTable information on star
STAR	Catalog information on star
FORMS |	
SCANFORMS	Print scan and BG scan lists
SPECFORMS	Print spectrometer forms

18.6.1 Catalog access

In order to obtain information on the observed stars, click on UTILITIES|LIST|STARS. If a scantable is loaded, you will be presented a list of observed stars from which to select. Otherwise, you have to enter a star name.

Chapter 19

Command line procedures

In this chapter we list some of the useful command line procedures. Note that not all tasks (e.g. plotting) are be performed easily at this level, for which therefore widget routines have been implemented. However, since all data is available at the command level, feel free to analyze it using IDL.

In the following, italic names, e.g. *value*, have to be replaced with variable names; if no type is specified in the procedure usage description, they will be allocated automatically and return the data requested. If a type is specified, e.g. *value*<_double>, the variable must be declared and initialized before. Optional arguments are in square brackets, e.g. *status*; if several optional arguments are listed, only one can be selected as such and arguments to the left then are compulsory! Also remember that you have to enclose character strings in quotes, unless you stored them in a variable of course.

19.1 Accessing data files

`hds_open`,*filename*<_char>,[*mode*<_char>][,*status*]

Opens a HDS data file. Only one file can be open at a time.

mode READ, UPDATE

status (returned) 0: no error; −1: error occurred

`hds_close`

Closes the current HDS file.

19.1.1 Loading configuration and logs

Several of these procedures have been modified to create the object for simulation purposes instead of reading it from an HDS file if that file is closed, or if the appropriate input parameters are passed to the procedure.

`get_systemid`[,*system_id*]

Get SystemId from HDS file, either NPOI or Mark_III. If parameter is omitted, SystemId will be stored in common block variable *systemid*. If file is not open, set default system ID (NPOI).

`get_datum`[,*datum*]

Read Date from HDS file. Format: yyyy-mm-dd. If parameter is omitted, date will be stored in common block variable *date*. If file is not open, set default date (the next day).

`get_geoparms`[[,*sysid*][,*datum*]]

Load GeoParms object from HDS file. If file not open, set default site coordinates corresponding to IDL variable systemid. If *sysid* or *datum* are defined, file is closed and obejct is created from inputs. Also reads catalogs/npoi/mark3.dat for earth rotation information.

`get_genconfig`[,*stations*]

Load GenConfig object from HDS file. If file not open or *stations* are defined, close file and create object from inputs.

`get_metroconfig`

Load MetroConfig object from HDS file.

`get_sysconfig`[,*sysid=sysid*],[,*datum=datum*] [,*stations=stations*]

Load GenConfig, MetroConfig, and GeoParms. Input parameters, if defined, cause creation of objects from inputs rather than reading the from file. In this case the file will be closed.

`get_observerlog`

Get ObserverLog from HDS file.

`get_constrictorlog`

Get ConstrictorLog from HDS file.

`get_format`,*format*

Get Format of HDS file, either CHAMELEON, CONSTRICTOR, INCHWORM, or COBRA.

19.1.2 Loading tables

`get_numscan`,*num_scan*

Read NumScan object for number of scans in HDS file.

`get_numbgscan`,*num_bg_scan*

Read NumBGScan object in HDS file for number of background scans.

`get_scantable`

Read data relevant to scansin HDS file in order to fill the scantable. The scantable is a somewhat antiquated concept, since it could be easily absorbed into the scans structure.

`get_bgtable`

Read objects in HDS file in order to fill the background table

`get_stationtable`,[*stationfile*<_char>]

Read *stationfile* with station coordinate information. If not specified, read default files *stations.config*, *fdl.config*, and *3way.config* or *6way.config* depending on GenConfig.BeamCombinerID. Remember to run `calcastrom` if you have scan data loaded in order to update the astrometry.

19.1.3 Loading data

Note that there are some procedures which package several tasks, e.g., `get_pointdata` includes loading of the configuration, scantable, etc., and the loading of point data for all scans.

`get_records`

Read raw data from HDS file. Since there is only one scan per file, no scan number needs to be specified. Note that all records all loaded.

`get_points`,[*scans*<_integer>]

Read point data for list of scans. First scan number is always 1! Note that configuration and table data must have been read prior to this call! The default is to load point data for all scans.

`get_scans`

Read scan data into scans. Note that configuration and table data must have been read prior to this call!

`get_bgscans`

Read background scan data into bgscans. Note that configuration and table data must have been read prior to this call!

`get_motiongroup`,*group*<_integer>

Read motion group data from *.inch* file.

`get_astromcorrgroup`,*group*<_integer>

Read astromcorr group data from *.inch* file.

`get_metrogroup`,*group*<_integer>

Compount procedure for `get_motiongroup` and `get_astromcorrgroup`. Note that the first group is index 1!

`get_star`,*star*<_char>

Read point data for *star* (e.g. FKV0193). Note that configuration and table data must have been read prior to this call!

`get_rawdata`,[*file*<_char>]

Compound procedure. Open file if necessary and read configuration, table, and raw data.

`get_pointdata`,[*file*<_char>]

Compound procedure. Read point data for all scans. Get auxilliary data. Open file if necessary.

`get_scandata`,[*file*<_char>]

Compound procedure. Read scan data and auxilliary data from *.cha* file. Do astrometry. This command is sufficient to read an entire *.cha* file. Open file if necessary.

`get_metrodata`,[*file*<_char>]

Compound procedure. Read metrology data from *.inch* file. Also read general configuration and scantable data. Allocate and fill in startable.

`get_data`,[*file*<_char>]

Compound procedure for all of the above. Checks file format to see which of the above compound routines will be called to read the data.

19.2 PointData manipulation

19.2.1 Astrometry

The following procedures are specific to the reduction of point astrometry data.

`dispinit`,[*stars*<_char>],[*class*=*class*]

Initialize the dispersion corrections. In order for calibrated delays to be processed by `average`, both dispersion and metrology corrections have to be initialized or computed. Initialize (zero) the dispersion corrections for the specified stars. Initialization for all stars is the default. A *class* (ALL, POINT, SCAN) can be specified using keyword *class*; the default is all classes. For scan data, initialization means setting the dispersion corrected delays equal to the averaged FDL data.

`dispflag`,*channels*,[*beams*]

Flag the visibility phase for the specified channels and beams. If none are specified, do channels 11 and 21 to 32 in all beams. If beams are not specified, but channels are, assume this selection is valid for all beams. If channels is a linear array and beams are specified, use the same channel selection for the specified beams. If channels is a two-dimensional array, the first index refers to the beam, and the second index to the channels to be flagged for the beam specified in the second parameter.

`dispclose`

Enforce the closure relations. Since the fringe delays computed for the coherent integration of the complex visibilities are baseline based, they do not necessarily close, i.e. add up to zero. This procedure makes sure they do, and adds the correction to the phases.

`unwrapphase`

Unwrap the visibility phase as a function of wavelength.

`dispcorr`

Compute the dispersion correction. Place the results in common block.

`metroinit`

Initialize (zero) the metrology delay corrections. They can be recomputed using `metrocorr` or `astrocorr`.

`metrocorr`

Compute delay corrections from (x, y, z) solutions provided by INCHWORM and stored in variables *parx*, *pary*, and *parz*. Interpolate to the epochs of the point data. Results are stored in variable *metropos*.

`astrocorr`

Compute delay corrections by interpolation of the INCHWORM solutions already converted to delays and stored in variable *metropath* to the observed point data times. The results are stored in *metropos*. Note that `metrocorr` and `astrocorr` store their results in the same variable, since they should give very similar if not identical results.

19.2.2 Other

`unwraptriple`

Unwrap triple phase as a function of time.

`calcgeo`

Compute geometric delays for point data.

`average`

Average point data currently loaded, and place results in scan structure scans.

19.3 Writing HDS objects

`put_genconfig`

Write GenConfig to HDS file. Erase existing object.

`put_geoparms`

Write GeoParms to HDS file. Erase existing object.

`put_sysconfig`

Compound procedure.

`put_observerlog`
`put_constrictorlog`
`put_date`,*date*
`put_userid`,*user_id*
`put_systemid`,*system_id*
`put_format`,*format*
`put_scandata`,*filename*

Write entire scan data set to file. Erase existing objects.

`put_stationtable`
`put_mark3data`

19.4 Analysis procedures

`hb`(*uvd*<_real>, *v2*<_real>)

Function. Return diameter of star for given *uv*-distance and squared visibility. Uses calibration implemented in polynomial expressions.

`caldiameter`(*starid*<_char>,*calid*<_char>)

Function. Return the diameter of a star using the calibrator and the uncalibrated estimated squared visibilities.

`npoirates`,*starid*,*predictedrate*

Propagate the stellar flux through the atmosphere and NPOI, and return the predicted count rate for every NPOI channel. Configuration must be loaded.

19.5 List procedures

`list_names`[,*names*<_char>]

List HDS objects at current level. Store in parameter, if given.

`list_tree`

List HDS tree beginning at current level.

`list_scans`[,*index*,/coh]

List observed scans, stars, times, stations, and codes. If *index* is specified as an integer vactor, list only these scan IDs. If *index* is a StarID string, only scans for this star will be listed. The keyword */coh* restricts the list to coherent scans only.

`list_bgscans`

List stars and back ground scans.

`list_stars`[,*stars*<_char>]

List stars present in currently loaded data. Store in parameter, if given. Also list scan numbers and embedded scan IDs. The latter, listed in parenthesis after the scan number, are the scan numbers which were used by the embedded system during the observations.

`list_stations`[,*stations*<_char>]

List stations loaded. Store in parameter, if given.

`list_summary_chameleon`

List summary for observations in CHAMELEON.

`list_summary_amoeba`

List summary for loaded data in AMOEBA.

`list_logs`

List all logs, i.e. ObserverLog, ConstrictorLog, ChameleonLog.

`list_flagreasons`,*class*<_char>

List flag reasons for class specified.

19.6 Date conversion functions

All inputs for these functions and procedures can be arrays!

`julian`(*year*<_integer>,*month*<_integer>,*day*<_real>)

Function; returns Julian day including fraction. Day can be fractional. Example: print,julian(1998,12,16.2)

`jd2date`,*jdf*<_real>,*year*,*month*,*day*

Procedure; returns year, month, and day (including fraction) for input (fractional) Julian date.

`by2jd`(*year*<_real>)

Function; returns Julian date corrresponding to input Besselian year.

`jy2jd`(*year*<_real>)

Function; returns Julian date for input Julian year.

`jd2jy`(*jd*<_real>)

Function; returns Julian year for input Julian date.

`jd2by`(*jd*<_real>)

Function; returns Besselian year for input Julian date.

19.7 Astrometry procedures

All arguments for these procedures and functions can be arrays. All require existence of GeoParms variable. A default can be created by using `get_geoparms`

`utc2ut1`(*utc*<_real>)

Function. Returns UT1 in seconds, with UTC input in seconds.

`ut12gst`(*utc*<_real>[,*ut1*<_real>])

Function. Returns GST in hours, inputs are in seconds.

`hourangle`(*gst*<_real>,*ra*<_real>)

Function. Return hourangle in hours, with inputs also in hours.

`zenithangle`(*ha*<_real>,*dec*<_real>)

Function. Return zenithangle in degrees with inputs *ha* in hours and *dec* in degrees.

`mirrorangle`(*ha*<_real>,*dec*<_real>)

Function. Return mirrorangle in degrees with inputs *ha* in hours and *dec* in degrees.

`horizon2equatorial`(*coord*<_real>)

Function. Return equatorial coordinates for input horizontal coordinates X=east, Y=north, and Z=up.

`equatorial2horizon`(*coord*<_real>)

Function. Return horizon coordinates for input equatorial coordinates X=projection of local meridian on equator, Y=east, and Z=north.

`angle2horizon`(*coord*<_real>)

Function. Return horizon coordinates for input azimuth and elevation angle in degrees. *No GeoParms required.*

`equatorial2angle`(*coord*<_real>)

Function. Return hour angle in hours and declination in degrees for input equatorial coordinates. *No GeoParms required.*

`topostar`,*times*<_real>,*stars*<_char>,*startable*,*ra*,*dec*

Compute apparent declinations and right ascensions for a list of times (in seconds UTC) and corresponding stars. (The arrays *time* and *stars* have to have the same number of elements therefore.) The startable is a standard STARBASE startable. The geoparms structure has to be loaded.

`calcastrom`

Procedure. Compute astrometric variables in scans structure.

`calcvis`

Procedure. Compute estimated visibilities, i.e. normalized visibilities.

`referencestation`,*station*<_char>[,*class*<_char>]

Change reference station for data of specified class (default all classes) to *station*.

`precess`,*theta*,*jy*,*ra*,*dec*,*direction*

Precess J2000 position angles (theta) to current Julian Year epoch (jy) for direction=1, from current to J2000 for direction=-1. (see: Heintz, "Double stars", p. 33). Input theta is in radians, dec in degrees, ra in hours; output is in radians.

19.8 Data base routines

`obsbase`,*files*<_char>

Create (if non-existent) or update observation data base *catalogs/npoi/obsbase* as a database of star tables from *.con* or *.cha* files. File specification can be either through a string array or or a single wildcard, e.g. *files* = '*.cha'. You have to have write permission to this data base file.

`obsstars`[,*stars*]

Print to screen or return list of stars contained in data base.

`obsnights`[,*nights*]

Print to screen or return list of nights contained in data base.

`obsdates`,*star*<_char>[,*dates*, *scans*]

Print to screen or return list of dates and number of scans when star was observed.

Chapter 20

Discussion of reduction issues

20.1 Sub-arrays

With the advent of the 6-way NPOI configuration in which the observer can switch in and out stations, the number of participating stations can change throughout the night. This is equivalent to VLBI, where different sub-arrays are observing. In $\mathcal{OYSTER}$, this is (so far) the only allowed exemption from the required invariability of the system configuration during observations. It is handled using a field in the scan table (*scantable.station*), which, for every scan, indicates which stations were used. When reading point data, this so called scan configuration is used to flag the data involving unused stations, as CONSTRICTOR has processed all data, good or bad.

Some calibration procedures must consider which configuration was used, and therefore require input as to which scans are to be combined in the analysis. In these cases, the scan selection directive has the list of different scan configurations attached to the standard directives. This acts as a simple scan selection, but based on a criterion (here: the scan configuration). There is a command line function, `scanconfig()`, which can be used for quick information regarding scan configurations, as follows.

`scanconfig`()

Return list of different scan configurations used.

`scanconfig`(*scan*)

Return scan configuration used for the specified scan ID.

`scanconfig`(*config*<_char>)

Return scan IDs which correspond to the specified scan configuration.

20.2 Bias correction of visibilities

Since Wittkowski et al. (2001), the importance of improved bias corrections of the amplitudes has been recognized. A good fraction of the non-Poisson bias was removed from the 3-station data through the use of Dave's "z^2" correction in CONSTRICTOR. For the new 6-station data,

CONSTRICTOR allows one to specify one, two, or three fringe modulation values (k-values) at which the amplitude is measured and then used in a polynomial fit of order zero, one, or two, respectively, to compute a bias at other modulations through interpolation. In the case of zero-order bias correction in CONSTRICTOR, a remaining bias can be studied by plotting the visibilities of incoherent scans (i.e. fringe-less data) versus the photon rate. Here is how.

After all editing of the point data has been done, one should set the default back ground rate (which is zero) using `defaultbg` or REDUCE|BG DATA|DEFAULT BGi, and zero bias coefficients (genconfig.bias=0, only needed when starting over) *before* averaging the data. Then use (use CALIBRATE|SYSTEM) to bring up the system calibration widget. Both squared visibilities (V2Bias) and triple amplitudes (TABias, if two or more baselines are on the same detector) need to be plotted, but only for spectrometers without interpolated bias corrections from CONSTRICTOR. Plot all fits for all channels for every configuration. The fit coefficients are stored in the *genconfig.v2bias* and *genconfig.tabias* variables. (If scan selection directive "All" is selected, the fit coefficients will not be stored!) Then re-load the back ground data (`get_bgdata`) and re-average. The bias coefficients will be written into the output file with the averaged data, when done after the calibration.

Note that the triple amplitude bias correction currently is not handled in a strict fashion. This is because it is the complex triple product which is biased. CONSTRICTOR applies a correction to the complex triple product based on Poisson noise if all three baselines are from the same detector. The remaining biases studied as mentioned above in the amplitude are due to non-Poisson detector statistics. If the amplitudes require additional bias compensation, so would of course the closure phases. But this is not currently done for lack of an analytical expression for the bias.

Another issue with the bias correction using incoherent scans is that they are currently combined in a single fit, whereas ideally each coherent scan would be corrected with the incoherent scan immeadiately following it. Implementation has been delayed however because of the new possibility to do bias correction with unused fringe frequencies.

20.3 Photometry

With this we do not mean to talk about using NPOI as an interferometer. Instead, photometric calibration is an important part of checking the system and calibrating visibility amplitudes of the 6-station array.

20.4 Calibration of visibilities

In $\mathcal{OYSTER}$, the visibility amplitudes and phases of the program stars are calibrated using calibrator stars, for which those variables are modeled and the resulting model coefficients transferred to the program star data.

In practical terms, the software keeps four "copies" of the amplitudes (squared and triple), which correspond to the uncalibrated amplitudes, the uncalibrated but normalized amplitudes, the calibrated amplitudes, and the normalized and calibrated amplitudes. Normalization is based on a model of the star, currently a single disk with a diameter taken from a file compiled with a STARBASE procedure. When calibrating the data, the user should begin by looking at the calibrated and normalized amplitudes first to see whether the data has already been

calibrated (for example when a *.cha* file was read). In addition, this data type is the only one which should approach unity for the calibrator stars after calibration. (In the case of visibility phases, there are no normalized quantities.)

When writing calibrated data into a *.cha* output file, both the uncalibrated and calibrated visibilities are written, but not the normalized data. This is because normalization is done every time the data is read from the output file. The calibration table, which can be used to undo specific calibration and which is updated during the calibration process is only needed for this purpose; any calibration can be reset without it and thus start over for a new calibration.

20.5 Astrometry data reduction

20.5.1 Introduction

Astrometry with NPOI is based on the measurement of fringe delays, which are related to the baseline orientation and star position. There are three different corrections to the uncalibrated FDL delays, which are due to atmospheric refractive index fluctuations, path length variations in the feed system, and motions of the siderostat pivots. We call the application of these corrections to the raw FDL delays calibration of the delays. $\mathcal{OYSTER}$, similar to the visibility calibration, has storage for three calibrated delays, i.e. group, dry, and wet delays. These are the result of the dispersion correction procedure (`dispcorr`), which computes cumulative corrections based on the white light fringe position (group delay), the dry and wet air dispersion constants, respectively. All subsequent metrology corrections are *only* applied to these three calibrated delays. Note that $\mathcal{OYSTER}$ does not have separate variables for delays with and without metrology corrections, for example. The user has to keep track of the actual calibration status of the delays.

20.5.2 CONSTRICTOR parameters for astrometric reductions

Atmospheric dispersion corrections are based on the variation of the visibility phase with wavelength. In order to increase the SNR for these data, we coherently integrate the complex visibility for typically 200 ms. The coherent integration is done using the COBRA procedure `cohstrictor`. In order to provide the necessary input file for this step, the following CONSTRICTOR parameters should be used.

- OutFile = YYYY-MM-DD.coh. The *.coh* extension is standard for coherent integration.
- Raw = On. Write the raw data of every scan into a separate HDS file. These files are needed by `cohstrictor` to compute the complex visibilities and fringe delays.
- NumAv = 100. Integrate 100 2ms samples. This seems to be a good trade-off between SNR and possible systematic error at longer integration times.
- Metrology = On. Pass on metrology data

20.5.3 Coherent integration

As explained in the chapter on COBRA, CONSTRICTOR no longer does coherent integrations due to the complexity of the steps involved. Instead, procedure `cohstrictor` is used, with a

single parameter telling it the file name of the CONSTRICTOR output. (Do not confuse this procedure with another COBRA procedure, `constrictor`, which is a general purpose procedure to replace or add certain variables in *.con* files.)

20.5.4 Dispersion correction

OYSTER procedure `dispcorr` is used to compute group delays and dispersion corrections for the data read from the *.coh* file.

The dispersion corrections are computed for every data point, as well as the corrections due to the motions of the siderostat pivots. The latter are solutions for the X, Y, and Z coordinates done by INCHWORM. They are transformed into delays and interpolated to the time stamps of the data points by `metrocorr`. Both are applied when the point data is averaged. (The delay corrections due to pivot motions are also available in the data file from INCHWORM, in which case the `astrocorr` procedure is used to just do the interpolation. The results of both procedures are stored in the same variable.)

20.5.5 Metrology corrections

After averaging the point data, the only remaining delay correction is the fringe position of the white light source used to monitor the feed system path length. This is implemented as a toggle, i.e. a procedure which applies the correction when called the first time, then removes it when called again, and so forth. The procedure name is `whitecorr`.

For greater flexibility, the siderostat metrology corrections can also be postponed until after the averaging, in which case they are derived from the averaged XYZ pivot coordinates. This is done by the procedure `pivotcorr`, the scan data equivalent to the point data procedure `metrocorr`. It too is implemented as a toggle. It is important however to use `pivotcorr` *before* `whitecorr`, since `pivotcorr` will have to correct the white light delays too! This is because the white light fringe position not only depends on the feed system path, but also on the pivot motion.

If one decides to apply pivot and constant term corrections after the data has been averaged (so that the changes may be studied more easily), the situation may arise where raw pivot coordinates from the INCHWORM file have to be averaged; this can be done using the `inchav` procedure.

20.5.6 A standard recipe

So here are the steps for a standard astrometry reduction.

`get_data,'2001-03-16.coh'` (Read a CONSTRICTOR file with coherent integrations.)

`hds_close`

REDUCE|BG DATA|DEFAULT BG (To make sure every scan has a back ground rate and set it to zero since we don't care about the visibility calibration.)

REDUCE|POINT DATA|ASTROMETRY|PHASEEDIT (Flag all channels higher than 20 as well as the laser channel 11. Data after about May 2002 only need channel 12 of spectrometer 1 flagged. This is done automatically.)

`flagphase,2,1,chan=indgen(16)+1` Optional flagging of baseline 1 in beam 2, in this example. This is needed if the same baseline occurs twice in the data. In the case of a reference baseline, the one in the first beam would be overwritten by the one in the second beam, and so on, in the dispersion correction procedure.

REDUCE|POINT DATA|ASTROMETRY|PHASEWRAP (Unwrap the phases as a function of channel.)

REDUCE|POINT DATA|ASTROMETRY|DISPCORR (Compute dispersion correction.)

REDUCE|POINT DATA|ASTROMETRY|PLOT (Plot VACDELAY, i.e. the dispersion corrected delay relative to the predicted geometric delay and flag. Use PLOT|AUTO. The reference station values are zero by definition.)

`hds_open,'2001-03-16.inch'` (Open INCHWORM file.)

`get_metroconfig`

`get_motiongroup,1`

`hds_close`

`pivotfit`. An optional step which fits siderostat pivot coordinates. RMS residuals should be on the order of a couple of microns. After this step, plotting ParXYZ corr. will display the residuals (see next step).

REDUCE|INCH DATA|PLOT. Plot azimuth versus elevation, connecting data points with lines. Smooth curves should be the result. This is a test whether proper siderostat models are loaded. Also plot ParX,Y,Z to look for outliers.

REDUCE|INCH DATA|INIT (For now, we plan on computing and applying the metrology correction *after* averaging, so we have to initialize the metrology correction to zero for the calibrated delays not to be flagged during averaging.)

`average`

`inchav`

CALIBRATE|ASTROMETRY|METROCORR (Compute and apply constant term and pivot corrections.)

CALIBRATE|ASTROMETRY|SOLVE (Compute astrometric solutions)

20.6 Imaging simulations

To use $\mathcal{OYSTER}$ for imaging simulations, first create a data set using the `fakedata` procedure. This will create scans, for which one can compute model values using `calcmodel` after reading a model file. `calcmodel` will compute the complex visibilities (*scans.complexvis*) and squared visibilities (*scans.vissqm*). If one wants to write this data into a FITS file, nothing needs to be copied since the procedure `put_fits` uses the complex visibilities to derive the FITS visibility

variables amplitude and phase. If noise is to be added to the visibilities, it should be added to the real and imaginary parts of scans.complexvis.

The data can then be mapped in AIPS with or without using phase self-calibration.

Chapter 21

HDS procedures

We list here all available procedures for the HDS access. Italic names, e.g. *value*, have to be replaced with variable names; they will be allocated automatically and return the data requested. If a type is specified, e.g. *value*<_double>, the variable must be declared and initialized before. Optional arguments are in square brackets, e.g. *status*; if several optional arguments are listed, only one can be selected as such and arguments to the left then are compulsory! Please refer to the HDS manual for explanations on the individual commands.

21.1 hds_

`hds_open`,*name*<_char>,[*mode*<_char>],[*status*
`hds_new`,*filename*<_char>],[*name*<_char>],[*type*<_char>],[*status*

21.2 cmp_

`cmp_shape`,*name*<_char>,*ndim*,*dims*
`cmp_get0i`,*name*<_char>,*value*
`cmp_get1i`,*name*<_char>,*num*<_long>,*values*
`cmp_getnd`,*name*<_char>,*ndim*<_long>,*dims*<_long>,*values*
`cmp_get0d`,*name*<_char>,*value*
`cmp_get1d`,*name*<_char>,*num*<_long>,*values*
`cmp_getnd`,*name*<_char>,*ndim*<_long>,*dims*<_long>,*values*
`cmp_get1r`,*name*<_char>,*num*<_long>,*values*
`cmp_get0c`,*name*<_char>,*data*
`cmp_get1c`,*name*<_char>,*num*<_long>,*data*
`cmp_getnc`,*name*<_char>,*data*<_char>
`cmp_put0c`,*name*<_char>,*string*<_char>
`cmp_put0i`,*name*<_char>,*value*<_long>
`cmp_put0r`,*name*<_char>,*value*<_float>
`cmp_put0d`,*name*<_char>,*value*<_double>
`cmp_put1i`,*name*<_char>,*values*<_long>
`cmp_put1r`,*name*<_char>,*values*<_float>
`cmp_put1d`,*name*<_char>,*values*<_double>

`cmp_putnd`,*name*<_char>,*ndim*<_long>,*dims*<_long>,*values*<_double>
`cmp_putnr`,*name*<_char>,*ndim*<_long>,*dims*<_long>,*values*<_float>
`cmp_putni`,*name*<_char>,*ndim*<_long>,*dims*<_long>,*values*<_long>
`cmp_putnc`,*name*<_char>,*ndim*<_long>,*dims*<_long>,*values*<_char>
`cmp_put1c`,*name*<_char>,*data*<_char>

21.3 dat_

`dat_name`,*name*
`dat_type`,*type*
`dat_ncomp`,*ncomp*
`dat_index`,*index*<_long>
`dat_find`,*name*<_char>
`dat_annul`
`dat_cell`,*ndim*<_long>,*cell*<_long>
`dat_shape`,*ndim*,*dims*
`dat_get1d`,*num*<_long>,*values*
`dat_get1r`,*num*<_long>,*values*
`dat_get1i`,*num*<_long>,*values*
`dat_getnd`,*ndim*<_long>,*dims*<_long>,*values*
`dat_getnr`,*ndim*<_long>,*dims*<_long>,*values*
`dat_getni`,*ndim*<_long>,*dims*<_long>,*values*
`dat_clen`,*clen*
`dat_size`,*size*
`dat_prim`,*reply*
`dat_there`,*name*<_char>,*reply*
`dat_erase`,*name*<_char>
`dat_new0c`,*name*<_char>,*len*<_long>
`dat_new`,*name*<_char>,*type*<_char>,*ndim*<_long>,*dims*<_long>

Chapter 22

Programmer's reference

22.1 Widget routines

In the following, we will list the IDL widget procedures connected to buttons in the main widget. The names of all these procedures start with `ww_`.

22.1.1 File

`ww_file`

- `ww_fileopen`
- `ww_filecancel`
- `ww_filehelp`

22.1.2 Access|Browse

`ww_access`

- `ww_look`
  - `ww_lookdestroyed`
- `ww_find`

22.1.3 Access|Interferometry

`ww_access` *cont'd*

- `get_GenConfig`
  - `get_Date`
  - `get_SystemId`
  - `get_Format`
- `get_GeoParms`
- `get_scantable`
- `get_bgtable`

get_bgscans
get_ObserverLog
get_ConstrictorLog

22.1.4 Access|PointData

ww_access *cont'd*

ww_loadstarsel
ww_loadstar

22.1.5 Access|Write

ww_write

put_Syslog
put_ConstrictorLog
put_GeoParms
put_GenConfig
put_ScanData
put_mark3data

22.1.6 Reduce

ww_reduce

ww_reducept
ww_reducebg
ww_reducedl

22.1.7 Reduce|PointData|Plot

ww_reducept

init_plotsel
ww_plot
    ww_plotdestroyed
    ww_plotoptions
    ww_plothelp
    ww_plotok

ww_plot *cont'd*

ww_setxaxis
ww_setyaxis

ww_setzaxis
ww_datasel
ww_setslice
ww_setstdir
ww_setstarsel
ww_setwsize

ww_datasel

ww_setyinbeam
ww_setyoutbeam
ww_setybldir
ww_setytrdir
ww_setychdir
ww_setyptdir
ww_setysel
ww_yseldestroyed

22.1.8 Calibrate

ww_calibrate

ww_cal
ww_plot

ww_astrom

ww_cal

ww_caldestroyed
ww_calindicator
ww_calfunction
ww_caloptions
ww_calstars
ww_calok
ww_listcalreasons
ww_caledit

22.1.9 Catalog

ww_catalog

ww_starinfo

22.1.10 WAVE

```
ww_wave

    ww_restore
    ww_restorecancel
    ww_restorehelp
    ww_quit
```

22.2 Organization of data

The data inside CHAMELEON is organized in tables (e.g. ScanTable), structures (e.g. GenConfig), arrays of structures (e.g. scans), and plain arrays (e.g. VisSq). Some of the choices made might be different from what one would make today if one were to write CHAMELEON from scratch, but obviously can't be changed that easily anymore.

Common blocks exist for the major data groups, i.e.:

- RawData: unaveraged, typically 2 ms integrations, flat arrays (bin counts, laser positions, quad cell counts)
- MetroData: unaveraged, typically 1/4 to several Hz, flat arrays (temperature data, siderostat laser metrology, etc.; very incomplete since INCHWORM is mainly used to process these data.)
- InchData: 15 s averaged integrations, output from INCHWORM, flat arrays
- PointData: 1 s averaged fringe data, output from CONSTRICTOR, flat arrays
- ScanData: scan averaged data (*scans, bgscans*), output from CHAMELEON, as well as data used in AMOEBA, velocities (spectrosopy) , positions (relative astrometry), magnitudes (photometry), arrays of structures

All flat array data have a pair of indices per group containing first and last elements for every corresponding scan loaded. Depending on size due to sampling time, some arrays have only data from a single scan, others have all night's data. In particular, point data is meant to be loaded one star at a time, overwriting the previous copy. The scan data is always allocated for the whole night.

Tables are identical to arrays of structures in IDL, but were not in PV-WAVE. Since the latter was the first language for CHAMELEON, we still speak of tables. Two tables accompany the interferometric scan data, the *scantable* and the *bgtable* (back ground table). They exist for historical reasons, related to being convenient containers for auxilliary scan related data when reading the point data, but they could have been incorporated into the scan structures. Aside from the flag tables and the station table, the *startable* is the only other tables worth mentioning, as it contains all stellar data and is subject to the manipulations through the STARBASE procedures.

The interferometric configuration is stored in structures (*genconfig, metroconfig, geoparms*), and arrays of structures (*geninfo, geoinfo*). The latter are expansions of single date configurations to multiple day configurations for use in AMOEBA.

In general, any interferometric data from a single night are not buffered. Thus, the two standard procedures to read data into CHAMELEON, `get_scandata` and `get_xdr` empty the buffer and only initialize the *geoinfo* and *geninfo* structures for use in AMOEBA. Similarly, any procedures creating a CHAMELEON data set from other formats do the same, with the exception of `get_oifits` because these files may contain multiple data sets. Finally, `fakedata`, which simulates a data set, also empties the buffer.

Finally, model and image data are contained in two common blocks.

22.3 Array configuration

22.3.1 Optical path lengths

The total optical path through the NPOI can be found by adding various contributions listed in the files *npoi/stations.config*, *npoi/fdl.config*, and *npoi/6way.config*. For the purpose of calculating delay constant term and optical path lengths, we break up the paths into segments of four types, associated with siderostat station, delay line, beam combiner number (ID), and beam combiner input number. It is best to make the break between station- and delay-line associated lengths at the intersection of the N and E arms. However, the west arm never reaches that point. We treat the feed pipes associated with delay lines 4, 5, and 6 as if they run to the N- and E-arm intersection. If they feed the W arm, the path length is reduced by 1.4455m, which is why D for the west arm stations is reduced by that amount.

In the file *stations.config*, D contains the path from the siderostat pivot point to the mirror at the intersection of the N and E arms, but *excluding* the vertical path in the elevator can which belongs to the delay line based path.

In file *fdl.config*, the paths do not include about 86m of path from the mirror at the intersection of the N and E arms through the feed system to the FDL entrance.

In file *6way.config*, about 4m of path through the beam combiner are not included. That is to say that the delays in this file and the previous one are *relative* paths. This is a result from their having been created using the `npoiconfig` procedure from measurements of the white light fringe positions in different configurations.

22.3.2 Basic definitions and sign conventions

This section is partially derived from "Basic definitions for the USNO and NRL Optical Interferometers" by D. Buscher.

A baseline vector B_{12} is oriented *from* telescope 1 *to* telescope 2, i.e. the baseline coordinates are the coordinates of station 2 *minus* the coordinates from station 1. In the aperture plane, the u coordinate increases to the East, and the v coordinate increases to the North. (Remember though that uv coverages are plotted in $\mathcal{OYSTER}$ as seen on the sky, so that the source structure and visibility function can be easily superposed.)

In PEARL, a map on the sky is allocated with the x-coordinate corresponding to RA, which increases to the left (East). The relationship between visibility and map is then

$$V(u,v) = \int B(\xi,\eta) exp(-j2\pi[u\xi + v\eta]) d\xi\, d\eta$$

22.4 Command line procedures

22.5 Plotting

22.5.1 Adding plot variables

Plot variables are identified by a sequence number in $\mathcal{OYSTER}$, and are defined in terms of $\mathcal{OYSTER}$ data variables in `set_plotdata` (and the corresponding function `set_ploterr`. The corresponding plot variable name is defined in the `set_axitems` function, which should have exactly the same ordering. The following additional information needs to be entered for each plot variable.

Each plot variable's ID is entered in `set_plotlabels` for the index label it requires so that it is displayed together with the name in the plots. In `set_plotloops`, the loop count is set to one for those indices which do *not* occur in the plot variable. These functions and procedures are all found in *plotting.pro*. Finally, the plot variable's ID need to be entered in `ww_setxaxis`, `ww_setyaxis`, and `set_zaxis` if it's selection requires and additional index selection by the user.

22.5.2 Adding plot classes

A plot class is basically defined by the list of plot variables displayed in the x,y, and z axis selections of the plot selection widget. The plot selection widget is recycled upon selecting a different plot class. Any plot variable can be put on more than one selection list associated with a plot classes. Variables are selected for a plot class because of a specific purpose of these plots. Therefore, the definition of a plot class requires the following information to be coded. In `ww_plot`, the appropriate `checkdata` call and plot widget title need to be specified. Furthermore, in the same widget procedure, the name of the new plot class needs to be added to each selection button if required. Then, in `ww_plotok`, an entry must be made for the new class for the plotting procedure associated with the new class. Note that so far all plots in $\mathcal{OYSTER}$ are handled by just three different procedures, `plotdata`, `plotncal`, and `plotuv`. This is to have similar kinds of plots handled by the same procedure in order to make the plotting code shorter. However, some kinds of plots are sufficiently different in their logic so as to require a different plotting procdure to be written. Finally, as far as the plot widgets are concerned, the new class needs to be added for each necessary index selection in `ww_indexsel`.

Whereas the widget routines for plotting in $\mathcal{OYSTER}$ are found in *plotwidget.pro*, the plot procedures and initialization functions are found in *plotting.pro*. Here we have to add an entry of the new class in `init_class` for the definition of the auxilliary information which is displayed if the user selects the "Identify" button. Then, which plot variables are to be listed in the plot selection widget is defined in `set_axitems`. Finally, the information of how multiple plots should be arranged in a plot for each plot class is specified in the plot procedure associated with the new plot class.

Part VI

CONSTRICTOR

Chapter 23

Introduction

For completeness' sake, we include here a description of the raw data averaging software, called CONSTRICTOR, which produces the input files for $\mathcal{OYSTER}$. CONSTRICTOR is written in C, but uses a mixed C and FORTRAN library for HDS file access. (This library is described in the $\mathcal{OYSTER}$ manual.)

CONSTRICTOR reads NPOI raw data files and observation logs, and decodes and integrates data such as visibilities, delays, photon rates, etc. NPOI raw data is written by the embedded system in form of packets, with different types depending on which data they hold. The packet definitions are contained in *packetdefs.h*. The packet headers, all conforming to the same format, contain information as to the type of the packet, its length and a time stamp.

CONSTRICTOR first reads the packet headers from all raw files associated with one night of observing and creates a combined packet directory in which all packets are in chronological order. Packet directories for the individual raw files are stored on disk (working directory), and can be read later for a faster restart on CONSTRICTOR should that become necessary. (More recently, $\mathcal{OYSTER}$ has been upgraded to create these auxiliary files for CONSTRICTOR with the added benefit of additional data integrity checks.) CONSTRICTOR then reads the FILE_HEADER and SYS_CONFIG packets for date and configuration information. It then proceeds to process all packets whose time stamps fall in between start and stop times defined in the input parameter file. The packets of the high rate data types, FRINGE_DATA (or BG_DATA for 6-way data), FDL_POSITION, and NAT_COUNTS, are processed by the *fringeav.c*, *fdlav.c*, and *natav.c* modules, respectively. Individual 2ms records of these packets are aligned (synchronized), and missing matches discarded. The integration length is defined by the number of records, *not* by a time interval. This is a pecularity which one has to keep in mind especially when reducing data for nights with very bad seeing.

In earlier versions of CONSTRICTOR, it was enabled to do both incoherent and coherent integrations of the visibilities. Since the latter has never worked well, this capability has been removed and is now part of $\mathcal{OYSTER}$'s COBRA procedures. The lean version of CONSTRICTOR therefore serves two purposes now: provide an output file with incoherently averaged data, and, optionally, reformat the raw data and store individual scans in separate HDS files for COBRA.

Chapter 24

How to use CONSTRICTOR

24.1 NPOI raw data files

High rate data files are files with the extensions *.fringeData* and *.alignData*. Metrology data files have the extension *.sidData.?*, where the index refers to the SIDcon ID, *not* the siderostat ID. Other files include the observer log (extension *.log*) and the scan log (extension *.starlog*).

24.2 Preparation of parameter file

The parameter file contains information for CONSTRICTOR on the location of the raw data files, start and stop times, integration time, etc. These are parameters used to control the output, but not parameters describing system configuration.

The parameter file, *YYYY-MM-DD.par*, can now be created using the `rawlist` procedure of $\mathcal{OYSTER}$. If you want to create the parameter file for the previous night, just run `rawlist` from the $\mathcal{OYSTER}$ prompt, for other dates you should supply the date, e.g. `rawlist,'2002-12-13'`. This procedure will look for files *YYYY-MMDD.** in datacon11 if run on the site data reduction computer octans. On any other computer, the raw data files should be in the current directory. Should you want to exclude certain files from being processed, compress them since `rawlist` ignores compressed files.

The procedure will do some basic file integrity tests, and it will also update header time stamps (in the packet directories) with the first time stamp found in the packet body. This is to prevent buffer overflows in CONSTRICTOR when synchronizing the records.

The `rawlist` procedure will also write the packet directories (*.dir* files) and packet listings (*.lst* files). The latter were previously produced by the pktlist program.

Check beginning and tail of the listing for each file. There should be an END_OF_DISK packet. If not, there will be an appropriate entry for the MaxPacket parameter in the parameter file. Also check if the data at the tail of the listing is from the same night as the data at the beginning. If the timestamps are not monotonically increasing in the entire listing, this is an indication that DATAcon crashed and you are looking at data from a previous night at the end of the disk. (EXCEPTION for metrology data: if there is no END_OF_DISK packet, another file for the same siderostat (see extension number) might have it if the file was broken into two or more pieces.

24.3 Parameters

- InFiles: The order of the input files is irrelevant.
- DirFiles: These must be in the same order as InFiles and MaxPacket.
- MaxPacket: The maximum number of packets to process for each file
- DayNumber: Even though this number is in the data, some packet headers have the wrong number and it makes it a whole lot easier to have the user check and enter this number correctly. If observations started before 0 UT, choose the day number of that previous day. CONSTRICTOR will add a multiple of 24 hours to the time tag when the day number changes. Remember to set the StopTime to, say, 48 hours if you use a DayNumber of one less than the actual day number.
- StartTime:
- StopTime: These times are for day number corrected time tags, i.e. if the day number changes during the observations, 24 hours are added to the tags and compared to the StartTime and StopTime parameters. This means you may have to specify a StopTime of 48 hours.
- LogFile: The observers log file, usually named YYYY-MMDD.log. If none is found, none is included in the output.
- ScnFile: The star log file, usually names YYYY-MMDD-XXXXXX.starlog. If none is, none is included in the output.
- OutFile: The name of the output file, should be YYYY-MM-DD.con, or YYYY-MM-DD.coh. The latter extension can be used as a memo for files which will have coherently integrated visibilities added later by $\mathcal{OYSTER}$. Please do not use other extensions.
- Raw: Write the raw data of every scan into a separate HDS file. This can then be read by the COBRA scripts of $\mathcal{OYSTER}$. (Off)
- Lock: Only average continuous locks. (Off)
- Triple: Compute and write the triple product. (On)
- NumAv: Number of samples in an incoherent integration. (500)
- RefStation: Which station to reference delays to. Choose one which is common to all spectrometers. (1)
- SmartFDLAverage:Average the difference between measured and pred. delays? (Off)
- Metrology: Decode and pass on metrology data (On)
- Nat: Process NAT data (On)

Here is an example of a CONSTRICTOR parameter file.

```
InFiles = "/datacon11/2003-0330-104722.fringeData","/datacon11/2003-0330-105135.fringeData","/d
DirFiles = "2003-03-30.1.dir","2003-03-30.2.dir","2003-03-30.3.dir","2003-03-30.4.dir","2003-03
ir","2003-03-30.6.dir","2003-03-30.7.dir","2003-03-30.8.dir"
MaxPacket =    5076,    5059,    4620,    4943,    5541,   17518,    1931,    1931
DayNumber =       89
StartTime = "00:00:00"
StopTime = "24:00:00"
LogFile = /datacon11/2003-0330.log
ScnFile = /datacon11/2003-0330-030621.starLog
OutFile = 2003-03-30.con
Raw = Off
Lock = Off
Triple = On
NumAv = 500
RefStation = 2
SmartFDLAverage = Off
Metrology = On
Nat = On
```

24.4 System configuration

As described in Chapter VI, the system configuration is not yet automatically produced by the control system. Neither is the *system.config* file, which is used by CONSTRICTOR for system configuration information, used to control the NPOI configuration. Ideally, the system configuration file is deposited in the same directory used for the observing list the day before observing. A script on sextans automatically transfers this file to the proper system area. If this is not done, the file *system.config* can always be placed in the directory where CONSTRICTOR is run, and will be loaded automatically, overriding any system configuration embedded in the raw data stream.

The system configuration is supposed to be written by the control system into the SYS_CONFIG packet stored in the raw data stream. This however is not currently implemented and instead the text packet, in KEYIN format, is fetched from disk (file *system.config*). Therefore, the stored system configuration can be wrong, if the configuration file was out of date. In this case, placing this file in the current directory for CONSTRICTOR will cause the software to load this file automatically and use it instead of the SYS_CONFIG packet. Make sure this file does not exist in the current directory if you want to use the SYS_CONFIG packet instead.

Here is an example of a system configuration.

```
MAX_WAV=16
MAX_FRFREQ=6
MAX_SID=4

Latitude = 35.0966666666667
Longitude = -111.535
```

```
Altitude = 2200.66

TDTminusUTC = 60.184
UT1minusUTC = 0

InstrCohInt = 2

NumBin = 64
BeamcombinerID = 2

InputBeam.NumSid = 6
InputBeam.StationID = "E02", "AC0", "AE0", "AW0", "W07", "AN0"
InputBeam.SiderostatID = 6, 1, 2, 3, 5, 4
InputBeam.Stroke = -1.0e-6, -3.0e-6, -3.0e-6, +3.0e-6, +2.0e-6, +2.0e-6
InputBeam.FDLTankID = 1, 2, 3, 4, 5, 6
InputBeam.BCInputID = 1, 2, 3, 4, 5, 6
InputBeam.StationCoord(01:04) =   5.599326,  -1.792619,   0.010397,   8.429653
InputBeam.StationCoord(05:08) =  -0.499984,  -5.639917,   0.000035,  13.415900
InputBeam.StationCoord(09:12) =  16.989235, -12.896476,  -0.001264,  31.406569
InputBeam.StationCoord(13:16) = -20.389907, -15.497607,  -0.000635,  27.541854
InputBeam.StationCoord(17:20) = -44.203123, -32.988367,  -0.081429,  56.867758
InputBeam.StationCoord(21:24) =  -0.627300,  17.212581,  -0.008760,  20.285819

OutputBeam.NumOutBeam = 2
OutputBeam.BCOutputID = 2, 3
OutputBeam.APDArrayID = 2, 3
OutputBeam.NumSid = 4, 4
OutputBeam.NumBiasFreq = 1, 3
OutputBeam.NumSpecChan = 16, 16
OutputBeam.BiasMod(01:06) = 8, 7, 10, 5, 3, 5
OutputBeam.BiasMod(07:12) = 1, 7, 10, 8, 3, 5
OutputBeam.StationID(01:04) = 1, 3, 4, 6
OutputBeam.StationID(05:08) = 1, 2, 4, 5

OutputBeam.Wavelength(01:04) = 849.4e-9, 820.9e-9, 793.9e-9, 768.3e-9
OutputBeam.Wavelength(05:08) = 744.2e-9, 722.9e-9, 701.5e-9, 683.1e-9
OutputBeam.Wavelength(09:12) = 664.6e-9, 648.9e-9, 617.7e-9, 603.5e-9
OutputBeam.Wavelength(13:16) = 590.7e-9, 577.9e-9, 566.5e-9, 556.6e-9

OutputBeam.Wavelength(17:20) = 849.4e-9, 820.9e-9, 793.9e-9, 768.3e-9
OutputBeam.Wavelength(21:24) = 744.2e-9, 722.9e-9, 701.5e-9, 683.1e-9
OutputBeam.Wavelength(25:28) = 664.6e-9, 648.9e-9, 617.7e-9, 603.5e-9
OutputBeam.Wavelength(29:32) = 590.7e-9, 577.9e-9, 566.5e-9, 556.6e-9

OutputBeam.WavelengthErr(01:04) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.WavelengthErr(05:08) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
```

```
OutputBeam.WavelengthErr(09:12) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.WavelengthErr(13:16) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9

OutputBeam.WavelengthErr(17:20) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.WavelengthErr(21:24) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.WavelengthErr(25:28) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.WavelengthErr(29:32) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9

OutputBeam.ChanWidth(01:04) = 31.3e-9,  28.9e-9,  26.7e-9,  24.7e-9
OutputBeam.ChanWidth(05:08) = 22.9e-9,  21.3e-9,  19.8e-9,  18.5e-9
OutputBeam.ChanWidth(09:12) = 17.2e-9,  16.1e-9,  15.1e-9,  14.2e-9
OutputBeam.ChanWidth(13:16) = 13.4e-9,  12.7e-9,  12.0e-9,  11.3e-9

OutputBeam.ChanWidth(17:20) = 31.3e-9,  28.9e-9,  26.7e-9,  24.7e-9
OutputBeam.ChanWidth(21:24) = 22.9e-9,  21.3e-9,  19.8e-9,  18.5e-9
OutputBeam.ChanWidth(25:28) = 17.2e-9,  16.1e-9,  15.1e-9,  14.2e-9
OutputBeam.ChanWidth(29:32) = 13.4e-9,  12.7e-9,  12.0e-9,  11.3e-9

OutputBeam.ChanWidthErr(01:04) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.ChanWidthErr(05:08) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.ChanWidthErr(09:12) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.ChanWidthErr(13:16) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9

OutputBeam.ChanWidthErr(17:20) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.ChanWidthErr(21:24) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.ChanWidthErr(25:28) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
OutputBeam.ChanWidthErr(29:32) = 0.1e-9, 0.1e-9, 0.1e-9, 0.1e-9
```

Note the OutputBeam.NumSid and OutputBeam.StationID parameters; these parameters replace the old OutputBeam.FringeMod and OutputBeam.InputPair parameters which specified the baselines and their modulations directly. Even though CONSTRICTOR still understands the old parameters, the new ones are easier to use since CONSTRICTOR will compute the combinations and their modulations amplitudes automatically from the list of stations in each spectrometer and their strokes. OutputBeam.StationID tells which stations are in a given spectrometer, by giving its index in the InputBeam.StationID array (starting with 1). (That means that th values specified should always be smaller or equal to the number of stations used, a common mistake!)

MAX_SID, like the other MAX parameters, is used to tell CONSTRICTOR the number of places allocated, but not necessarily used, for the specifications in the KEYIN style system configuration. For example, OutputBeam.StationID in the KEYIN file is a one-dimensional array (KEYIN does not recognize more than one dimension), but needs to be converted to a two-dimensional array which specifies for each output beam what stations it contains. The MAX_SID parameter is used to determine where in the KEYIN vector the next output beam section begins (here: after the first 4 numbers).

Similarly, the MAX_FRFREQ is used to decode the OutputBeam.BiasMod numbers. Again,

the actual number of baselines can be less (but not more of course) than this parameter allows.

Also note that the FDLTankID numbers are used to determine at which position in the FDL_POSITION packet one can find the delay data for a given station. (This is only true for 6-station data; in the case of the 3-station data all three delay lines occupy the first three dimensions in the FDL_POSITION packet.) The same holds for SiderostatID, which actually should have been called SidCon numbers. This is because NAT, SidModel, and Motor counts are stored in the order of SidCon IDs, *not* in the station order. CONSTRICTOR uses SiderostatID to put the data into the right place. Here for example, the E02 NAT data is not the first entry in its packet, but the sixth.

One word on the BeamCombinerID. It is one for the 3-station combiner, 2 for the 6-station combiner, and 3 for the 6-station combiner used for astrometry. The only difference between the last two cases is whether or not the white light (α Lab) fringe modulation needs to be doubled. If BeamCombinerID=3, then the modulation has to doubled.

24.5 Running CONSTRICTOR

Type: /home/cah/constrictor/constrictor

You will see a '*' prompt. This is KEYIN communication; you can modify parameters interactively (which you should NOT, unless you edit the parameter file accordingly). Type, e.g., @1997-08-02.par, and KEYIN will read the parameter file. Type show, and you will get the current setting of the parameters. Then, if everything is OK (check carefully!), hit '/'. CONSTRICTOR will now open all files, create packet listings which are saved in the .dir files (or read a .dir file if available from `rawlist`), and process all data (falling in between StartTime and StopTime). At the end, you will be returned to KEYIN, at which point you should type quit to exit CONSTRICTOR.

24.6 Programmer's reference

Parts of CONSTRICTOR got fairly complicated over time due to pecularities of the raw data, the requirement of backward compatibility, and other issues.

Part VII

INCHWORM

Chapter 25

Introduction

INCHWORM, written by N. Elias II, is *not* part of the $\mathcal{OYSTER}$ software, but we document a few instructions here on how to use it.

To create a .inch file from a .con file, startup INCHWORM and then type inchworm inch1 /gemini/npoi/con/2003-03-15.con -dir. You can delete the inch1 object by typing inch1 delete.

Then open the file and go to Raw|ScanData, double click on sid 1, and go to Edit|De-glitch. In this widget deselect tuple (1,2,4) and (1,3,5), and then click on Find. Fix every glitch listed by double-clicking the glitch and then, judging by the plot, the laser which glitched. Select this laser (Calc) and click Fix. Note that lasers 2&4, as well as 3&5 are degenerate since they are 180 degrees apart. Also, lasers 2&5 and 3&4 are on the same side of the plate and should therefore move in the same way. You do not have to work on the glitches in sequence, but note that of course every fix will shift all laser values from current to end by that amount. Typically, there is up to a three count difference between calculated and predicted values of the glitch. 100 counts = 0.5 μm. Any notes can be entered into the INCHWORM log using View—Logs—InchwormLog.

After fixing all glitches, go to Average, (Re)create|ScanData, then Calculate|By Time, then Calibrate and then Done.

To prepare the motion group, go to Motion,)(Re)create|Go, Calculate, Calculate, Done, Done. To exit, go to File|Done.

Part VIII

COBRA

Chapter 26

Introduction

A collection scripts capable of reading HDS formatted NPOI raw data for analysis and display. There is also a small class of procedures which read packets directly from a NPOI raw file. These are described at the end of this part.

26.1 Reading raw HDS scan files

In the first implementation of optional raw data output in CONSTRICTOR, NPOI raw data of a scan was written into a single HDS formatted file. However, long scans (e.g. scans for atmospheric studies exceed 30 minutes) and the advent of the 6-way fringe engine made the current logic necessary, which is as follows. Scans will be divided into sub-scans, which have an additional extension after the scan number, identifiying the sequence number. Sub-scan files can still be read individually, but the standard procedure is to read and concatenate at least the time stamps and delay line metrology data from all sub-scans, and then to read the fringe data individually.

`get_rawdata`,[*file_spec*<_char>]

> Procedure to read raw time and delay data from one or more sub-scan files specified by *file_spec*. Example: get_rawdata,'2001-10-25.raw.013'. Note that one does *not* include the sub-scan extension, as the procedure itself checks how many sub-scan files are present on disk. The data from all files found is read and concatenated. If there is only one sub-scan file, it will be left open. If no file is specified, it is assumed that a file is open, in which case the time and delay data of just this file will be read. The file will be left open.

`get_bincounts`,*spectrometer*<_int>

> Read the bin count data for the specified spectrometer. A raw HDS file must be open. The spectrometer number, i.e. the output beam, is stored in a common block for shared reference.

`get_data`,*filename*

> This standard procedure to read any kind of NPOI standard data files can be used to read a single raw file if the full filename is specified. All spectrometers will be

read, and therefore the cautionary notes as to the spectrometer index in *bincounts* apply. The file will be left open.

26.2 Frames

At NPOI, a fringe frame is a two-dimensional array of count rates, whereby one dimension corresponds to the 8 or 64 phase bins across the fringe, and the other dimension corresponds to the channel axis. A frame is recorded every 2 m1997-05-15.chas. The frames are stored in *bincounts*, which can have dimensions, in this order, of the number of spectrometers, number of bins, number of channels, and the number of frames. This structure is the same whether or not one is using the 3 or 6 way beam combiners. All COBRA functions are guaranteed to work if a single spectrometer is selected; only a few accept frame or visibility variables with a spectrometer index, as noted specifically below. Therefore, the default is that the bincounts array does *not* have a spectrometer index.

`displayframe`, *c*

Display frames on the TV. Any index selections in the different dimensions can be displayed, provided all phase bins are always selected. Examples: displayframe,bincounts(0,*,*,0), displayframe,bincounts(0,*,0,0:99).

`rotateframe`(*c*, *l*, *a*, *g*)

Function. Apply rotation by group delay *d* and airpath *a* to fringe frame *c* and return rotated frame.

`fringeframe`(*a*, *g*, *l*, *v*, *n*[, */poisson*])

Function. Return a simulated fringe frame for a given path in air, *a*, delay tracking offset (group delay), *g*, wavelength grid, *l*, visibility amplitude, *v*, and photonrate, *n*. Optionally include Poisson noise. *a* and *g* can be arrays, as well as *v* and *n*. The number of bins is inferred from the *genconfig.numbin* variable. The default number of bins is 8.

26.3 Visibilities

`fringevis`(*bincounts*),[*k*]

Function. Return complex visibility for bincounts. The parameter *k* selects a particular baseline, i.e. fringe frequency. Examples: v=fringevis(bincounts(*,0,*),7), which returns the (complex) visibility of channel 1, and the baseline with the frequency number 7. If no sub-array of *bincounts* is selected, then the returned visibility array has the same dimensions as the input, minus the bin index.

`fringevissq`(*bincounts*<_real>),[*k*,*n*]

Function. Return the (averaged) squared visibility amplitude for bincounts. The parameter *n* selects how many samples will be averaged.

`visrotate`(*v*, *l*, *d*)

Function. Return rotated complex visibility for input visibility *v*, wavelength scale *l*, and group delay *d*.

26.4 Fringe delays

`fdldelay`(*baseline*[,*spectrometer*, */scan*])

Function. Return residual FDL delay for specified baseline. The baseline parameter can either be a string variable (e.g. 'AW0-AE0'), or an integer in conjuction with the spectrometer number specifiying an entry in *GenConfig.BaselineId*. The *scan* keyword specifies whether or not all records belonging to the scan will be returned. If not, then only the records associated with the currently loaded bincount data (if any) will be returned. The default is *scan=1*.

`visdft`(*v*, *l*, *d*)

Function. Return complex Fourier transform of visibility with respect to wavenumber, computed at delay values *d*. *l* are the channel wavelengths in the same units as *d*.

`groupdelay`(*p*, *l*, *d*)

Function. Return position of peak of amplitude of complex power spectrum *p*.

`whitedelay`(*v*, *l*)

Function. Return white light phase delay for complex visibility *v*. Caution: phases should be unwrapped before!

26.5 Photon rates

`fringenphot`(*bincounts*<_real>)

Function. Return photonrate.

26.6 NAT data

`natoffset`(*c*)

Function. Return x and y offset (in units of the airy disk) of the image on the nat quad cell.

26.7 Compound plot procedures

These procedures use more elementary functions to produce diagnostic plots.

`plot_fdldelay`,*baseline*<_int>[,*beam*<_int>][,*/slide*]

Plot the residual FDL delay in microns. If keyword *slide* is 1, then open a sliding plot widget.

`plot_fdlpower`,*baseline*<_int>[,*beam*<_int>][,*/sf*]

Plot a power spectrum (or structure function if *sf=1*) of the delay. The time stamps must be integer multiples of the instrumental coherent integration time (2 ms currently at NPOI). Example: plot_delayseries, fdlstamp(),fdldelay('AE0-AW0').

`plot_vispower`,*baseline*<_int>[,*beam*<_int>][,*numav=numav*]

Plot visibility power spectra in a special browser plot.

`plot_fringepower`[,*channels=channels*][,*numav=numav*]

Plot mean of squared visibility amplitude averaged over *numav* samples (default 10 samples) as a function of k frequency.

`plot_fringeimage`[,*channels=channels*]

Display image of fringes from the frame data.

`plot_images`,*frames*

Display fringe frames in a special browser window. Example: plot_images,bincounts

`plot_powerpeaks`,*baseline*<_int>[,*beam*<_int>,*scanfile*<_char>] [,numav=numav]

Compute and plot scatter plots of primary and secondary peak heights in the fringe powerspectra.

`plot_spectrumpeaks`,*baseline*<_int>[,*beam*<_int>,*scanfile*<_char>] [,channels=channels][,numav=numav]

Compute fringe spectra of the squared visibility amplitude averaged incoherently over numav samples as a function of fringe frequency. Make a scatter plot of the peak amplitude at the k value corresponding to the selected baseline, versus the maximum peak amplitude over all the other fringe frequencies which are not supposed to contain any signal.

`plot_groupdelay`,*baseline*<_int>[,*beam*<_int>,*scanfile*<_char>] [,numav=numav][,classic=classic][,slide=slide]

Plot the groupdelay.

`plot_fringephase`,*baseline*<_int>[,*beam*<_int>,*scanfile*<_char>]

For the specified baseline, plot the corrected fringe phase. If *beam* is not specified, use currently loaded data. With the *datum* parameter, all available scan files will be read and analysed.

`plot_dispsol`,*baseline*<_int>[,*beam*<_int>,*scanfile*<_char>] [,compute=compute][,classic=classic]

Plot the dispersion corrected delays, along with the raw FDL delays for comparison.

`plot_ratehist`,*channels*

Plot count rate histograms for the specified channels.

`plot_coherence`,*baseline*,*channels*

Plot coherence analysis for the selected baseline and channel.

`plot_nathist`,*sid*

Plot NAT count rate histograms.

Chapter 27

NPOI raw packet data files

A small number of procedures capable of directly accessing NPOI raw packet files.

`packetdir`,*file*

> Function. Return packet directory for specified file. This is an array of structures, each one specifying file, day number, time, type, length, and offset in bytes from the beginning of the file for a packet. Example: pdr=packetdir('2001-1121-131547.fringeData').

`packettype`,*type*

> Function. Return human-readable name of packet type specified through *type*. Example: print,packettype('0x000b0000').

`readpacket`,*packet_dir_record*

> Function. Read and return packet specified in argument, which is an entry in the packet directory. Example: p=readpacket,pdr(17).

`packetlist`, *YYYY-MMDD*

> Open all raw packet files corresponding do embedded system date YYYY-MMDD. Create GUI displaying packetlistings for each file, and a combined list of stars. If one or more stars are selected (click and/or drag mouse cursor), scans corresponding to the star selection are listed, which in turn can be clicked to display the packets corresponding to a specific scan. If packets are clicked (i.e. selected), the packet is read (if that type is implemented), and some information is displayed and/or stored in $\mathcal{OYSTER}$ global variables, like *Date, GenConfig*, etc. The packet contents are made available in the common block variable *packet*.

Part IX

VOLVOX

Chapter 28

Introduction

28.1 About VOLVOX

A small number of procedures dealing with global astrometric fits to interferometric delay data.

Part X

PEARL

Chapter 29

Introduction

29.1 About PEARL

A collection of procedures capable of imaging interferometric data. These were written to develop an algorithm for combining multi-channel data by accounting for the wavelength dependence of the image if composed of black-body radiators (or stellar atmospheres). A corresponding multi-channel CLEAN was implemented together with a self-calibration routine, following a difference mapping scheme. In the latter, a model (list of CLEAN components) is built-up incrementally between phase self-calibrations. The CLEAN algorithm is controlled with a slider widget to adjust the number of components to be added to the model, updating the residual image for each component added or subtracted.

PEARL will work with the complex visibilities, which have to be initialized with the observed visibility and closure phase data using the `set_complexvis` procedure. For testing purposes, if PEARL is called directly after computing model visibilities with `calcmodel`, the complex visibilities used by PEARL correspond exactly to the model.

The PEARL widget has two HELP buttons for further information on how to use it. The final PEARL image consists of a white light image and two maps, one of the effective temperature and the other one of the surface gravity. Together, these fully constrain images of stellar surfaces and multiple stellar systems.

29.2 Getting started

First, let us start with a simple example which will also test the code of PEARL. Execute the following commands to read a data set delivered with *OYSTER* and the model of a double star fit to these data. Compute the complex model visibilities (`calcmodel`) to have a "perfect" data set for which one should be able to get a good image.

```
cd oyster/lab/data/2001
oyster
get_data,'1997-05-01.cha'
readmodel,'zetuma.model'
binary_model.period=200  ; Binary will move during obs. otherwise
scans.vissqcerr=abs(scans.vissqcerr) ; Use all data
scans.triplephasecerr=abs(scans.triplephasecerr) ; Use all data
```

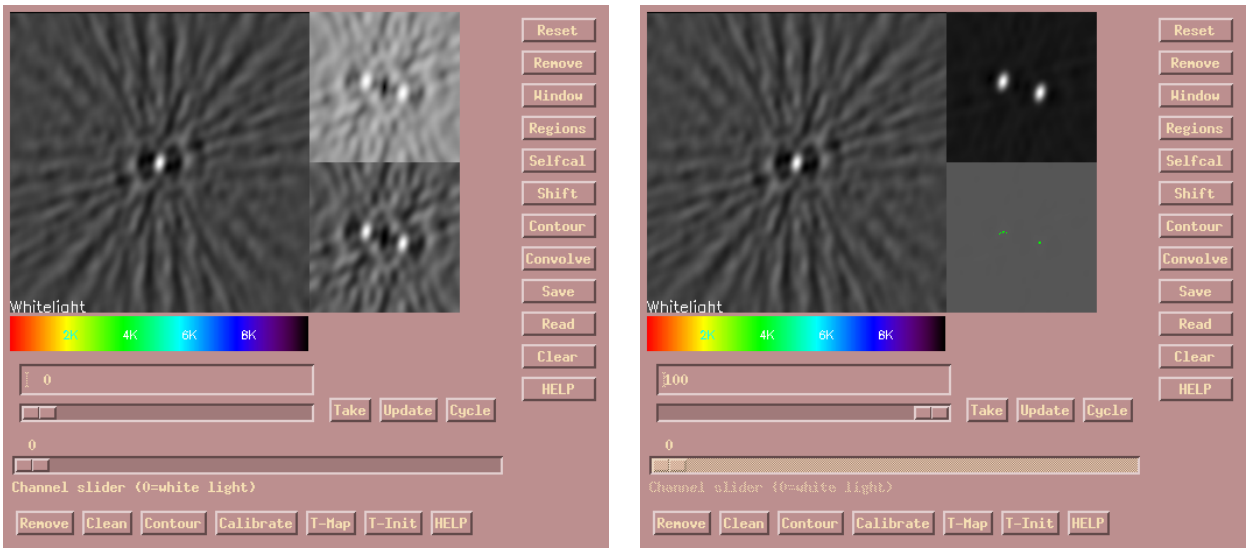

Figure 29.1: Main PEARL GUI

```
calcmodel ; Compute model visibilities
pearl,'FKV0497',cellsize=0.3,imsize=129
```

After starting PEARL, you will be presented with GUI as shown in Fig. 29.1.

Drag the slider under the large display of the synthesized point spread function (aka "dirty beam") to the right to run an interactive version of the CLEAN algorithm (the counter above the slider shows the number of CLEAN components added to the model and subtracted from the "dirty image" shown in then lower-right display (green dots show the locations of the CLEAN components). You can also drag the slider to the left to remove the CLEAN components from the model. The convolution of the model with the CLEAN beam (central peak of the dirty beam) is shown in the small upper-right display.

When you click the Update button, the current model is used to compute the complex model visibilites, run a phase self-calibration, and display the residual image in the lower right panel, and the final image (CLEAN components convolved with the CLEAN beam added to the residual image) in the upper right panel.

29.3 Phase self-calibration

Image reconstruction based on the Fourier transform of the complex visibilities requires the visibility phases in addition to the amplitudes. Since phase information is only available as closure phases, phase self-calibration is used with an initial model to minimize the differences between model closure phases and observed phases by adjusting telescope-based phase offsets (which caused by the atmosphere). The procedure `set_complexvis` performs this operation using a point-source model and must be called before the use of `pearl`. Within the latter, the phase self-calibration is performed with the current image by `selfcal` (button SelfCal).

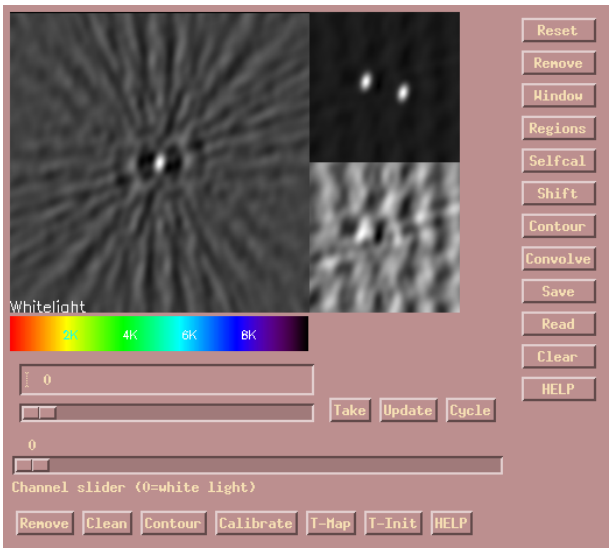

Figure 29.2: Main PEARL GUI

29.4 Programmer's reference

The `pearldata` routine is the first one called after starting `pearl`. The data used are the complex visibilities (*scans.complexvis*) and their weights (*scans.complexweight*), the *uv* coordinates, and the photometry (*scans.photometry*). The measurements are stored in one-dimensional vectors, and auxiliary information such as scan IDs, spectrometers, baselines, and channels are stored in the same vector format. Only measurements with the same time stamp are considered to be belonging to the same interferometer and can be combined in the self-calibration process.

29.5 Beam procedures and functions

`displaybeam`,*starid*<_CHAR>

> Procedure; compute the dirty beam for a given star and display it on the screen. Example: displaybeam,'FKV0621'

`contourbeam`,*starid*<_CHAR>

> Procedure; plot a contour map of the dirty beam.

`cleanbeam`,*beam*

> Function; return parameters of an ellipse fitted to the half-maximum points of the dirty beam. Example: print,cleanbeam(dirtybeam('FKV0621'))

Part XI

Appendices

Appendix A

Plot variables and their indices

Some data selection widgets for plotting display more indices than needed for some of the plot variables in order to avoid recycling of the data selection widget if the variables change. In the following tables, an overview is presented over the indices applicable to the plot variables. They are organized by class, even though, strictly speaking, an $\mathcal{OYSTER}$ plot class refers only to a specific selection of variables listed in a plot selection widget. More precisely, the integration interval (if applicable) is the same for all variables for a table.

Also note that the IDL variables in *italic* font denote a derivation from this variable, i.e. the data is not exactly identical to the variable contents. Where the IDL variable field is left blank, the formula is more complicated and might involve more than one variable.

Table A.1: Plot variables (class point) and their indices

Plot variable	IDL name	inbeam	outbeam	triple	channel	baseline	point
Time	pointtime						pt
PointNo							pt
VisSq	vissq		ob		ch	bl	pt
VisAmp	visamp		ob		ch	bl	pt
VisPhase	visphase		ob		ch	bl	pt
TripleAmp	tripleamp			tr	ch		pt
TriplePhase	triplephase			tr	ch		pt
PhotonRate	photonrate		ob		ch		pt
FDLPath	*fdlpos*	ib					pt
FDLDelay	*fdlpos*	ib					pt
DelayJitter	delayjitter		ob			bl	pt
NATJitter	natjitter	ib					pt
GrpDelay	grpdelay	ib					pt
DryDelay	drydelay	ib					pt
WetDelay	wetdelay	ib					pt
VacDelay		ib					pt
MetroDelay	*metropos*	ib					pt

Table A.2: Plot variables (class bg) and their indices

Plot variable	IDL name	inbeam	outbeam	triple	channel	baseline	point
Time	bgscans.time						sc
ScanNo							sc
BGRate	bgscans.rate		ob		ch		sc

Table A.3: Plot variables (class scan) and their indices

Plot variable	IDL name	inbeam	outbeam	triple	channel	baseline	point
Time	scans.time						sc
ScanNo							sc
VisSq	scans.vissq		ob		ch	bl	sc
VisSq c	scans.vissqc		ob		ch	bl	sc
VisSq/e	scans.vissqe		ob		ch	bl	sc
VisSq c/e	scans.vissqec		ob		ch	bl	sc
TripleAmp	scans.tripleamp			tr	ch		sc
TripleAmp c	scans.tripleampc			tr	ch		sc
TripleAmp/e	scans.tripleampe			tr	ch		sc
TripleAmp c/e	scans.tripleampec			tr	ch		sc
PhotonRate	scans.photonrate		ob		ch		sc
NATCounts	scans.natcounts	ib					sc
BeamCounts	*scans.natcounts*		ob				sc
SpecCounts	*scans.photonrate*		ob				sc
BackgndRate	scans.backgndrate		ob		ch		sc
FDLPath	scans.fdlpos	ib					sc
DelayJitter	scans.delayjitter		ob			bl	sc
TrackJitter	*scans.natjitter*		ob			bl	sc
FDLDelay	*scans.fdlpos*	ib					sc
GeoDelay	scans.geodelay	ib					sc
uv-radius	*scans.uvw*		ob		ch	bl	sc
HourAngle	scans.ha						sc
ZenithAngle	scans.za						sc
MirrorAngle	scans.ma						sc
U	*scans.uvw*		ob		ch	bl	sc
V	*scans.uvw*		ob		ch	bl	sc
W	*scans.uvw*		ob		ch	bl	sc
FDL_O-C		ib					sc
Grp_O-C		ib					sc
Dry_O-C		ib					sc
Wet_O-C		ib					sc
VisSq m	scans.vissqm		ob		ch	bl	sc
TripleAmp m	scans.tripleampm			tr	ch		sc
TriplePhase m	scans.triplephasem			tr	ch		sc
ModelDelay	scans.modeldelay	ib					sc
MetroDelay	scans.metrodelay	ib					sc

Table A.4: Plot variables (class metro) and their indices

Plot variable	IDL name	inbeam	outbeam	triple	channel	baseline	point
MetroTime	metrotime						pt
Par X	parx	ib					pt
Par Y	pary	ib					pt
Par Z	parz	ib					pt
MetroPath	*metropath*	ib					pt
MetroDelay	*metropath*	ib					pt

Table A.5: Miscellaneous plot variables and their indices

Plot variable	IDL name	inbeam	outbeam	triple	channel	baseline	point
Wavelength	genconfig.wavelength		ob		ch		
Channel					ch		
BLlength			ob			bl	
Baseline			ob			bl	
Triple				tr			

Appendix B

CONSTRICTOR output file structure

The data are in an HDS (Hierarchical Data System [Starlink Project]) structure, somewhat analogous to a directory structure. It differs from a directory structure in that a "directory" can actually be an array. Each element of the array will have the same set of sub-directories. Since the structure is like a directory tree, the order of variables listed below is not meaningful. Furthermore, variables can be added or removed in the future. Therefore, this list amounts to a list of the data we think we want, and of the names we have agreed to give them.

In addition to the natural array dimensions such as spectral channel number, point number, etc., we use one more to label the real and imaginary parts of a complex number, indicated by "`[R/I]`". The uncertainties for complex numbers have three components, which may be the major axis, minor axis, and position angle of the error ellipse.

For locations, we use three-element vectors with "East, North, Up" as the order of the elements

For directions, we use two-element vectors with the first element (θ) being the angle from the zenith, and the second element (ϕ) being the angle in the horizontal plane, measured from east through north.

We have not specified the units in the database description. For lengths, we will use meters. Time stamps are offsets from 0 h UT of `DataSet.Date`. For time stamps and sample intervals, we will use milliseconds. We use `<_integer>` (which is int*4, with a range of +/-2e9, or +/-68 days) for time stamps, and `<_double>` for times (e.g., start and stop times).

The format for the descriptions is:

```
VariableName [IndexLimit(source)][ ... ] <Data type>
```

So, for instance,

```
ComplexVis[NumPoint][NumBaseline(beam)][NumSpecChan(beam)][R/I]
```

under ScanData.PointData.OutputBeam[NumOutBeam] is a four-dimensional array, with the first dimension ranging from 0 to NumPoint - 1, the second from 0 to NumBaseline(beam) - 1 (where NumBaseline can differ from one beam combiner output beam to another), and the third ranging from 0 to NumSpecChan(beam) - 1. In the fourth index, 0 corresponds to the real part and 1 to the imaginary part. In this example, [NumPoint] could have been indicated as [NumPoint(scan)]; however, we suppress listing the source of the parameter if that source is in the same object as the indexed variable. In this example, the variable NumScan is in the ScanData object that contains ComplexVis, so the source for NumPoint is suppressed, but NumOutBeam is in the GenConfig object, so the source for NumBaseline is indicated.

We use the C convention, where the last index listed is the most rapidly varying.

Tables are not part of the HDS system, but they are easily implemented. A table, such as ScanData, is simply a structure like any other, containing a set of substructures, each of which is an array. What makes it a table is our agreement that the items (a primitive, a structure, or a sub-array) pointed to by the first index of each array correspond with one another, e.g., PointData[1] (where PointData[] is itself a complicated structure) was taken between StartTime[1] and StopTime[1] while we observed StarID[1], and so on.

We use two kinds of table here: <Table> and <ExtTable>. Both are slightly different from the table type described in the previous paragraph. Both types contain one integer scalar quantity giving the number of rows (e.g., NumData), followed by the contents of the table, i.e., one or more arrays of variables for which the first index runs from 1 to (NumData - 1). The <ExtTable> (extended table) type also allows array variables whose first index does not correspond with row number. However, the primitives that make up that variable must be array primitives whose first index does correspond with row number.

```
Session
    Format <_char>
    Date <_char>
    SystemID <_char>
    ObserverLog <_char>
    ConstrictorLog <_char>
    GeoParms
        Latitude <_double>
        Longitude <_double>
        Altitude <_double>
        EarthRadius <_double>
        J2 <_double>
        LeapSeconds <_integer>
        EarthRotation <_double>
    GenConfig
        InstrCohInt <_double>
        BeamCombinerID <_integer>
        NumLaserP2P <_integer>
        P2PLaunchPlate [NumLaserP2P] <_integer>
        P2PRetroPlate[NumLaserP2P] <_integer>
        MasterPlateID <_char*>
        NumPlate <_integer>
```

```
    Plate [NumPlate]
        NumCluster <_integer>
        PlateEmbedded <_integer>
        PlateID <_char*>
        PlateLoc [3] <_double>
        PlateLocErr [3] <_double>
    InputBeam <Table>
        NumSid <_integer>
        StationID [NumSid] <_char>
        SiderostatID [NumSid] <_integer>
        DelayLineID [NumSid] <_integer>
        BCInputID [NumSid] <_integer>
        StarTrackerID [NumSid] <_integer>
        StationCoord [NumSid][4] <_double> four coords incl delay
    OutputBeam <Table>
        NumOutBeam <_integer>
        SpectrometerID [NumOutBeam] <_char>
        NumBaseline [NumOutBeam] <_integer>
        NumSpecChan [NumOutBeam] <_integer>
        FringeMod [MaxNumBaseline][NumOutBeam] <_integer>
        BaselineID [MaxNumBaseline][NumOutBeam] <_char>
        Wavelength [MaxNumSpecChan][NumOutBeam] <_double>
        WavelengthErr [MaxNumSpecChan][NumOutBeam] <_double>
        ChanWidth [MaxNumSpecChan][NumOutBeam] <_double>
        ChanWidthErr [MaxNumSpecChan][NumOutBeam] <_double>
    Triple <Table>
        NumTriple <_integer>
        OutputBeam [NumTriple][3] <_integer>
        Baseline [NumTriple][3] <_integer>
        NumSpecChan [NumTriple] <_integer>
        SpecChan [NumTriple][3] <_integer>
SidMetConfig [NumPlate]
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
    Theta [NumLaser] <_integer>
    ThetaErr [NumLaser] <_integer>
    Phi [NumLaser] <_integer>
    PhiErr [NumLaser] <_integer>
    LaunchInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
```

```
        AirGapThickErr [NumAirGap] <_double>
OptAnchConfig [NumPlate][MaxNumCluster]
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
    Theta [NumLaser] <_integer>
    ThetaErr [NumLaser] <_integer>
    Phi [NumLaser] <_integer>
    PhiErr [NumLaser] <_integer>
    LaunchInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
Pier2PierConfig
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
    Theta [NumLaser] <_integer>
    ThetaErr [NumLaser] <_integer>
    Phi [NumLaser] <_integer>
    PhiErr [NumLaser] <_integer>
    LaunchInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
```

```
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
ExtCatEyeConfig [NumPlate]
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
    Theta [NumLaser] <_integer>
    ThetaErr [NumLaser] <_integer>
    Phi [NumLaser] <_integer>
    PhiErr [NumLaser] <_integer>
    LaunchInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
PlateExpConfig [NumPlate]
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
    Theta [NumLaser] <_integer>
    ThetaErr [NumLaser] <_integer>
    Phi [NumLaser] <_integer>
    PhiErr [NumLaser] <_integer>
    LaunchInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
```

```
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
MetAirTempConf [NumPlate]
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
MetSolidTmpConf [NumPlate]
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
MetPressConf [NumPlate]
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
MetHumConf [NumPlate]
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
LabAirTempConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
```

```
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
LabSolidTmpConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
LabPressConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
LabHumConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
DLPressConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
FBPressConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
```

```
    LocErr  [NumSensor][3] <_double>
WxAirTempConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
WxPressConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
WxHumConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
WindSpeedConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
WindDirConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    Cross [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
BGScanData <ExtTable>
    NumBGScan <_integer>
    ScanID [NumBGScan] <_integer>
    Time [NumBGScan] <_double>
```

```
    RA [NumBGScan] <_double>
    Dec [NumBGScan] <_double>
    OutputBeam [NumOutBeam] <ExtColumn>
        Rate [NumBGScan][NumSpecChan(beam)] <_real>
ScanData <Table>
    NumScan <_integer>
    ScanID [NumScan] <_integer>
    StartTime [NumScan] <_double>
    StopTime [NumScan] <_double>
    Code [NumScan] <_integer>
    NumPoint [NumScan] <_integer>
    StarID [NumScan] <_char>
    WASA_Image [NumScan] intensity and position(s) of stars in field
    NumCoh [NumScan] <_integer> instr. coh. int.s per CONSTRICTOR coh. int.
    NumIncoh [NumScan] <_integer>
    PointData [NumScan] <ExtTable>
        NumPoint <_integer>
        Time [NumPoint] <_double> time of mid-point
        SoftDelay [NumPoint][NumSid] <_double>
        InputBeam [NumSid]
            FDLPos [NumPoint] <_double>
            FDLPosErr [NumPoint] <_double>
            NATJitter [NumPoint] <_real>
            NATJitterErr [NumPoint] <_real>
        OutputBeam [NumOutBeam] <ExtColumn>
            SoftDelay [NumPoint][NumBaseline(beam)] <_real>
            SoftDelayErr [NumPoint][NumBaseline(beam)] <_real>
            DelayJitter [NumPoint][NumBaseline(beam)] <_real>
            VisSq [NumPoint][NumBaseline(beam)][NumSpecChan(beam)] <_real>
            VisSqErr [NumPoint][NumBaseline(beam)][NumSpecChan(beam)] <_real>
            ComplexVis [NumPoint][NumBaseline(beam)][NumSpecChan(beam)][R/I] <_real>
            ComplexVisErr [NumPoint][NumBaseline(beam)][NumSpecChan(beam)][A/B/C] <_real>
            PhotonRate [NumPoint][NumSpecChan(beam)] <_real>
            PhotonRateErr [NumPoint][NumSpecChan(beam)] <_real>
        Triple [NumTriple] <ExtColumn>
            ComplTriple [NumPoint][NumTripleChan][R/I] <_real>
            ComplTripleErr [NumPoint][NumTripleChan][R/I] <_real>
SidMetData [NumPlate]
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumLaser(plate)] <_integer>
OptAnchData [NumPlate][MaxNumCluster]
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumLaserMax(plate,cluster)] <_integer>
Pier2PierData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumLaser(plate)] <_integer>
ExtCatEyeData [NumPlate]
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumLaser(plate)] <_integer>
PlateExpData [NumPlate]
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumLaser(plate)] <_integer>
MetAirTempData [NumPlate]
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor(plate)] <_word>
MetSolidTmpData [NumPlate]
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor(plate)] <_word>
```

```
MetPressureData [NumPlate]
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor(plate)] <_word>
MetHumidityData [NumPlate]
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor(plate)] <_word>
LabAirTempData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
LabSolidTmpData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
LabPressData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
LabHumData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
DLPressData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
FBPressData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
WxAirTempData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
WxPressureData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
WxHumidityData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
WindSpeedData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
WindDirData
    NumData <_integer>
    TimeStamp [NumData] <_integer>
    Data [NumData][NumSensor] <_word>
```

Appendix C

CHAMELEON file structure

The file format of averaged data is described here. It closely follows the format of the CONSTRICTOR files.

```
DATASET
     FORMAT          <_CHAR>                                  CONSTRICTOR, CHAMELEON, INCHWORM
     DATE            <_CHAR>                                  YYYY-MM-DD (UT date of first data point)
     SYSTEMID        <_CHAR>                                  NPOI, Mark_III
     USERID          <_CHAR>                                  Login name
     OBSERVERLOG     <_CHAR>
     CONSTRICTORLOG  <_CHAR>                                  Parameters used to run CONSTRICTOR
     GEOPARMS
          LATITUDE        <_DOUBLE>                           [degrees N]
          LONGITUDE       <_DOUBLE>                           [degrees E]
          ALTITUDE        <_DOUBLE>                           [m]
          EARTHRADIUS     <_DOUBLE>                           [m]
          J2              <_DOUBLE>
          TAI-UTC         <_DOUBLE>                           [s]
          TDT-TAI         <_DOUBLE>                           [s]
     GENCONFIG
          INSTRCOHINT     <_DOUBLE>                           Instrumental coherent integration time, [ms]
          BEAMCOMBINERID  <_INTEGER>
          REFSTATION      <_INTEGER>                          Station to which delays are referenced; First = 1
          INPUTBEAM
               NUMSID          <_INTEGER>
               SIDEROSTATID   [NumSid] <_INTEGER>              Unique ID
               BCINPUTID      [NumSid] <_INTEGER>              Which beam combiner input?
               STARTRACKERID  [NumSid] <_INTEGER>              Unique ID of quadcell
               STATIONID      [NumSid] <_CHAR*3>               e.g. AW0, E02
               DELAYLINEID    [NumSid] <_INTEGER>
               STATIONCOORD   [     4][NumSid] <_DOUBLE>       x=east, y=north, z=up, d=delay constant, all in [m]
          OUTPUTBEAM
               NUMOUTBEAM      <_INTEGER>                    Also the number of spectrometers
               NUMBASELINE    [NumOutBeam] <_INTEGER>          Number of baselines in output for spectrometer
               NUMSPECCHAN    [NumOutBeam] <_INTEGER>          Number of spectral channels for each spectrometer
               SPECTROMETERID [NumOutBeam] <_CHAR*7>           Unique ID
               BASELINEID     [MaxNumBaseline][NumOutBeam] <_CHAR*7>     Made up of StationIDs, e.g. AW0-E02
               WAVELENGTH     [MaxNumSpecChan][NumOutBeam] <_DOUBLE>     [m]
               WAVELENGTHERR  [MaxNumSpecChan][NumOutBeam] <_DOUBLE>     [m]
               CHANWIDTH      [MaxNumSpecChan][NumOutBeam] <_DOUBLE>     [m]
               CHANWIDTHERR   [MaxNumSpecChan][NumOutBeam] <_DOUBLE>     [m]
               FRINGEMOD      [MaxNumBaseline][NumOutBeam] <_INTEGER>
          TRIPLE
               NUMTRIPLE       <_INTEGER>
               OUTPUTBEAM     [     3][     1] <_INTEGER>          For each baseline, which output beam is it from?
               BASELINE       [     3][     1] <_INTEGER>          For each baseline, which baseline in that output beam is it?
```

```
          SPECCHAN       [    32][     3][     1] <_INTEGER>  For each baseline, which channel number of these output beams
          NUMSPECCHAN    [     1] <_INTEGER>                  How many spectral channels in this triple?
SCANDATA        <TABLE>
     NUMSCAN         <_INTEGER>
     SCANID         [NumScan] <_INTEGER>                                    Unique observed
     STARID         [NumScan] <_CHAR*7>                                     e.g. BSC1948
     SCANTIME       [NumScan] <_DOUBLE>                                     [s] from 0UT of date
     OUTPUTBEAM     [NumOutBeam] <EXTCOLUMN>
          VISSQ          [NumSpecChan(beam)][NumBaseline(beam)][NumScan] <_REAL>
          VISSQERR       [NumSpecChan(beam)][NumBaseline(beam)][NumScan] <_REAL>
          VISSQC         [NumSpecChan(beam)][NumBaseline(beam)][NumScan] <_REAL>
          VISSQCERR      [NumSpecChan(beam)][NumBaseline(beam)][NumScan] <_REAL>
          DELAYJITTER    [NumBaseline(beam)][NumScan] <_REAL>             [m]
          DELAYJITTERERR [NumBaseline(beam)][NumScan] <_REAL>             [m]
          PHOTONRATE     [NumSpecChan(beam)][NumScan] <_REAL>             per coherent integration
          PHOTONRATEERR  [NumSpecChan(beam)][NumScan] <_REAL>
          BACKGNDRATE    [NumSpecChan(beam)][NumScan] <_REAL>             per coherent integration
          BACKGNDERR     [NumSpecChan(beam)][NumScan] <_REAL>
     TRIPLE         [NumTriple] <EXTCOLUMN>
          COMPLTRIPLE    [     2][NumTripleChan(triple)][NumScan] <_REAL>
          COMPLTRIPLEERR [     2][NumTripleChan(triple)][NumScan] <_REAL>
          TRIPLEAMP      [NumTripleChan(triple)][NumScan] <_REAL>
          TRIPLEAMPERR   [NumTripleChan(triple)][NumScan] <_REAL>
          TRIPLEPHASE    [NumTripleChan(triple)][NumScan] <_REAL>
          TRIPLEPHASEERR [NumTripleChan(triple)][NumScan] <_REAL>
          TRIPLEAMPC     [NumTripleChan(triple)][NumScan] <_REAL>
          TRIPLEAMPCERR  [NumTripleChan(triple)][NumScan] <_REAL>
          TRIPLEPHASEC   [NumTripleChan(triple)][NumScan] <_REAL>
          TRIPLEPHASECERR[NumTripleChan(triple)][NumScan] <_REAL>
     INPUTBEAM      [NumSid] <EXTCOLUMN>
          FDLPOS         [NumScan] <_DOUBLE>                              [m], relative to ReferenceStation
          FDLPOSERR      [NumScan] <_REAL>
          GRPDELAY       [NumScan] <_DOUBLE>                              [m], including group delay
          GRPDELAYERR    [NumScan] <_REAL>
          DRYDELAY       [NumScan] <_DOUBLE>                              [m], dry air dispersion corrected
          DRYDELAYERR    [NumScan] <_REAL>
          WETDELAY       [NumScan] <_DOUBLE>                              [m], wet air dispersion corrected
          WETDELAYERR    [NumScan] <_REAL>
          NATJITTER      [NumScan] <_REAL>
          NATJITTERERR   [NumScan] <_REAL>
```

Appendix D

COBRA file structure

The HDS file format of raw data is described here.

```
SESSION  <SESSION>
     DATE  <_CHAR*10>
     SYSTEMID  <_CHAR*4>
     FORMAT  <_CHAR*11>
     GENCONFIG  <>
          INSTRCOHINT  <_DOUBLE>
          BEAMCOMBINERID  <_INTEGER>
          REFSTATION  <_INTEGER>
          INPUTBEAM  <TABLE>
               NUMSID  <_INTEGER>
               STATIONID [     3] <_CHAR*16>
               SIDEROSTATID [     3] <_INTEGER>
               DELAYLINEID [     3] <_INTEGER>
               BCINPUTID [     3] <_INTEGER>
               STARTRACKERID [     3] <_INTEGER>
               STATIONCOORD [     4][     3] <_DOUBLE>
          OUTPUTBEAM  <TABLE>
               NUMOUTBEAM  <_INTEGER>
               SPECTROMETERID [     3] <_CHAR*16>
               NUMSPECCHAN [     3] <_INTEGER>
               NUMBASELINE [     3] <_INTEGER>
               FRINGEMOD [     1][     3] <_INTEGER>
               BASELINEID [     1][     3] <_CHAR*256>
               WAVELENGTH [    32][     3] <_DOUBLE>
               WAVELENGTHERR [    32][     3] <_DOUBLE>
               CHANWIDTH [    32][     3] <_DOUBLE>
               CHANWIDTHERR [    32][     3] <_DOUBLE>
          TRIPLE  <TABLE>
               NUMTRIPLE  <_INTEGER>
               OUTPUTBEAM [     3][     1] <_INTEGER>
               BASELINE [     3][     1] <_INTEGER>
               NUMSPECCHAN [     1] <_INTEGER>
               SPECCHAN [    32][     3][     1] <_INTEGER>
     GEOPARMS  <>
          LATITUDE  <_DOUBLE>
          LONGITUDE  <_DOUBLE>
          ALTITUDE  <_DOUBLE>
          EARTHRADIUS  <_DOUBLE>
          J2  <_DOUBLE>
     SCANDATA  <>
          NUMSCAN  <_INTEGER>
          SCANID [     1] <_INTEGER>
          NUMREC [     1] <_INTEGER>
          STARID [     1] <_CHAR*12>
```

```
STARTTIME [     1] <_DOUBLE>
STOPTIME [     1] <_DOUBLE>
RAWDATA [     1] <TABLE>
     TIME [ 42200] <_DOUBLE>
     OUTPUTBEAM [     3] <>
          SOFTDELAY [     1][ 42200] <_REAL>
          BINCOUNTS [     8][    32][ 42200] <_BYTE>
     INPUTBEAM [     3] <>
          FDLDELAY [ 42200] <_DOUBLE>
          QUAD [     4][ 42200] <_REAL>
```

Appendix E

INCHWORM file structure

The HDS file format of INCHWORM data is described here, as of the last recorded revision September 28, 1995.

Note: All the metrology subsystems have not been included under the Motion HDS object yet. It will not be necessary to have all the metrology subsystem data under the same cell of the Motion HDS object; some cells will have solutions from different subsystems (done in different ways), that can be combined in the INCHWORM software.

```
Session
    Format <_char*>     Format in the constrictor file
    FormatInch <_char*> Format in the inchworm file
    Date <_char*>       Date in the constrictor file
    DateInch <_char*>   Date the inchworm file was created
    SystemID <_char*>
    GenConfig
        NumLaserP2P <_integer>
        P2PLaunchPlate [NumLaserP2P] <_integer>
        P2PRetroPlate[NumLaserP2P] <_integer>
        MasterPlateID <_char*>
        NumPlate <_integer>
        Plate [NumPlate]
            NumCluster <_integer>
            PlateEmbedded <_integer>
            PlateID <_char*>
            PlateLoc [3] <_double>
            PlateLocErr [3] <_double>
    SidMetConfig [NumPlate]
        NumLaser <_integer>
        CountsPerWaveln <_integer>
        LaserWavelength <_double>
        SampleInterval <_integer>
        IFBox [NumLaser] <_integer>
        Channel [NumLaser] <_integer>
        Theta [NumLaser] <_integer>
        ThetaErr [NumLaser] <_integer>
        Phi [NumLaser] <_integer>
        PhiErr [NumLaser] <_integer>
        LaunchInfo [NumLaser]
            NumGlass <_integer>
            NumAirGap <_integer>
            Loc [3] <_double>
            LocErr [3] <_double>
            GlassThick [NumGlass] <_double>
```

```
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
OptAnchConfig [NumPlate][MaxNumCluster]
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
    Theta [NumLaser] <_integer>
    ThetaErr [NumLaser] <_integer>
    Phi [NumLaser] <_integer>
    PhiErr [NumLaser] <_integer>
    LaunchInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
Pier2PierConfig
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
    Theta [NumLaser] <_integer>
    ThetaErr [NumLaser] <_integer>
    Phi [NumLaser] <_integer>
    PhiErr [NumLaser] <_integer>
    LaunchInfo [NumLaser]
```

```
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
ExtCatEyeConfig [NumPlate]
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
    Theta [NumLaser] <_integer>
    ThetaErr [NumLaser] <_integer>
    Phi [NumLaser] <_integer>
    PhiErr [NumLaser] <_integer>
    LaunchInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
    RetroInfo [NumLaser]
        NumGlass <_integer>
        NumAirGap <_integer>
        Loc [3] <_double>
        LocErr [3] <_double>
        GlassThick [NumGlass] <_double>
        GlassThickErr [NumGlass] <_double>
        GlassCode [NumGlass] <_integer>
        ExFrac [NumGlass] <_double>
        ExFracErr [NumGlass] <_double>
        AirGapThick [NumAirGap] <_double>
        AirGapThickErr [NumAirGap] <_double>
PlateExpConfig [NumPlate]
    NumLaser <_integer>
    CountsPerWaveln <_integer>
    LaserWavelength <_double>
    SampleInterval <_integer>
    IFBox [NumLaser] <_integer>
    Channel [NumLaser] <_integer>
```

```
        Theta [NumLaser] <_integer>
        ThetaErr [NumLaser] <_integer>
        Phi [NumLaser] <_integer>
        PhiErr [NumLaser] <_integer>
        LaunchInfo [NumLaser]
            NumGlass <_integer>
            NumAirGap <_integer>
            Loc [3] <_double>
            LocErr [3] <_double>
            GlassThick [NumGlass] <_double>
            GlassThickErr [NumGlass] <_double>
            GlassCode [NumGlass] <_integer>
            ExFrac [NumGlass] <_double>
            ExFracErr [NumGlass] <_double>
            AirGapThick [NumAirGap] <_double>
            AirGapThickErr [NumAirGap] <_double>
        RetroInfo [NumLaser]
            NumGlass <_integer>
            NumAirGap <_integer>
            Loc [3] <_double>
            LocErr [3] <_double>
            GlassThick [NumGlass] <_double>
            GlassThickErr [NumGlass] <_double>
            GlassCode [NumGlass] <_integer>
            ExFrac [NumGlass] <_double>
            ExFracErr [NumGlass] <_double>
            AirGapThick [NumAirGap] <_double>
            AirGapThickErr [NumAirGap] <_double>
    MetAirTempConf [NumPlate]
        NumSensor <_integer>
        SampleInterval <_integer>
        Chain [NumSensor] <_integer>
        BRAD [NumSensor] <_integer>
        Offset [NumSensor] <_double>
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
        LocErr  [NumSensor][3] <_double>
    MetSolidTmpConf [NumPlate]
        NumSensor <_integer>
        SampleInterval <_integer>
        Chain [NumSensor] <_integer>
        BRAD [NumSensor] <_integer>
        Offset [NumSensor] <_double>
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
        LocErr  [NumSensor][3] <_double>
    MetPressConf [NumPlate]
        NumSensor <_integer>
        SampleInterval <_integer>
        Chain [NumSensor] <_integer>
        BRAD [NumSensor] <_integer>
        Offset [NumSensor] <_double>
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
        LocErr  [NumSensor][3] <_double>
    MetHumConf [NumPlate]
```

```
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    CrossEnv [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
LabAirTempConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    CrossEnv [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
LabSolidTmpConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    CrossEnv [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
LabPressConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    CrossEnv [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
LabHumConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    CrossEnv [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
DLPressConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
```

```
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
        LocErr  [NumSensor][3] <_double>
    FBPressConf
        NumSensor <_integer>
        SampleInterval <_integer>
        Chain [NumSensor] <_integer>
        BRAD [NumSensor] <_integer>
        Offset [NumSensor] <_double>
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
        LocErr  [NumSensor][3] <_double>
    WxAirTempConf
        NumSensor <_integer>
        SampleInterval <_integer>
        Chain [NumSensor] <_integer>
        BRAD [NumSensor] <_integer>
        Offset [NumSensor] <_double>
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
        LocErr  [NumSensor][3] <_double>
    WxPressConf
        NumSensor <_integer>
        SampleInterval <_integer>
        Chain [NumSensor] <_integer>
        BRAD [NumSensor] <_integer>
        Offset [NumSensor] <_double>
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
        LocErr  [NumSensor][3] <_double>
    WxHumConf
        NumSensor <_integer>
        SampleInterval <_integer>
        Chain [NumSensor] <_integer>
        BRAD [NumSensor] <_integer>
        Offset [NumSensor] <_double>
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
        LocErr  [NumSensor][3] <_double>
    WindSpeedConf
        NumSensor <_integer>
        SampleInterval <_integer>
        Chain [NumSensor] <_integer>
        BRAD [NumSensor] <_integer>
        Offset [NumSensor] <_double>
        OffsetErr [NumSensor] <_double>
        Scale [NumSensor] <_double>
        ScaleErr [NumSensor] <_double>
        CrossEnv [NumSensor] <_double>
        Loc [NumSensor][3] <_double>
```

```
    LocErr  [NumSensor][3] <_double>
WindDirConf
    NumSensor <_integer>
    SampleInterval <_integer>
    Chain [NumSensor] <_integer>
    BRAD [NumSensor] <_integer>
    Offset [NumSensor] <_double>
    OffsetErr [NumSensor] <_double>
    Scale [NumSensor] <_double>
    ScaleErr [NumSensor] <_double>
    CrossEnv [NumSensor] <_double>
    Loc [NumSensor][3] <_double>
    LocErr  [NumSensor][3] <_double>
SysLog <_char*>
ConstrictorLog <_char*>
ScanData <Table>
    NumScan <_integer>
    ScanID [NumScan] <_integer>
    StartTime [NumScan] <_double>
    StopTime [NumScan] <_double>
InchwormLog <_char*>
NumMap <_integer>
Map [NumMap]
    SidMetMap [NumPlate]
        NumLaser <_integer>
        LaunchEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
        RetroEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
    ExtCatEyeMap [NumPlate]
        NumLaser <_integer>
        LaunchEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
        RetroEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
    PlateExpMap [NumPlate]
        NumLaser <_integer>
        LaunchEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
        RetroEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
    OptAnchMap [NumPlate][NumClusterMax]
        NumLaser <_integer>
        LaunchEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
```

```
            MetHumiditySen <_integer>
        RetroEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
    Pier2PierMap
        NumLaser <_integer>
        LaunchEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
        RetroEnd [NumLaser]
            MetAirTempSen <_integer>
            MetSolidTmpSen <_integer>
            MetPressureSen <_integer>
            MetHumiditySen <_integer>
NumAveGroup <_integer>
AveGroup[NumAveGroup]
    NumAveIn <_integer>
    NumAveOut <_integer>
    TimeAveIn <_integer>
    TimeAveOut <_integer>
    NumTime <_integer>
    Time [NumTime] <_double>
    SidMetData [NumPlate][NumLaserMax]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    ExtCatEyeData [NumPlate][NumLaserMax]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    PlateExpData [NumPlate][NumLaserMax]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    MetAirTempData [NumPlate][NumSensorMax]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    MetSolidTmpData [NumPlate][NumSensorMax]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    MetPressData [NumPlate][NumSensorMax]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    MetHumData [NumPlate][NumSensorMax]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    OptAnchData [NumPlate][NumClusterMax][NumLaserMax]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    Pier2PierData [NumLaser]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    LabAirTempData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
```

```
        DataErr [NumTime] <_double>
    LabSolidTmpData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    LabPressData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    LabHumData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    DLPressData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    FBPressData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    WxAirTempData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    WxPressData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    WxHumData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    WindSpeedData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    WindDirData [NumSensor]
        NumData [NumTime] <_integer>
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
NumEnvCorrGroup <_integer>
EnvCorrGroup [NumEnvCorrGroup]
    AveGroup <_integer>
    Map <_integer>
    Refs <_char*>
    SidMetData [NumPlate][NumLaserMax]
        EnvSensorFlag [8] <_integer> # 1=air temperature, etc.
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    ExtCatEyeData [NumPlate][NumLaserMax]
        EnvSensorFlag [8] <_integer> # 1=air temperature, etc.
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    PlateExpData [NumPlate][NumLaserMax]
        EnvSensorFlag [8] <_integer> # 1=air temperature, etc.
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    OptAnchData [NumPlate][NumClusterMax][NumLaserMax]
        EnvSensorFlag [8] <_integer> # 1=air temperature, etc.
        Data [NumTime] <_double>
        DataErr [NumTime] <_double>
    Pier2PierData [NumLaser]
        EnvSensorFlag [8] <_integer> # 1=air temperature, etc.
        Data [NumTime] <_double>
```

```
        DataErr [NumTime] <_double>
NumMotionGroup <_integer>
MotionGroup [NumMotionGroup]
    AveGroup <_integer>
    EnvCorrGroup <_integer>
    ZeroFlag <_integer>
    Mode <_integer>
    PlateList [NumPlateList] <_integer>
    ClusterList [NumPlateList][NumClusterList] <_integer>
    LaserListSid [NumPlateList][NumLaserList] <_integer>
    LaserListOA [NumPlateList][NumClusterList][NumLaserList] <_integer>
    LaserListP2P [NumLaserList] <_integer>
    Siderostat [NumPlate]
        NumPar <_integer>   # NumPar = 3
        Par [NumPar] <_char*>
        NumCorr <_integer>  # NumCorr = NumPar(NumPar-1)/2
        Corr [NumCorr] <_char*>
        ReducedChi2 [NumTime] <_double>
        XCorr [NumTime][NumCorr] <_double>
        NumSingVal [NumTime] <_integer>
        SingValFlag [NumTime][NumPar] <_integer>
        WRatio [NumTime][NumPar] <_double>
        Par [NumTime][NumPar] <_double>
        ParFitErr [NumTime][NumPar] <_double>
        ParThErr [NumTime][NumPar] <_double>
        MotCorrFlags [NumFlag] <_char*>
    OpticalAnchor [NumPlate]
        NumPar <_integer>   # NumPar = 6
        Par [NumPar] <_char*>
        NumCorr <_integer>  # NumCorr = NumPar(NumPar-1)/2
        Corr [NumCorr] <_char*>
        ReducedChi2 [NumTime] <_double>
        XCorr [NumTime][NumCorr] <_double>
        NumSingVal [NumTime] <_integer>
        SingValFlag [NumTime][NumPar] <_integer>
        WRatio [NumTime][NumPar] <_double>
        Par [NumTime][NumPar] <_double>
        ParFitErr [NumTime][NumPar] <_double>
        ParThErr [NumTime][NumPar] <_double>
        MotCorrFlags [NumFlag] <_char*>
    Pier2Pier
        NumPar <_integer>   # NumPar = 3(NumPlate-1)
        Par [NumPar] <_char*>
        NumCorr <_integer>  # NumCorr = NumPar(NumPar-1)/2
        Corr [NumCorr] <_char*>
        ReducedChi2 [NumTime] <_double>
        XCorr [NumTime][NumCorr] <_double>
        NumSingVal [NumTime] <_integer>
        SingValFlag [NumTime][NumPar] <_integer>
        WRatio [NumTime][NumPar] <_double>
        Par [NumTime][NumPar] <_double>
        ParFitErr [NumTime][NumPar] <_double>
        ParThErr [NumTime][NumPar] <_double>
        MotCorrFlags [NumFlag] <_char*>
    AllBedrock
        NumPar <_integer>   # NumPar = 6NumPlate
        Par [NumPar] <_char*>
        NumCorr <_integer>  # NumCorr = NumPar(NumPar-1)/2
        Corr [NumCorr] <_char*>
        ReducedChi2 [NumTime] <_double>
        XCorr [NumTime][NumCorr] <_double>
        NumSingVal [NumTime] <_integer>
        SingValFlag [NumTime][NumPar] <_integer>
        WRatio [NumTime][NumPar] <_double>
        Par [NumTime][NumPar] <_double>
```

```
ParFitErr [NumTime][NumPar] <_double>
ParThErr [NumTime][NumPar] <_double>
```