Mustang Obit User Documentation

Obit: Merx mollis mortibus nuper

Draft version: 0.1 June 13, 2008

Abstract

This document describes the user interface to the Obit software for analyzing data from the Mustang (GBT 3mm bolometer) camera and the currently established techniques for such analysis. The Obit package provides a python based interactive, scripting and compiled task interface to C language libraries. General information about the Obit package can be obtained from http://www.cv.nrao.edu/∼bcotton/Obit.html. A detailed script for processing Mustang data along with extensive annotation is given.

Contents

1.1 Introduction . . . 3
1.2 Obtaining Software . . . 3
1.3 Starting ObitTalk . . . 4
1.4 Quick Guide to ObitTalk . . . 5
1.5 Calibration Approach . . . 6
1.6 OTF Imaging . . . 7
1.7 Using CLEAN to Derive Sky Model . . . 8
1.8 OTF Data Format . . . 9
1.9 Examining OTF Data Using fv . . . 10
1.10 Mustang Data Analysis . . . 10
1.10.1 Calibration and Background Estimation . . . 13
1.10.2 Reading Data . . . 14
1.10.3 Pointing Calibration . . . 15
1.10.4 Annotated Processing Script . . . 15
1.10.5 Modifying the Script . . . 32
1.11 Major Obit Single Dish Routines . . . 34
1.11.1 GBTUtil.UpdateOTF . . . 34
1.11.2 GBTUtil.GetTargetPos . . . 35
1.11.3 OTF.ClearCal . . . 35
1.11.4 OTFGetSoln.POTFGetDummyCal . . . 35
1.11.5 OTFGetSoln.POTFGetSolnPARGain . . . 35
1.11.6 OTF.Soln2Cal . . . 36
1.11.7 OTFGetSoln.PFilter . . . 36
1.11.8 OTFGetSoln.PMBBase . . . 36
1.11.9 OTF.makeImage . . . 37
1.11.10 OTF.ResidCal . . . 38
1.11.11 OTF.SelfCal . . . 39
1.11.12 CleanOTF.PClean . . . 39
1.11.13 PARCal.CleanSkyModel . . . 40
1.11.14 PARCal.FitCal . . . 41
1.11.15 PARCal.InitCal . . . 43
1.11.16 PARCal.PlotData . . . 44

1.1 Introduction

The Obit package provides a flexible environment for developing astronomical data analysis techniques but also provides sufficient efficiency to effectively apply these techniques to real astronomical applications. This is achieved by a python interface to C language libraries. This interface provides interactive, scripting and compiled task capabilities. Obit includes a package for analyzing single dish "On-the-Fly" (OTF) imaging data such as that from the Mustang (GBT 3mm bolometer) camera.

The techniques for using the Obit software, as well as the software itself, are still undergoing development with the increasing experience with the instrument. The Mustang data analysis is currently largely implemented by means of python scripts which can be easily modified but which requires some detailed knowledge. When the techniques become well established, they will be converted into a compiled task given a set of parameters.

The following sections tell how to obtain the Obit software and a simplified guide to the python user interface (ObitTalk). Subsequent sections describe the general calibration and imaging techniques employed, a description of the data format as well as a discussion of the current analysis script. In general, Obit allows multiple native data formats and both FITS and AIPS image and tables formats are supported. However, there is no suitable AIPS equivalent of the OTF format so only a FITS version is supported. More information about Obit can be obtained from http://www.cv.nrao.edu/∼bcotton/Obit.html. Documents of particular relevance are

- Preprint of paper describing the Obit package (PASP 2008, **120**, 439)
  ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/Obit.pdf
- ObitTalk (python interface) User document
  ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/ObitTalk.pdf
- ObitTalk software architecture
  document ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/ObitTalkSoft.pdf
- Obit C language software architecture document
  ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/OBITdoc.pdf
- Obit Single disk OTF architecture document
  ftp://ftp.cv.nrao.edu/NRAO-staff/bcotton/Obit/ObitSD.pdf
- Obit C language documentation
  http://www.cv.nrao.edu/ bcotton/Obit/ObitDoxygen/html/index.html
- ObitSD C language documentation
  http://www.cv.nrao.edu/ bcotton/Obit/ObitSDDoxygen/html/index.html

1.2 Obtaining Software

Obit and related software is available from http://www.cv.nrao.edu/∼bcotton/Obit.html. At present there is no system of stable releases so using the anonymous Subversion (SVN) interface is recommended. Obit depends heavily on third party software which is described on this page. Support of the Obit package is limited. The components of the Obit/ObitTalk package are:

- Obit
  Basic Obit package and the support for images and radio interferometry.

- ObitSD
  Obit “On The Fly” (OTF) single dish imaging package.
- ObitView
  Image display used by Obit.
- ObitTalk
  Scripting and interactive interface to Obit software.

These software packages come with installation instructions and config scripts to build them. Of particular interest is a tarball distribution of Obit which includes the third party software and an installation script. This is available from the Obit home page.

1.3 Starting ObitTalk

The interactive and scripting python interface is ObitTalk which is a python interpreter with basic Obit packages preloaded. For using “standard” installations of Obit on NRAO machines setting the OBIT, OBITSD and PYTHONPATH environment variables as described in the following is unnecessary. For an installation using the Obit installation tarball, use the setup.sh or setup.csh scripts generated by the installation script in the root directory of the Obit installation to set these variables.

Before starting ObitTalk, the environment variables OBIT, OBITSD and PYTHONPATH may need to be set. The variables OBIT and OBITSD tells ObitTalk where to find the Obit task parameter definition (TDF) files and executables and should point to the base of the Obit and ObitSD directory trees. In particular, $OBITSD/TDF should point to the directory with the TDF files and $OBITSD/bin/ should give the directory with the executables. In tcsh, if Obit is installed in your home directory in subdirectory Obit you could use:

```
% setenv OBIT "$HOME/Obit"
```

Set PYTHONPATH to the standard Obit python directories and any directories in which you want to put your own python modules. For example, using tcsh and setting PYTHONPATH to use both ObitSD and Obit modules:

```
% setenv PYTHONPATH "$OBITSD/python":"$OBIT/python"
```

Note the colon separating the directory names.

If you wish to use the ObitView image display, you can start it before ObitTalk. If ObitView is in your path:

```
% ObitView &
```

will start the display server. If this fails to start the display, see the discussion of ObitView in the ObitTalk User Manual.

Then, if the script ObitTalk is in your path:

```
% ObitTalk [scriptname]
```

should start ObitTalk. If the environment variable AIPS_ROOT is defined, ObitTalk will make AIPS tasks and data available. If the optional script name is given, then the python interpreter will do some simple AIPS initialization and execute the python script “scriptname”. If no script is specified then ObitTalk will ask for your AIPS number and do its AIPS initialization (if AIPS is available) and go into an interactive python session. The python prompts are:

```
>>>
```

1.4 Quick Guide to ObitTalk

Python is a basically "Object–oriented" language and much of the ObitTalk software follows this methodology. "Object–oriented" in this context means little more than variables are more substantial than that floats and strings and simple arrays (although these also exist). A python (hence ObitTalk) variable is a relatively arbitrary thing and can be a scalar number, string, an array or list of variables or the interface to a dataset such as an image or OTF data. In addition, functions which operate on the object can be attached to the object.

In ObitTalk, the interface to a data set is assigned to a variable and this variable is used to specify operations on that dataset. Similarly, the interface to a task (compiled program executed outside of the python interpreter) is an object with parameters and functions as members.

The usual object–oriented syntax is that "class methods" (functions which can operate on an object) are invoked like this:

```
>>> object.function(arguments)
```

where "object" is the python object, "function" is the function name and "arguments" are the additional arguments, the object is implicitly an argument, by convention called "self" in python. In python documentation of function interfaces, "self" appears as the first argument of the function although it is invoked as shown above. Note: many Obit functions have objects as explicit arguments, these use the form:

```
>>> module.function(object, other_arguments)
```

where module is the name of the module defining function.

Before a module can be used, it must first be imported into python. This is done using the python "import" command:

```
>>> import OTF
```

to import the basic OTF package. The documentation on that package can then be obtained:

```
>>> help(OTF)
```

for the entire package or

```
>>> help(OTF.AtmCal)
```

for a non class function or for a class function:

```
otf=OTF.OTF("otf")  # An object of type OTF
>>> help(otf.Open)
```

The following modules are of interest to single dish imaging:

- **OTF -** OTF ("On the Fly") data
- **OTFDesc -** OTF data descriptor (header)
- **OTFUtil -** OTF Utilities
- **OTFRec -** OTF data record class
- **OTFGetAtmCor -** OTF Atmospheric correction utilities
- **OTFGetSoln -** OTF calibration solution utilities
- **OTFSoln2Cal -** Utilities to convert OTF solution to calibration tables
- **GBTUtil -** GBT OTF Utilities

- **CCBUtil -** GBT CCB utility package
- **CleanOTF -** Single dish (Hogbom) CLEAN
- **GBTDCROTF -** Convert GBD DCR data to OTF format
- **PARCal -** Mustang calibration/editing utilities
- **History -** History class
- **Image -** Image class
- **ImageDesc -** Image Descriptor (header)
- **ImageMosaic -** Image Mosaic class
- **ImageUtil -** Image utilities
- **InfoList -** Obit associative array for control info
- **ODisplay -** Interface to ObitView display
- **OErr -** Obit message/error class
- **OPlot -** pgplot interface
- **TableDesc -** Table descriptor (header) class
- **TableList -** Table list for data object (Image, UVData, OTF)
- **Table -** Table class
- **TableUtil -** Table utilities
- **History -** History class

1.5 Calibration Approach

The principle difficulty with imaging short wavelength radio continuum single dish data is the large and variable background signal. This background comes from the atmosphere, the telescope and the instrument itself and can vary on a wide variety of timescales. The offset of the measured signals from zero is particularly difficult to determine and generally must be determined from some knowledge of the (astronomically interesting) sky. The basic approach in Obit Single Dish software (ObitSD) is to iteratively model the background signals, image the sky, subtract a model of the astronomical sky from the data and reestimate the backgrounds. The estimate of the background is determined from the residual data (sky model subtracted) by a low pass filtering. As the quality of the model improves, higher temporal frequency components of the residuals are included. This technique uses the redundancy in the data to separate the constant astronomical sky from the variable background signals. The zero level is set essentially by the constraint that the sky brightness in some regions imaged are zero. This technique works best if the telescope beams can be swept over the field of view sufficiently fast that the modulation of the signal due to the beam sweeping over the astronomical sky is faster than the time scale of the variations of the background signals.

The redundancy in the data are of two forms. The first is that each resolution element in the image is covered multiple times by the instrument, hopefully along different trajectories on the sky

as in the “basket–weaving” technique. These multiple observations of each piece of the sky help separate the constant portion of the signal from the time variable component.

For instruments with multiple pixels, such as the Mustang bolometer array, there is additional information. The contributions of the atmosphere will be essentially the same in all detectors, i.e. a common mode signal, as the various beams strongly overlap through the regions of the atmosphere in which the bulk of the brightness variations occur. In the case of Mustang, much of the background variation from the instrument itself (e.g. the 1.4 Hz variations due to the refrigerator pump) are also common mode. For sources smaller than the detector array, these common mode backgrounds are easily distinguished from the source signals which appear in only a few pixels.

The implementation of the ObitSD calibration is an iterative one. The OTF data is kept in a single FITS file and the calibration is manipulated with tables of a pair of types; a “solution” (OTFSoln) table and a total “calibration” (OTFCal) table. The structures of these tables are identical and contain a number of multiplicative and additive (both per detector and common) and other corrections. The usage of the two types is different. A “solution” table is an incremental set of calibrations, usually derived from data after the application of a total calibration table. A new total calibration table is then derived by the application of the solution table to the previous calibration table. This new total calibration can then be applied to the data. In addition to the calibration tables, there is a “flagging” table (OTFFlag) which can be used to describe data to be ignored (actually given zero weight). More details of these tables are described in section 1.8.

1.6 OTF Imaging

In the “On-the-Fly” imaging technique, the telescope is swept across the field of view in a pattern that will cover all of the field to be imaged while sampling data at a constant rate. The data sampling should be fast compared the the time it takes the beam to move its own width or the sky will be smeared out. In this mode, the sky is sampled at uniform times but at positions not constrained to the pixels on a well defined grid. As this is the case, the actual position on the sky of each detector must be accurately known at all times.

The conversion of the “randomly” sampled data to a regular grid proceeds in a number of steps collectively called “griding”.

1. “Convolution” and re-sampling on a grid
   Each sample is considered as a delta function and is multiplied by a continuous “griding” (AKA “convolution”) function. The gridding function is centered on the datum and is sampled on a grid centered on the pixel nearest to the datum and which is of finite support. This results in a grid of data*griding_function as well as a grid of griding_function values.

2. Accumulation onto a grid
   The data times the griding function and the griding function samples are accumulated onto a pair of grids covering the region of the sky to be imaged.

3. Normalization
   When all of the data have been multiplied by the griding function, re sampled and accumulated onto the grids, the image is normalized by dividing the sum of the data*griding_function grid by the sum of the griding_function grid on a pixel-by-pixel basis.

4. deMode [optional]
   If the bulk of the pixels in an image are expected to have no emission, this can be enforced by using the deMode option. The mode of the pixel distribution is determined, this is presumed

to be the actual background level which should be zero, and the value of the mode is subtracted from all pixels in the image.

In general, not all data are of the same quality, e.g. varying sensitivity among detectors, and each datum may be assigned a statistical weight. These weights are included in the process described above by replacing the griding function with the product of the griding function times the weight of the datum being gridded.

In the griding procedure, the griding function plays a critical role and the function chosen is generally a compromise. One of the compromises is sensitivity against resolution. Using a "fatter" griding function will include more data in each pixel and thus give lower noise at the cost of reduced resolution.

Obit currently provides the following griding functions as separable functions on a rectangular grid, generally the default parameter values are adequate.

- **Pillbox**
  The pillbox function is 1 inside the area of the pixel and 0 outside. The following parameters are allowed:
  - Parm[0] = half-width in cells of support [def 0.5]
  - Parm[1] = Expansion factor
- **Gaussian**
  This is a Gaussian centered on the central image grid cell. The following parameters are allowed:
  - Parm[0] = half-width in cells of support [def 3.0]
  - Parm[1] = Gaussian with as fraction of raw beam [def 1.0]
- **Exponential*Sinc**
  The following parameters are allowed: This is as exponential times a Sinc (sin x/x) centered on the central image grid cell. The following parameters are allowed:
  - Parm[0] = half-width in cells of support [def 2.0]
  - Parm[1] = 1/sinc factor (cells) [def 1.55]
  - Parm[2] = 1/exp factor (cells) [def 2.52]
  - Parm[3] = exp power [def 2.0]
- **Spherodial wave function**
  This is a prolate spheroidal wave function centered on the central image grid cell. The following parameters are allowed:
  - Parm[0] = half-width in cells of support [def 3.0]
  - Parm[1] = Alpha [def 5.0]

1.7 Using CLEAN to Derive Sky Model

In order to use the method outlined in Section 1.5, a sky model must be derived from the image obtained from the data. The use of a griding function in the image formation in general has a point spread function (psf) larger than the intrinsic resolution of the telescope so is not directly useful.

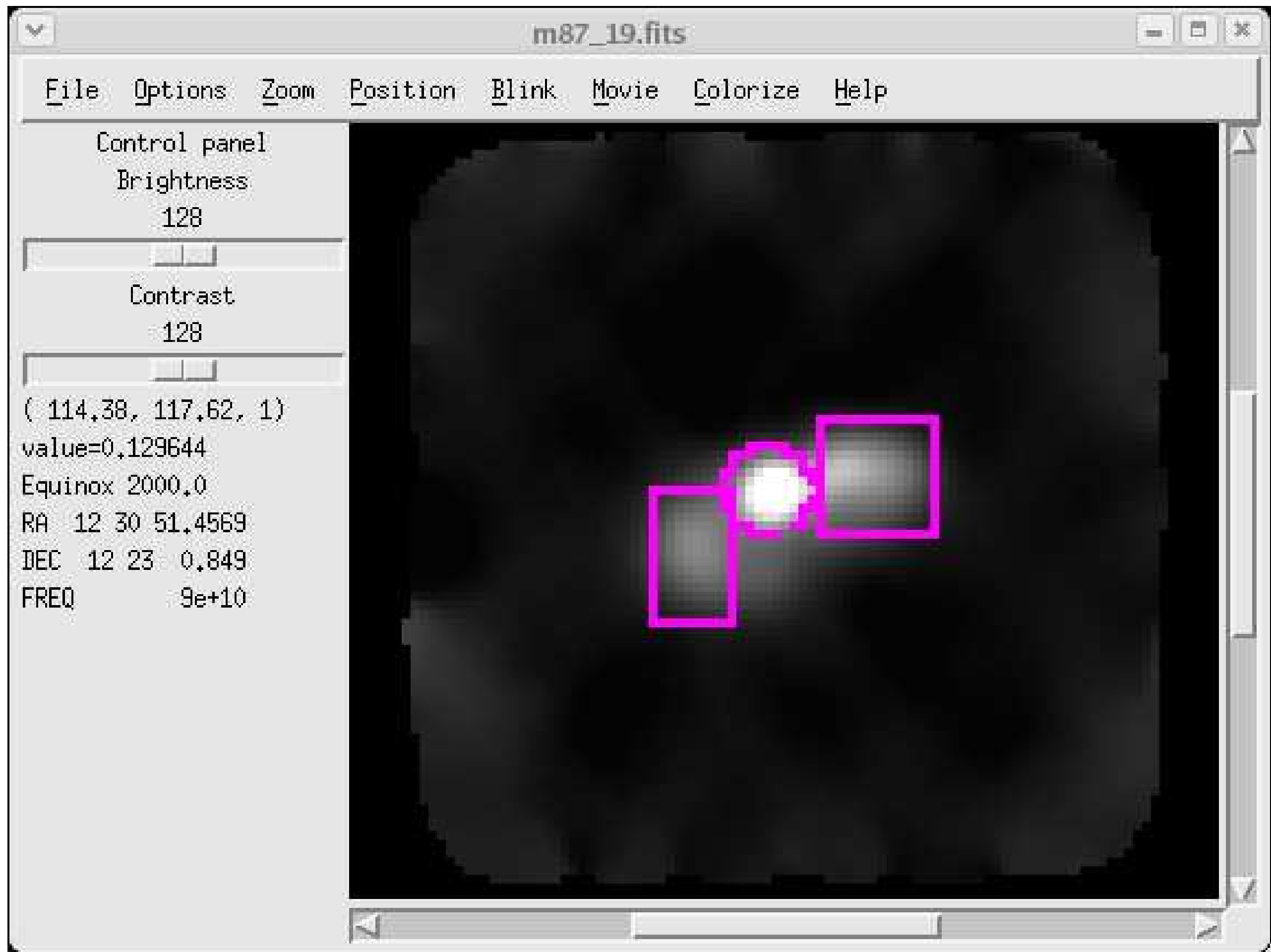

Figure 1.1: Screen-shot of interactive CLEAN window editing using ObitView.

In addition, there are artifacts in the image resulting from imperfect background estimation that should not be included in the sky model.

The sky model used in ObitSD is that obtained from a CLEAN deconvolution of the image derived from the data. This has the advantage that the CLEAN components can be restored with an approximation of the actual telescope beam, meaning that the telescope response can then be determined from a simple interpolation of the CLEAN restored image. In addition, CLEAN windowing can be used to constrain the region in which emission is allowed. In the CLEAN sky model, the residuals are not included which gives the region outside of the CLEAN window zero flux density. CLEAN windows can be either hard-coded into the processing script, set interactively or use Obit's automatic windowing algorithm. A screen-shot of an interactive window setting session using ObitView is given in Figure 1.1

1.8 OTF Data Format

The GBT archives data on a per scan basis with each logical component of the system (e.g antenna controller and receiver) logging its own data into separate FITS files. For use in the ObitSD OTF package, these files are read and converted into a single FITS file in a form similar to a relational database. The data are kept in a single table with antenna pointing positions interpolated from the

Antenna FITS files together with other information. Separate tables contain auxiliary information such as target information, offsets of each detector from the antenna pointing and a scan index. In addition, calibration and editing can be specified as tables which can be applied as the data are read. The standard tables are described in the following:

- OTFScanData
  Raw sky brightness data, pointing and other time variable information. A row in this table corresponds to all data at a given time.
- OTFArrayGeom
  Table giving the geometric offsets of a feed/detector array from the pointing axis of the telescope.
- OTFTarget
  Table of sources or targets. These are referred to in the OTFScanData table as an index into this table.
- OTFIndex
  Scan table [optional] giving start and stop times and row numbers in the OTFScanData table as well as targets etc. This index is used to improve data access times.
- OTFFlag
  Table describing "flagged" data - data to be ignored.
- OTFCal
  Cumulative calibration table. This table gives multiplicative and additive corrections to the raw sky brightness measurements in the OTFScanData table as well as corrections to the nominal telescope pointing direction.
- OTFSoln
  Differential calibration ("Solution") table.

A screen-shot of fv displaying an OTF FITS file with data and calibration tables is shown in Figure 1.2.

1.9 Examining OTF Data Using fv

The plotting facilities in Obit are currently limited (see section 1.11.16 or in ObitTalk, import PARCal; help(PARCal.PlotData)) and the most general way to view Mustang data is using the fv utility from GSFC. A display of a segment of data from a number of detectors is shown in Figure 1.3.

1.10 Mustang Data Analysis

The following describes in some detail the current state of Mustang data analysis in Obit. This is currently in the form of a script which is modified to the details of the processing. This allows both modifying the detailed parameters of the processing and the steps actually involved.

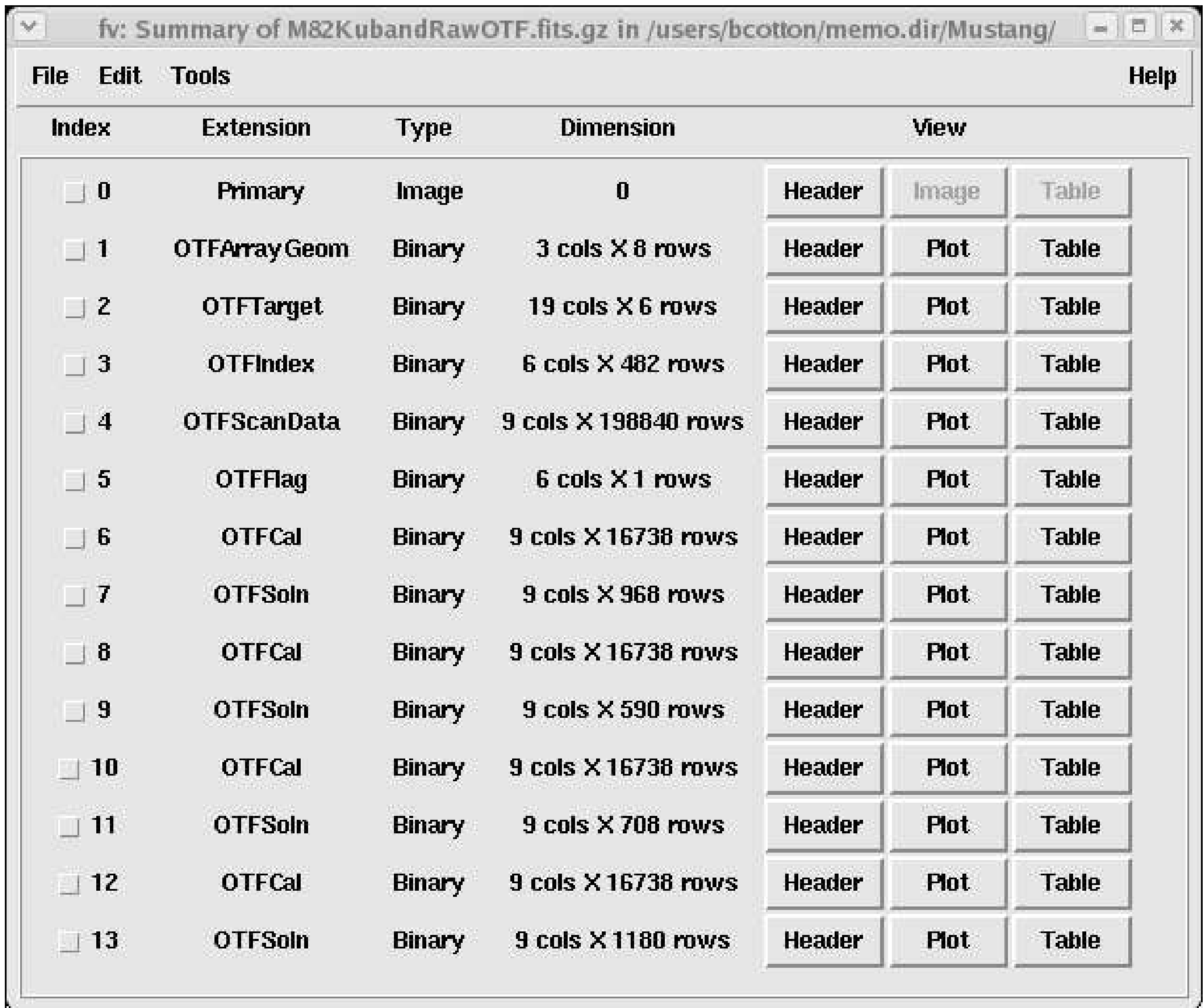

Figure 1.2: Screen-shot of the fv display of an OTF FITS file containing data, calibration and editing tables.

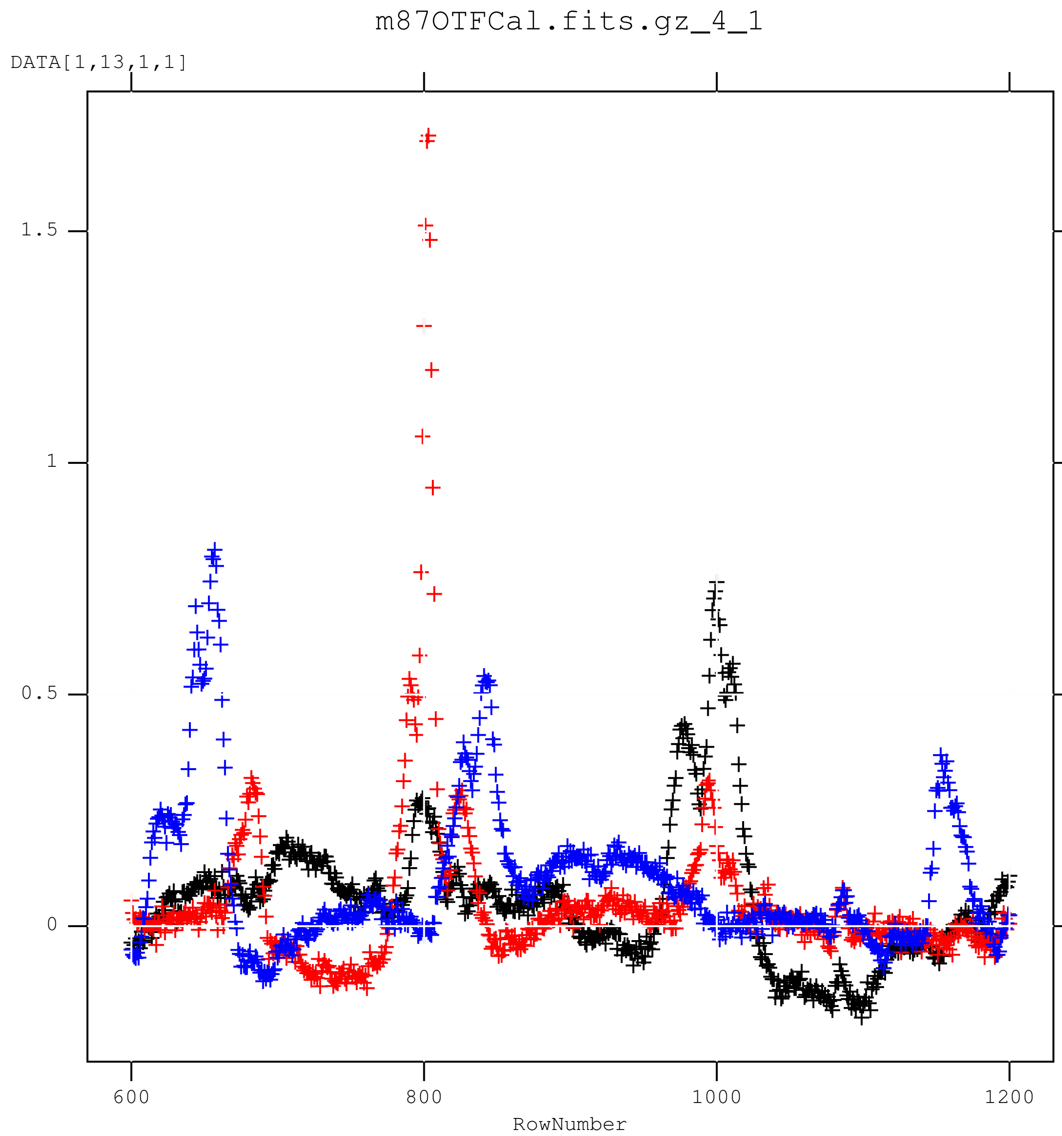

Figure 1.3: fv display of a section of data from several detectors.

1.10.1 Calibration and Background Estimation

Calibration consists of a number of steps allowing the Mustang detector outputs to be converted into Jy, data weights to be estimated, and the background signal levels estimated. This information is entered into "solution" (OTFSoln) tables and accumulated into "calibration" (OTFCal) tables. Individual calibration steps are described in the following; each of these generates an OTFSoln table.

Gain and Weight Calibration

The gain of each detector is determined from scans with a cyclic firing of the Mustang internal calibration source. The function OTFGetSoln.POTFGetSolnPARGain reads through an entire OTF data file and in scans in which the cal signal is switched on and off, determines the difference (strength of the cal). Then this routine calculates the multiplicative terms to convert counts into Jy given the cal strength in Jy. This routine also (optionally) determines the statistical data weights from the RMS in all scans converted into Jy.

The weight calibration removes variations in data, calculates the cal signal deflection and the weight of each detector in each scan from the inverse variance of data. For rach detector, the following is done:

1. Determined cal from averaging each segment and differencing transitions;
2. Corrects data for cal value
3. Fit polynomial (5th order) to data
4. Determine rms deviation
5. Clip points > 5 sigma from 0
6. Refit polynomial
7. Redetermine rms deviation
8. Flag scans with weights with entries further than 10X low or 5X high from the median weight

Baseline Calibration

The "baseline" for each detector in each scan is estimated using OTFGetSoln.PFilter which performs a low pass filter on the data from each detector and estimates the background for each detector from this filtered data stream.

Common Atmosphere/ Detector Offset Calibration

For multi-beam systems such as Mustang, the atmospheric contributions to the background are expected to be largely common among all the detectors. The routine OTFGetSoln.PMBBase fits a polynomial common atmosphere and a single offset per detector per scan.

Automated Editing

Mustang data frequently contains spikes which should be removed from the data stream. This is done using the statistics of each residual data stream (i.e. the data after a source model is subtracted) by a comparison of the RMS of the values in each detector's residual data stream with the detector model data stream (data stream in which the data has been replaced by the model). In each time period of a specified length, the RMS about the mean of each the residual and model data streams are determined for each detector. If a residual RMS exceeds the larger of a fixed value or a multiple of the model RMS, then an entry for that detector is made for that time sample in the OTFFlag table causing subsequent processing to ignore that data segment. The model data stream is used to avoid confusing minor errors arounnd bright sources with bad data.

Residual Calibration

To estimate background fluctuations on short time scales comparable to the time required for the telescope beam to traverse the region of emission, the sky model (CLEAN model) needs to be subtracted from the data. To the degree that the sky model is correct, the residuals will be purely background signals. This subtraction is performed using OTFUtil.PSubImage. The resulting residual data can be converted into background estimates by an appropriate low-pass filtering. The details of the filtering will depend on whether the background signal to be estimated is dominated by components common to all detectors or specific to individual detectors. The residual calibration process is performed by OTF.ResidCal and supports the following filtering types:

- solType="Gain"
  Solve for multiplicative term from "cals" in data for each detector.
- solType="Offset"
  Solve for additive terms from residuals to the model for each detector. The median residual in each solInt for each detector is used as the background estimate.
- solType="GainOffset"
  Solve both gain and offset.
- solType="Filter"
  Additive terms from filtered residuals to the model for each detector. The time series of detector residuals for each detector is low pass filtered to determine the background estimates.
- solType="MultiBeam"
  A common mode background for all detectors is estimated. The center averaged median of the individual detectors in each integration is low pass filtered to remove timescales shorter than solInt to obtain the background estimate.

Since the cal signal is not generally used during Mustang observations, the "Gain" solution types are not applicable; the "MultiBeam" is the most useful for Mustang data.

1.10.2 Reading Data

Before the data can be processed, it must be converted from the form stored in the GBT archive to the form used by the Obit software. This is done using function GBTUtil.UpdateOTF which looks at the ScanLog.fits file in the DataRoot directory (as defined in the relevant script) to see which scans, if any, are not already included in the output OTF data file. These scans are then

read using task PAROTF and appended to the output OTF data file. This allows updates from the GBT archive during the observations to obtain the latest data.

PAROTF performs a number of operations on the archive data. Most important of these is to average the 1 kHz samples to typically 20 Hz. Another operation is to filter the 1.4 oscillations of the detectors due to the pump frequency of the refrigerator. A sample of data before and after this filtering is shown in Figure 1.4.

Time stream filtering:

1. Averaging.
   The data streams are time averaged using boxcar averaging to the requested integration time.

2. Remove jumps in baseline
   Determines a 9 point running median and when there is a persistent jump not associated with a cal state change, then the following data is adjusted by the difference. Multiple jumps may be detected.

3. Filter 1.4 Hz signal from refrigerators
   If the requested averaging time is 50 or 100 msec, then a IIR notch filter is applied to the averaged data. These filters use a third order Bessle bandstop filter between 1.38 and 1.45 Hz. For other averaging times a notch filter using the ObitTimeFilter class is used to filter between 1.39 and 1.44 Hz.

The following script fragment shows how to convert data from the archive to OTF format

```
import GBTUtil
# Root of archive (or copy) data directory
DataRoot="/media/usbdisk/bcotton/FITS/TPAR_17/"

outFile   = "TPAR_19OTF.fits"      # Name of output FITS file
outDisk   = 0                      # 0 means current working directory

# Update OTF with any new scans in archive
outOTF = GBTUtil.UpdateOTF ("PAROTF","Rcvr_PAR",outFile, outDisk, DataRoot, err, \
                           avgTime=0.05)
OErr.printErrMsg(err, "Error creating/updating OTF data object")
```

1.10.3 Pointing Calibration

Pointing offsets can be obtained from daisy scans on calibrators at the correct focus. This can be done using FitCal in python module PARCal.

1.10.4 Annotated Processing Script

The following is a template script for processing Mustang data with a segment-by-segment explanation. This script may be obtained from $OBITSD/share/scripts/scriptMustangTemplate.py. If ObitView is to be used for interactive CLEAN window editing, it should be started from the directory in which the input OTF data to be processed resides. The FITS file PARGaussBeam.fits should also be present in the same directory, this file gives the beamshape assumed for the GBT. This file (in gzip compressed form) is available in $OBITSD/share/data/PARGaussBeam.fits.gz. The processing script should also be executed from the command line in this directory as

```
> ObitTalk script_name
```

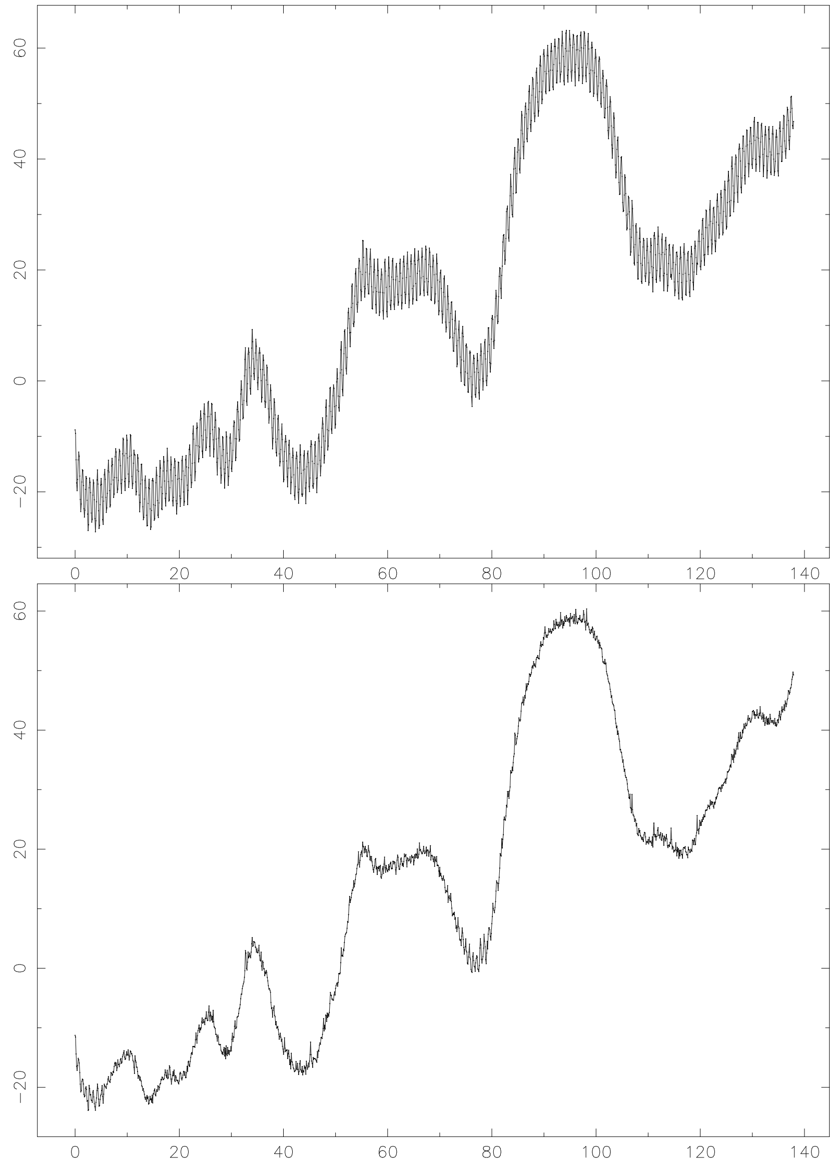

Figure 1.4: The upper plot shows the time sequence of 10 Hz averaged data from a single Mustang detector as a function of time (seconds). The lower plot shows the same data after a 1.4 Hz notch filter.

The details of the script can be edited, however, generally this will only involve the parameters defined in the section described in Section 1.10.4.

The execution of this script will result in a number of output files in directory ./FITSdata whose names are derived from the name of the target ("target" in the following).

- `targetDirty.fits` The final "dirty" image
- `targetClean.fits` The final "CLEAN" image
- `targetWt.fits` The weight image, sum of data weights times griding weights. These are needed to properly combine data from different observations.
- `targetOTFCal.fits` Final calibrated and edited OTF data being imaged.

Once an image has been created, the procedure can be rerun starting with the final result of the previous run. To use this feature, copy the targetClean.fits file to the input data area and re-execute the script.

Obit Initialization

The following imports the Obit modules the script will use and then initializes Obit. Part of this initialization is defining a directory (./FITSdata) in which output, scratch and other temporary files will be written. This directory should exist before executing the script.

```
# Template ObitTalk script for processing Mustang data
import OSystem, OErr, InfoList, Image, Table, TableUtil, History, ODisplay
import OTF, OTFUtil, CleanOTF, OTFGetSoln, OTF
import GBTUtil, FITSDir

# Init Obit
err=OErr.OErr()
FITS = ["./FITSdata"]
ObitSys=OSystem.OSystem  ("Mustang", 1, 1, 1, ["None"], len(FITS), FITS, True, False, err)
OErr.printErrMsg(err, "Error with Obit startup")
```

Defining Parameters

The following specifies the parameters to be used in processing, these are:

- `target` is a list of target names to be imaged as specified in the OTFTarget table. The center of the field imaged will be the position of first target specified as obtained from the OTFTarget table.
- `scans` gives the beginning and end scan number to be considered, note that only targets named in `target` in this range will actually be included.
- `doOffset` if True, solve for offsets on each detector with 3 times the time scale before each common mode ("MultiBeam") residual calibration. If this is false, only the common model calibration is used.
- `deMode` if True then subtract the mode in the pixel distribution of each dirty image as it is formed. This is to help stabilize extended emission and to damp instabilities.

- `inFile` is the name of the input OTF format FITS file.
- `feeds` is a list of detectors to include, an empty list means use all feeds.
- `cells` is the cell spacing of the output images in arcsec
- `nx, ny` are the dimensions of the image in pixels.
- `niter` is the number of iterations of CLEAN
- `gain` is the CLEAN loop gain
- `minFlux` is the minimum abs. flux density to which to CLEAN
- `CLEANbox` is a list of lists defining the CLEAN boxes. Each (inner) list consists of either: `[blc_x, blc_y, trc_x, trc_y]` giving the bottom left corner (blc) and top right corner (trc) x and y (1-relative) pixel numbers to define a rectangular box, or, `[-1, radius, x, y]` to define a circular box of radius `radius` pixels centered on pixel `(x,y)` (1-relative).
- `CalJy` gives the value of the Mustang internal calibration signal in Jy. If a single value is given, it is used for all detectors; alternately a value per detector can be given.
- `solInt` is the minimum integration time for smoothing residuals.
- `BLInt` is the baseline filter time in sec
- `AtmInt` is the atmospheric filter time in sec
- `tau0` is the zenith opacity in nepers. This can be either a single, constant, scalar, or a time ordered list of [time (days), zenity_opacity(nepers)]. Zenith opacities may be measured from a tipping scan or estimated from surface weather data. CLEO has a facility to derive a set of estimated opacities.
- `soln` is a list of time scales (sec) for each cycle of the iterative calibration process. The number of entries in the list determines the number of cycles.
- `PointTab` List of pointing offsets in time order each list entry is of the form [time(day) d Xel (asec), d el (asec)]
- `flagver` Flag (OTFFlag) table to be used. There is no action if no OTFFlag table exists and automated flagging is not enabled.
- `flagInt` Automated flagging interval (sec), $<= 0 ->$ no automated flagging. The automated flagging procedure is to image the selected data with the current best calibration, CLEAN the dirty image and form a sky model. This sky model is then used to determine both a residual and model data streams. A comparison of these streams using parameters `maxRMS` and `maxRatio` proceeds as described in section 1.10.1.
- `maxRMS` Maximum allowable detector residual RMS in Jy in in each flagInt of the automated flagging.
- `maxRatio` Maximum allowable ratio of the detector residual RMS to the equivalent model RMS in each flagInt of the automated flagging

In the following script segment, generic values of these parameters are set and then redefined for particular targets.

```
##########################################################################################
# Define parameters

# Root of data directory
DataRoot="/media/usbdisk/bcotton/FITS/TPAR_17/"
DataRoot = None  # To suppress attempt to update from archive

# Target to image and range of scans
target=["eskimo"]
scans = [55,72]                 # range of scans SET THIS
target = ["m87"]
scans = [93,104]               # range of scans SET THIS
target=["orionKL","bolorionN","bolorionS"]
scans = [20,41]                 # range of scans SET THIS

# Define data
# OTF file
inFile   = "TPAR_19OTF.fits"      # Full TPAR_19
inDisk   = 0                      # 0 means current working directory
outDisk  = 1                      # Where resultant files will go (./FITSdata)

# Use target name to define output files
outFile  = target[0]+"OTFCal.fits"        # Output calibrated data
dirtFile = "!"+target[0]+"Dirty.fits"     # Final output dirty image
cleanFile= "!"+target[0]+"Clean.fits"     # Final CLEAN output image
wtFile   = "!"+target[0]+"Wt.fits"        # Weighting image
BeamFile = "PARGaussBeam.fits"            # Dirty Gaussian beam
priorFile= target[0]+"Clean.fits"         # Prior model?

# List of feeds to use, empty list = all
feeds=[]

# Default Image Info
timerange=[0.0,1.0]               # time range in days, All times
cells = 2.0                       # cell spacing in asec
nx = 256                          # no. cells in x
ny = 256                          # no. cells in y
niter   = 1000                    # Number of iteration of CLEAN
gain    = 0.1                     # CLEAN loop gain
minFlux = 0.001                   # Minimum image brightness to CLEAN
CLEANbox=[[-1,10, nx/2+1,ny/2+1]] # Clean window, circular at center
flagver = 1                       # Flag table

# Default Calibration info
CalJy = [38.5]                  # Cal values in Jy, one for all or per detector
solInt = 1.0                    # Min solint in seconds
```

```
BLInt  = 20.0                   # Baseline filter time in sec
AtmInt = 10.0                   # Atmospheric filter time in sec

# Default editing info
flagver   = 1                   # Flag table
flagInt   = 5.0                 # Flagging interval (sec), <=0 -> no flagging
maxRMS    = 1.0                 # Maximum allowable detector residual RMS in Jy
maxRatio  = 3.0                 # Max. allowable ratio to equivalent model RMS

# Table of pointing offsets in time order [time(day) d Xel (asec), d el (asec)]
PointTab=[\
    [0.0, 0.0, 0.0], \
    [1.0, 0.0, 0.0]]
#    *********** SET THIS ********

# Table of opacities in time order [time(day), zenith opacity(nepers)]
tau0 = [[0.0000, 0.100],       \
        [1.0000, 0.100]]
#    *********** SET THIS ********

# set of time scales for iterations
soln = [(5*solInt), (2*solInt), (solInt)]
doOffset = True    # Do Offset cal before each MultiBeam cal?
deMode   = True    # Subtract the mode of the image when forming

# The following resets parameters for particular objects
# Orion
if target[0]=="bolorionN":
    BLInt  = 30.0                   # Baseline filter time in sec
    AtmInt = 20.0                   # Atmospheric filter time in sec
    nx = 500; ny=500
    solInt = 1.0
    soln = [6*solInt, 3*solInt, solInt, solInt]
    CLEANbox=[[177,49,332,333]]
    niter = 50000
    gain = 0.05
    # Crab
if target[0]=="crab":
    BLInt  = 15.0                   # Baseline filter time in sec
    AtmInt = 10.0                   # Atmospheric filter time in sec
    nx = 500; ny=500
    solInt = 1.0
    niter=50000
    gain=0.05
    #deMode   = False    # Don't Subtract the mode of the image when forming
    #doOffset = False    # Don't Do Offset cal before each MultiBeam cal?
    soln = [6*solInt, 3*solInt, solInt, solInt]
    CLEANbox=[[-1,101,252,253]]
```

```
    minWt   = 0.00001  # Minimum weight in imaging wrt maximum - big variation here
# Eskimo
if target[0]=="eskimo":
    inFile   = "EskimoRawOTF.fits"    # Eskimo nebula + cal only
    solInt = 2.0
    soln = [3*solInt, 2*solInt, solInt]
    CLEANbox=[[-1,16,129,126]]
    niter = 500
    doOffset = False
# M87
if target[0]=="m87":
    inFile   = "M87RawOTF.fits"     # M87 data
    niter = 5000
    solInt = 1.0
    soln = [(5*solInt),3*(solInt), (solInt), (solInt)]
    CLEANbox=[[95,110,149,140]]
    CLEANbox=[[-1,11,136,131],[-1,15,120,124],[-1,8,130,120],[-1,13,102,125]]
```

Updating Data During the Observations

When the analysis is proceeding during observations, the following will update the data file with any new scans not already included. No data update will be attempted if DataRoot = None, otherwise, it should be the root of the GBT archive for the observing session.

```
################################# Update data ########################################
# Update OTF with any new scans in archive
inOTF = GBTUtil.UpdateOTF ("PAROTF","Rcvr_PAR",inFile, inDisk, DataRoot, err
                            avgTime=0.05)
OErr.printErrMsg(err, "Error creating/updating input data object")
inInfo = inOTF.List
print "Processing", target, "scans", scans
```

Prior CLEAN model

If a prior CLEAN model is to be used, it should be initialized. The "dirty beam" (GBT resolution) image object is created.

```
########################### If prior model given #############################
# dirty beam
dirtyBeam  = Image.newPFImage("dirty beam", BeamFile,  inDisk,  True, err)
OErr.printErrMsg(err, "Error initializing dirty beam")

if priorFile and FITSDir.PExist(priorFile, inDisk, err):
    print "Using prior model in",priorFile
    prior  = Image.newPFImage("prior model", priorFile,  inDisk,  True, err)
    PSF    = dirtyBeam
else:
    prior = None
```

```
    PSF   = None
```

Initialize Calibration

The initial calibration is performed in the python module PARCal using function InitCal.

```
################################### Initial calibration ##############################
PARCal.InitCal(inOTF, target, err,\
               flagver=flagver, CalJy=CalJy, BLInt=BLInt, AtmInt=AtmInt,tau0=tau0, \
               prior=prior, PSF=PSF, PointTab=PointTab)
```

The following describes the functionality of PARCal.InitCal. The first phase of the script deletes any previous calibration tables from the data set and then creates a new initial OTFCal table (version no. 1) with an time increment of a quarter of the `solInt`.

```
################################### Initialize calibration ##############################
# delete any prior calibration tables
print "Remove previous calibration"
OTF.ClearCal(inOTF,err)

# Create an initial dummy table with a interval 1/4 of the shortest
# Filter type solution interval.
inter = solInt/4
OTFGetSoln.POTFGetDummyCal (inOTF, inOTF, inter, 1, 1, err)
```

Gain/Weight calibration

The next step is to determine the detector gains and weights from calibration scans, these are written the OTFSoln table 1. The values from a calibration scan are propagated in time until the next calibration scan; this operation is done on the entire data set. The solution table derived is then applied to OTFCal no. 1 producing OTFCal no. 2. OTFSoln tables are applied to OTFCal tables using OTF.Soln2Cal whose parameters are passed in OTF.Soln2CalInput. See help(OTF.Soln2Cal) for details.

```
################################### Gain/Weight calibration ############################
# Gain/Weight calibration
print "Gain/Weight  calibration"
inInfo = inOTF.List
inInfo.set("calJy", CalJy)
inInfo.set("doWate", True) # Do weight calibration

OTFGetSoln.POTFGetSolnPARGain(inOTF, inOTF, err)

# Update OTF Cal table
OTF.Soln2CalInput["InData"]  = inOTF        # Input data object
OTF.Soln2CalInput["oldCal"]  = 1            # Use initial cal table
OTF.Soln2CalInput["newCal"]  = 2            # New cal table
OTF.Soln2Cal(err, OTF.Soln2CalInput)        # Apply
```

Target specification

In subsequent processing steps, only data for the targets named in `target`, scans in the range given in `scans` and times in `timerange` are processed.

```
################################## Target specification #################################

inInfo.set("Targets", target)        # select only target data
inInfo.set("Stokes", "    ")         # Set Stokes
inInfo.set("timeRange", timerange)   # Set timerange
inInfo.set("Scans", scans)           # Select scans
inInfo.set("doCalSelect", True)
inInfo.set("flagVer", flagver)
inInfo.set("gainUse", 0)
```

Residual data from prior model for Baseline Cal

If an Image exists in the input data area with the same name as the output CLEAN image in the output data area (./FITSdata) then its CLEAN model is used as the initial guess of the source. A residual set of data is calculated applying the gain/weight calibration and this residual data is used in the the subsequent Baseline filter calibration. If no prior model exists, then the input OTF data is used.

```
    ########################### Residual data from prior model ############################
    # Get prior model, if any and compute residual OTF
    if prior!=None:
        print "Using prior model"
        gainuse = 0
        inInfo.set("gainUse", gainuse)
        inInfo.set("doCalib", 1)
        inInfo.set("flagVer", flagver)
        resid = OTFUtil.PSubModel(inData, None, prior, PSF, err)
        OErr.printErrMsg(err, "Error with residial data")
    else:
        print "No prior model used"
        resid = inData
```

Baseline filter

The initial background estimation is the Baseline filter which estimates the background after applying OTFCal table no. 2 and then a low pass filter removing fluctuations on timescales shorter than BLInt for each individual detector. This generates OTFSoln no. 2 which is applied to OTFCal no. 2 to generate OTFCal no. 3.

```
################################## Baseline filter ####################################
print "Baseline Filter"
solint = BLInt/86400.0
inInfo.set("solInt", solint)
inInfo.set("doCalSelect", True)
inInfo.set("flagVer", flagver)
gainuse=2
```

```
inInfo.set("gainUse", gainuse)
inInfo.set("doCalib", 1)
OTFGetSoln.PFilter(inOTF, inOTF, err)
# Soln2Cal parameters for filter cal (most defaulted)
OTF.Soln2CalInput["InData"]  = inOTF       # Input data object
OTF.Soln2CalInput["oldCal"]  = 2           # Use Gain cal output
OTF.Soln2CalInput["newCal"]  = 3           # New calibration
OTF.Soln2Cal(err, OTF.Soln2CalInput)  # Apply
```

Residual data from prior model for Atmosphere Cal

If an Image exists in the input data area with the same name as the output CLEAN image in the output data area (./FITSdata) then its CLEAN model is used as the initial guess of the source. A residual set of data is calculated applying the Baseline calibration and this residual data is used in the the subsequent Atmospheric (common mode) calibration. If no prior model exists, then the input OTF data is used.

```
    ########################### Residual data from prior model ##############################
    # Get prior model, if any and compute residual OTF
    if prior!=None:
        print "Using prior model"
        gainuse = 0
        inInfo.set("gainUse", gainuse)
        inInfo.set("doCalib", 1)
        inInfo.set("flagVer", flagver)
        resid = OTFUtil.PSubModel(inData, None, prior, PSF, err)
        OErr.printErrMsg(err, "Error with residial data")
    else:
        print "No prior model used"
        resid = inData
```

Common atmosphere + offset

The next calibration, after applying OTFCal no. 3, is a common “atmospheric” term and a single detector offset term in each scan. This generates OTFSoln no. 3 which is applied to OTFCal no. 3 to generate OTFCal no. 4.

```
print "Common atmosphere removal"
inInfo.set("Tau0", tau0)                    # Opacity table
inInfo.set("doCalSelect", True)
inInfo.set("flagVer", flagver)
gainuse=0
inInfo.set("gainUse", gainuse)
inInfo.set("doCalib", 1)
solint = AtmInt/86400.0
inInfo.set("solInt", solint)
clipsig = 5.0
inInfo.set("ClipSig", clipsig)
```

```
plotDet = -10
inInfo.set("plotDet", plotDet)
OTFGetSoln.PMBBase(resid, inOTF, err)

# Soln2Cal for Atm cal
OTF.Soln2CalInput["InData"]  = inOTF       # Input data object
OTF.Soln2CalInput["oldCal"]  = 3           # Use baseline cal output
OTF.Soln2CalInput["newCal"]  = 4           # New cal table
OTF.Soln2Cal(err, OTF.Soln2CalInput)       # Apply
```

Write pointing corrections to Soln table

If a table of pointing corrections is given in PointTab, this is written to a OTFSoln table and applied to the OTFCal table.

```
    ################################ Pointing correction ###################################
if PointTab != None:
    print "Apply pointing corrections"
    inInfo.set("POffset", PointTab)
    OTFGetSoln.POTFGetSolnPointTab(inData, inData, err)
    OErr.printErrMsg(err, "Error with pointing corrections")

    # Soln2Cal for Point cal
    OTF.Soln2CalInput["InData"]  = inData       # Input data object
    OTF.Soln2CalInput["oldCal"]  = 4           # Use baseline cal output
    OTF.Soln2CalInput["newCal"]  = 5           # New cal table
    OTF.Soln2Cal(err, OTF.Soln2CalInput)       # Apply
    OErr.printErrMsg(err, "Error updating Cal table with Soln")
```

Editing data

If editing parameters are specified, the data is edited by first imaging the data with the current calibration and then CLEANing it to form a sky model. Then flagging is done using a statistical comparison of the residual data streams and the model data streams. Data to be discarded are described in the OTFFlag table.

```
################################## Editing data #############################
if flagInt>0.0:
    # CleanSkyModelInput
    input = PARCal.CleanSkyModelInput
    input["InData"] = inOTF
    input["DirtyName"] = cleanFile
    input["CleanName"] = dirtFile
    input["outDisk"]= outDisk
    input["PSF"]    = dirtyBeam
    input["Niter"]  = niter
    input["Gain"]   = gain
    input["scan"]   = scans
```

```
    input["target"] = target
    input["nx"]     = nx
    input["ny"]     = ny
    input["xCells"] = cells
    input["yCells"] = cells
    input["Window"] = CLEANbox

    # Make sky model
    CCimage = PARCal.CleanSkyModel (err, input=input)

    # Editing
    if flagver<=0:
        flagver = 1
    inInfo = inOTF.List
    inInfo.set("maxRMS", maxRMS)
    inInfo.set("maxRatio", maxRatio)
    inInfo.set("flagVer", flagver)
    inInfo.set("FGVer", flagver)
    inInfo.set("doCalSelect", True)
    inInfo.set("doCalib", 1)
    inInfo.set("gainUse", 0)
    OTFGetSoln.PFlag(inOTF, CCimage, inOTF, 1, err)
    OErr.printErr(err)
```

Write calibrated output

In this section, the current calibration is applied to the target data and a new OTF data file is written. This file is used for subsequent calibration and imaging and will contain the final calibration when the procedure is done. A new, initial OTFCal table (no. 1) is generated.

```
# Apply current calibration and use result for remaining calibration
print "Write calibrated data "
# Delete output if it exists */
if FITSDir.PExist (outFile, outDisk, err):
    zapOTF = OTF.newPOTF("Output data", outFile, outDisk, True, err)
    zapOTF.Zap(err)
outOTF = OTF.newPOTF("Output data", outFile, outDisk, False, err)
OTF.PClone(inOTF, outOTF, err)     # Same structure etc
OErr.printErrMsg(err, "Error initializing output")

# Set time range
inInfo.set("timeRange", timerange)
inInfo.set("doCalSelect", True)
inInfo.set("flagVer", flagver)
gainuse=0
inInfo.set("gainUse", gainuse)
inInfo.set("doCalib", 1)
```

```
# Copy/calibrate
OTF.PCopy(inOTF, outOTF,  err)
OErr.printErrMsg(err, "Error selecting data")

# Create an initial dummy table with a interval 1/4 of the shortest
# Filter type solution interval.
inter = solInt/4
OTFGetSoln.POTFGetDummyCal (outOTF, outOTF, inter, 1, 1, err)
inInfo = outOTF.List
```

Setting Imaging/Calibration Parameters

The following section first looks up the position of the specified target from the input OTF data and then uses previously specified parameters to initialize the structures which control subsequent processing.

`OTF.ImageInput` is a python structure (dict) containing the imaging parameters, see `help(OTF.makeImage)` for details. This structure is filled with parameters as specified above.

`OTF.ResidCalInput` is a python structure (dict) containing the residual data calibration parameters, see `help(OTF.ResidCal)` for details.

```
################################### Set parameters ##############################
# Get position from OTF
pos =  GBTUtil.GetTargetPos(inOTF, target[0], err)
ra  = pos[0]                       # ra of center
dec = pos[1]                       # dec of center

# Imaging parameters
OTF.ImageInput["InData"]  = outOTF
OTF.ImageInput["disk"]    = outDisk
OTF.ImageInput["OutName"] = dirtFile
OTF.ImageInput["Beam"]    = dirtyBeam
OTF.ImageInput["OutWeight"]= wtFile
OTF.ImageInput["ra"]  = ra                    # Center RA
OTF.ImageInput["dec"] = dec                   # Center Dec
OTF.ImageInput["xCells"] = cells              # "X" cell spacing
OTF.ImageInput["yCells"] = cells              # "Y" cell spacing
OTF.ImageInput["nx"] = nx                     # number of cells in X
OTF.ImageInput["ny"] = ny                     # number of cells in Y
OTF.ImageInput["gainUse"] = 0                 # Which cal table to apply, -1 = none
OTF.ImageInput["flagVer"] = flagver           # Which flag table to apply, -1 = none
OTF.ImageInput["minWt"] = 1.0e-2              # Minimum weight in imaging - includes data weight
OTF.ImageInput["ConvType"] = 5                # Convolving fn = pillbox, 3 = Gaussian,

# Calibration parameters (some reset in loop)
OTF.ResidCalInput["InData"]  = outOTF       # Input data object
OTF.ResidCalInput["solType"] = "MultiBeam" # Solution type
OTF.ResidCalInput["solInt"]  = 500.0       # Solution interval (sec)
OTF.ResidCalInput["minFlux"] = 0.0         # Minimum Model brightness to use
```

```
OTF.ResidCalInput["Clip"]    = 1000.0      # Minimum image brightness to use in model
OTF.ResidCalInput["gainUse"] = 1           # Prior calibration, 0-> highest
OTF.ResidCalInput["minEl"]   = -90.0       # minimum elevation
OTF.ResidCalInput["flagVer"] = flagver     # Which flag table to apply, -1 = none
```

Create initial image

Next, make the initial dirty image applying OTFCal no. 1.

```
############################### Create initial image ###################################
print "Make Dirty Image"

OTF.ImageInput["gainUse"] = 0               # Which cal table
if len(feeds)>0:
    inInfo.set("Feeds", feeds)              # Which feeds
DirtyImg = OTF.makeImage(err, OTF.ImageInput)
OErr.printErrMsg(err, "Error making initial image")
```

CLEAN parameters

Now fill in previously specified CLEAN parameters into the CleanOTF.CleanInput dict. See CleanOTF.PClean for details. The image display object, `disp` is also created here. The actual CLEANing is performed in OTF.SelfCal. Finally, the OTF.Soln2CalInput parameters are reset for use in OTF.SelfCal.

```
################################## CLEAN parameters ####################################
# Image display
disp = ODisplay.ODisplay("ObitView", "ObitView", err)

# Image for cleaning
CleanImg = Image.newPFImage("Clean Image", cleanFile, outDisk, False, err)

# Create CleanOTF
CleanObj = CleanOTF.PCreate("Clean", DirtyImg, dirtyBeam, CleanImg, err)
OErr.printErrMsg(err, "Error creating CLEAN object")

# CLEAN parameters
CleanOTF.CleanInput["CleanOTF"] = CleanObj     # Clean object
CleanOTF.CleanInput["disp"]     = disp         # Image display object
CleanOTF.CleanInput["Patch"]    = 40           # Beam patch
CleanOTF.CleanInput["Niter"]    = niter        # Number of iterations
CleanOTF.CleanInput["Gain"]     = gain         # CLEAN loop gain
CleanOTF.CleanInput["BeamSize"] = 8.0/3600.0   # CLEAN restoring beam size in deg
CleanOTF.CleanInput["minFlux"]  = minFlux      # Minimum image brightness to CLEAN
CleanOTF.CleanInput["CCVer"]    = 1            # Clean components table version
CleanOTF.CleanInput["Feeds"]    = feeds        # list of feeds to use
CleanOTF.CleanInput["noResid"]  = True         # Don't include residuals in calibration
```

```
#  CLEAN window
for win in CLEANbox:
    CleanOTF.PAddWindow(CleanObj, win, err)

# Reset Soln2Cal parameters for self cal
OTF.Soln2CalInput["InData"]  = outOTF      # Input data object
OTF.Soln2CalInput["oldCal"]  = 1           # Input cal table
OTF.Soln2CalInput["newCal"]  = 2           # New cal table
```

Self calibration loop

This section contains the "self calibration" loop in which the backgrounds and sky model are iteratively determined. This loop is of two forms depending on the value of `doOffset`. If `doOffset` is True then each cycle consists of two parts; the initial calibration is per detector but with a time scale three times that specified in `soln`. This is followed by a common mode ("MultiBeam") imaging and calibration based on the "Offset" calibration. If `doOffset` is False, each cycle consists of only the ("MultiBeam") imaging and calibration. Each part of the self calibration cycle is performed in OTF.SelfCal and consists of:

1. Form a new dirty image using the most recent OTFCal calibration. If `deMode` is True, the mode of the pixel distribution is subtracted from each pixel.
2. If ObitView is available, display dirty image and allow interactive specification of the CLEAN window.
3. Clean the image
4. Make a "restored" image with the full telescope resolution but without residuals, clip below zero. This is the sky model.
5. Scale the resultant sky model image to Jy/beam in the new beam size.
6. Subtract the sky model from the calibrated OTF data by interpolating the value at the location of each data sample and writing a scratch copy of the data.
7. Derive the background signal estimations as specified by `soln` and write to OTFSoln 1 or 2 for the first and second parts of the cycle.
8. Apply OTFSoln no. 1 (or 2) to OTFCal no. 1 (or 2) and write OTFCal no. 2 (or 3).

The application of calibration here differs from the previous in that each cycle starts from the initial (dummy) calibration. The time scale of the solutions for each cycle are obtained from the `soln` array specified earlier. The number of intervals in `soln` determines the number of iterations.

```
################################# Self calibration loop #################################
# For each iteration specify the solution interval in soln
# Depending on doOffset, do pair of calibrations, the first with
# solType="Offset" with interval 3*si followed by a "MultiBeam"
# solution with interval si or simply
# a "MultiBeam" solution with interval si
```

```
count=0
OTF.ImageInput["gainUse"]    = 1                             # Which cal table to apply
OTF.ResidCalInput["gainUse"] = OTF.ImageInput["gainUse"]     # Prior calibration,
inInfo.set("deMode", False)  # Don't remove mode first cycle
if doOffset:
    for si in soln:
        count = count+1
        print "\n *** Self calibration loop ",count,"si=",3*si,"Offset"
        # First calibration of pair
        OTF.ResidCalInput["solInt"]  = 3*si
        OTF.ResidCalInput["solType"] = "Offset"
        OTF.Soln2CalInput["oldCal"]  = 1
        OTF.Soln2CalInput["newCal"]  = 2           # (Re)Use 2
        OTF.SelfCal(err, OTF.ImageInput, CleanOTF.CleanInput, OTF.ResidCalInput, OTF.Soln2CalIn
        OErr.printErrMsg(err, "Error in self cal")
        OTF.ImageInput["gainUse"]    = OTF.Soln2CalInput["newCal"]   # Which cal table to apply
        OTF.ResidCalInput["gainUse"] = OTF.Soln2CalInput["newCal"]   # Prior calibration
        # Second
        print "\n *** second calibration of loop ",count,"si=",si,"MultiBeam"
        OTF.ResidCalInput["solInt"]  = si
        OTF.ResidCalInput["solType"] = "MultiBeam"
        OTF.Soln2CalInput["oldCal"]  = 2
        OTF.Soln2CalInput["newCal"]  = 3           # (Re)Use 3
        OTF.SelfCal(err, OTF.ImageInput, CleanOTF.CleanInput, OTF.ResidCalInput, OTF.Soln2CalIn
        OTF.ImageInput["gainUse"]    = OTF.Soln2CalInput["newCal"]   # Which cal table to apply
        OTF.ResidCalInput["gainUse"] = 1   # Prior calibration for next cycle
        # Cleanup Soln tables
        outOTF.ZapTable("OTFSoln",1,err)
        outOTF.ZapTable("OTFSoln",2,err)
        inInfo.set("deMode", deMode) # remove mode
else:   # Only MultiBeam
    for si in soln:
        count = count+1
        print "\n *** calibration loop ",count,"si=",si,"MultiBeam"
        OTF.ResidCalInput["solInt"]  = si
        OTF.ResidCalInput["solType"] = "MultiBeam"
        OTF.Soln2CalInput["oldCal"]  = 1
        OTF.Soln2CalInput["newCal"]  = 2
        OTF.SelfCal(err, OTF.ImageInput, CleanOTF.CleanInput, OTF.ResidCalInput, OTF.Soln2CalIn
        OTF.ImageInput["gainUse"]    = OTF.Soln2CalInput["newCal"]   # Which cal table to apply
        OTF.ResidCalInput["gainUse"] = 1   # Prior calibration for next cycle
        # Cleanup Soln tables
        outOTF.ZapTable("OTFSoln",1,err)
        inInfo.set("deMode", deMode) # remove mode

print 'Finished with loop, final image'
```

Final image/CLEAN

After the calibration loop is finished, form a final dirty image, allow editing of the CLEAN window and CLEAN. After the CLEAN, the components in each pixel are summed in the CC table.

```
# Final image
DirtyImg = OTF.makeImage(err,  OTF.ImageInput)
OErr.printErrMsg(err, "Error in final dirty image")

# Final Clean
resid = CleanObj.Clean                        # Copy image just produced
Image.PCopy(DirtyImg, resid, err)             # to clean (residual)
CleanOTF.CleanInput["noResid"]  = False       # include residuals
CleanOTF.PClean(err,  CleanOTF.CleanInput)
OErr.printErrMsg(err, "Error Cleaning")

# Compress CC tables
cctab = CleanImg.NewTable(Table.READWRITE, "AIPS CC", 1, err)
TableUtil.PCCMerge(cctab, cctab, err)
OErr.printErrMsg(err, "Error merging CC table")
```

History

This section copies any processing history from the input data to the output CLEAN and dirty images and calibrated data and adds the current processing history records.

```
print "Copy history"
# Loop over dirty, clean images, output data
for img in [DirtyImg, CleanImg, outOTF]:
    inHistory  = History.History("in history", inOTF.List, err)
    outHistory = History.History("out history", img.List, err)
    History.PCopy(inHistory, outHistory, err)

    # Add this programs history
    outHistory.Open(History.READWRITE, err)
    outHistory.TimeStamp(" Start Obit "+ObitSys.pgmName,err)
    outHistory.WriteRec(-1,ObitSys.pgmName+" inFile = "+inFile,err)
    outHistory.WriteRec(-1,ObitSys.pgmName+" target = "+str(target),err)
    outHistory.WriteRec(-1,ObitSys.pgmName+" scans = "+str(scans),err)
    outHistory.WriteRec(-1,ObitSys.pgmName+" timerange = "+str(timerange),err)
    outHistory.Close(err)
    # Copy history to header
    inHistory  = History.History("in history",  img.List, err)
    outHistory = History.History("out history", img.List, err)
    History.PCopy2Header(inHistory, outHistory, err)
    OErr.printErrMsg(err, "Error with history")
```

Display CLEAN image, shutdown

This section displays the final CLEAN image and shuts down Obit.

```
###################### Display final CLEAN image, shutdown ######################
print "Display final CLEAN image"
ODisplay.PImage(disp, CleanImg, err)
OErr.printErrMsg(err, "Error displaying output image")

# Shutdown Obit
OErr.printErr(err)
del ObitSys
```

1.10.5 Modifying the Script

Most of the background signals seen by Mustang are common to all detectors and for a single, small, isolated source it is relatively straightforward to separate celestial sky and background. The general approach of this script is to begin the process by initially estimating the backgrounds from the measurements, implicitly assuming the celestial sky is empty. The initial calibrations need to use sufficiently long timescales not to remove actual emission. Following this is an iterative refinement of a (celestial) sky model and a background model. The sky model is determined by imaging the calibrated data and CLEANing the resultant "dirty" image. This sky model is then subtracted from the data and a refined estimate of the background derived, leading to an improved calibration. If an *a priori* sky model is available, either from separate observations or a previous processing of the same data, this model can be used in the initial calibration stages.

The details of the script may well need to be modified on a per case basis. The optimum strategy differs for isolated, small sources, weak extended emission and strong extended emission. The most critical aspect is the proper handling of extended emission, especially on scales larger than the footprint of the detector on the sky. Given the variability of the common mode backgrounds from both the atmosphere and the instrument, there is an inherent ambiguity between large scale celestial structure and time variable backgrounds. An erroneous large scale feature (either positive or negative) in the sky model can get mapped into the calibration. Part of the observing strategy is to sweep the telescope over sources on time scales fast compared to the atmospheric fluctuations and to cover each pixel many times in hopes that the systematic offsets will be uncorrelated. Unfortunately, common mode fluctuations from the instrument are on rather short time scales; the dominant of these is the 1.4 Hz variations seen in Figure 1.4 which is largely removed by the notch filter in `PAROTF`.

Strong and especially extended emission in the field is a particular problem as each detector sees different parts of the source, but for emission larger then the size of the array on the sky, the fluctuations in detector output are largely common mode. The initial calibration phases will only include a sky model if an *a priori* model is given and otherwise will try to remove this structure. A long time scale in these calibrations helps reduce this effect at the cost of adding negative regions around bright emission. This effect can be largely removed using techniques discussed below.

Some of the more critical parameters in the script are described in the following:

1. `CalJy`
   This value or list of values sets the flux density scale for the data by specifying the equivalent flux density of the Mustang internal cal signal. The value depends on the Mustang setup but the currently favored setup gives 38.5. If a single value is given, it is used for all detectors; values can also be specified on a per detector basis.

2. `BLInt`
   This is the timescale in seconds of the per detector "Baseline" calibration. Scales shorter than

this will not be included in the calibration. If the field observed contains extended emission, this needs to be as long as practical. However, for the common mode parts of the background to be accurately modeled and removed, the signals from the different detectors need to be adjusted to each other and this calibration is one of the principle ways of doing this. If an initial model of the field is provided, this value can be shortened.

3. `AtmInt`
   The bulk of the "Atmospheric" calibration is a common model correction with minimum time scale `AtmInt`. There is also a single, per scan adjustment of the detector offset. If there is extended emission in the field, especially on scales larger than the area covered by the detector array, `AtmInt` should be longer than the time for the antenna to traverse the region of emission. This can be relaxed somewhat if an initial model of the field is provided.

4. `soln`
   This parameter is the most important controlling this script. It determines the number of cycles of calibration and the time–scales of the residual filtering. The initial timescales should be longer than the time it takes the antenna to traverse the region of emission as the residual calibration is capable of removing (or adding) structure on large scales. Subsequent cycles decrease the minimum timescale.

5. `solInt`
   This parameter defines the minimum smoothing time for the various calibration phases and is used to determine the time increment in the initial OTFCal table. As the time stamps in the initial OTFCal table will be the ones in all derived OTFCal tables, `solInt` needs to be short enough that the residuals filtered to this timescale will be adequately sampled. This sampling of the OTFCal table has an increment 1/4 of `solInt` which is reasonably conservative. The principle reason for not making `solInt` smaller is that the OTFCal tables are rather large and can dwarf in size the actual data.

6. `doOffset`
   When there is bright emission in the field, the initial per detector calibrations will be in error due to the uncorrected emission from these sources. The common mode calibration cannot remove the resultant per detector variable offsets. If `doOffset` is True, then each residual calibration cycle has two parts starting with a per detector ("Offset") calibration with a time scale three times that specified in `soln` for the common mode calibration in the second part. An imaging and residual calibration is carried out for each part of the calibration cycle. This additional ("Offset") calibration may degrade the results for fields containing only weak emission.

7. `deMode`
   Systematic offsets of the zero level in a dirty image can increase or decrease the total flux density CLEANed. This can adversely affect the quality of the image derived, causing the total flux density in the image to increase or decrease with cycle. If `deMode` is True, then the mode of the pixel distribution (most common value in a pixel value histogram) is taken to be the "zero" level and is subtracted from the dirty image. This works best if much of the region imaged is empty of emission; in the limit of very widespread emission, this feature may remove true emission.

8. `niter, minFlux`
   These values control the depth of the CLEAN. If the CLEAN is too shallow, some of the

emission will not be represented in the model and the missing emission may be removed by subsequent calibration. `niter` is the maximum number of CLEAN iterations (components) and `minFlux` is the minimum abs. value residual to CLEAN.

9. `CLEANbox`
   This sets the region in which CLEAN may place the centers of components; regions more than a beam width from the edge of the box are implicitly set to zero in the sky model and any actual emission there may be removed or distorted by subsequent calibration. The default window is a single small circular box in the center of the field. The boxes of the CLEAN window can either be specified in the script or edited manually using ObitView before each CLEAN. The ObitView message window gives the parameters of the CLEAN window at the end of an editing session and these values can be written into the script.

1.11 Major Obit Single Dish Routines

This section documents the major routines used in this script.

1.11.1 GBTUtil.UpdateOTF

```
UpdateOTF(OTFTask, Rcvr, outFile, outDisk, DataRoot, err, \
     offTime=None, avgTime=None, config=None, scanNo=None, doBS=None, \
     dataNorm=None)
    Update OTF and return object

    return Python OTF object
    Update an OTF object with scans not yet in the file.
    Read DataRoot/scanLog.fits and determine which scans are available and
    are not already in the output OTF.
    These scans are appended and the output OTF returned.
    Does some data validity checks.

    OTFTask  = Name of Obit task to read GBT archive and write to OTF format
               DCR = "DCROTF"
               PAR = "PAROTF" (Mustang)
               CCB = "CCBOTF"
    Rcvr     = directory name with data for receiver.
               DCR = "DCR"
               PAR = "Rcvr_PAR" (Mustang)
               CCB = "CCB26_40"
    outFile  = Name of output OTF FITS file
    outDisk  = disk number for outFile, 0=> current working directory
    DataRoot = Root of GBT archive for current project
               If None, don't attempt
    err      = Python Obit Error/message stack

    Optional, backend specific values
    offTime  = PAR, CCB Offset in sec to be added to time
    avgTime  = PAR  Data averaging time in seconds
```

```
config   = PAR  Path of configuration file
scanNo   = PAR, CCB, replace GBT scan number with this value
doBS     = CCB  Output beamswitched data?
dataNorm = CCB  Normalization factors for beamswitched data
```

1.11.2 GBTUtil.GetTargetPos

```
GetTargetPos(OTF, Target, err)
    Get target position from OTFTarget table

    return [raepo, decepo] in deg
    Loop through target table on OTF looking for Target and return position

    OTF      = OTF data file to check
    Target   = Name of target e.g. "MARS"
```

1.11.3 OTF.ClearCal

```
ClearCal(inOTF, err)
    Delete calibration tables on an OTF

    Removes all OTFSoln and OTFCal tables
    inOTF    = Extant Python OTF
    err      = Python Obit Error/message stack
```

1.11.4 OTFGetSoln.POTFGetDummyCal

```
POTFGetDummyCal(inOTF, outOTF, inter, ver, ncoef, err)
    Create dummy OTFCal table table (applying will not modify data)

    returns OTFCal table with solutions
    inOTF   = input Python Obit OTF
    outOTF  = Python Obit OTF onto which the cal table is to be appended.
    inter   = time interval (sec) between entries
    ver     = OTFCal table version
    ncoef   = Number of coefficients (across array feeds) in table
    err     = Python Obit Error/message stack
```

1.11.5 OTFGetSoln.POTFGetSolnPARGain

```
POTFGetSolnPARGain(inOTF, outOTF, err)
    Determine Penn Array type gain calibration from data

    Determine instrumental gain from Mustang-like cal measurements
    Mustang-like = slow switching, many samples between state changes
    Average On and Off for each detector and difference.
    Mult factor = calJy/(cal_on-cal_off) per detector, may be negative.
    Data scan averaged and repeated for any subsequent scans without cal On data
    Write Soln entries at beginning and end of each scan.
```

```
Note: Any scans prior to a scan with calibration data will be flagged.
Calibration parameters are on the inOTF info member.
"calJy"     OBIT_float (*,1,1) Calibrator value in Jy per detector [def 1.0] .
            Duplicates if only one given.
returns OTFSoln table with solutions
inOTF   = Python Obit OTF from which the solution is to be determined
          prior calibration/selection applied if requested
outOTF  = Python Obit OTF onto which the solution table is to be appended.
err     = Python Obit Error/message stack
```

1.11.6 OTF.Soln2Cal

```
Soln2Cal(err, input={'InData': None, 'newCal': 0, 'oldCal': -1, 'soln': 0, 'structure': ['Soln2
'soln', 'input soln table version'), ('oldCal', 'input cal table version, -1=none'), ('newCal',
    Apply a Soln (solution) table to a Cal (calibration) table.

    err     = Python Obit Error/message stack
    input   = input parameter dictionary

    Input dictionary entries:
    InData = Python input OTF to calibrate
    soln   = Soln table version number to apply, 0-> high
    oldCal = input Cal table version number, -1 means none, 0->high
    newCal = output Cal table version number, 0->new
```

1.11.7 OTFGetSoln.PFilter

```
PFilter(inOTF, outOTF, err)
    Determine offset calibration for an OTF by time filtering a residual data set.

    The time series of each detector is filtered to remove structure on time
    scales shorter than solInt.
    Scans in excess of 5000 samples will be broken into several.
    Calibration parameters are on the inOTF info member.
    "solInt"    float scalar Solution interval in days [def 10 sec].
                This should not exceed 1000 samples.  Solutions will be truncated
                at this limit.
    "minEl"     float scalar Minimum elevation allowed (deg)
    returns OTFSoln table with solutions
    inOTF   = Python Obit OTF (residual) from which the solution is to be determined
    outOTF  = Python Obit OTF onto which the solution table is to be appended.
    err     = Python Obit Error/message stack
```

1.11.8 OTFGetSoln.PMBBase

```
PMBBase(inOTF, outOTF, err)
    Fits polynomial additive term on median averages of a residual data set.

    Each solution interval in a scan is median averaged
```

```
    (average of 9 points around the median) and then a polynomial fitted.
    Calibration parameters are on the inOTF info member.
    "SolInt"    float scalar Solution interval in days [def 10 sec].
                This should not exceed 5000 samples.  Solutions will be truncated
                at this limit.
    "minEl"     float scalar Minimum elevation allowed (deg)
    "Order"     Polynomial order [def 1]
    "clipSig"   data outside of range +/- CLIP sigma are blanked [def large]
    "plotDet"   Detector number to plot per scan [def =-1 = none]
    returns OTFSoln table with solutions
    inOTF   = Python Obit OTF (residual) from which the solution is to be determined
    outOTF  = Python Obit OTF onto which the solution table is to be appended.
    err     = Python Obit Error/message stack
```

1.11.9 OTF.makeImage

```
makeImage(err, input={'Beam': None, 'Clip': 1e+19, 'ConvParm':
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 'ConvType': 3,
    'Disk': 1, 'InData': None, 'OutName': None, 'OutWeight': None,
    'Wt': None, 'dec': 0.0, ...})

    Image an OTF.

    Data is convolved and re-sampled onto the specified grid.
    Image is created and returned on success.
    err     = Python Obit Error/message stack
    input   = input parameter dictionary

    Input dictionary entries:
    InData  = input Python OTF to image
    OutName = name of output image file
    OutWeight =  Output griding weight file name
    Disk    = disk number for output image file
    ra      = center RA (deg)
    dec     = center Dec (deg)
    nx      = number of pixels in "x" = RA
    ny      = number of pixels in 'Y' = dec
    xCells  = Cell spacing in x (asec)
    yCells  = Cell spacing in y (asec)
    minWt   = minimum summed weight in gridded image [def 0.1]
    Clip    = data values with abs. value larger are set zero weight
    ConvType= Convolving function Type 0=pillbox,3=Gaussian,4=exp*sinc,5=Sph wave
    ConvParm= Convolving function parameters depends on ConvType
      Type 2 = Sinc, (poor function - don't use)
        Parm[0] = half-width in cells,
        Parm[1] = Expansion factor
      Type 3 = Gaussian,
        Parm[0] = half-width in cells,[def 3.0]
```

```
      Parm[1] = Gaussian with as fraction or raw beam [def 1.0]
    Type 4 = Exp*Sinc
      Parm[0] = half-width in cells, [def 2.0]
      Parm[1] = 1/sinc factor (cells) [def 1.55]
      Parm[2] = 1/exp factor (cells) [def 2.52]
      Parm[3] = exp power [def 2.0]
    Type 5 = Spherodial wave
      Parm[0] = half-width in cells [def 3.0]
      Parm[1] = Alpha [def 5.0]
      Parm[2] = Expansion factor [not used]

gainUse = version number of prior table (Soln or Cal) to apply, -1 is none
flagVer = version number of flagging table to apply, -1 is none
doBeam  = Beam convolved with convolving Fn image desired? [def True]
Beam    = Actual instrumental Beam to use, else Gaussian [def None]
Wt      = Image to save griding weight array [def None], overrides OutWeight
```

1.11.10 OTF.ResidCal

```
ResidCal(err, input={'Clip': 1e+20, 'InData': None, 'Model': None,
    'ModelDesc': None, 'calJy': [1.0, 1.0], 'flagVer': -1, 'gainUse': -1,
    'minEl': 0.0, 'minFlux': -10000.0, 'minRMS': 0.0, ...})

    Determine residual calibration for an OTF.

    Determines a solution table for an OTF by one of a number of techniques using
    residuals from a model image.
    Returns the version number of the Soln Table on success.
    err     = Python Obit Error/message stack
    input   = input parameter dictionary

    Input dictionary entries:
    InData  = Python input OTF to calibrate
    Model   = Python input model FArray, "None" means do not subtract model image
    ModelDesc= Python input model ImageDesc
    minFlux = Minimum brightness in model
    solInt = solution interval (sec)
    solType = solution type:
       "Gain" solve for multiplicative term from "cals" in data.
             (solInt, minRMS, minEl, calJy)
       "Offset" Solve for additive terms from residuals to the model.
             (solInt, minEl)
       "GainOffset" Solve both gain and offset
             (solInt, minRMS, minEl, calJy)
       "Filter"  Additive terms from filters residuals to the model.
             (solInt, minEl)
        "MultiBeam" Multibeam solution
             (solInt, minEl)
```

```
    minEl  = minimum elevation (deg)
    minRMS = Minimum RMS residual to solution
    calJy  = Noise cal value in Jy per detector
    gainUse = version number of prior table (Soln or Cal) to apply, -1 is none
    flagVer = version number of flagging table to apply, -1 is none
```

1.11.11 OTF.SelfCal

```
SelfCal(err, ImageInp={'Beam': None, 'Clip': 1e+19, 'ConvParm':
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 'ConvType': 3,
    'Disk': 1, 'InData': None, 'OutName': None, 'OutWeight': None,
    'Wt': None, 'dec': 0.0, ...}, CleanInp=None,
    ResidCalInp={'Clip': 1e+20, 'InData': None, 'Model': None,'ModelDesc': None,
      'calJy': [1.0, 1.0], 'flagVer': -1, 'gainUse': -1, 'minEl': 0.0,
      'minFlux': -10000.0, 'minRMS': 0.0, ...},
    Soln2CalInp={'InData': None, 'newCal': 0, 'oldCal': -1, 'soln': 0,
      'structure': ['Soln2Cal', [('InData', 'Input OTF'), (
      'soln', 'input soln table version'), ('oldCal',
      'input cal table version, -1=none'), ('newCal', 'output cal table')]]})

    Self calibrate an OTF

    Image an OTF, optionally Clean, determine residual calibration,
    apply to Soln to Cal table.  If the Clean is done, then the CLEAN result is
    used as the model in the ResidCal, otherwise the dirty image from Image is.
    err         = Python Obit Error/message stack
    ImageInp    = input parameter dictionary for Image
    CleanInp    = input parameter dictionary for Clean, "None"-> no Clean requested
                  May be modified to point to the result of the Image step
    ResidCalInp = input parameter dictionary for ResidCal
                  Will be modified to give correct derived model image
    Soln2CalInp = input parameter dictionary for Soln2Cal
```

1.11.12 CleanOTF.PClean

```
PClean(err, input={'BeamSize': 0.0, 'CCVer': 0, 'CleanOTF': None,
    'Factor': 0.0, 'Gain': 0.10000000000000001, 'Niter': 100,
    'Patch': 100, 'Plane': [1, 1, 1, 1, 1], 'autoWindow': False,
    'minFlux': 0.0, ...})

    Performs image based CLEAN

    The peak in the image is iteratively found and then the beam
    times a fraction of the peak is subtracted and the process is iterated.
    err     = Python Obit Error/message stack
    input   = input parameter dictionary

    Input dictionary entries:
    CleanOTF  = Input CleanOTF,
```

```
    Niter       = Maximum number of CLEAN iterations
    Patch       = Beam patch in pixels [def 100]
    maxPixel    = Maximum number of residuals [def 20000]
    BeamSize    = Restoring beam (deg)
    Gain        = CLEAN loop gain
    minFlux     = Minimun flux density (Jy)
    noResid     = If True do not include residuals in restored image
    Factor      = CLEAN depth factor
    Plane       = Plane being processed, 1-rel indices of axes 3-?
    autoWindow  = True if autoWindow feature wanted.
    CCVer       = CC table version number
```

1.11.13 PARCal.CleanSkyModel

```
CleanSkyModel(err, input={'BeamSize': 0.0, 'CleanName': None, \
           'ConvParm': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], \
           'ConvType': 5, 'DirtyName': None, 'Gain':   0.1, \
           'InData': None, 'Niter': 100, 'PSF': None, 'Patch': 100, ...})
    Create a CLEAN image sky model of an OTF data set

    Does imaging and cleaning of specified data.
    Returns an image model of the CLEAN components convolved with PSF
    err     = Python Obit Error/message stack
    input   = input parameter dictionary, for interactive use, the function input
              will display the contents in human readable format.

    Input dictionary entries:
    InData  = Python input OTF to calibrate
    DirtyName = Dirty image name, None = scratch
    CleanName = Clean image name, None = scratch
    outDisk = Disk number for output files
    PSF     = Image with telescope psf
    scans   = scan range to image
    target  = list of targets to include, center will be position of first
    nx      = number of pixels in 'X' = RA
    ny      = number of pixels in 'Y' = dec
    xCells  = Cell spacing in x (asec)
    yCells  = Cell spacing in y (asec)
    ConvType= Convolving function Type 0=pillbox,3=Gaussian,4=exp*sinc,5=Sph wave
    ConvParm= Convolving function parameters depends on ConvType
      Type 2 = Sinc, (poor function - don't use)
        Parm[0] = halfwidth in cells,
        Parm[1] = Expansion factor
      Type 3 = Gaussian,
        Parm[0] = halfwidth in cells,[def 3.0]
        Parm[1] = Gaussian with as fraction or raw beam [def 1.0]
      Type 4 = Exp*Sinc
        Parm[0] = halfwidth in cells, [def 2.0]
```

```
      Parm[1] = 1/sinc factor (cells) [def 1.55]
      Parm[2] = 1/exp factor (cells) [def 2.52]
      Parm[3] = exp power [def 2.0]
    Type 5 = Spherodial wave
      Parm[0] = halfwidth in cells [def 3.0]
      Parm[1] = Alpha [def 5.0]
      Parm[2] = Expansion factor [not used]
  gainUse = version number of prior table (Soln or Cal) to apply, -1 is none
  flagVer = version number of flagging table to apply, -1 is none
  Niter       = Maximum number of CLEAN iterations
  Patch       = Beam patch in pixels [def 100]
  BeamSize    = Restoring beam (deg)
  Gain        = CLEAN loop gain
  Window      = list of Clean windows
  autoWindow  = True if autoWindow feature wanted.
```

1.11.14 PARCal.FitCal

```
FitCal(err, input={'BeamSize': 0.0, \
       'ConvParm': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,  0.0], \
        'ConvType': 5, 'Gain': 0.10000000000000001, 'InData': None, \
        'Niter': 100, 'PSF': None, 'Patch': 100, 'autoWindow':False, \
        'disp': None, ...})
    Image and fit a sequence of calibrator scans

    Does imaging and iterative calibration of a sequence of scans on a calibrator
    (Target).
    Calibration is by imaging  and performing a CLEAN to derive a sky model
    and then subtracting the sky model from the data.  An estimate of the
    residual background is derived from filtering the residuals.
    The number of calibration cycles is determined by the number of entries in
    soln and each cycle may consist of a "Common" calibration, or a "Detector"
    calibration or both.
    Data are imaged as grouped in scanList, scans in each list in scanList
    are imaged together
    Returns an array of dict's, one per list in scanList with
        "Time"   Center time in Days
        "Target" Source Name
        "Peak"   Peak flux density
        "Gauss"  Gaussian parameters (major, minor axis size (asec), PA (deg))
        "elOff"  Offset in elevation in asec
        "azOff"  Offset in azimuth ( actually Xel)
    err     = Python Obit Error/message stack
    input   = input parameter dictionary, for interactive use, the function input
              will display the contents in human readable format.

    Input dictionary entries:
    InData  = Python input OTF to calibrate
```

```
scrDisk = Disk number for scratch files
PSF     = Image with telescope psf
scanList= list of lists of target/scans to be imaged together
          list of forn [target, scan_1...]
disp    = Image display to show final "dirty" maps
save    = if True, save derived images, else delete
          These will be in scrDisk with names of the form
          target.+scan+.CalImage.fits
nx      = number of pixels in "x" = RA
ny      = number of pixels in 'Y' = dec
xCells  = Cell spacing in x (asec)
yCells  = Cell spacing in y (asec)
ConvType= Convolving function Type 0=pillbox,3=Gaussian,4=exp*sinc,5=Sph wave
ConvParm= Convolving function parameters depends on ConvType
  Type 2 = Sinc, (poor function - don't use)
    Parm[0] = halfwidth in cells,
    Parm[1] = Expansion factor
  Type 3 = Gaussian,
    Parm[0] = halfwidth in cells,[def 3.0]
    Parm[1] = Gaussian with as fraction or raw beam [def 1.0]
  Type 4 = Exp*Sinc
    Parm[0] = halfwidth in cells, [def 2.0]
    Parm[1] = 1/sinc factor (cells) [def 1.55]
    Parm[2] = 1/exp factor (cells) [def 2.52]
    Parm[3] = exp power [def 2.0]
  Type 5 = Spherodial wave
    Parm[0] = halfwidth in cells [def 3.0]
    Parm[1] = Alpha [def 5.0]
    Parm[2] = Expansion factor [not used]
gainUse = version number of prior table (Soln or Cal) to apply, -1 is none
flagVer = version number of flagging table to apply, -1 is none
Niter       = Maximum number of CLEAN iterations
Patch       = Beam patch in pixels [def 100]
BeamSize    = Restoring beam (deg)
Gain        = CLEAN loop gain
autoWindow  = True if autoWindow feature wanted.
solInt      = solution interval (sec)
solType     = solution type:
   "Common"   solve for common mode.additive effects on timescales longer
           than solInt
   "Detector" Solve for detector additive terms on timescales longer
           than 3 * solInt
   "Both" Both Common and Detector solutions each calibration cycle
soln        = list of solution intervals, one cycle per interval
              NO value should be less than solInt
```

1.11.15 PARCal.InitCal

```
InitCal(inData, targets, err, flagver=1, CalJy=[38.5], BLInt=30.0,   \
        AtmInt=20.0, tau0=0.10000000000000001, PointTab=None, \
        prior=None, PSF=None)
    Initial calibration of Mustang (PAR) data

    Any prior calibration tables are removed and then does the following:
      1) Generate an initial Cal table with a time increment of 1/4 of the
         lesser of BLInt and AtmInt
      2) Gain and Weight calibration, conversion to Jy uses CalJy
         CalJy can contain either a single value for all detectors or
         one value per detector.  This is the value of the cal in Jy.
      3) a "Baseline" per detector offsets on timescales longer than BLInt
         are determined and applied
      4) "Atmosphere" calibration determining one offset per detector per scan
         and a common mode offset on time scales longer than AtmInt.
         Opacity corrections are based on tau0 (zenith opacity in nepers)
      5) If PointTab is specified, pointing corrections are applied.
    When the procedure is finished, data cal be calibrated using the highest
    Cal table.
    If prior is given it is a Clean image if the target to be subtracted
    prior to the Baseline and Atmosphere calibration.
    Note: this only makes sense when all targets are covered by prior.
    If this option is used the instrumental PSF must also be provided in PSF.

    inData  = OTF data set to be calibrated
    targets = list of target names, empty list = all
    err     = Python Obit Error/message stack
    flagver =
    CalJy   = Array of the cal in Jy.  CalJy can contain either a single value
              for all detectors or one value per detector.
    BLInt   = Baseline filter shortest timescale in sec
    AtmInt  = Atmospheric filter shortest timescale in sec
    tau0    = zenith opacity in nepers
              this can be either a scalar constant opacity, or a table in the form
              of a list of lists of time (days) and opacity, e.g.:
    PointTab= a table of pointing offsets in time order
            [time(day) d Xel (asec), d el (asec)]
            Such a table can be generated from a dataset by FitCal
    prior   = If given, a CLEAN image covering all targets given,
            This model will be subtracted from the data prior to
            "Baseline" and "Atmosphere" calibration
            If this option is used the instrumental PSF must also be provided
            in PSF.
    PSF     = If prior is given, this is the instrumental PSF to use in the
            subtraction.
```

1.11.16 PARCal.PlotData

This describes the plplot implementation.

```
PlotData(inData, targets, scans, feeds, err, output='None', \
        bgcolor=0, nx=1, ny=1)
    Plot selected OTF data

    Plot data in inData selected by targets and scans
    inData  = OTF data set to be plotted, any calibration and editing will be applied
    targets = list of target names, empty list = all
    scans   = Range of scan number, 0's => all
    feeds   = list of feeds to plot
    err     = Python Obit Error/message stack
    output  = name and type of output device:
              "None"  interactive prompt
              "xwin"  X-Window (Xlib)
              "gcw"   Gnome Canvas Widget (interacts with ObitTalk)
              "ps"    PostScript File (monochrome)
              "psc"   PostScript File (color)
              "xfig"  Fig file
              "png"   PNG file
              "jpeg"  JPEG file
              "gif"   GIF file
              "null"  Null device
    bgcolor   = background color index (1-15), symbolic names:
                BLACK, RED(default), YELLOW, GREEN,
                AQUAMARINE, PINK, WHEAT, GRAY, BROWN,
                BLUE, BLUEVIOLET, CYAN, TURQUOISE,
                MAGENTA, SALMON, WHITE
    nx        = Number of horizontal subpages
    ny        = Number of vertical subpages
```