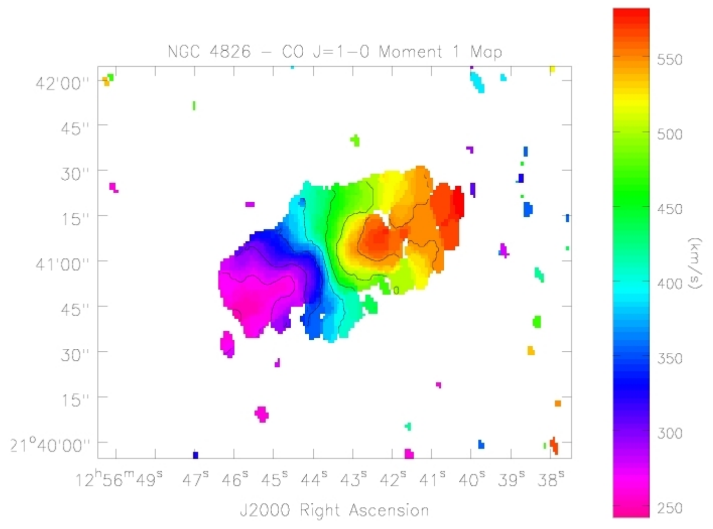
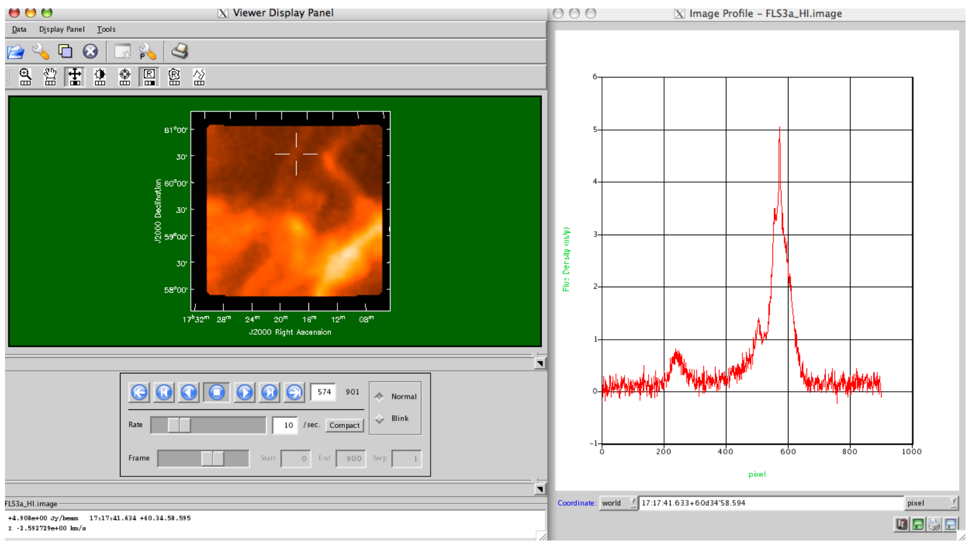
CASA Toolkit Guide





```
574 901
10 /sec.
0 900 1
FLS3a_HI.image
+4.908e+00 Jy/beam   17:17:41.634 +60.34.58.595
I -2.592729e+00 km/s
17:17:41.633+60d34'58.594
```

Version: April 20, 2007

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This document describes how to calibrate and image interferometric and single-dish radio astronomical data using the CASA (Common Astronomy Software Application) package. CASA is a suite of astronomical data reduction tools and tasks that can be run via the IPython interface to Python. CASA is being developed in order to fulfill the data post-processing requirements of the ALMA and EVLA projects, but also provides basic and advanced capabilities useful for the analysis of data from other radio, millimeter, and submillimeter telescopes.

The CASA home page can be found at:

- `http://casa.nrao.edu`

From there you can find documentation and assistance for the use of the package. Currently, CASA is in an **alpha release** and this should be taken into account as users begin to learn the package.

Tools in CASA provide the full capability of the package, and are the atomic functions that form the basis of data reduction. Tasks represent the more streamlined operations that a typical user would carry out — in many cases these are Python interface scripts to the tools, but with specific, limited access to them and a standardized interface for parameter setting. The idea for having tasks is that they are simpler to use, provide a more familiar interface, and are easier to learn for most astronomers who are familiar with radio interferometric data reduction (and hopefully for novice users as well).

For the moment, the audience is assumed to have some basic grasp of the fundamentals of synthesis imaging, so details of how a radio interferometer or telescope works and why the data needs to undergo calibration in order to make synthesis images are left to other documentation — a good place to start might be Synthesis Imaging in Radio Astronomy II (1999, ASP Conference Series Vol. 180, eds. Taylor, Carilli & Perley).

The CASA Reference Library consists of:

- CASA **Synthesis & Single Dish Reduction Cookbook** — task-based data analysis walk-through and instructions;
- CASA **in-line help** — accessed using `help` in the **casapy** interface;

- The **CASA Toolkit Guide** — this document; useful when the tasks do not have everything you want and you need more power and functionality, also contains more detailed descriptions of the philosophy of data analysis;
- The **CASA User Reference Manual** — to find out what a specific task or tool does and how to use it.

Chapter 2

Data Import, Handling, and Export

2.1 CASA Measurement Sets

Data is handled in CASA via the `table` system. In particular, visibility data are stored in a CASA table known as a Measurement Set (MS). Note that images are handled through special `image` tables, although standard FITS I/O is also supported. Images and `image` data are described in a separate chapter.

Unless your data was previously processed by CASA or software based upon its predecessor `aips++`, you will need to import it into CASA as an MS. Supported formats include some "standard" flavors of UVFITS, the VLA "Export" archive format, and most recently, the ALMA Science Data Model (ASDM) format.

2.1.1 Measurement Set Structure

Visibility data are stored in a CASA table known as a Measurement Set (MS). An MS consists of a main table containing the visibility data and associated sub-tables containing auxiliary or secondary information. Table 2.1 lists data selection parameters which may be used during a typical data reduction session while Table 2.2 identifies the commonly accessed components of the `MAIN` table in data sets.

Each row in a data column in the MS (e.g. `DATA`, `ALMA_PHAS_CORR`, `CORRECTED_DATA`) contains a matrix of observed complex visibilities at a single time stamp, for a single baseline in a single spectral window. The shape of the data matrix is given by the number of channels and the number of correlations (voltage-products) formed by the correlator for an array.

All CASA data files, including Measurement Sets, are written into the current working directory by default, with each CASA table represented as a separate sub-directory. MS names therefore need only comply with UNIX file or directory naming conventions, and can be referred to from within CASA directly, or via full path names.

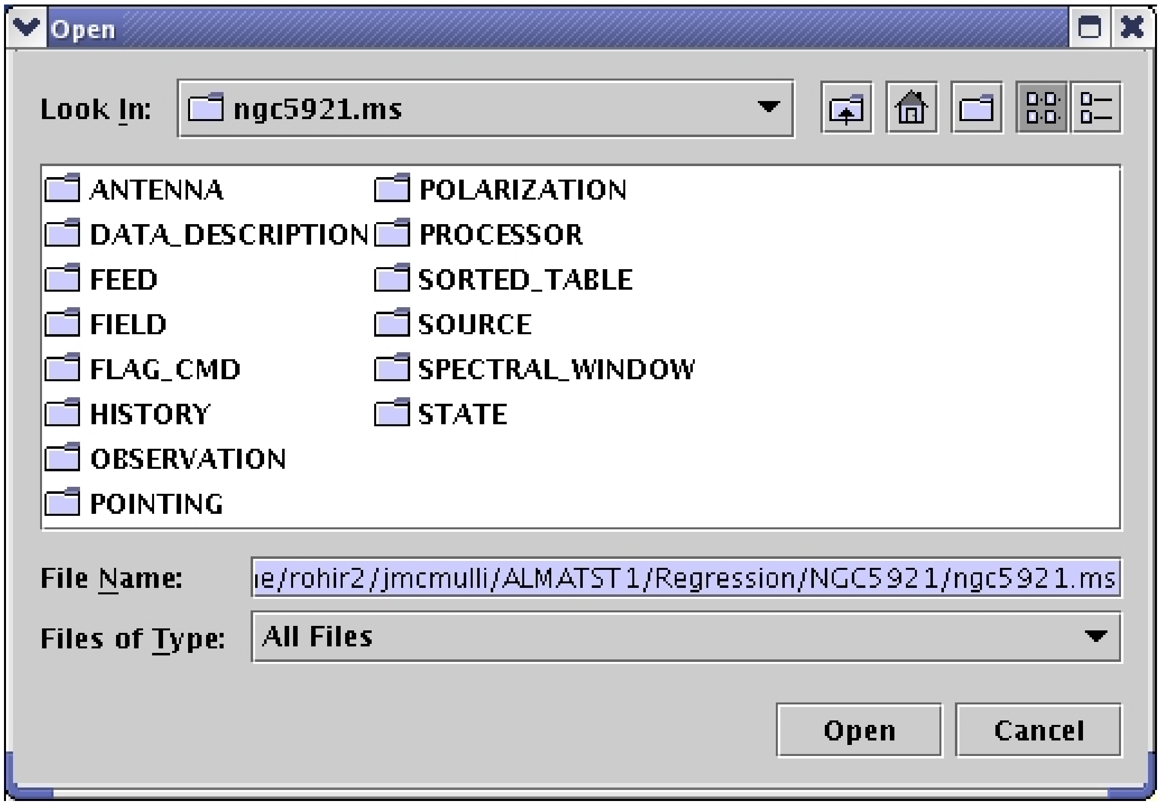
Table 2.1: Common Data Selection Parameters

Parameter	**Contents**
ANTENNA1	First antenna in baseline
ANTENNA2	Second antenna in baseline
FIELD_ID	Field (source no.) identification
DATA_DESC_ID	Spectral window number, polarization identifier pair (IF no.)
ARRAY_ID	Subarray number
OBSERVATION_ID	Observation identification
POLARIZATION_ID	Polarization identification
SCAN_NUMBER	Scan number
TIME	Integration midpoint time
UVW	UVW coordinates

Note: when you examine table entries like `FIELD_ID` or `DATA_DESC_ID` with the table browser, you will see 0-based numbers.

2.2 UVFITS Import and Export

Import and export methods include `ms.fromfits` and `ms.tofits`.



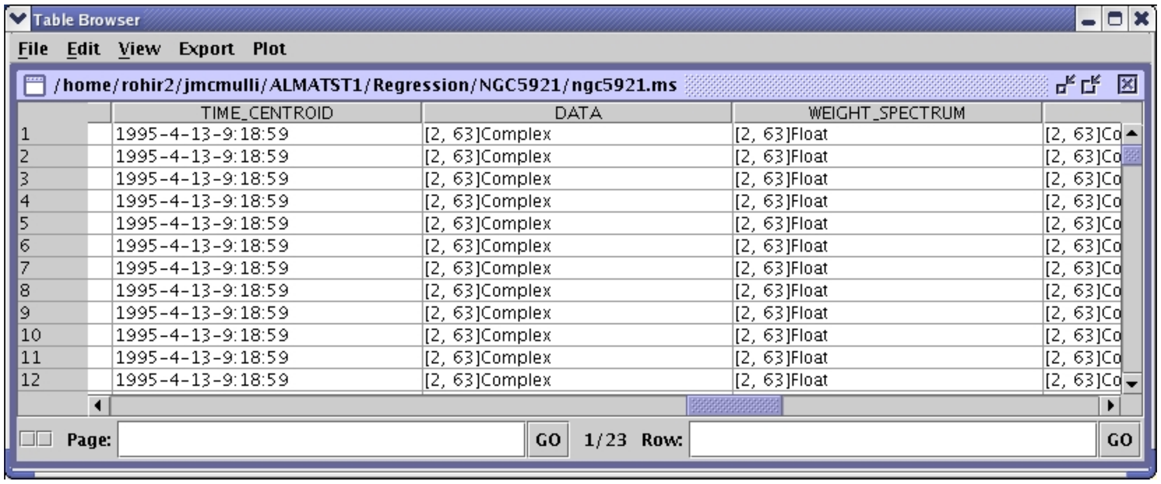
Look In: ngc5921.ms

ANTENNA
DATA_DESCRIPTION
FEED
FIELD
FLAG_CMD
HISTORY
OBSERVATION
POINTING
POLARIZATION
PROCESSOR
SORTED_TABLE
SOURCE
SPECTRAL_WINDOW
STATE

File Name: ıe/rohir2/jmcmulli/ALMATST1/Regression/NGC5921/ngc5921.ms

Files of Type: All Files

Figure 2.1: The structure of a Measurement Set. The tables which compose a Measurement Set named ngc5921.ms on disk.



/home/rohir2/jmcmulli/ALMATST1/Regression/NGC5921/ngc5921.ms

		TIME_CENTROID	DATA	WEIGHT_SPECTRUM	
1		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
2		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
3		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
4		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
5		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
6		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
7		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
8		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
9		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
10		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
11		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co
12		1995-4-13-9:18:59	[2, 63]Complex	[2, 63]Float	[2, 63]Co

Figure 2.2: A few of the MAIN table columns in a Measurement Set.

Table 2.2: Commonly accessed `MAIN` Table components for ALMA data

Column	**Format** **Comments**
DATA	Complex(N_c, N_f) Complex visibility matrix =ALMA_PHASE_CORR by default
WEIGHT_SPECTRUM	Float(N_c) Weight for whole data matrix
ALMA_PHASE_CORR	Complex(N_c, N_f) On-line phase corrected complex visibility matrix *(Not in VLA data)*
ALMA_NO_PHAS_CORR	Bool(N_c, N_f) Complex visibility matrix that has not been phase corrected *(Not in VLA data)*
ALMA_PHAS_CORR_FLAG_ROW	Bool(N_c, N_f) Flag to use phase-corrected data or not, Default=F *(not in VLA data)*
CORRECTED_DATA	Complex(N_c, N_f) Corrected data created by calibrater or imager tools
MODEL_DATA	Complex(N_c, N_f) Model data created by calibrater or imager tools
IMAGING_WEIGHT	Float(N_c) created by calibrater or imager tools
FLAG	Bool(N_c, N_f) cumulative data flags

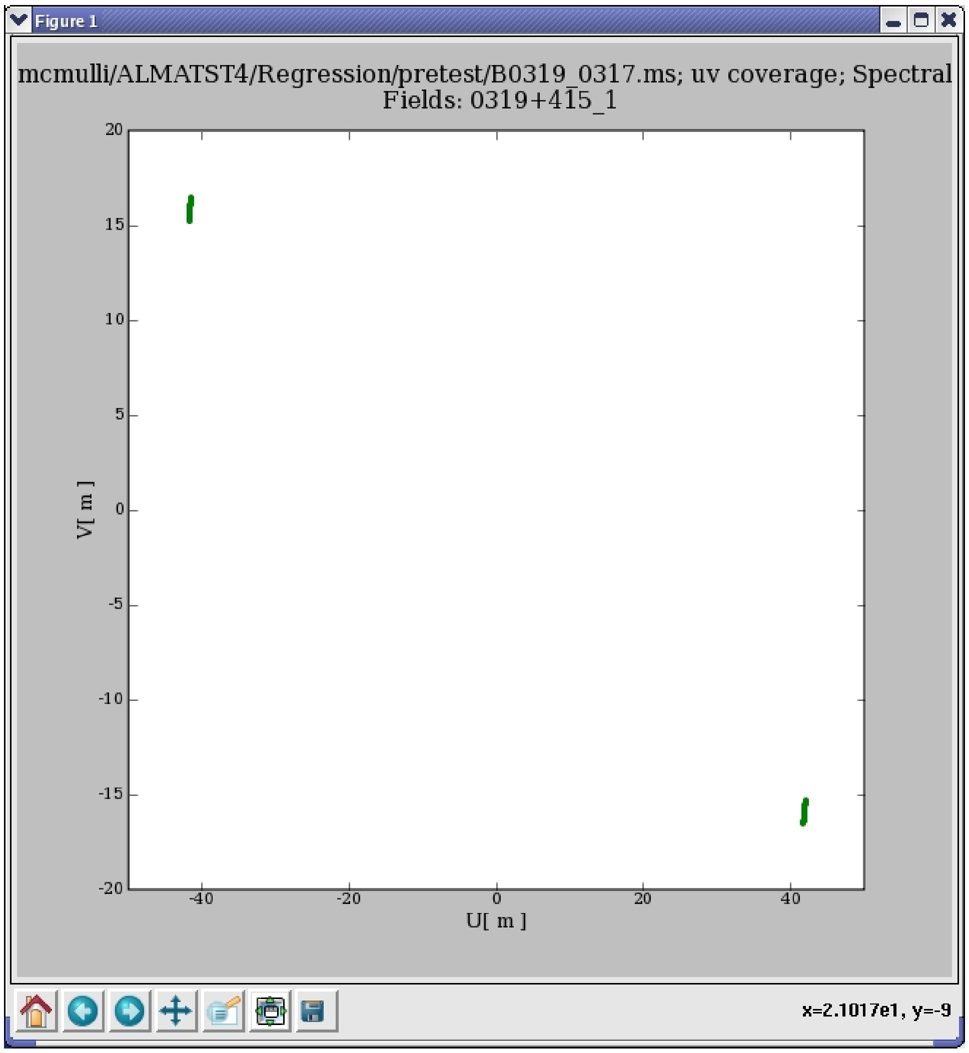


Figure 2.3: matplotlib plotter. The buttons on the lower left are: 1,2,3) **Home, Back and Forward**. Click to navigate between previously defined views (akin to web navigation), 4) **pan**. Click and drag to pan to a new position, 5) **zoom**. Click to define a rectangular region for zooming, 6) **Subplot Configuration**. Click to configure the parameters of the subplot and spaces for the figures, 7) **Save**. Click to launch a file save dialog box. The cursor readout is on the bottom right.

Chapter 3

Data Examination and Editing

3.1 Plot the Data

CASA uses the `matplotlib` plotting library to display its plots. You can find information on `matplotlib` at `http://matplotlib.sourceforge.net/`.

3.1.1 Plot Symbols

The `plotsymbol` defines both the line or symbol for the data being drawn as well as the color; from the matplotlib online documentation (e.g., type `pl.plot?` for help):

```
The following line styles are supported:
    -     : solid line
    --    : dashed line
    -.    : dash-dot line
    :     : dotted line
    .     : points
    ,     : pixels
    o     : circle symbols
    ^     : triangle up symbols
    v     : triangle down symbols
    <     : triangle left symbols
    >     : triangle right symbols
    s     : square symbols
    +     : plus symbols
    x     : cross symbols
    D     : diamond symbols
    d     : thin diamond symbols
    1     : tripod down symbols
    2     : tripod up symbols
    3     : tripod left symbols
    4     : tripod right symbols
    h     : hexagon symbols
    H     : rotated hexagon symbols
```

```
        p     : pentagon symbols
        |     : vertical line symbols
        _     : horizontal line symbols
        steps : use gnuplot style 'steps' # kwarg only
    The following color strings are supported
        b  : blue
        g  : green
        r  : red
        c  : cyan
        m  : magenta
        y  : yellow
        k  : black
        w  : white
    Line styles and colors are combined in a single format string, as in
    'bo' for blue circles.
```

3.1.2 Pylab - `matplotlib` library interface

For even more functionality, you can access the `pl` tool directly using `pylab` functions that allow one to annotate, alter, or add to any plot displayed in the matplotlib plotter (e.g. `plotxy`). `matplotlib` commands can be found at:

`http://matplotlib.sourceforge.net/pylab_commands.html`

The `pl.clf()` method is perhaps the most useful native command as it will clear the contents of the plotter frame.

A quick synopsis of relevant commands for altering plots are found below.

```
  pl.axhline - draw a horizontal line across axes
  pl.axvline - draw a vertical line across axes
  pl.axhspan - draw a horizontal bar across axes
  pl.axvspan - draw a vertical bar across axes
  pl.clf     - clear the current figure
  pl.close   - close the plotter (subsequent plots will reopen a plotter window)
  pl.ion     - turn interaction mode on (default - this indicates that any plotting command
               will be seen immediately after the command)
  pl.ioff    - turn off interaction mode (a 'show' command is required to see commands after
               this mode has been enabled)
  pl.savefig - save the current figure
  pl.subplot - make a subplot (numrows, numcols, axesnum)
  pl.text    - add some text at location (x,y)
  pl.title   - add/change the title
  pl.xlim    - set/get the xlimits
  pl.ylim    - set/get the ylimits
  pl.xlabel  - add/change the xlabel
  pl.ylabel  - add/change the ylabel
```

In addition, there are a range of mathematical functions provided (trigonometric, matrix algebra, logs, etc). Again, see the pylab documentation for help.

3.2 Data Flagging

Chapter 4

Synthesis Calibration

4.1 Calibration Philosophy

The visibilities measured by an instrument must be calibrated before formation of an image. This is becausethe wavefronts received and processed by the observational hardwarehave been corrupted by a variety of effects. These include the effects of transmission through the atmosphere, the imperfect details amplified electronic (digital) signal and transmission through thesignal processing system, and the effects of formation of the cross-power spectra by a correlator. Calibration is the process of reversing these effects to arrive at corrected visibilities which resemble as closely as possible the visibilities that would have been measured in vacuum by a perfect system. The subject of this chapter of the cookbook is the determination of these effects by using the visibility data itself.

The relationship between the observed and ideal (desired) visibilities on the baseline between antennas i and j may be expressed by the Measurement Equation:

$\vec{V}_{ij} \; = \; J_{ij} \; \vec{V}_{ij}^{\;\mathrm{IDEAL}}$

where $\vec{V}_{ij}$ represents the observed visbility, $\vec{V}_{ij}^{\;\mathrm{IDEAL}}$ represents the corresponding ideal visibilities, and J_{ij} represents the accumulation of all corruptions affecting baseline ij. The visibilities are indicated as vectors spanning the four correlation combinations which can be formed from dual-polarization signals. These four correlations are related directly to the Stokes parameters which fully describe the radiation. The J_{ij} term is therefore a 4×4 matrix.

Most of the effects contained in J_{ij} (indeed, the most important of them) are antenna-based, i.e., they arise from measurable physical properties of (or above) individual antenna elements in a synthesis array. Thus, adequate calibration of an array of N_{ant} antennas forming $N_{ant}(N_{ant}-1)/2$ baseline visibilities is usually achieved through the determination of only N_{ant} factors, such that $J_{ij} = J_i \otimes J_j^*$. For the rest of this chapter, we will usually assume that J_{ij} is factorable in this way, unless otherwise noted.

As implied above, J_{ij} may also be factored into the sequence of specific corrupting effects, each having their own particular (relative) importance and physical origin, which determines their unique algebra. Including the most commonly considered effects, the Measurement Equation can be written:

$\vec{V}_{ij} \ = \ M_{ij} \ B_{ij} \ G_{ij} \ D_{ij} \ E_{ij} \ P_{ij} \ T_{ij} \ \vec{V}_{ij}^{\ \mathrm{IDEAL}}$
where:

- T_{ij} = Polarization-independent multiplicative effects introduced by the troposphere, such as opacity and path-length variation.
- P_{ij} = Parallactic angle, which describes the orientation of the polarization coordinates on the plane of the sky. This term varies according to the type of the antenna mount.
- E_{ij} = Effects introduced by properties of the optical components of the telescopes, such as the collecting area's dependence on elevation.
- D_{ij} = Instrumental polarization response. "D-terms" describe the polarization leakage between feeds (e.g. how much the R-polarized feed picked up L-polarized emission, and vice versa).
- G_{ij} = Electronic gain response due to components in the signal path between the feed and the correlator. This complex gain term G_{ij} includes the scale factor for absolute flux density calibration, and may include phase and amplitude corrections due to changes in the atmosphere (in lieu of T_{ij}). These gains are polarization-dependent.
- B_{ij} = Bandpass (frequency-dependent) response, such as that introduced by spectral filters in the electronic transmission system
- M_{ij} = Baseline-based correlator (non-closing) errors. By definition, these are not factorable into antenna-based parts.

Note that the terms are listed in the order in which they affect the incoming wavefront (G and B represent an arbitrary sequence of such terms depending upon the details of the particular electronic system). Note that M differs from all of the rest in that it is not antenna-based, and thus not factorable into terms for each antenna.

As written above, the measurement equation is very general; not all observations will require treatment of all effects, depending upon the desired dynamic range. E.g., bandpass need only be considered for continuum observations if observed in a channelized mode and very high dynamic range is desired. Similarly, instrumental polarization calibration can usually be omitted when observing (only) total intensity using circular feeds. Ultimately, however, each of these effects occurs at some level, and a complete treatment will yield the most accurate calibration. Modern high-sensitivity instruments such as ALMA and EVLA will likely require a more general calibration treatment for similar observations with older arrays in order to reach the advertised dynamic ranges on strong sources.

In practice, it is usually far too difficult to adequately measure most calibration effects absolutely (as if in the laboratory) for use in calibration. The effects are usually far too changable. Instead, the calibration is achieved by making observations of calibrator sources on the appropriate timescales for the relevant effects, and solving the measurement equation for them using the fact that we have $N_{ant}(N_{ant}-1)/2$ measurements and only N_{ant} factors to determine (except for M which is only sparingly used). (*Note: By partitioning the calibration factors into a series of consecutive effects,* *it might appear that the number of free parameters is some multiple of N_{ant}, but the relative algebra and timescales of the different effects, as well as the the multiplicity of observed polarizations and channels compensate, and it can be shown that the problem remains well-determined until, perhaps, the effects are direction-dependent within the field of view. Limited solvers for such effects are under study; the calibrater tool currently only handles effects which may be assumed constant within the field of view. Corrections for the primary beam are handled in the imager tool.*) Once determined, these terms are used to correct the visibilities measured for the scientific target. This procedure is known as cross-calibration (when only phase is considered, it is called phase-referencing).

The best calibrators are point sources at the phase center (constant visibility amplitude, zero phase), with sufficient flux density to determine the calibration factors with adequate SNR on the relevant timescale. The primary gain calibrator must be sufficiently close to the target on the sky so that its observations sample the same atmospheric effects. A bandpass calibrator usually must be sufficiently strong (or observed with sufficient duration) to provide adequate per-channel sensitivity for a useful calibration. In practice, several calibrators are usually observed, each with properties suitable for one or more of the required calibrations.

Synthesis calibration is inherently a bootstrapping process. First, the dominant calibration term is determined, and then, using this result, more subtle effects are solved for, until the full set of required calibration terms is available for application to the target field. The solutions for each successive term are relative to the previous terms. Occasionally, when the several calibration terms are not sufficiently orthogonal, it is useful to re-solve for earlier types using the results for later types, in effect, reducing the effect of the later terms on the solution for earlier ones, and thus better isolating them. This idea is a generalization of the traditional concept of self-calibration, where initial imaging of the target source supplies the visibility model for a re-solve of the gain calibration (G or T). Iteration tends toward convergence to a statistically optimal image. In general, the quality of each calibration and of the source model are mutually dependent. In principle, as long as the solution for any calibration component (or the source model itself) is likely to improve substantially through the use of new information (provided by other improved solutions), it is worthwhile to continue this process.

In practice, these concepts motivate certain patterns of calibration for different types of observation, and the calibrater tool in CASA is designed to accomodate these patterns in a general and flexible manner. For a spectral line total intensity observation, the pattern is usually:

1. Solve for G on the bandpass calibrator
2. Solve for B on the bandpass calibrator, using G
3. Solve for G on the primary gain (near-target) and flux density calibrators, using B solutions just obtained
4. Scale G solutions for the primary gain calibrator according to the flux density calibrator solutions
5. Apply G and B solutions to the target data
6. Image the calibrated target data

If opacity and gain curve information are relevant and available, these types are incorporated in each of the steps (in future, an actual solve for opacity from appropriate data may be folded into this process):

1. Solve for G on the bandpass calibrator, using T (opacity) and E (gain curve) solutions already derived.
2. Solve for B on the bandpass calibrator, using G, T (opacity), and E (gain curve) solutions.
3. Solve for G on primary gain (near-target) and flux density calibrators, using B, T (opacity), and E (gain curve) solutions.
4. Scale G solutions for the primary gain calibrator according to the flux density calibrator solutions
5. Apply T (opacity), E (gain curve), G, and B solutions to the target data
6. Image the calibrated target data

For continuum polarimetry, the typical pattern is:

1. Solve for G on the polarization calibrator, using (analytical) P solutions.
2. Solve for D on the polarization calibrator, using P and G solutions.
3. Solve for G on primary gain and flux density calibrators, using P and D solutions.
4. Scale G solutions for the primary gain calibrator according to the flux density calibrator solutions.
5. Apply P, D, and G solutions to target data.
6. Image the calibrated target data.

For a spectro-polarimetry observation, these two examples would be folded together.

In all cases the calibrator model must be adequate at each solve step. At high dynamic range and/or high resolution, many calibrators which are nominally assumed to be point sources become slightly resolved. If this has biased the calibration solutions, the offending calibrator may be imaged at any point in the process and the resulting model used to improve the calibration. Finally, if sufficiently strong, the target may be self-calibrated as well.

4.2 General Calibrater Mechanics

The calibrater tasks/tool are designed to solve and apply solutions for all of the solution types listed above (and more are in the works). This leads to a single basic sequence of execution for all solves, regardless of type:

1. Set the calibrator model visibilities
2. Select the visibility data which will be used to solve for a calibration type
3. Arrange to apply any already-known calibration types (the first time through, none may yet be available)
4. Arrange to solve for a specific calibration type, including specification of the solution timescale and other specifics
5. Execute the solve process
6. Repeat 1-4 for all required types, using each result, as it becomes available, in step 2, and perhaps repeating for some types to improve the solutions

By itself, this sequence doesn't guarantee success; the data provided for the solve must have sufficient SNR on the appropriate timescale, and must provide sufficient leverage for the solution (e.g., D solutions require data taken over a sufficient range of parallactic angle in order to separate the source polarization contribution from the instrumental polarization).

4.3 Baseline-based Calibration

Some calibration cannot be factored into antenna-based factors. Such effects occur physically after the signals from each antenna have been combined in the correlator. These may occur as a consequence of finite time- and frequency-averaging over other calibration factors before they are corrected. Any loss of coherence in such averaging gives rise to residual baseline-based errors which depend on the details of the *combination* of antenna-based errors on the baseline. Also, if only a single baseline is available, a single calibration solution for that baseline is obtainable, and short of additional information, it simply cannot be factored into unique terms for each antenna. Therefore, such situations necessarily require baseline-based calibration solutions.

It is important to note that source structure also introduces baseline-based information in the visibilities, and so it can be difficult to distiguish baseline-based errors from the information in which we are interested, namely, the astronomincal visibilities (which are imaged). Therefore, baseline based calibration factors should be used with great care, to avoid changing the desired astrophysical information in ways that cannot be determined. As indicated below, there are some calibration circumstances where such extreme measures are warranted (e.g., resolved bandpass calibrators), but the careful observer should always consider how different calibration factors affect the data, and this is especially important for baseline-based factors.

4.3.1 M, MF solutions (Generic Baseline-based Gain)

The calibration types M and MF provide the baseline-based analogs of G and B, respectively. M provides for (per-spectral window) time-dependent gain calibration for each baseline independently. MF provides for a frequency-dependent (within each spectral window) version of M. One or the other of these types can be used to compensate for any residual closure errors, usually after all antenna-based calibrations have converged. Since these types can absorb legitimate visibility information from the data, they should be used with great care, and usually only on sources for which there is no doubt about the source structure. It is therefore largely meaningless to use them in self-calibration—they will only reinforce the current source model, and can even artifically absorb thermal noise in the data.

To solve for M on an hourly timescale, using previously determined G and B solutions:

```
cb.reset()                                # Reset the calibrater tool
cb.setdata(msselect='FIELD_ID IN [0,1]') # Restrict data selection
cb.setapply(type='G',table='cal.G')      # Apply existing G solution
cb.setapply(type='B',table='cal.B')      # Apply existing G solution

cb.setsolve(type='M',table='cal.M',      # Setup to solve for M on
    t=3600)                               #  an hourly timescale,
                                          #  write solutions to a table on
                                          #  disk called 'cal.M'
cb.solve()                                # Solve
```

Note that `refant` is, of course, meaningless for baseline-based calibration types.

To apply these solutions, along with G and B:

```
cb.reset()                                   # Reset calibrater tool
cb.setdata(msselect='FIELD_ID IN [0,1,2]')  # Restrict data selection
cb.setapply(type='G',table='cal.G')         # Apply G solutions
cb.setapply(type='B',table='cal.B')         # Apply B solutions
cb.setapply(type='M',table='cal.M')         # Apply M solutions
cb.correct()                                 # Correct data and write to
                                             #  CORRECTED_DATA column in MS
```

Use of MF is essentially identical, except that it will probably be used on even longer timescales than M (as for B, compared to G).

An M solution can be especially useful for obtaining normalized bandpass solutions from significantly resolved calibrators where a source model adequate for pre-B G-solving is not available. Ordinarily, if the bandpass calibrator is nearly point-like, we first solve for G (using a point-like model):

$$\vec{V}_{ij} = G_{ij}\,\vec{V}_{ij}^{\,\mathrm{point}}$$

Then, use this G to solve for B (using the same model):

$$\vec{V}_{ij} = B_{ij}\left(G_{ij}\,\vec{V}_{ij}^{\,\mathrm{point}}\right)$$

However, we will get (at best) awkwardly scaled and/or poorly converged B solutions if our source model is not sufficient for the either of these solutions. In this circumstance, we can use M to absorb both time-dependent gain variations during the bandpass calibration scan *and* the source structure. First solve for M, using a point-like model:

$$\vec{V}_{ij} = M_{ij}\,\vec{V}_{ij}^{\,\mathrm{point}}$$

Then, use this solution to solve for B:

$$\left(M_{ij}^{-1}\ \vec{V}_{ij}\right) \ = \ B_{ij}\ \vec{V}_{ij}^{\,\mathrm{point}}$$

The resulting B solution is nearly as good as one obtained with a good, resolved source model and G. It is somewhat less sensitive because more free parameters have been introduced, but this technique can often be useful. Note that the data for the bandpass calibrator, after correction by B and M, will be that of a point source, since M has absorbed the source structure information. This information has been sacrificed for a decent bandpass calibration applicable to other sources.

4.3.2 K solutions (Baseline-based fringe-fitting)

Fringe-fitting is essentially a generalization of ordinary phase calibration (the phase part of G, B, M, and/or MF) in which coefficients of the Taylor expansion of the phase in time and frequency are solved for. Usually, the expansion is only taken to first order in time and frequency. In this case, the phase can be written:

$$\phi = \phi(t_o, \nu_o) + 2\pi(\nu - \nu_o)\tau + 2\pi\nu(t - t_o)\dot{\tau}$$

In this equation, t_o and ν_o are a fiducial time and frequency within the solution interval (where the phase is $\phi(t_o, \nu_o)$), and τ and $\dot{\tau}$ are the delay (phase slope in frequency) and delay-rate (phase slope in time), respectively.

Note: As written, this equation defines the *group delay* (by virtue of referring the frequency to the fiducial value) where the delay-rate is not necessarily the time derivative of the delay. For most current observations, this is the case simply because the noise on the delay solution far exceeds the variation implied by the delay-rates. The equation can also be written in terms of the *phase delay*, where the ν_o is dropped from the second term, and the delay-rate is exactly the derivative of the delay. This mode will be useful for the wide-bandwidth instruments of the near future, and will be an option available for users of K.

Evidently, fringe-fitting permits more accurate phase tracking in both time and frequency when delay and delay-rate errors are large. Such errors originate in independent clock and position errors of antennas and in errors in the direction to radio sources. Fringe-fitting is normally associated with VLBI because these clock and geometry errors are more significant there than for smaller connected-element arrays, but the wide bandwidths and high frequencies of future instruments will likely find fringe-fitting useful.

The current implementation of fringe-fitting in CASA is baseline-based and quite simple. It solves the above equation for $\phi(t_o, \nu_o)$, τ, and $\dot{\tau}$ for each baseline independently, assuming less than one cycle of phase over the bandwidth and duration of the solution interval.

Note: Solutions outside the first ambiguity require implementation of a first-guess mechanism (an FFT) to reduce the phase to within the first ambiguity. This is still under development.

A separate phase and delay is determined for each polarization, but only one rate is determined for both.

Note: This is almost always justified since the phase and delay correspond to signal path lengths, and the delay rate corresponds to the speed of the signal along the path. The phase and delay are likely to be different for different polarizations since these signals traverse different electronics with different path lengths, but the speed at which the signal travels is the same. In any case, this constraint will be relaxable in a future version.

To obtain a scan-based K solution, with an existing MF solution applied:

```
cb.setdata(msselect='FIELD_ID==0')        # select data
cb.setapply(type='MF',table='cal.MF')     # arrange apply of MF
cb.setsolve(type='K',table='cal.K',t=0)   # arrange solve for K
cb.solve()                                # solve
```

Application is like any other type:

```
cb.setdata(msselect='FIELD_ID IN [0,1]')  # select data
cb.setapply(type='MF',table='cal.MF')     # arrange apply of MF
cb.setapply(type='K',table='cal.K')       # arrange apply of K
cb.correct()                                # correct
```

Note that only *nearest* interpolation is available for application of K. This will be remedied in the near future.

Chapter 5

Imaging Data

This chapter describes how to make and deconvolve images starting from a calibrated MeasurementSet.

5.1 The `im` Toolkit

The rm tool for imaging and deconvolution is imager (`im`). It has been designed to do gridding/degridding of visibility data, Fourier transforms, deconvolution using various methods, etc. A complete overview of the capabilities of Imager can be found in the User Reference Manual for the `imager` tool on the web:

```
(http://aips2.aoc.nrao.edu/weekly/docs/user/SynthesisRef/node118.html#imager);
```

Note: at this time the Reference Manual still refers to the *Glish* syntax so you will have to convert this to python syntax.

There are two sorts of tool functions in `im`; passive and active. The passive functions like `setimage`, `setdata` etc. set the state of the `im` tool. Then, active functions such as `makeimage`, `clean` etc. act on the data according to the previously set state.

To use an `im` tool, one must first construct it from a MeasurementSet and then configure it (set its "state"). When you are finished, you destroy the tool (which of course does not affect the actual MeasurementSet on disk). To construct an `imager` tool for a measurement set located at /home/data/data.ms:

```
im.open('/home/data/data.ms')            # Load data into imager tool
im.summary()                             # Summarize state of tool
im.close()                               # End the tool
```

The `summary` function is always useful to summarize the current state of the `imager` tool.

5.2 Setting up the `im` Tool

5.2.1 Data Selection

To make an image from the visibilities one needs to select the part of the data from the MeasurementSet that is going to be used (e.g which pointing or channels or spectral window that is going to be imaged). This is done with the imager.setdata function. If you don't run this function, you get all of the data.

Here are some examples (refer to the User Reference Manual for more details):

```
im.setdata(mode='channel', nchan=15,      # Select 15 channels, third
             spwid=2, fieldid=3)          #  spectral window  and fourth field
                                          # REMEMBER: everything is 0-based!

im.setdata(mode='channel', nchan=256,     # Select every second channel
             step=2, spwid=2, field=3)    #  to make 256 total channels

im.setdata(mode='velocity', nchan=63,     # Select data based on
             mstart=quantity(20.,'km/s'), #  radio velocity
             mstep=quantity(-100.,'m/s'),
             spwid=2,
             fieldid=3)
im.setdata(mode='opticalvelocity',        # Select data based on
             nchan=63,
             mstart=quantity(20.'km/s'),  #  optical velocity
             mstep=quantity(-100.,'m/s')) #

# Consider a more complex data selection with diferent number of channels and
# different spectral windows:

im.setdata(mode='channel', spwid=[1,2],   # Select spectral windows 2 & 3,
             start=[3,4],                 #  spwid 1: start at 4, select 10 chans
             nchan=[10,12], step=[1,1])   #  spwid 2: start at 5, select 12 chans
                                          #  step by 1 channel for each spwid
```

Other functions like `imager.filter` and `imager.uvrange` can also further reduce the data selected.

5.2.2 Setting Image Parameters

To set the actual imaging parameters use the `imager.setimage` function. This is a **required** function in `Imager` (in fact, the only one presently). This function is passive; it just sets state so that when you tell `Imager` to do something (like make an image), it knows what to do.

This function controls things like image size, cell size, spectral and polarization selection, sampling, phasecenter, and number of facets (for wide-field imaging). Images constructed afterwards (using *e.g.* the `imager.clean` function) will have these parameters.

One important point to note is that the image size does NOT have to be a power of two. Just choose some reasonably composite even number close to the number that you want. **Do not** use a prime number since the Fourier Transform will then be very slow.

Another important point is that the data selection between `setdata` and `setimage` should be aligned, that is, there must be some overlap (e.g., don't select fieldid=1 in `setdata` and fieldid=2 in `setimage`). If you aren't doing any averaging, then the data selection fields (`fieldid, mode, nchan, start, spwid, etc`) should be the same in both function calls; see below for more details.

As an example, to set the image parameters for a simple image cube:

```
im.setimage(nx=600, ny=600,             # Define image to be 600x600 pixels
   cellx=quantity(0.5,'arcsec'),
   celly=quantity(0.5,'arcsec'),        #  with 0.5'' pixels for
   fieldid=2, mode='channel',           #  field 3 and image 20 channels
   nchan=20, start=10)                  #  starting at channel 11.
```

The phase center is that of the specified field center. In addition, each selected channel will be imaged as one channel of the output image.

5.2.3 Channel Selection and Combination

Functions `imager.setdata` and `imager.setimage` both have arguments `nchan, start` and `step`. `setdata` is used to select channels. `setimage` is used to specify how many **output** channels the image will have, and how the input channels should be combined (averaging, gridding).

You should call `setdata` first. After you have called this function, subsequent active `Imager` function calls only have available to them the data that you selected.

Note however, that when you call `setimage`, the specification of the channels is still in an absolute sense. Thus, if you set `nchan=10, start=20` in `setdata`, and you wish to image all of these channels (let us say one image plane per channel), you must **also** set `nchan=10, start=20` in `setimage`.

You might think that you have already selected the 20 channels you want with `setdata` and therefore in function `setimage` setting `start=1` would select the first of those 20 channels. This is **not** the way it works. If you asked for, say, channel 1 with `setimage` when you did **not** select channel 1 with function `setdata`, that channel would be imaged but empty in the output image.

You use the channel selection parameters in `setimage` to specify how the selected channels will be combined and how many output channels the image will have. Basically, there are two sorts of ways that you might like to use the channels that you have selected.

Firstly, in a multi-frequency synthesis image, all of the selected channels are gridded onto the *uv* plane to reduce band-width smearing (which would be incurred if you averaged the channels and then gridded). In this case, the `step` argument is not generally relevant; leave it at 1 if explicitly 'mfs' is used. For example:

```
im.setdata(mode='channel',      # Select 32 channels and start at channel 11
   nchan=32, start=10, step=1)
im.setimage(mode='mfs',         # In mfs mode, create the average of
   nchan=1, start=0, step=1)    #  the 32 channels selected above
```

Or, if you want a straight average:

```
im.setdata(mode='channel',      # select 32 channels starting at channel 11
   nchan=32, start=10, step=1)
im.setimage(mode='channel',     # Select 1 channel - this will be the
   nchan=1, start=10, step=32)  #  average of the 32 channels selected above
```

Secondly, when you make a spectral-line cube, you may wish to select/combine channels in a variety of ways according to the science you want to do. Here are some examples.

```
im.setdata(mode='channel',       # Select 200 consecutive channels starting
   nchan=200, start=10, step=1)  #  with channel 11
im.setimage(mode='channel',      # Form an image plane for each
   nchan=200, start=10, step=1)  #  selected channel


im.setdata(mode='channel',       # Select 200 channels starting with channel
   nchan=200, start=10, step=5)  #  11, pick every fifth channel
im.setimage(mode='channel',      # Form an image plane for each
   nchan=200, start=10, step=5)  #  selected channel


im.setdata(mode='channel',       # Select 200 channels starting with channel
   nchan=200, start=10, step=5)  #  11, pick every fifth channel
im.setimage(mode='channel',      # Each channel of the image is formed by
   nchan=100, start=10, step=10) #  averaging 2 successively selected channels
```

In the above example, channels 11, 16, 21, 26, 31, etc... are selected. During imaging, channels 11 and 16 will be averaged to form output image channel 0. Channels 21 and 26 are averaged to form output channel 1 and so on.

Of course you could also use `mode='mfs'` when combining groups of channels if you want an output image of more than one channel. In this case the combination is done by gridding rather than averaging.

Now to an example when one wants to make a cube image from multiple spectral windows:

```
im.setdata(mode='channel',               # Select 600 data channels from
   spwid=[0,1,2], nchan=[200,200,200],   #  3 spectral windows(200 from each)
   start=[0,0,0], step=[1,1,1],          #  start at channel 1, step by 1,
   fieldid=1)                            #  for field 2
im.setimage(mode='channel',              # Create 150 channels, average
   spwid=[0,1,2], nchan=150,             #  4 channels for each image
   start=0, step=4,                      #  plane.
   fieldid=1)                            #
```

In the above example we select 600 data channels from 3 spectral windows (200 from each spectral window). Then in the `setimage` step we make imager combine 4 channels to make one image channel. Note, the spectral windows should have some overlapping channels for this procedure.

`imager` will figure out what the overlap is and create a continuous image cube from all 3 spectral windows.

Now consider an example in which all data in spectral windows 1 and 2 are selected. Then define an image in terms of velocity values:

```
im.setdata(fieldid=1, spwid=[0,1])     # Select field 2 and sp windows 1 & 2
im.setimage(nx=800, ny=800,            # Image will have 800x800 pixels
   cellx=quantity(0.5,'arcsec'),
   celly=quantity(0.5,'arcsec'),       # 0.5'' on a side, create 30 channels,
   mode='velocity', nchan=30,          #  start at -10km/s and step by
   mstart=quantity(-10.,'km/s'),
   mstep=quantity(1.8,'km/s'),         #  1.8km/s.  Do this for field 2
   spwid=[0,1], fieldid=1)             #  and use both spectral windows
```

Examples for mosaics are given in Section 5.6.5.

5.3 Weighting

The above steps show how to set up the `Imager` tool as desired. In addition, before imaging one may wish to set some values in the MeasurementSet itself. This is necessary for visibility weighting. The visibility imaging weights are computed prior to the actual imaging and stored in a column of the MeasurementSet called "IMAGING_WEIGHT".

The first time an `Imager` tool is attached to a MeasurementSet it initializes the imaging weights to the natural weighting scheme. Thereafter, whatever weighting scheme you last used is the one you will get if you don't explicitly run one of the weighting functions.

The values in the IMAGING_WEIGHT column are set, changed, and examined by the following functions.

- `im.weight` – sets the column using one of natural, uniform, or Briggs (robust) weighting. For the latter two methods, one can specify the field of view over which the minimization of the sidelobes is done (thus achieving what is often called super-uniform weighting). Below are some examples of how to set imaging weights:

  ```
  im.weight(type='natural')               # Natural weighting
  im.weight(type='uniform')               # Uniform over entire field of view
  im.weight(type='uniform', npixels=300)  # Uniform over specified size
  im.weight(type='briggs', rmode='norm',  # An example of Briggs
     robust=0.5)                          #  weighting
  ```

- `im.filter` – applies an optimal filter for a given shape (often called 'tapering'). In the following example we apply a Gaussian filter:

  ```
  im.filter(type='gaussian',              # Apply a Gaussian taper
     bmaj=quantity(4.0,'arcsec'),
     bmin=quantity(2.5,'arcsec'),         #  that is 4x2.5 arcsec in size
     bpa=quantity(60.,'deg'))             #  at a position angle of 60deg
  ```

- `im.plotweights` – plots the imaging weights either as a function of *uv* distance or on a gridded plane.
- `im.sensitivity` – calculates and returns the sensitivity of the resulting weighting both absolutely and relative to natural weighting.
- `im.fitpsf` – calculates the dirty point spread function and returns the best fitting Gaussian.

WARNING: All the weighting schemes modify the MeasurementSet and thus are conserved even after destroying the Imager tool and may no longer be suitable for subsequent imaging. You can of course reset the weighting scheme with the im.weight function.

5.4 Creating Images

It may be helpful to use the im.advise function to help determine the cell size, as well as the number of pixels for a given field of view.

Imagine we want to make an image over a field of view of 10 arcmin. The `im.advise` function will return the maximum pixel size required and number of pixels. If the number of facets required is larger than 1 then one needs a wide-field imaging algorithm as described in the `wide-field` Section below . The recommendations should always be considered in the context of your imaging goals before being used. For example, to use the `advise` function on a split MeasurementSet that contains only calibrated source data:

```
im.open('source.ms')                         # Create the imager tool
im.advise(fieldofview=quantity(10.,'arcmin'))  # Provide advice if FOV=10'

# Logger will report something like:
  Maximum uv distance = 31068.1 wavelengths
  Recommended cell size < 3.31956 arcsec
  Recommended number of pixels = 192
  Dispersion in uv, w distance = 16011.2, 6277.93 wavelengths
  Best fitting plane is w = -0.0543279 * u + -0.418078 * v
  Dispersion in fitted w = 4882.22 wavelengths
  Wide field cleaning is not necessary
```

It is often useful to make a dirty image of the field before deconvolution. Use the `im.makeimage` function to make a dirty image and the point spread function:

```
im.makeimage(type='corrected', # Make a dirty image and store
     image='dirty.image')      #  it in the file called 'dirty.image'

im.makeimage(type='psf',       # Make a PSF image and store it
     image='dirty.beam')       #  in the file on disk called 'dirty.beam'
```

5.5 Deconvolution

At this point, you are ready to deconvolve the point spread function (usually called the dirty beam) from the dirty image. The goal of this step is to correct the dirty image for the sidelobes in the point spread function which arise from the limited Fourier plane coverage in your observation.

The available deconvolution algorithms are CLEAN, MEM and multiscale-CLEAN. Each are described below.

5.5.1 CLEAN deconvolution

Högbom, Clark and multi-scale variants of the CLEAN algorithm are available using the `im.clean` function.

If you use a multi-scale algorithm, you should also set up the scale sizes via the `im.setscales` function.

CLEAN hints

For data using a PSF with high sidelobes, data that hasn't been properly calibrated, or in mosaics in which the PSF used is very different between different fields, the clean residuals may begin to diverge (rather than converge to zero). In these cases, there are a number of controls to help rein the cleaning process:

- Reconcile the visibilities often and regenerate new residuals from the residual visibilities.
- If using one of the multi-field algorithms (mf), the `setmfcontrol` parameters provide additional controls on the process.:
  - cyclefactor: The threshold in a major cycle by default is 1.5 times the product of the peak residual and the maximum outer sidelobe. This can be increased to raise the threshold.
  - cyclespeedup: Number of iterations after which it will increase the threshold in a major cycle by 2.
- Manually set `niter` to a small value and restart clean multiple times. `im` subtracts the model visibility everytime it is started with a model image. This wya, one can force a major cycle every `niter` iterations.
- Use a mask around the region where one believes there is real emission. In confused regions or with strong sources, use interactive masking to change the mask by looking at the residual after some iteration of clean.
- Use a very low gain.
- Re-calibrate, edit the data.

Example: Force major cycles:

```
im.open("orion.ms" )                      # load data into imager (im) tool
im.setdata(mode="none" ,                  # set all channels
        spwid=[0, 1] ,                    # set spectral windows 0 and 1
        fieldid=6)                        # set field 7
im.setimage(nx=500, ny=500,               # set 500 x 500
        cellx=quantity(2.0,'arcsec'),     # set cells to be 2"x2"
        celly=quantity(2.0,'arcsec'),
        stokes="I" ,                      # set Stokes I
        spwid=[0,1],                      # image spectral windows 1 and 2
        fieldid=6)                        # image field 7
im.weight(type="briggs",                  # Briggs weighting parameters
        robust=0,
        fieldofview=quantity(10.,arcsec'))
im.make('field_7c')                       # make a blank image to use as starting model
im.setscales(scalemethod='uservector',    # set multiscale clean scales
        uservector=[0,3,10,50])
im.setmfcontrol(stoplargenegatives=-1)    # continue component search even if
                                          # we've found negative components

  for k in range(10):                        # do a loop to force major cycles based
     im.clean(algorithm='mfmultiscale',   #  on the number of iterations
     niter=2000,                          # in the initial trial, 10000 iterations
                                          # diverged so we choose a 2000 iterations
                                          # (fraction of 10000) to force the major cycle
     gain=0.1, threshold='0.005Jy',       #  force major cycles every 2000 iterations
     model=['field_7c'],
     residual=['field_7c.residual'],
     image=['field_7c.restored'],
     mask=['orion.mask2'])                # use a tight mask around emission
                                          # you can generate this with interactivemask
```

5.5.2 Maximum Entropy Method Deconvolution (MEM)

Maximum entropy and maximum emptiness methods are available. Functions that implement MEM are `im.mem` and tt dc.mem.

```
  im.setdata(mode='channel',        # Select 200 consecutive channels starting
     nchan=200, start=10, step=1)  #  with channel 11
  im.setimage(mode='channel',       # Form an image plane for each
     nchan=200, start=10, step=1)  #  selected channel
  im.mem(algorithm='entropy',       # Select maximum entropy algorithm
     niter=10,                      # niter is smaller than that used in CLEAN as
                                    # it is not the components but the number of
                                    # iterations to maximize the entropy
     sigma=quantity(0.1,'Jy'),      # target noise to achieve in residual image
     targetflux=quantity(10.,'Jy'),# an estimate of the total flux in the image -
                                    # if uncertain, start with 1/10th to 1/2 of the
                                    # expected source flux
     constrainflux=F,               # set this to 'T' if you want the total flux fixed
                                    # to the target flux specified above
```

```
    prior='priorimage',          # if available, an image that gives some knowledge of
                                 # how the flux is distributed. This is not necessary.
    image=['maxen.image'],       # output images
    model=['maxen.model'],
    residual=['maxen.resid'],
    mask=['mask.im'])            # If needed you can specify a region to constrain the
                                 # flux
```

MEM hints

How to set up MEM so it can converge; how to decide when to stop MEM:

- Run MEM with only a few iterations (niter=5-10) and no mask. The image will be poor but you can use it to define a mask. This is critical as MEM can easily diverge if you have complicated emission and no mask. Go through this procedure, increasing niter a bit until you have a good mask.
- Start with about half of the flux density expected in the final image. This allows MEM adequate cycles to achieve convergence. If you start with the full known flux density MEM may not converge to the correct flux. If you start with too little flux density, MEM may obtain a divergent flux. If in doubt, start with a small value, and watch the convergence. If you get no convergence on the final map flux with 20 to 50 iterations, increase the input Target Flux until you see MEM behaving reasonably. Note: when using a single-dish image as the starting model, the target flux should be set to the value of the single-dish flux since this is a known, rigid constraint.
- Use a sigma input that is about or lower than what you expect for the final image. This is not too critical. If it is clearly too high, MEM will stop before it has obtained a good image. You want to make sure sigma is low enough that MEM continues to deconvolve the image until it has converged to a good solution. You will probably stop MEM manually anyway based on the displayprogress plot.
- When you have arrived at good initial inputs to MEM and you have a good mask, the displayprogress plot will show discontinuities between each major cycle but steadily increasing flux. If, during a cycle, the peak flux and sigma show signs of instability (they get a small bump). This is the first sign that MEM is digging too deep. You will want to stop MEM before the peak and sigma become unstable.
- If you stop the iterations before MEM gets unstale will get you a nice flat residual map and a good deconvolved image with low background. If you allow MEM to dig deeper, you will get increased background, stripes and a worse residual image.
- When in doubt, make lots of images and compare. Also, compare MEM with CLEAN or multi-scale CLEAN.

The deconvolution methods do not keep a list of CLEAN components in sequence. Instead the CLEAN components are immediately added to the model image. This allows interoperability of the CLEAN and MEM algorithms as well the deconvolution functions in the `Deconvolver` tool. It also allows editing of the resulting images using the Image tool.

The deconvolution functions can return the residual and restored images, as well as the updated model image. In addition, there are `imager.residual` and `imager.restore` functions that also compute residual and restored images. The distinction is that in the former, the residual is that calculated by the deconvolution method at the end of iteration, whereas the latter returns the residual calculated by going back to the original visibility data (except for the cases of multi-field and wide-field algorithms).

An example of using Clark CLEAN deconvolution with the `imager` tool is given below:

```
im.clean(algorithm='clark',     # Clean a single field with Clark clean
     niter=2000, gain=0.2,      #  using 2000 iterations, set loop gain=0.2,
     model=['model_2000'],      #  The model image is called model_2000,
     residual=['residual_2000'],#  the residual image is residual_2000,
     image=['restored_2000'])   #  the final restored image is restored_2000.
                                #  All images are written to disk.
```

If the model image does not pre-exist it will be created and filled with the CLEAN delta function components found. If it does pre-exist (it may be non-empty (perhaps the result of a prior deconvolution) its Fourier transform is first subracted. This is also how you would continue to CLEAN deeper from a prior run of `clean or mem`.

Note that the model image is updated when this function finishes. So if it was non-empty on input, you should save a copy first, if you wish to preserve that model.

5.5.3 Multi-scale CLEAN

Background on CLEAN versus Multi-scale CLEAN: Delta function CLEAN algorithms, such as the Clark or Hogbom algorithms, must use a modest gain factor (traditionally 0.1) in order to image extended emission in a reasonable manner. Otherwise, emissions and sidelobes get confused and error striping can result.

Multi-scale CLEAN algorithms attempt to recognize that the emission in the sky usually comes in a variety of scale sizes; it decomposes the image into Gaussians of the specified scale sizes. This means it does not spend huge amounts of time CLEANing large extended sources pretending that they are really collections of delta functions. However, note: because you are CLEANing multiple scale sizes, multi-scale CLEAN generally takes longer than CLEAN to run.

The `im.setscales` function allows you to choose these scales, either with an automatic method (you say how many scales you want) or by direct specification of the scale sizes in pixels.

Multi-scale CLEAN does not suffer from the same problems as delta function CLEAN and can use a larger value of the gain factor. Sometimes an oscillatory pattern will occur in the total cleaned flux with the peak residuals bouncing back and forth between positive and negative. If this occurs, try reducing the gain factor or try reducing the size of the largest scale.

As an example:

```
im.setdata(mode='channel', nchan=15,   # Select 15 channels,
     start=10, step=1,                 #  starting with channel 11, use
     spwid=2, fieldid=3)               #  spectral window 3 and field 4
im.setimage(nx=600, ny=600,            # Set up the imaging parameters
     cellx=quantity(0.5,'arcsec'),     #  Combine all channels into a single
     celly=quantity(0.5,'arcsec'),
     mode='mfs',                       #  plane using multi-frequency synthesis.
     spwid=2, fieldid=3))
im.setscales(scalemethod='nscales',    # Set up multi-scale components
     nscales=3)                        # Specify number of scales, allow imager
                                       #  to determine the scale sizes.
im.clean(algorithm='msclean',          # Deconvolve using multi-scale CLEAN
           model=['model'],            #  Call the model image 'model' and
           image=['restored'],         #  restored image 'restored'
           niter=100, gain=0.3)        #  clean down 100 iterations using a gain loop
                                       #  of 0.3.
```

Multi-scale CLEAN hints

- Use im.setscales to set scalemethod='uservector'. Look at the general distribution of emission and choose scales that are appropriate for the emission (e.g., start with 0 and 3 pixels; add scale sizes until the largest scale is about the size of the largest diffuse clump in the map). For example, assume you have emission that covers a 4 arcminute area, and there is significant diffuse emission that looks like it has a size scale of about 30 - 40 arcseconds. With 1 arcsecond pixels, choose deconvolution scales of [0,3,10,30] pixels. You can go smaller (e.g., [0,2,5,10]), but don't go much larger.
- The more scale sizes you choose, the more memory it will take to deconvolve the image. Be conservative.
- The largest scale you choose should fit in any deconvolution masks you set (or multi-scale CLEAN will not be able to use the largest scale).
- If you have a mosaic where the fields are not sampled very quickly (e.g., field uv coverages differ somewhat) then the PSF will also differ between fields, sometimes significantly. CASA finds an average PSF that is common to all fields and cleans down to a certain level in each major cycle. In almost any mosaic that is taken with current instrumentation, fields are observed at different times and can be observed at slightly different frequencies. Thus, the PSFs between fields will differ enough that convergence in multi-scale CLEAN can be challenging. Try: in imager.setmfcontrol, start with cyclefactor=3 and cyclespeedup=500.
- While multi-scale CLEAN is running, watch the progress. If there is significant large-scale emission, initial major cycles will only get flux on the largest scale. As the number of cycles increase, more scales will begin to accumulate flux. Eventually, smaller scales will go negative; don't panic. The smaller scales are compensating for errors in the large scale flux distribution. Stop the iterations when you see minimal or negative flux on all scales. We recommend using the im.setmfcontrol method with the stoplargenegatives=-1 setting. Notice as the number of iterations increases, the final threshold obtained in the residual image decreases.

5.6 Mosaicing

The Fourier transform relationship between the Fourier plane and the image plane must include the primary beam:

$V(u) = \int A(x)I(x)e^{-i2\pi ux}dx$

Where $V(u)$ is the measured visibilities; $A(x)$ is the primary beam response; and $I(x)$ is the true brightness distribution on the sky.

Hence, given the image, one can simulate the corresponding *uv* data. However, given the data and desiring the image, we have an inverse problem to solve.

Early attempts at mosaicing treated each field independently, deconvolving and self-calibrating each, and then sewing the overlapping fields' images together via the basic mosaicing equation:

$I(x) = \frac{\sum_f I_f(x)A_f(x)}{\sum_f A_f^2(x)}$.

Where the subscript f refers to each field in the mosaic. However, Cornwell (1988) demonstrated that better results can be obtained via a simultaneous deconvolution of the data from all the fields. This simultaneous deconvolution was achieved by using maximum entropy (MEM) or maximum emptiness as a solution engine to solve the inverse problem. Total power could be added as additional fields with their own primary beam. However, MEM's positivity bias, which is detrimental to low SNR imaging, led to a search for other algorithms to image multi-field data.

Sault et al. (1996) have implemented mosaicing algorithms which can use either CLEAN or MEM for simultaneous deconvolution.

5.6.1 The CASA Mosaicing Algorithm

Cornwell, Holdaway, and Uson (1994) proposed the following mosaicing algorithm: generate the mosaic of the dirty images and a single approximate point-spread function (PSF), and then proceed with any conventional single field deconvolution algorithm. For high-quality Fourier-plane coverage and similar PSF's for all fields in the mosaic, this approach is not limited by the differences in the approximate PSF and each field's actual PSF until the possible image dynamic range exceeded a few hundred to one.

CASA takes this approach to mosaicing a step further: perform an incremental deconvolution of the residuals with the approximate PSF, with an exact subtraction of the cumulative model brightness distribution at the end of each incremental "major cycle" (similar in concept to the major cycles of the Clark CLEAN).

If all of the fields are observed with many short snapshots over and over again (this is the preferred way to make a mosaic observation) then each field will have similar Fourier coverage and hence similar synthesized beams. An approximate PSF can be created which is a fairly good match to the actual PSF of each of the fields. Also, if the sky-coverage of the observed fields is Nyquist or better, then the approximate, shift-invariant PSF will be a reasonable match to the actual PSF of sources at various locations across the mosaic. The residual visibilities from each field can be transformed and mosaiced to make a single residual mosaic image. This mosaic image can be deconvolved with the deconvolution method of your choice; for example, with Clark CLEAN, Multiscale CLEAN, maximum entropy, or maximum emptiness.

The deconvolution algorithm cannot deconvolve arbitrarily deeply, because at some level the discrepancies between our approximate shift-invariant PSF and the true PSF at any location in the image will become apparent, and we will start "cleaning" error flux. Hence, we need to stop deconvolving when we have gotten down to the level of these PSF discrepancies. At this point, we take the part of the model brightness distribution we have just deconvolved and calculate model visibilities (using the measurement equation) and subtract them from the (corrected) data visibilities. To the extent that the primary beam and sky pointing are exact, the visibility subtraction is also exact. The residual visibilities can then be re-mosaiced, but the peak residual is at a much lower level. The process of deconvolving with the approximate, shift-invariant PSF then continues, and another increment to the model brightness distribution is formed, removed from the remaining residual visibilities, and added to the cumulative model brightness distribution. Borrowing from the Clark CLEAN's terminology, we call each cycle of incremental deconvolution and exact visibility subtraction a "major cycle".

5.6.2 How to Set Up a Mosaic Image

5.6.2.1 Set the Data Fields

Mosaicing can be a time consuming process, so it may be worthwhile to make a restricted version of the mosaic first. For example, you may want to image a few fields at lower resolution to reduce the number of pixels you are imaging. Eventually, you will want to image most or all of the observed fields. Use the `setdata` function to restrict the data over which you will generate your image, e.g.,

```
  im.setdata(fieldid=range(0,4))       # Select first 4 fields
```

Set the Image

One of the fields must be specified in `im.setimage` to provide the direction of the resultant image's reference pixel. For example, with a 25 pointing (5 x 5 raster) observation, field 13 could be the central field:

```
  im.setdata(fieldid=range(0,25))    # Select all 25 pointings
  im.setimage(nx=256, ny=256,        # Create a 256x256 resultant image
     cellx=quantity(3.,"arcsec"),    # Choose a cell size of 3x3 arcseconds
     celly=quantity(3.,"arcsec"),
     fieldid=12)                     # Select field 13 as the center
```

5.6.2.2 Setting the Voltage pattern (primary beam)

We must account for the primary beam pattern when imaging the same region but from different pointings. CASA currently has primary beam patterns stored for:

1. ATCA

2. GBT
3. GMRT
4. HATCREEK
5. NMA
6. NRAO12M
7. NRAO140FT
8. OVRO
9. VLA
10. WSRT
11. OTHER

in addition, there are specific beams available for each of the following:

1. ATCA_L1, ATCA_L2, ATCA_L3, ATCA_S, ATCA_C, ATCA_X, GBT, GMRT, HATCREEK, NRAO12M, NRAO140FT, OVRO, VLA, VLA_INVERSE, VLA_NVSS, VLA_2NULL, VLA_4, VLA_P, VLA_L, VLA_C, VLA_X, VLA_U, VLA_K, VLA_Q, WSRT, WSRT_LOW

```
im.setvp(dovp=T)  # dovp=do the voltage pattern correction
                   # this will default to use the voltage pattern for the
                   # telescope used and frequency observed. If no such
                   # default exists, a warning is issued.
```

although rare, if a voltage pattern is not provided, you can specify your own voltage pattern and bind it to the telescopes in a MeasurementSet by using the `vpmanager` (voltage pattern manager. The Vpmanager will produce a table describing the different telescopes' voltage patterns, and this table can be used by Imager.

See the vpsection of the User Reference Manual for details:

```
http://aips2.nrao.edu/stable/docs/user/SynthesisRef/node228.html
```

5.6.2.3 Mosaic Weighting

If you use "uniform" or "briggs" weighting, the weighting details will depend upon the way the data are gridded. However, if all fields are specified in function `setdata`, then the weights from all fields will be gridded onto a single grid for the purposes of calculating the weights. This is valid for natural weighting but will not be valid for other types of weighting. You probably want to weight the data on a field-by-field basis:

```
  im.weight(type="uniform",mosaic=T)  # This will weight each field separately
```

Mosaic Options

For mosaicing, there is a key option which can be used. In particular, the gridding can use the primary beam as the convolution function. This is achieved using the option `ftmachine="mosaic"`.

```
  im.setoptions(ftmachine='mosaic')   # do a mosaic gridding
```

The normal gridder uses a spheroidal function to grid the data. For mosaics, using the Fourier transform of the primary beam as the gridding function has some advantages.

After a major cycle, the primary beam correction using the mosaic gridder leaves the (uncleaned) noise floor flat while it gets the deconvolved component scaled properly for flux scale. The use of the spheroidal gridding function, after correcting for the primary beam, tends to lift the noise at the edge of the beams and thus is non-optimal for images with emission extending to the edge of the mosaic.

In addition, controls are available using the `setmfcontrol` function to help adjust the cycle parameters (the conditions for ending the deconvolution cycles) for multi-field (and wide-field) imaging.

```
  # Set the image plane flux scale type to "SAULT"
  # Set a cutoff for the minimum primary beam level to use
  # constpb is used for SAULT weighting to set the flux scale constant above
  # this level.
  im.setmfcontrol(scaletype='SAULT', # This selects Sault weighting which uses a
                                     # primary beam function that is flat across most
                                     # of the mosaiced image but is attenuated at
                                     # the edges of the mosaic so that the noise level
                                     # does not become excessive there
                  minpb=0.07,        # minimum primary beam level to use in each field
                  constpb=0.4)       # The noise is constant above this primary
                                     # beam level
```

5.6.3 Mosaic Deconvolution

To image and deconvolve, use either imager's CLEAN or MEM functions. Only algorithms with the "mf" prefix will perform multi-field imaging correctly (i.e. algorithm "clark" will grid the data from all specified fields onto the same grid, resulting in a very confused image. CLEAN's mosaicing methods include `mfclark`, `mfhogbom`, and `mfmultiscale`, while MEM's mosaicing methods include `mfentropy` and `mfemptiness`. Some hints for CLEAN and MEM are provided at 5.6.4.2 and 5.6.4.3, respectively. A full example of how to create a mosaic image is given below. A detailed script is given at the end of this chapter.

```
im.open('data.ms')  # load data into imager
                                                # choose all fields (1-10) and all channels
im.setdata(mode='channel', nchan=62, start=0,   # restrict the data to the
     step=1,fieldid=range(0,10), spwid=[0,1])   # first two spectral windows
im.setimage(nx=640, ny=256,                     # select the output image size
     cellx=quantity(0.5,'arcsec'),              # and cell size properties
     celly=quantity(0.5,'arcsec'),stokes='I',
     mode='channel',
     nchan=18,start=2,step=5,fieldid=0,
     spwid=[0,1])                              # do 18 channels starting
                                               # at channel 3 and averaging 5
                                               # channels; use field 1 as the mosaic
                                               # field center
im.setvp(dovp=T)                               # do primary beam correction
im.weight(type="uniform",mosaic=T)             # use uniform weighting for each field
im.setoptions(padding=1.0, ftmachine='mosaic') # do mosaic gridding
im.clean(algorithm="mfclark" , niter=1000,     # use mfclark algorithm
     gain=0.1, threshold=quantity(0.005,'Jy'),
     model=["src.all.cln.model"],              # stop cleaning at a threshold
                                               # of 0.005 Jy
     image=["srcmos"], residual=["src.all.cln.resid"])
```

5.6.4 Mosaic Details

5.6.4.1 Controlling the Major Cycles

The key to making the incremental deconvolution in CASA multi-field imaging successful lies in controlling just how deeply to deconvolve in the major cycles. The control parameters discussed here can be set with`im.setmfcontrol`. Deconvolving too deeply with the approximate PSF will waste computation time as slightly different PSF sidelobes add and subtract components. Not deconvolving deeply enough also causes problems as it necessitates more major cycles, again slowing down the computation time.

- by increasing the `cyclefactor`. The major cycle cleaning threshold is a fraction of the peak image residual. That residual fraction is determined by the cyclefactor multiplied by the peak negative sidelobe of the approximate PSF. Stopping the major cycle cleaning sooner can be accomplished by increasing the cyclefactor. Values ranging between 2 and 4 are common.
- by decreasing the `cyclespeedup`. What if the `cyclefactor` is set too low? The cycle threshold as calculated above can be made to drift upwards by setting the `cyclespeedup`. The threshold will double every `cyclespeedup` iterations in the major cycle until the major cycle stops. If the `cyclespeedup` is less than or equal to 0.0, no adjustments to the calculated cycle threshold will be made.

In addition to these cycle control parameters, which are applicable to mosaicing with CLEAN, Multi-Scale CLEAN, MEM, or Maximum Emptiness, there are two more control arguments set by `im.setmfcontrol` which are applicable only to Multi-Scale CLEAN. These are `stoplargenegatives` and `stoppointmode`, which are discussed below.

5.6.4.2 Multi-Scale CLEAN Hints

Sometimes in the first few iterations of Multi-scale CLEAN in the mosaicing context, the largest scale will be dominated by large negative residuals (i.e. this is just the negative bowl, integrated over the area of the largest clean-component scale). One way to fix this is to make the largest scale smaller. Another way is to use a tighter mask which excludes finding large scale components in the bowl region. And a third, ad hoc way is to stop the major cycle when a negative component is found on the largest scale. This allows the exact subtraction to proceed, often resulting in a reduced bowl level. Stopping the major cycle upon encountering a negative component on the largest scale should only be performed on the first few cycles, as during subsequent cycles small amplitude large scale components may be required to make adjustments in the image. The number of cycles for which stopping when a negative component is found on the largest scale can be controlled by the parameter `stoplargenegatives` in `imager.setmfcontrol`. As smaller scales may require negative components to correct for errors made by "over-cleaning" in the larger cycles, no restriction should be placed on negative components from smaller size scales.

5.6.4.3 MEM Hints

If there are bright unresolved or barely resolved sources in the field, it may be advantageous to perform a Clark Clean down to the level of the peak extended emission, or include a component list in the model, because MEM does not work well on bright point-like sources.

The maximum entropy/emptiness algorithm has been modified to fit into the incremental deconvolution/major cycle framework adopted by mosaicing in CASA. These algorithms deal with both the incremental brightness distribution, which it seeks to solve given the approximate PSF and the residual mosaic image, and the cumulative brightness distribution for the calculation of the entropy function and its gradients. When maximum entropy starts up, it typically takes the algorithm four or five iterations to "find itself" and get the control parameters set to achieve the proper balance between the gradients of the entropy and the χ^2. Once a good balance is struck, the algorithm makes marked progress towards convergence. At the end of a major cycle, the relevant control parameters are saved and used for the next major cycle.

An example similar to the single field case but with the algorithm changed to 'mfentropy' for multiple fields:

```
im.setdata(mode='channel',         # Select 200 consecutive channels starting at 11
     fields=range(0,10),           # Select 10 mosaic fields
     nchan=200, start=10, step=1)  #
im.setimage(mode='channel',        # Form an image plane for each selected channel
     nx=300,ny=300,                # Create a 300x300 pixel image with
     cellx=quantity(2.,'arcsec'),
     celly=quantity(2.,'arcsec'),  # 2"x2" pixels
     nchan=200,start=10,step=1,    # use all channels selected in setdata
     field=5)                      # Use field 6 as the mosaic center
im.mem(algorithm='mfentropy',      # Select maximum entropy algorithm for mosaic
     niter=10,                     # niter is smaller than that used in CLEAN as
                                   # it is not the components but the number of
                                   # iterations to maximize the entropy
     sigma=quantity(0.1,'Jy'),     # target noise to achieve in residual image
     targetflux=quantity(10.,'Jy'),# an estimate of the total flux in the image -
                                   # if uncertain, use 1/10 of the expected flux
     constrainflux=F,              # set this to 'T' if you want the total flux fixed
```

```
                                # to the target flux specified above
     prior='priorimage',         # if available, an image that gives some knowledge of
                                # how the flux is distributed. This is not necessary.
     image=['maxen.image'],      # output images
     model=['maxen.model'],
     residual=['maxen.resid'],
     mask=['mask.im'])           # If needed you can specify a region to constrain the
                                # flux
```

5.6.4.4 Flux Scale Images

When correcting for the effects of the primary beam to achieve an accurate and uniform flux scale across the image (i.e. by dividing by the primary beam in the case of a single field observation), the noise and errors at the edge of the mosaic sky coverage are amplified. The noise amplification distracts from the visual beauty of the image and may complicate the image's display and interpretation.

Sault et al (1996) endorse a different image plane weighting in the mosaicing which results in a constant noise level across the image, but has a variable flux scale, e.g., for Nyquist sampled mosaics, flux scale decreases outside of the overlap region of the mosaic field - just as the single field fluxscale decreases toward the edge of the primary beam.

In CASA an image plane weighting similar to Sault's scheme has been implemented, but the noise goes to zero outside the region covered by the primary beams. The flux scale is position dependent, but it is mostly flat over most of the mosaic sky coverage (depending on how `constpb` is set within the `setmfcontrol` function and how much overlap there is between pointing centers). In order to get a constant flux image across the mosaic, one must divide by the fluxscale image (See Section 7 for details on manipulating images). The flux scale images can be created by setting the `fluxscale` argument in Imager's `setmfcontrol` function (in case an inverse-Sault weighting is needed to correct the image). Regions outside the multi-field primary beam pattern will have a zero value.

5.6.4.5 Masks with Mosaic Fields

Routinely in single field deconvolution, only the inner quarter of an image is deconvolved so that the sidelobes from this region can be correctly subtracted from the entire image. However, in the multi-field case, such a restriction usually does not exist. The major cycles only deconvolve down to a certain level, fixed by the sidelobe characteristics of the PSF. After that, the exact subtraction of the deconvolved flux is carried out. Typically, the exact subtraction is performed by multiplying the brightness distribution by a field's primary beam, convolving by that field's exact PSF, multiplying by the primary beam again, and subtracting from the previous cycle's residual mosaic. The two primary beam multiplications ensure that the far out effects of the PSF, which will not be correct due to the full-image deconvolution, will not effect the model brightness distribution.

If no mask is used, the major cycle deconvolution algorithms create a mask from the generalized primary beam pattern of all observed fields, with zero outside the outermost fields' primary beam main lobes. If you don't want this mask for some reason, you should supply your own mask image.

5.6.5 An Example Mosaic Script

The following script makes an interferometer-only mosaic image from the multi-field test measurementset which is distributed with CASA:

```
im.open('XCAS.ms')
# Use all the data for the mosaic
im.setdata(mode="none",
           spwid=[1:2], fieldid=[1:7])

# Use the first field as the image center
im.setimage(nx=256, ny=256, cellx="3arcsec",
            celly="3arcsec", stokes="I", doshift=F,
            mode="mfs", spwid=[1:2], fieldid=1,  facets=1)

# Weight each field individually
im.weight(type="uniform",mosaic=T)

# Use all the data for the mosaic
im.setdata(mode="none", nchan=1, start=1, step=1,
           spwid=[1:2], fieldid=[1:7])

# Using the mosaic gridder that uses the FT of the Primary Beam as the gridding function
im.setoptions(ftmachine='mosaic')

# Use the voltage pattern (primary beam) - it will lookup the telescope and use the
# appropriate PB for the observing frequency
im.setvp(dovp=T, usedefaultvp=T, dosquint=F)

# Make a MEM image  (using  mask image) - set names for results
maskname = ['mem.mask']              # maskname will be mem.mask
modname = ['mem.model']              # modname will be mem.model
imgname = ['mem.image']              # imgname will be mem.image
resname = ['mem.resid']              # resname will be mem.resid
scalename = ['mem.scale']            # scalename will be mem.scale

# set some multi-field control parameters.
im.setmfcontrol(cyclefactor=3.0,  # set to 3.0* max sidelobe * max residual
   cyclespeedup=20.0,             # threshold doubles in this many iterations
   fluxscale=scalename)           # name of fluxscale image
im.mem(algorithm='mfentropy',     # use multi-field Maximum Entropy algorithm
   niter=80, sigma='0.001Jy',     # number of iterations and target sigma
   targetflux='10.0Jy',           # target flux for final image
   constrainflux=F,               # constrain image to match target flux
   displayprogress=T,  fixed=F,   # display progress, don't keep model fixed
   complist='', prior='',         # set these to blank
   mask=maskname,                 # setup output file names
   model=modname,
   image=imgname,
   residual=resname)
```

```
# Make a multi-scale CLEAN image  for comparison
modname = ['msclean.model']         # modname will be msclean.model
imgname = ['msclean.image']         # imgname will be msclean.image
resname = ['msclean.resid']         # resname will be msclean.resid
scalename = ['msclean.scale']       # scalename will be msclean.scale

im.setscales(scalemethod='uservector',  # method by which scales are set
   uservector=[0.0, 3.0, 10.0, 20.0])   # vector of scale sizes in pixels

### please note the use of parameter stoplargenegatives
### which is set to false to let multi-scale clean to continue despite hitting
### a negative on the largest scale

im.setmfcontrol(cyclefactor=3.0,  # set to 3.0 * max sidelobe * max residual
   stoplargenegatives=-1,         # continue despite negatives
   fluxscale=scalename)           # name of fluxscale image
im.clean(algorithm='mfmultiscale',# use multi-field multi-scale algorithm
   niter=1000, gain=0.6,          # number of iterations and loop gain
   threshold='0Jy',               # stop cleaning at this threshold
   displayprogress=T, fixed=F,    # display progress, don't keep model fixed
   complist='',                   # name of component list
   mask=maskname,                 # set names for produced files
   model=modname,
   image=imgname,
   residual=resname)

# unload data from tool
im.close()
```

5.7 Imaging combined single dish/synthesis data

5.7.1 Methods

Below is a list of current methods used to combine single dish and synthesis data (Stanimirovic 2002).

- **Feathering** in the image domain (IMMERGE in miriad, IMERG in AIPS, im.feather in CASA). Images are feathered together in the Fourier plane. Intermediate size scales are down-weighted to give interferometer resolution while preserving single-dish total flux density.

  This is the fastest and least computer intensive way to add short-spacings. Also the most robust way relative to the other 3 methods which all require a non-linear deconvolution at the end.

- **Linear combination** - the single-dish image and the interferometer dirty image are linearly combined in the image plane, a composite beam is constructed, and the resulting composite image is deconvolved with the composite beam. Existing packages don't directly support this method but it can be done via image manipulation and subsequent deconvolution.

Straight linear combination is not optimal but this method has the advantage that it does not require either Fourier transformation of the single-dish data which can suffer severely from edge effects, nor deconvolution of the single-dish data which is especially uncertain and leads to amplification errors.

- **Model Image** - use the single-dish image as the starting model and subtract this model from the uv data. In the shortest uv spacing, the single-dish-sampled structure will be preserved in the model information. In the uv overlap region, the source structure can be modified from the single-dish flux density distribution during deconvolution using the interferometer uv data. In the interferometer-only region of the uv-plane, the deconvolution will proceed as usual with no single-dish constraints.

  this works effectively with good uv overlap (i.e., large single-dish).

- **Full Joint Deconvolution** - single-dish and interferometer data are gridded together. All regions of the uv-plane are jointly deconvolved.

  Theoretically the best way to do short spacing correction. This method depends heavily on a good estimate of the interferometer and single-dish noise variances.

Note: These techniques rely on images (data and masks) having the same number of axes. If errors are encountered with complaints about image shapes, use the image.summary function to look at the input images axes.

5.7.2 Feathering

In the image feathering technique, first the calibrated single-dish image is corrected for the beam response and the calibrated interferometric data are Fourier transformed and deconvolved to create an interferometer-only image.

The feathering technique does the following:

- The single-dish and interferometer images are Fourier transformed.
- The beam from the single-dish image is Fourier transformed (FTSDB(u,v)).
- The Fourier transform of the interferometer image is multiplied by (1-FTSDB(u,v)). This basically down weights the shorter spacing data from the interferometer image.
- The Fourier transform of the single-dish image is scaled by the volume ratio of the interferometer restoring beam to the single dish beam.
- The results from c and d are added and Fourier transformed back to the image plane.

The term feathering derives from the tapering or down-weighting of the data in this technique; the overlapping, shorter spacing data from the deconvolved interferometer image is weighted down compared to the single dish image while the overlapping, longer spacing data from the single-dish are weighted down compared to the interferometer image.

The tapering uses the transform of the low resolution point spread function. This can be specified as an input image or the appropriate telescope beam for the single-dish. The point spread function for a single dish image may also be calculated using `makeimage`.

Advice: Note that if you are feathering large images, be advised to have the number of pixels along the X and Y axes to be composite numbers and definitely not prime numbers. In general FFTs work much faster on even and composite numbers. You may use subimage function of the image tool to trim the number of pixels to something desirable.

Note: This method is analogous to the AIPS IMERG task and the MIRIAD immerge task with option 'feather'.

Feathering Example:

```
im.setvp(dovp=T)                  # Do primary beam correction; it will use the default
                                  # primary beam for the single-dish telescope
                                  # and frequency
im.feather(image='feathered.image'# Resulting, combined image
  highres='synthesis.image',      # Synthesis image
  lowres='singledish.image')      # Single-dish image
                                  # If the beam response of the single-dish telescope
                                  # is not stored in AIPS++ then, one can optionally
                                  # specify the 'lowpsf' image if  available.
```

The feather task works on both single plane images and on multi-plane images (data cubes). The synthesis and single-dish images must have the same number of axes however; for example, default images in CASA have axes of: 1) Direction (e.g., RA) 2) Direction (e.g., Dec), 3) Stokes (e.g., I), and 4) Spectral (e.g., frequency or velocity). These axes can be manipulated (added or deleted) as necessary using the `image` tool; this tool has not yet been ported to CASA .

5.7.3 uv-plane Combination

There are two principal methods for the uv-plane combination of single-dish and synthesis data.

- Use the single-dish image as a starting model for the deconvolution (CLEAN or MEM).
- Full joint deconvolution of the datasets.

Use single-dish image as starting model

In this first technique, a non-linear combination of the synthesis and single-dish data is performed in the uv plane. In the CASA implementation, the single-dish image is used to make a model (clean component) image which is then used as the starting point for the deconvolving algorithm to interpolate the missing short baselines.

During the deconvolution step in the CLEAN process, the interferometric data is extrapolated to shorter spacings. When starting CLEAN with a model from a single-dish image, the single-dish image is Fourier transformed and data in the area of uv-overlap is compared in a major cycle. The uv spacings provided by the single-dish image thus provides information (constraints) which help to prevent CLEAN from over-extrapolating in this region of the uv-plane.

The `imager` function, `setsdoptions` can be used to set a factor by which to scale the single-dish image, if necessary (typically to convert from K to Jy/beam).

The `sdpsf` parameter (optional) should be used if an external PSF image of the single dish is needed to calculate the beam parameters of the primary beam of the dish. This is usually needed if the single-dish image is from a non standard telescope or the beam is not in the CASA system; see the example in `feather` - 5.7.2 for generating a beam in this fashion.

A mask image can be provided to the clean algorithm. This mask image helps guide the clean (and mem) algorithms and should be chosen carefully based on the 'known' emission.

Eventually, it will be possible to create a mask image from an existing image via an `interactivemask` task. But this is not implemented yet. In CASA you will need to use the `makemask` task to create simple clean boxes.

There are two ways to get a model from a single-dish image:

- Directly convert the single-dish image from Jy/beam to Jy/pixel.
- Deconvolve the single-dish image with MEM or multiscale-CLEAN (using large scales only) to obtain a model image (Jy/pixel).

Example

Example: Directly convert the single-dish image to a model and use this as a starting model for the deconvolution.

```
im.open(filename='n4826_both.ms')    # Create imager tool using synthesis data
im.setdata(fieldid=range(0,7),       # Select mosaic fields 1-7
           spwid=[0,1,2])            # Select spectral windows 1-3
im.setimage(nx=256, ny=256,          # Resultant image will be 256x256
   cellx=quantity(1.,'arcsec'),
   celly=quantity(1.,'arcsec'),      #   with 1" pixels
   stokes='I',                       # Resultant image will be Stokes I
   mode='channel',                   # Define image planes by channel
   nchan=30,                         #   30 planes
   start=46,                         #   Starting with channel 47
   step=4,                           #   Averaging 4 channels
   fieldid=0,                        #   Use fieldid=1 as the phase center reference
   spwid=[0,1,2])                    #   Use spectral windows 1-3
im.setmfcontrol(scaletype="NONE",    # Set some multi-field processing parameters
                                     #   NONE indicates no scaling
   minpb=0.1)                        #   Level at which the primary beam will be applied
im.setvp(dovp=T)                     # Do primary beam correction

                                     # Make starting model image from single-dish image
im.makemodelfromsd(sdimage='n4826_12mchan.im', # specify single-dish image
   modelimage='n4826_joint1',        # specify name of output model image
   maskimage='n4826.mask')           # specify make image

                                     # joint deconvolution and clean
im.clean(algorithm='mfclark',        # Use multi-field clark algorithm
   model=['n4826_joint1'],           # Use model image generated above as the initial model
   gain=0.2, niter=500)              # set gain and iterations
                                     # you can specify a clean mask if desired - see
                                     # Section 5.3.4.1 on interactivemask
                                     # Note: if the output image name isn't specified, the
                                     # restored image name will be the model image name,
                                     # 'n4826_joint1' appended with '.restored'

im.close()                           # Close tool
```

When combining single-dish images (in feather or uv-plane combination), the single-dish image can be scaled by a factor (typically to convert from K to Jy/beam) using the `setsdoptions` function.

```
  im.setsdoptions(scale=0.5)
```

Full Joint Deconvolution

This functionality is currently under construction.

5.7.4 References for Single-Dish and Interferometric Data Combination

1. For a review of techniques see: Stanimirovic, S. 2002, astro-ph/0205329, "Short-Spacings Correction from the Single Dish perspective".

2. Emerson, D. T. 1974, *MNRAS*, **169**, 607

3. Helfer et al., 2003, *ApJS*, **145**, 259, Appendix B

4. Holdaway, M. A. 1999, in ASP Conf. Ser. 180, Synthesis Imaging in Radio Astronomy II, ed. ed. G. B. Taylor, C. L. Carilli, & R. A. Perley (San Francisco: ASP), 401

5. Stanimirovic, S., Staveley-Smith, L., Dickey, J. M., Sault, R. J., & Snowden, S. 1999, *MNRAS*, **302**, 417

6. Vogel, S. N., Wright, M. C. H., Plambeck, R. L., & Welch, W. J. 1984, *ApJ*, **283**, 655

5.8 Wide Field Imaging

The problem of imaging interferometric data for wide fields involves correcting for the non-co-planarity of the array when inverting the visibilities in the Fourier basis. The distortions in the image domain increase as a function of distance from the phase center. This distortion, if not corrected, limits the imaging dynamic range for fields with significant emission away from the phase center. The measurement equation for such an observation is

$V(u,v,w) = \int\int I(l,m)e^{2\pi\iota(ul+vm+w\sqrt{1-l^2-m^2})}\frac{dldm}{\sqrt{1-l^2-m^2}}$

with the usual meaning for the various symbols. The two algorithms in CASA for correcting for the non-co-planarity are called the "w-projection" and the "faceted imaging" algorithm.

5.8.1 W-Projection

The w-projection algorithm is a matched filter approach. A set of filters corresponding to various value of w for the $e^{2\pi\iota w\sqrt{1-l^2-m^2}}$ term are constructed and applied to the 2D co-planar visibility function to evaluate the 3D non-co-planarity visibilities as:

$V(u,v,w) = G(u,v,w) * V(u,v)$

where $G(u,v,w)$ are the filters. For fast evaluation of the left-hand side, G is pre-computed for a discrete set of w with uniform sampling in $\sqrt{w}$ and the filter corresponding to the nearest value of the w is used.

This effectively increases the limit before which the errors due to the non-coplanar array dominates. The errors in the PSF computed using the w-projection approach are smaller than those incurred in using the approximate PSF in the minor cycle of Clark-Clean. Effectively therefore, Clark-Clean (and its variants) uses one form of approximate PSF in the minor cycle, while w-projection uses another. The accuracy of the major cycle in w-projection depends on the resolution at the which the w-axis is sampled by the filters. For VLA L-band imaging, 256 samples along the w-axis is probably sufficient (set via the `facets` argument of the `setimage()` method of the `imager`).

5.8.2 Faceted Imaging

The tradition approach for wide-field imaging involves approximating the (curved) sky by a set of tangent planes. The size of the facets is set such that the 2D approximation of the measurement equation is valid on each facet. The part of sky covered by each facet is them imaged and deconvolved in the usual manner by phase rotating the visibilities to the center of each facet. The Clean-ed images for each of the facets are then appropriately rotated and projected onto a single 2D tangent image. The accuracy of the major cycle for this algorithm also depends on the number (and hence the size) of the facets, but the dependence is different from that of the w-projection algorithm.

It can be shown that the imaging on each facet in the image domain is equivalent to a shear operation in the visibility domain. The CASA implementation of the faceted imaging algorithm uses this technique, which effectively does the faceting in the visibility domain rather than in the image domain. The functional advantage of this approach is that the users see (and manage) a single 2D image as against multiple facet images when the faceting is done in the image domain. This is a significant advantage, when using faceted imaging.

5.8.3 Comparison of the two methods

The w-projection algorithm can be shown to be about 10x faster than the faceted algorithm. The major cost of major-minor cycle based deconvolution algorithms is the cost of the major cycle. The dominant cost of major cycle is in the the prediction of the visibilities, given a model image. Typically, in faceted imaging approach, each major cycle involves predicting the visibilities for each facet. Though the facets without any emission need not be predicted saving some compute time, this saving disappears for typical P- and L-bands fields which invariable have emission is most facets.

The disadvantage of the w-projection algorithm is that it needs the entire image for the minor cycle. Since a single image covers the entire field of view, this approach requires larger run-time memory. This can however be relaxed by putting smaller fields around the dominant flanking sources in the field of view, which can be of smaller sizes.

5.8.4 Example — Faceted Single Field Imaging

A typical script for imaging a Measurement Set named `MS` using the w-projection algorithm is as follows. The number of facets in this case corresponds to the number of discreet values of w for which the gridding convolution functions are evaluated. This ultimately determins the imaging dynamic range. However since the runtime is only weakly dependant on the number of facets, setting it to a high number like 256 is sufficient for VLA L-band imaging.

```
#
# Set the various paraments of imaging
#
MS       = ''                   # Name of the MS

ALGO     = 'mfclark'            # The algorithm to use for
                                  # deconvolution

IMSIZE   = [4096,4096]          # The image size

CELLSIZE = ['2arcsec','2arcsec'] # The cellsize

SPWID    = [1,2]                # Spectral windows to image

FIELDID  = 1                    # Field IDs to image

NFACETS  = 256                  # No. of w-planes.

STOKES   = 'I'                  # Stokes value

MODEL_IMAGE_NAME ='1046.im'     # Name of the image files.
                                # The restored Clean-ed image
                                # and the residual imgaes are
                                # stored in images with
                                # ``.clean'' and ``.res''
```

```
                                  # appended to this name.

    NITER    = 10000              # No. of Clean components

    THRESHOLD= '100e-3mJy'        # The Cleaning threshold.  The
                                  # iterations are stopped when
                                  # the magnitude of the
                                  # strongest components is lower
                                  # than this value.

    DOINTERACTIVE = F             # Set it to T to do interactive
                                  # box setting in-between iterations.

    im.open(MS)

    im.setdata(spwid=SPWID,fieldid=FIELDID,mode='channel')

    im.setimage(nx=IMSIZE[1],ny=IMSIZE[2],
                cellx=CELLSIZE[1],celly=CELLSIZE[2],
                facets=NFACETS,stokes=stokes,
                spwid=SPWID,fieldid=FIELDID)
                im.setoptions(ftmachine='wproject')

    im.clean(algorithm=ALGO,
             niter=NITER,threshold=THRESHOLD,
             interactive=DOINTERACTIVE,
             model=[MODEL_IMAGE_NAME],
             image=[MODEL_IMAGE_NAME+'.clean'],
             residual=[MODEL_IMAGE_NAME+'.res'])
```

5.8.5 Wide Field Imaging References

See the EVLA Memo: `http://www.aoc.nrao.edu/evla/geninfo/memoseries/evlamemo67.pdf`

Chapter 6

Displaying Images

This chapter describes how to display data with the `casaviewer` either as a stand-alone or through the `viewer` task. You can display both images and MeasurementSets.

6.1 Starting the casaviewer

`casaviewer` is the name of the stand-alone application that is available with a CASA installation. You can call this command from the command line in the following ways:

Start the `casaviewer` with no default image/MS loaded; it will pop up the `Load Data` frame and a blank, standard "Viewer Display Panel. Selecting a file on disk in the `Load Data` panel will provide options for how to display the data. Images can be displayed as: 1) Raster Image, 2) Contour Map, 3) Vector map or 4) Marker Map. MS's can only be displayed as raster.

```
  > casaviewer &
```

Start the `casaviewer` with the selected image; the image will be displayed in the `Viewer Display Panel`. If the image is a cube (more than one plane for frequency or polarization) then it will be one the first plane of the cube.

```
  > casaviewer image_filename &
```

Start the `casaviewer` with the selected MeasurementSet; note the additional parameter indicating that it is an ms; the default is 'image'.

```
  > casaviewer ms_filename ms &
```

In addition, within the casapy environment, there is a `viewer` task which can be used to call up an image (*Note: currently the parameter list of the* `viewer` *task needs to expand to handle the extra parameter to indicate an MS; until then, please use the stand-alone invocations for viewing MS's*):

```
viewer(imagename=None)
  View an image or visibility data set:

  Keyword arguments:
  imagename -- Name of file to visualize
          default: <unset>; example: imagename='ngc5921.image'
```

The `viewer` can be started as:

```
  CASA <4>: viewer
  --------> viewer()
or
CASA <5>: viewer 'ngc5921_task.image'
--------> viewer('ngc5921_task.image')
```

6.2 The viewer GUI

The main parts of the GUI are the menus:

- Data
  - Open - open an image from disk
  - Register - register selected image (menu expands to the right containing all loaded images)
  - Close - close selected image (menu expands to the right)
  - Adjust - open the adjust panel
  - Print - print the displayed image
  - Close Panel - close the Viewer Display Panel
  - Quit Viewer - currently disabled
- Display Panel
  - New Panel - create a new Viewer Display Panel
  - Panel Options - open the panel options frame
  - Print - print displayed image
  - Close Panel - close the Viewer Display Panel
- Tools
  - Currently blank - will hold annotations and image analysis tools

Below this are icons for fast access to some of these menu items:

- folder - Data:Open shortcut – pulls up Load Data panel

- wrench - Data:Adjust shortcut – pulls up Data Display Options panel
- panels - Data:Register shortcut – pull up menu of loaded data
- delete - Data:Close shortcut – closes/unloads selected data
- panel - Display Panel:New Panel
- panel wrench - Display Panel:Panel Options – pulls up Viewer Canvas Manager
- print - Display Panel:Print – print data

Important Bug Note: Please use the icon buttons whenever possible instead of the menus. The `Register` and `Close menus` especially are known to lead to viewer crashes in some cases. You'll usually find that the first four icon buttons are all you need. Click on the display panel titlebar then hover over the buttons for brief reminders of their purpose.

Below this are the eight mouse control buttons. These allow/show the assignment of the mouse buttons for different operations. Clicking in one of these buttons will re-assign a mouse button to that operation.

- **Zooming (magnifying glass icon)** Zooming is accomplished by pressing down the selected mouse button at the start point, dragging the mouse away from that point, and releasing the selected mouse button when the zoom box encloses the desired zoom area. Once the button is released, the zoom rectangle can be moved by clicking inside it with the selected mouse button and dragging it around. To zoom in, simply double click with the selected button inside the rectangle. Double clicking outside the rectangle will result in a zoom out.
- **Panning (hand icon)** Panning is accomplished by pressing down on the selected mouse button at the point you wish to move, dragging the mouse to the position where you want the first point moved to, and releasing the selected mouse button. *Note: The arrow keys, Page Up, Page Down, Home and End keys, and scroll wheel (if any) can also be used to scroll through your data once you have zoomed in. For these to work, the mouse must be over the display panel drawing area, but no mouse tool need be active. Note: this is currently not enabled.*
- **Stretch-shift colormap fiddling**
- **Brightness-contrast colormap fiddling**
- **Positioning** This enables the user to place a crosshair marker on the image to indicate a position. Depending on the context, the positions may be used to flag MeasurementSet data (not yet enabled) or display image spectral profiles (also not currently enabled). Click on the position to place the crosshair; once placed you can drag it to move to another location. Double click is not needed for this control.
- **Rectangle and Polygon region drawing** A rectangle region is generated exactly the same way as the zoom rectangle, and is set by double clicking within the rectangle. Polygon regions can be constructed by progressively clicking the selected mouse button at the desired

location of each vertex, and clicking in the same location twice to complete the polygon. Once constructed, it can be moved by dragging inside the polygon, and reshaped by dragging the various handles at the vertices.

- **Polyline drawing** A polyline can be constructed with this button selected. It is almost identical to the polygon region tool. Create points by clicking at the positions wanted and then double-click to finish the line.

Below this area is the actual display surface.

Below the display is the 'tape deck' which provides basic movement between image planes along a selected third dimension of an image cube. This set of buttons is only enabled when the first-registered image reports that it has more than one plane along the 'Z axis'. In the most common case, the animator controls the frequency channel being viewed. From left to right, the tape deck controls allow the user to:

- rewind to the start of the sequence (i.e., the first plane)
- step backwards by one plane
- play backwards, or repetitively step backwards
- stop any current play
- play forward, or repetitively step forward
- step forward by one plane
- fast forward to the end of the sequence

To the right of the tape deck is an editable text box indicating the current frame number and a sunken label showing the total number of frames. One can type a channel number into the current frame to jump to that channel. Below this is a slider for controlling the animation speed. To the right of this is the 'Full/Compact' toggle. In full mode, additional controls for blinking and for controlling the frame value and step are available; the default setting is for compact. In 'Blink' mode, when more than one raster image is registered in the Viewer Display Panel, the tapedeck will control which is being displayed at the moment. The images registered should cover the same portion of the sky, using the same coordinate projection.

6.3 Viewing a raster map

A raster map of an image shows pixel intensities in a two-dimensional cross-section of gridded data with colors selected from a finite set of (normally) smooth and continuous colors, i.e., a colormap.

Starting the `casaviewer` with an image as a raster map will look something like:

You will see the GUI which consists of two main windows, entitled "Viewer Display Panel" and "Load Data". In the "Load Data" panel, you will see all of the files in the current working directory along with their type (Image, MeasurementSet, etc). After selecting a file, you are presented with the available data types for these data. Clicking on the button `Raster Map` will create a display as above. The main parts of the ”Viewer Display Panel” GUI are discussed in the following Section.

6.4 Viewing a contour map

Viewing a contour image is similar the process above. A contour map shows lines of equal pixel intensity (e.g., flux density) in a two dimensional cross-section of gridded data. Contour maps are particularly useful for overlaying on raster images so that two different measurements of the same part of the sky can be shown simultaneously.

6.5 Viewing a MeasurementSet with visibility data

Visibility data can also be displayed and flagged directly from the viewer (*Note: flagging is not currently enabled*). For MeasurementSet files the only option for display is 'Raster' (similar to AIPS task TVFLG).

Note: There is also a bug in the current MS viewing which disables display of the data and flags; use the 'Adjust' panel 'Flagging Options' Menu to change the 'Show Flagged Regions' option to 'Masked to Background'. This will be the default for Patch 2.

6.6 Adjusting Display Parameters

The data display can be adjusted by the user as needed. The following illustrate the available options in the catagories of:

- Display axes
- Hidden axes
- Basic Settings
- Position tracking
- Axis labels
- Axis label properties

This older web page gives details of individual display options. Although it has not yet been integrated into the reference manual for the newer CASA, it is accurate in most cases:

`http://aips2.nrao.edu/daily/docs/user/Display/node267.html`

6.7 Adjusting Canvas Parameters/Multi-panel displays

The display area or Canvas can also be manipulated through two sets of values:

- Margins - specify the spacing for the left, right, top, and bottom margins
- Number of panels - specify the number of panels in x and y and the spacing between those panels.

The following illustrates a multi-panel display along with the Viewer Canvas Manager settings which created it.

6.8 Overlay contours on a raster map

Contours of either a second data set or the same data set can be used for comparison or to enhance visualization of the data. The Adjust Panel will have multiple tabs which allow adjusting each data set individually (Note tabs along the top). To enable this simply open up the Load Data panel (Use the Data menu or click on the Folder icon), select the data set and select Contour.

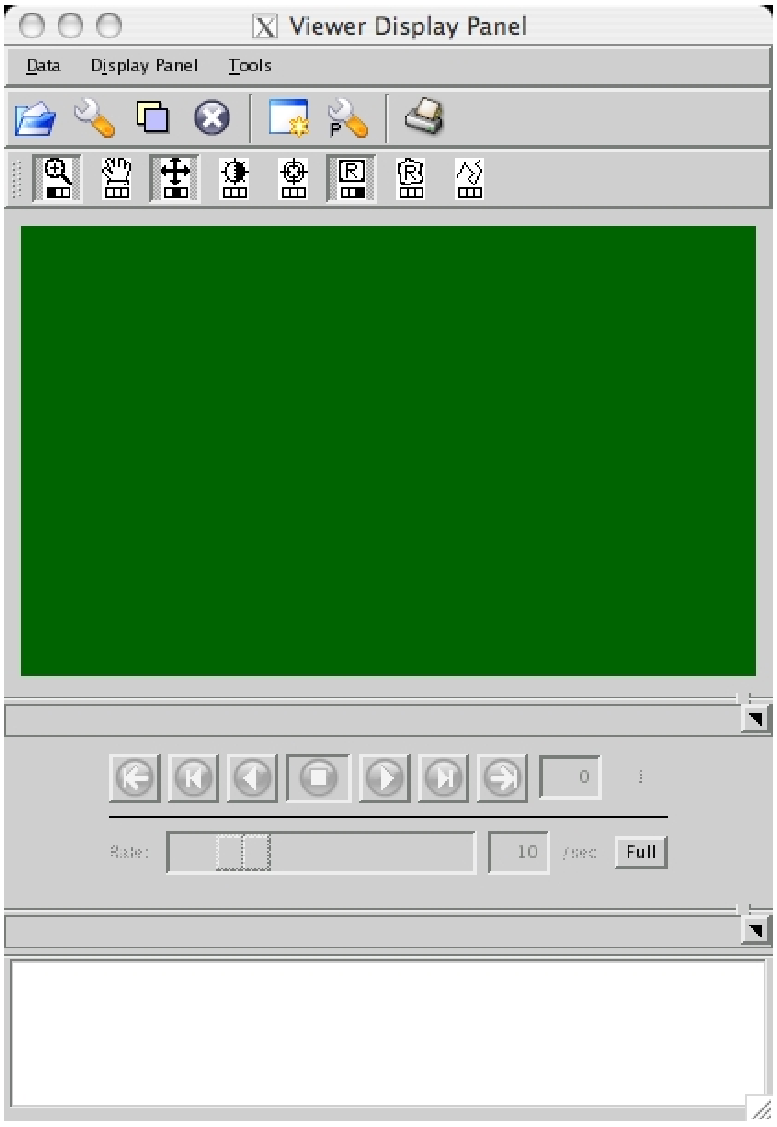
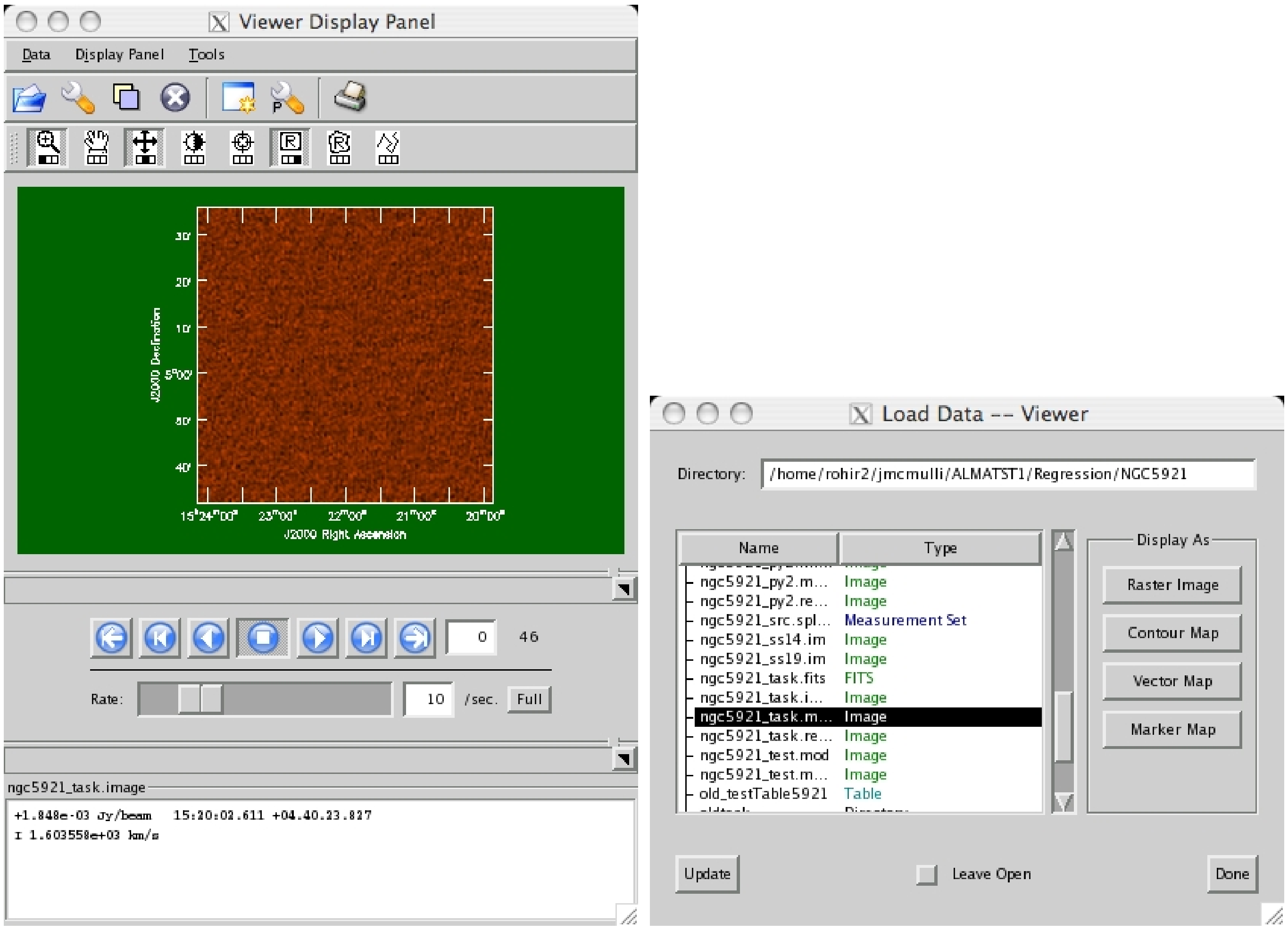


Figure 6.1: Viewer Display Panel with no data loaded. Each section of the GUI is explained below



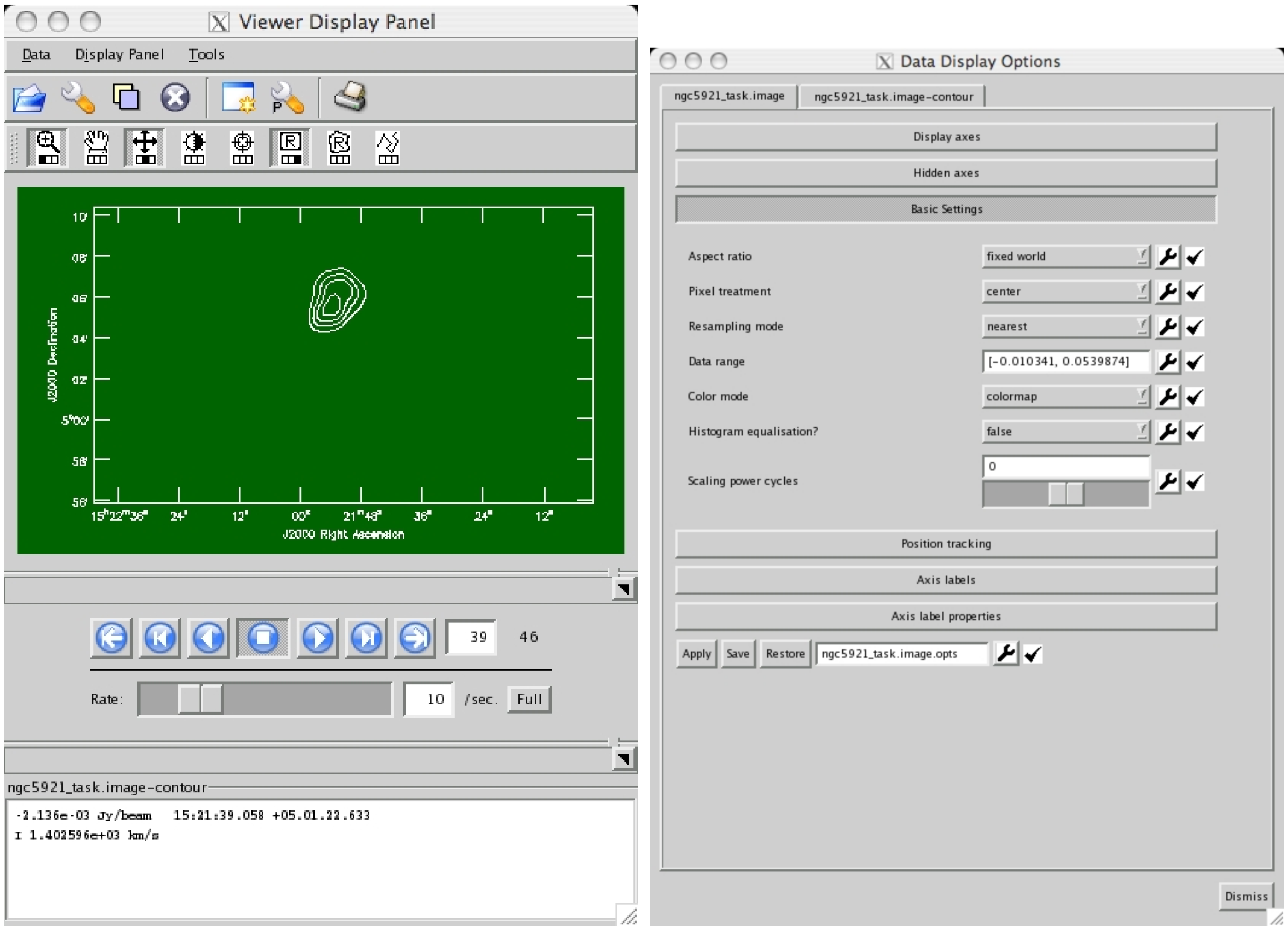
ngc5921_task.image

```
+1.848e-03 Jy/beam   15:20:02.611 +04.40.23.827
I 1.603558e+03 km/s
```

Directory: /home/rohir2/jmcmulli/ALMATST1/Regression/NGC5921

Name	Type
ngc5921_py2.m...	Image
ngc5921_py2.re...	Image
ngc5921_src.spl...	Measurement Set
ngc5921_ss14.im	Image
ngc5921_ss19.im	Image
ngc5921_task.fits	FITS
ngc5921_task.i...	Image
ngc5921_task.m...	Image
ngc5921_task.re...	Image
ngc5921_test.mod	Image
ngc5921_test.m...	Image
old_testTable5921	Table

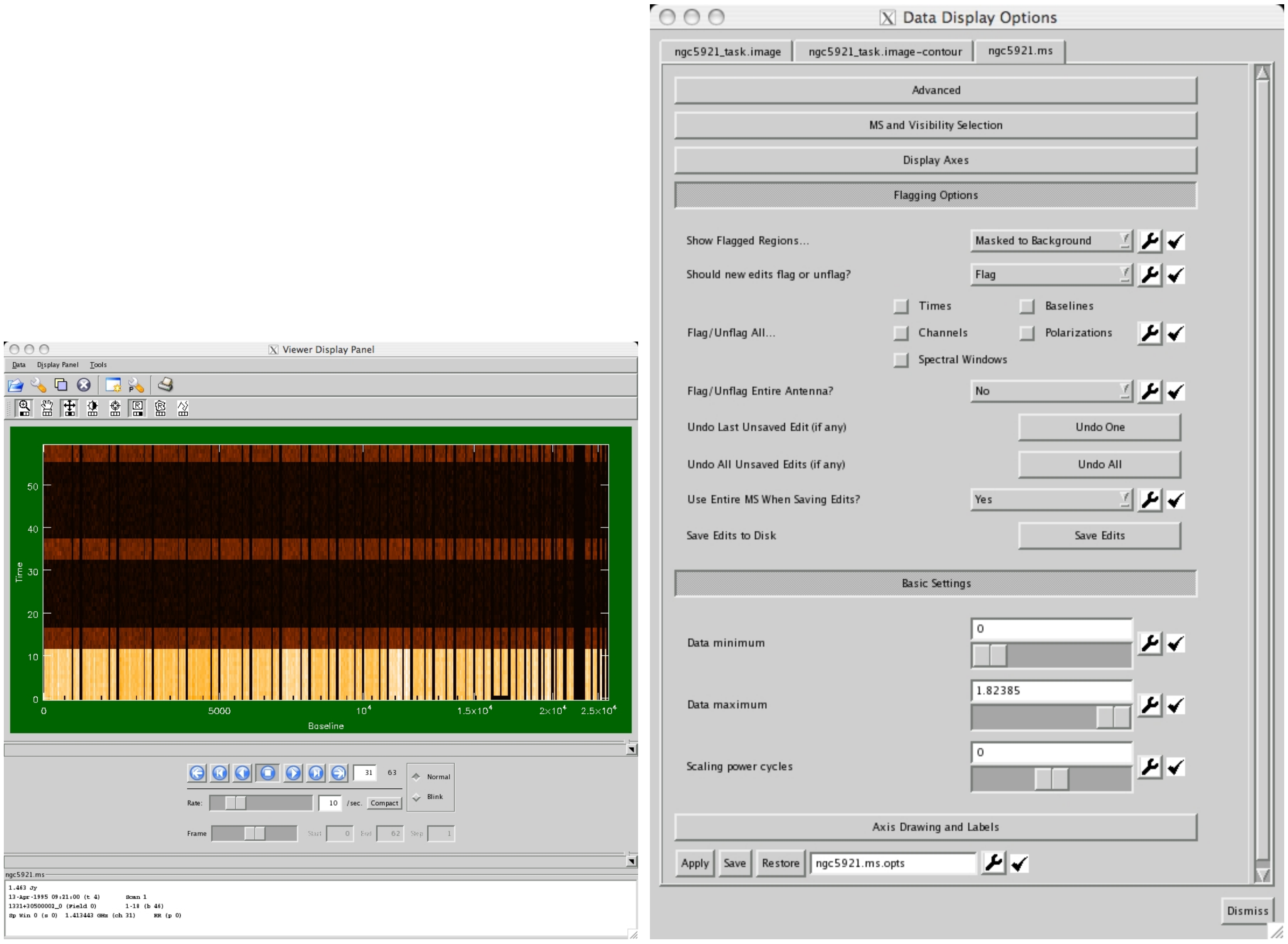
Figure 6.2: casaviewer: Illustration of a raster image in the Viewer Display Panel(left) and the Load Data panel (right).



```
-2.136e-03 Jy/beam   15:21:39.058 +05.01.22.633
I 1.402596e+03 km/s
```

Aspect ratio	fixed world
Pixel treatment	center
Resampling mode	nearest
Data range	[-0.010341, 0.0539874]
Color mode	colormap
Histogram equalisation?	false
Scaling power cycles	0

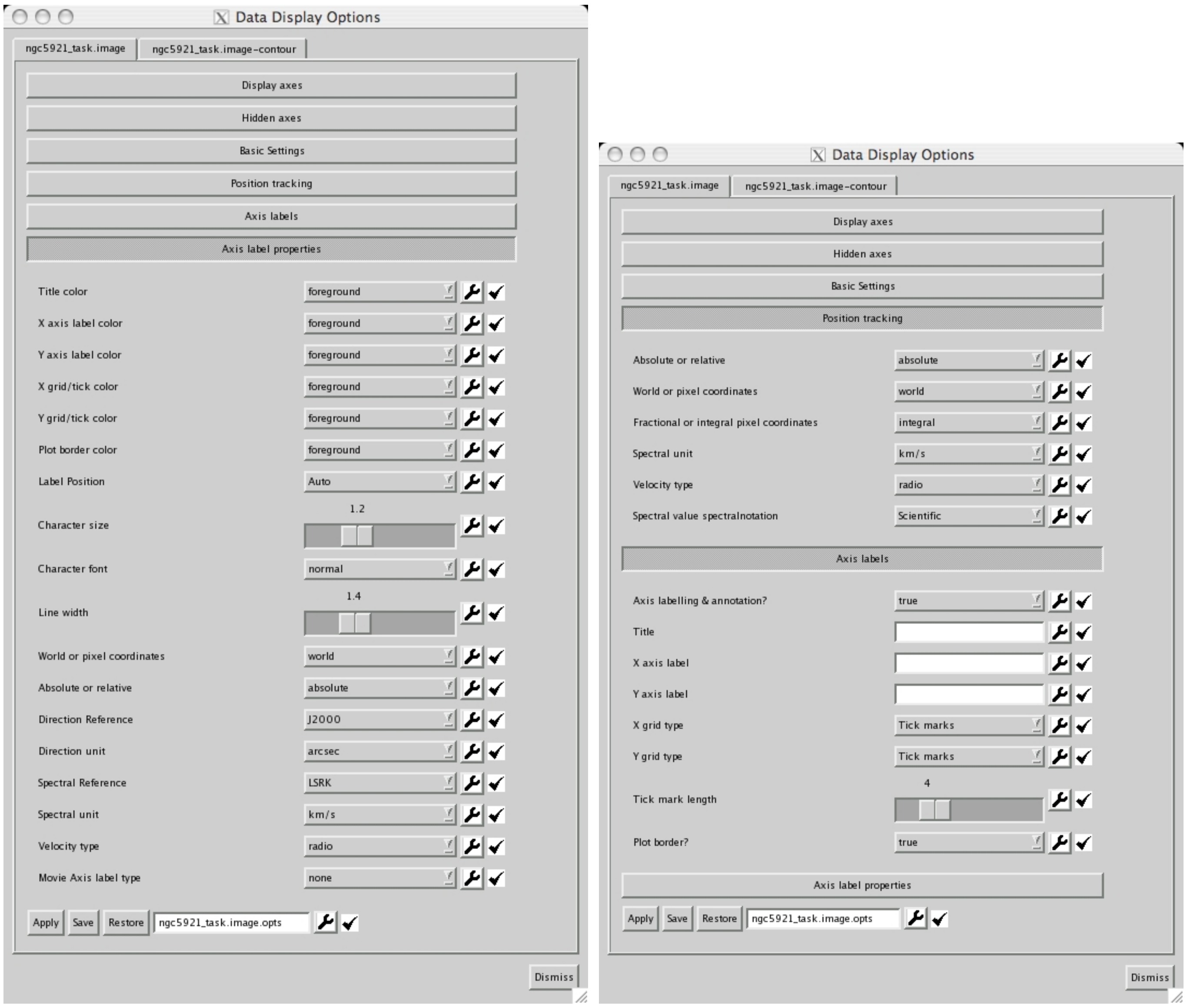
Figure 6.3: casaviewer: Illustration of a raster image in the Viewer Display Panel(left) and the Load Data panel (right).



```
ngc5921.ms
1.463 Jy
13-Apr-1995 09:21:00 (t 4)       Scan 1
1331+30500002_0 (Field 0)        1-18 (b 46)
Sp Win 0 (s 0)  1.413443 GHz (ch 31)     RR (p 0)
```

Show Flagged Regions... Masked to Background
Should new edits flag or unflag? Flag
Flag/Unflag All... Times, Baselines, Channels, Polarizations, Spectral Windows
Flag/Unflag Entire Antenna? No
Undo Last Unsaved Edit (if any)
Undo All Unsaved Edits (if any)
Use Entire MS When Saving Edits? Yes
Save Edits to Disk
Data minimum 0
Data maximum 1.82385
Scaling power cycles 0
ngc5921.ms.opts

Figure 6.4: casaviewer: Display of visibility data. The default axes are time vs. baseline.



Title color	foreground
X axis label color	foreground
Y axis label color	foreground
X grid/tick color	foreground
Y grid/tick color	foreground
Plot border color	foreground
Label Position	Auto
Character size	1.2
Character font	normal
Line width	1.4
World or pixel coordinates	world
Absolute or relative	absolute
Direction Reference	J2000
Direction unit	arcsec
Spectral Reference	LSRK
Spectral unit	km/s
Velocity type	radio
Movie Axis label type	none

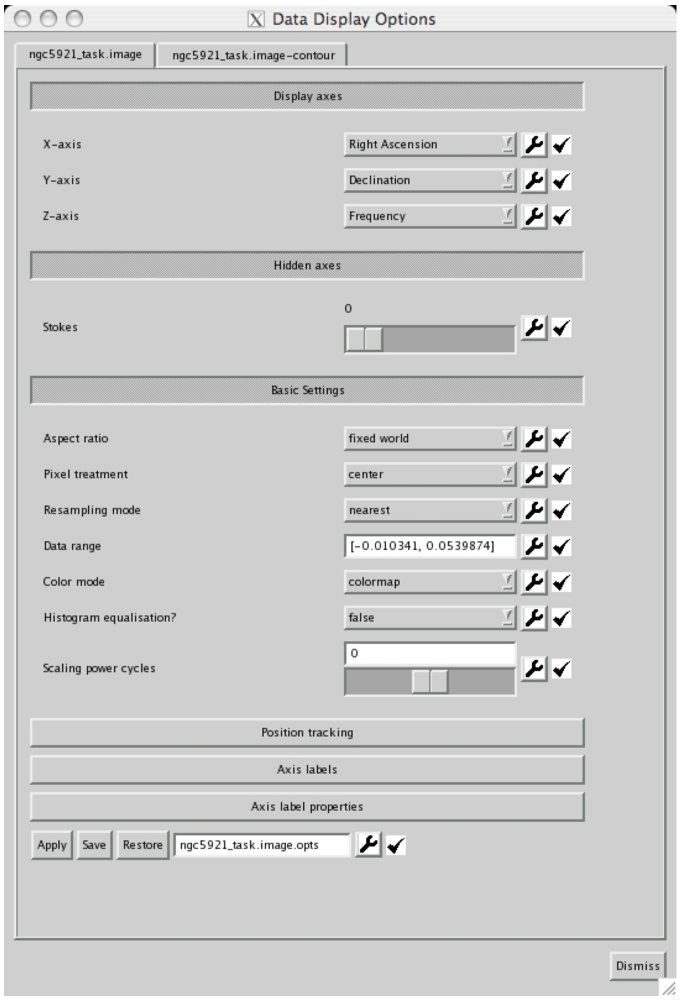
ngc5921_task.image.opts

Absolute or relative	absolute
World or pixel coordinates	world
Fractional or integral pixel coordinates	integral
Spectral unit	km/s
Velocity type	radio
Spectral value spectralnotation	Scientific

Axis labelling & annotation?	true
Title	
X axis label	
Y axis label	
X grid type	Tick marks
Y grid type	Tick marks
Tick mark length	4
Plot border?	true

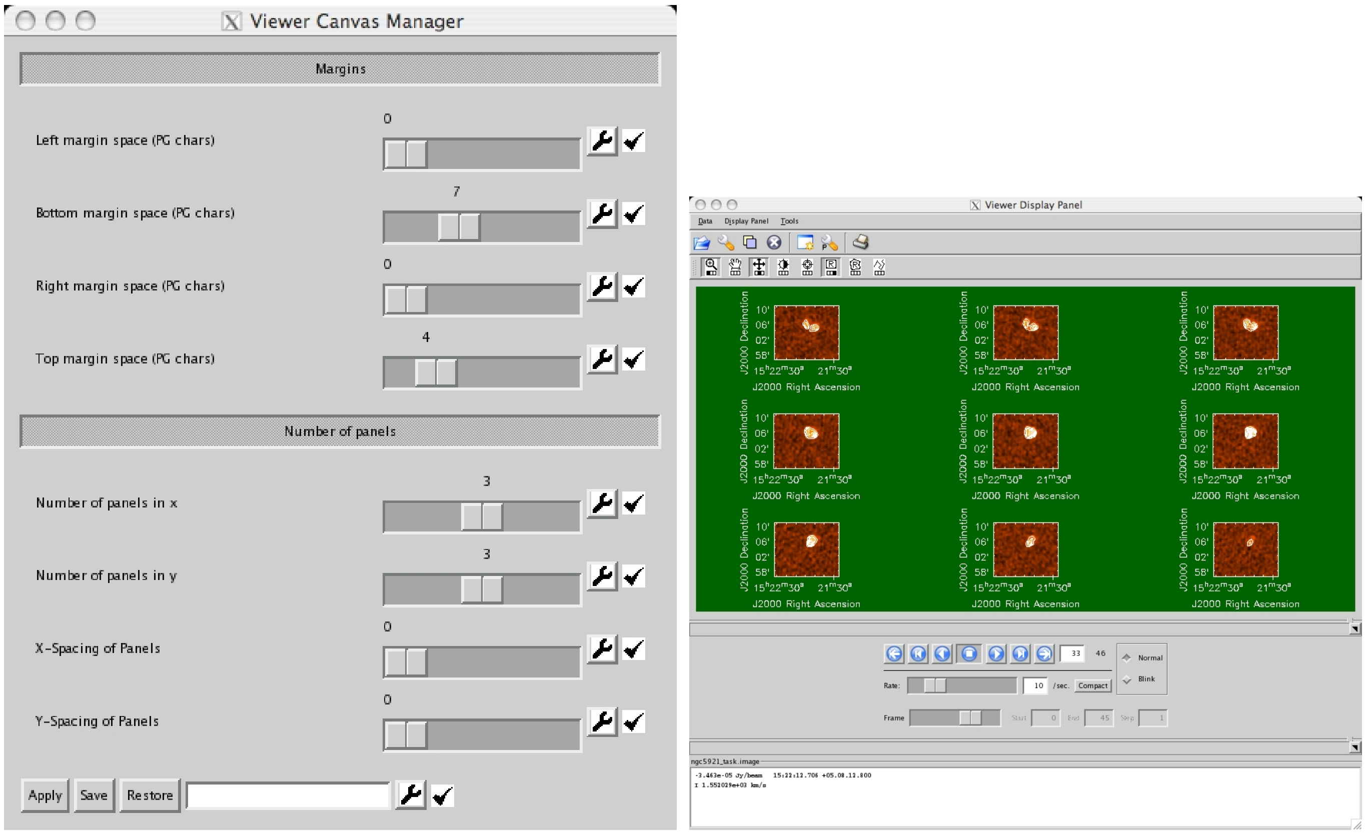
ngc5921_task.image.opts

Figure 6.5: casaviewer: Data display options. In the left panel, the Display axes, Hidden axes, and Basic Settings options are shown; in the right panel, the Position tracking and Axis labels options are shown.



X-axis: Right Ascension
Y-axis: Declination
Z-axis: Frequency
Stokes: 0
Aspect ratio: fixed world
Pixel treatment: center
Resampling mode: nearest
Data range: [-0.010341, 0.0539874]
Color mode: colormap
Histogram equalisation?: false
Scaling power cycles: 0
ngc5921_task.image.opts

Figure 6.6: casaviewer: Data display options. In this final, third panel , the Axis label properties are shown.



Viewer Canvas Manager

Margins

Left margin space (PG chars) 0

Bottom margin space (PG chars) 7

Right margin space (PG chars) 0

Top margin space (PG chars) 4

Number of panels

Number of panels in x 3

Number of panels in y 3

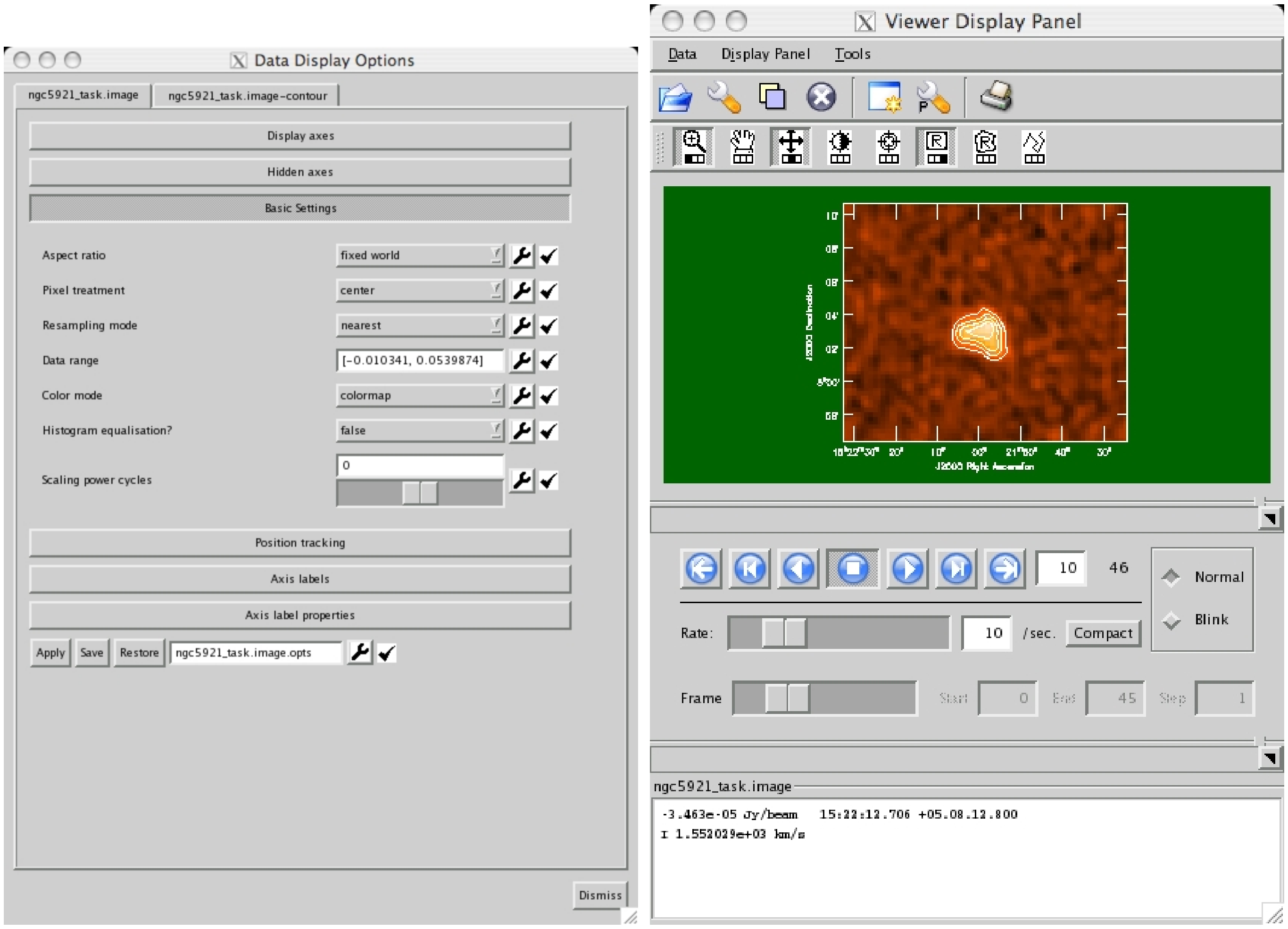
X-Spacing of Panels 0

Y-Spacing of Panels 0

ngc5921_task.image

```
-3.463e-05 Jy/beam   15:22:12.706 +05.08.12.800
I 1.552029e+03 km/s
```

Figure 6.7: casaviewer: A multi-panel display set up through the Viewer Canvas Manager.



Aspect ratio: fixed world
Pixel treatment: center
Resampling mode: nearest
Data range: [-0.010341, 0.0539874]
Color mode: colormap
Histogram equalisation?: false
Scaling power cycles: 0
ngc5921_task.image.opts

Frame: 10 of 46; Rate: 10 /sec.; Start 0, End 45, Step 1

ngc5921_task.image

```
-3.463e-05 Jy/beam   15:22:12.706 +05.08.12.800
I 1.552029e+03 km/s
```

Figure 6.8: casaviewer: Display contour overlay on top of a raster image.

Chapter 7

Image Analysis

7.1 Open a file

To open an image file for analysis, use the `ia.open` method, e.g.:

```
  ia.open('ngc5921_task.image')
```

7.2 Get an image summary

To get a summary of the properties of your image, use the `ia.summary` method, e.g.:

```
  imhead(imagename='image.im')
```

7.3 Image statistics

The `ia.statistics` method computes statistics in a (region) of an image.

```
  CASA <50>: ia.open('ngc5921_task.image')
  CASA <51>: ia.statistics() # Note: formatted output goes to the logger
  {'return': True,
   'statsout': {'blc': [0, 0, 0, 0],
               'blcf': '15:24:08.404, +04.31.59.181, I, 1.41281e+09Hz',
               'flux': [12.136708280654085],
               'max': [0.12773475050926208],
               'maxpos': [134, 134, 0, 38],
               'maxposf': '15:21:53.976, +05.05.29.998, I, 1.41374e+09Hz',
               'mean': [4.7899158775357123e-05],
               'min': [-0.023951441049575806],
               'minpos': [230, 0, 0, 15],
               'minposf': '15:20:17.679, +04.31.59.470, I, 1.41317e+09Hz',
               'npts': [3014656.0],
               'rms': [0.00421889778226614],
               'sigma': [0.0042186264443439979],
               'sum': [144.399486397083],
               'sumsq': [53.658156081703709],
               'trc': [255, 255, 0, 45],
               'trcf': '15:19:52.390, +05.35.44.246, I, 1.41391e+09Hz'}}
```

Restrict statistics to a region:

```
CASA <60>: blc=[0,0,0,23]

CASA <61>: trc=[255,255,0,23]

CASA <62>: bbox=ia.setregion(blc,trc)

CASA <63>: ia.statistics(region=bbox)

0%....10....20....30....40....50....60....70....80....90....100%
Warning no plotter attached.  Attach a plotter to get plots
  Out[63]:
{'return': True,
 'statsout': {'blc': [0, 0, 0, 23],
             'blcf': '15:24:08.404, +04.31.59.181, I, 1.41337e+09Hz',
             'flux': [0.21697188625158217],
             'max': [0.061052091419696808],
             'maxpos': [124, 132, 0, 23],
             'maxposf': '15:22:04.016, +05.04.59.999, I, 1.41337e+09Hz',
             'mean': [3.9390207550122095e-05],
             'min': [-0.018510516732931137],
             'minpos': [254, 20, 0, 23],
             'minposf': '15:19:53.588, +04.36.59.216, I, 1.41337e+09Hz',
             'npts': [65536.0],
             'rms': [0.0040461532771587372],
             'sigma': [0.0040459925613279676],
             'sum': [2.5814766420048016],
             'sumsq': [1.0729132921679772],
             'trc': [255, 255, 0, 23],
             'trcf': '15:19:52.390, +05.35.44.246, I, 1.41337e+09Hz'}}
```

You can also use the viewer to interactively obtain image statistics on a region:

```
viewer('imagename.im')
# Now use the right mouse button (default for region setting)
# create a region and then double click inside to obtain statistics on that region
# Currently this supports a single plane only and the output goes to your casapy
# terminal window as:

ngc5921_task.image

n           Std Dev     RMS         Mean        Variance    Sum
660         0.01262     0.0138      0.005622    0.0001592   3.711
```

```
  Flux         Med |Dev|    Quartile     Median       Min          Max
  0.3119       0.003758     0.004586     0.001434     -0.009671    0.06105
```

7.3.1 Moments

Make 0th and 1st moment maps. The example below takes an existing image cube ang generates 0th and 1st moment maps. No task yet exists to make moment masks so the examples below show you how to do this with the CASA tools:

```
  ia.open('mosaic.image')           # Open the image you want and attach it to a tool called ia
  ia.moments(outfile='mosaic.mom0', # Write the moment 0 map, mosaic.mom0, to disk
     moments=0, axis=3,             # Take the zeroth moment over velocity (axis 3,0-based)
     mask='indexin(4,[4:25])',      # Select channels 4-25
                                    # Darn: mask axis input must be 1-based (4=velocity now).
     includepix=[0.09,100.0])       # Only include pixels above 0.09 Jy
  ia.close()                        # Close the tool and detach from the MS

  ia.open('mosaic.image')           # Open the image you want and attach it to a tool called ia
  ia.moments(outfile='mosaic.mom1', # Write the moment 1 map, mosaic.mom1, to disk
     moments=1, axis=3,             # Take the first moment over velocity (axis 3 - 0-based)
     mask='indexin(4,[4:25])',      # Select channels 4-25, over velocity
     includepix=[0.09,10.0])        # Only include pixels above 0.09 Jy
  ia.close()                        # Close the tool and detach from the MS
```

Note: 4mar07 there is a bug with the moments in that it keeps the moment image locked so you can't look at it with the viewer. Type the following to clear all locks and then bring up the viewer:

```
  tb.clearlocks()
  viewer
```

Now the viewer will come up and you can look at the resulting image.

7.4 Image Math

Images can be scaled, manipulated and combined in various ways. The most straightforward method is to use the `imagecalc` tool to do a simple scaling:

```
  ia.open('n75.im')                 # open the image called n75.im
  ia.imagecalc(outfile='output.im', # divide the image by 2 and write new image
     pixels='infile.im/2')          #  to the file called output.im
  ia.close()                        # close the tool.
```

Now, say you have made a mosic that has uniform RMS across the image but the flux falls off as you move to the edge of the mosaic. You wrote out a `fluxscale` image called fluxscale.im that you can use to scale your mosaic to have the correct flux across the entire field (albiet it has increasing RMS noise as you go to the edge of the mosaic). For example:

```
ia.open('mosaic.im')                  # open the image called mosaic.im
ia.imagecalc(outfile='mosaic.correctflux.im', # divide it by the fluxscale.im
    pixels='(mosaic.image)*(fluxscale.im)');
ia.close()                          # close the tool.
```

Chapter 8

Single Dish Data Processing

For single-dish spectral calibration and analysis, CASA uses the ATNF Spectral Analysis Package (ASAP). This is imported as the `sd` tool, and forms the basis for a series of tasks (the "SDtasks") that encapsulate the functionality within the standard CASA task framework. ASAP was developed to support the Australian telescopes such as Mopra, Parkes, and Tidbinbilla, and we have adapted it for use within CASA for GBT and eventually ALMA data also. For details on ASAP, see the ASAP home page at ATNF:

- `http://www.atnf.csiro.au/computing/software/asap/`

You can also download the ASAP User Guide and Reference Manual at this web site. There is also a brief tutorial. Note that within CASA, the ASAP tools are prefaced with `sd.`, e.g. where it says in the ASAP User Guide to use `scantable` you will use `sd.scantable` in CASA. See § 8.2 for more information on the tools.

All of the ASAP functionality is available with a CASA installation. In the following, we outline how to access ASAP functionality within CASA with the tasks and tools, and the data flow for standard use cases.

If you run into trouble, be sure to check the list of known issues and features of ASAP and the SDtasks presented in § 8.4 first.

8.1 Guidelines for Use of ASAP and SDtasks in CASA

8.1.1 Environment Variables

There are a number of environment variables that the ASAP tools (and thus the SDtasks) use to help control their operation. These are described in the ASAP User Guide as being in the `.asaprc` file. Within CASA, these are contained in the Python dictionary `sd.rcParams` and are accessible through its keys and values. For SDtask users, the most important are the `verbose` parameter controlling the display of detailed messages from the tools. By default

```
    sd.rcParams['verbose'] = True
```

and you get lots of messages. Also), and the `scantable.storage` parameter controlling whether scantable operations are done in memory or on disk. The default

```
    sd.rcParams['scantable.storage'] = 'memory'
```

does it in memory (best choice if you have enough), while to force the scantables to disk use

```
    sd.rcParams['scantable.storage'] = 'disk'
```

which might be necessary to allow processing of large datasets. See § 8.2.1 for more details on the ASAP environment variables.

8.1.2 Assignment

Some ASAP methods and function require you to assign that method to a variable which you can then manipulate. This includes `sd.scantable` and `sd.selector`, which make objects. For example,

```
    s = sd.scantable('OrionS_rawACSmod', average=False)
```

8.1.3 Lists

For lists of scans or IFs, such as in `scanlist` and `iflist` in the SDtasks, the tasks and functions want a comma-separated Python list, e.g.

```
    scanlist = [241, 242, 243, 244, 245, 246]
```

You can use the Python `range` function to generate a list of consecutive numbers, e.g.

```
    scanlist = range(241,247)
```

giving the same list as above, e.g.

```
CASA <3>: scanlist=range(241,247)
CASA <4>: print scanlist
[241, 242, 243, 244, 245, 246]
```

You can also combine multiple ranges by summing lists

```
CASA <5>: scanlist=range(241,247) + range(251,255)
CASA <6>: print scanlist
[241, 242, 243, 244, 245, 246, 251, 252, 253, 254]
```

Note that in the future, the `sd` tools and SDtasks will use the same selection language as in the synthesis part of the package.

Spectral regions, such as those for setting masks, are pairs of min and max values for whatever spectral axis unit is currently chosen. These are fed into the tasks and tools as a list of lists, with each list element a list with the `[min,max]` for that sub-region, e.g.

```
    masklist=[[1000,3000], [5000,7000]].
```

8.1.4 Text

The SDtasks trap leading and trailing whitespace on string parameters (such as `infile` and `sdfile`), but ASAP does not, so be careful with setting string parameters. ASAP is also case-sensitive, with most parameters being upper-case, such as `ASAP` for the `sd.scantable.save` file format. The SDtasks are generally more forgiving.

8.2 Using The ASAP Toolkit Within CASA

ASAP is included with the CASA installation/build. It is not loaded upon start-up, however, and must be imported as a standard Python package. A convenience function exists for importing ASAP along with a set of prototype tasks for single dish analysis:

```
CASA <1>: asap_init
```

Once this is done, all of the ASAP functionality is now under the Python 'sd' tool. bf: Note: This means that if you are following the ASAP cookbook or documentation, all of the commands should be invoked with a 'sd.' before the native ASAP command.

The ASAP interface is essentially the same as that of the CASA toolkit, that is, there are groups of functionality (aka tools) which have the ability to operate on your data. Type:

```
CASA <4>: sd.<TAB>
sd.__class__               sd._validate_bool          sd.list_scans
sd.__date__                sd._validate_int           sd.mask_and
sd.__delattr__             sd.asapfitter              sd.mask_not
sd.__dict__                sd.asaplinefind            sd.mask_or
sd.__doc__                 sd.asaplog                 sd.merge
sd.__file__                sd.asaplotbase             sd.os
sd.__getattribute__        sd.asaplotgui              sd.plf
sd.__hash__                sd.asapmath                sd.plotter
sd.__init__                sd.asapplotter             sd.print_log
sd.__name__                sd.asapreader              sd.quotient
sd.__new__                 sd.average_time            sd.rc
sd.__path__                sd.calfs                   sd.rcParams
sd.__reduce__              sd.calnod                  sd.rcParamsDefault
sd.__reduce_ex__           sd.calps                   sd.rc_params
sd.__repr__                sd.commands                sd.rcdefaults
sd.__setattr__             sd.defaultParams           sd.reader
sd.__str__                 sd.dosigref                sd.scantable
sd.__version__             sd.dototalpower            sd.selector
sd._asap                   sd.fitter                  sd.simple_math
sd._asap_fname             sd.is_ipython              sd.sys
sd._asaplog                sd.linecatalog             sd.unique
sd._is_sequence_or_number  sd.linefinder              sd.version
sd._n_bools                sd.list_files              sd.welcome
sd._to_list                sd.list_rcparameters       sd.xyplotter
```

...to see the list of tools.

In particular, the following are essential for most reduction sessions:

- `sd.scantable` - the data structure for ASAP and the core methods for manipulating the data; allows importing data, making data selections, basic operations (averaging, baselines, etc) and setting data characteristics (e.g., frequencies, etc).
- `sd.selector` - selects a subset of data for subsequent operations
- `sd.fitter` - fit data
- `sd.plotter` - plotting facilities (uses `matplotlib`)

The `scantable` functions are used most often and can be applied to both the initial scantable and to any spectrum from that scan table. Type

```
sd.scantable.<TAB>
```

(using TAB completion) to see the full list.

8.2.1 Environment Variables

The `asaprc` environment variables are stored in the Python dictionary `sd.rcParams` in CASA. This contains a number of parameters that control how ASAP runs, for both tools and tasks. You can see what these are set to by typing at the CASA prompt:

```
  CASA <2>: sd.rcParams
  Out[2]:
{'insitu': True,
 'plotter.colours': '',
 'plotter.decimate': False,
 'plotter.ganged': True,
 'plotter.gui': True,
 'plotter.histogram': False,
 'plotter.linestyles': '',
 'plotter.panelling': 's',
 'plotter.papertype': 'A4',
 'plotter.stacking': 'p',
 'scantable.autoaverage': True,
 'scantable.freqframe': 'LSRK',
 'scantable.save': 'ASAP',
 'scantable.storage': 'memory',
 'scantable.verbosesummary': False,
 'useplotter': True,
 'verbose': True}
```

The use of these parameters is described in detail in the ASAP Users Guide.

You can also change these parameters through the `sd.rc` function. The use of this is described in `help sd.rc`:

```
CASA <3>: help(sd.rc)
Help on function rc in module asap:

rc(group, **kwargs)
    Set the current rc params.  Group is the grouping for the rc, eg
    for scantable.save the group is 'scantable', for plotter.stacking, the
    group is 'plotter', and so on.  kwargs is a list of attribute
    name/value pairs, eg

      rc('scantable', save='SDFITS')

    sets the current rc params and is equivalent to

      rcParams['scantable.save'] = 'SDFITS'

    Use rcdefaults to restore the default rc params after changes.
```

8.2.2 Import

Data can be loaded into ASAP by using the `scantable` function which will read a variety of recognized formats (RPFITS, varieties of SDFITS, and the CASA MeasurementSet). For example:

```
  CASA <1>: scans = sd.scantable('OrionS_rawACSmod', average=False)
  Importing OrionS_rawACSmod...
```

NOTE: It is important to use the `average=False` parameter setting as the calibration routines supporting GBT data require all of the individual times and phases.

NOTE: GBT data may need some pre-processing prior to using ASAP. In particular, the program which converts GBT raw data into CASA MeasurementSets tends to proliferate the number of spectral windows due to shifts in the tracking frequency; this is being worked on by GBT staff. In addition, GBT SDFITS is currently not readable by ASAP (in progress).

NOTE: The MeasurementSet to scantable conversion is able to deduce the reference and source data and assigns an '_r' to the reference data to comply with the ASAP conventions.

NOTE: GBT observing modes are identifiable in scantable in the name assignment: position switched ('_ps'), Nod ('_nod'), and frequency switched ('_fs'). These are combined with the reference data assignment. (For example, the reference data taken in position switched mode observation are assigned as '_psr'.)

Use the `summary` function to examine the data and get basic information:

```
CASA <8>: scans.summary()
--------------------------------------------------------------------------------
 Scan Table Summary
--------------------------------------------------------------------------------
Beams:         1
IFs:           26
Polarisations: 2   (linear)
```

```
Channels:      8192

Observer:      Joseph McMullin
Obs Date:      2006/01/19/01:45:58
Project:       AGBT06A_018_01
Obs. Type:     OffOn:PSWITCHOFF:TPWCAL
Antenna Name:  GBT
Flux Unit:     Jy
Rest Freqs:    [4.5490258e+10] [Hz]
Abcissa:       Channel
Selection:     none

Scan Source         Time      Integration
     Beam    Position (J2000)
          IF       Frame   RefVal          RefPix    Increment
--------------------------------------------------------------------------------
  20 OrionS_psr     01:45:58    4 x       30.0s
       0    05:15:13.5 -05.24.08.2
           0      LSRK   4.5489354e+10   4096    6104.233
           1      LSRK   4.5300785e+10   4096    6104.233
           2      LSRK   4.4074929e+10   4096    6104.233
           3      LSRK   4.4166215e+10   4096    6104.233
  21 OrionS_ps      01:48:38    4 x       30.0s
       0    05:35:13.5 -05.24.08.2
           0      LSRK   4.5489354e+10   4096    6104.233
           1      LSRK   4.5300785e+10   4096    6104.233
           2      LSRK   4.4074929e+10   4096    6104.233
           3      LSRK   4.4166215e+10   4096    6104.233
  22 OrionS_psr     01:51:21    4 x       30.0s
       0    05:15:13.5 -05.24.08.2
           0      LSRK   4.5489354e+10   4096    6104.233
           1      LSRK   4.5300785e+10   4096    6104.233
           2      LSRK   4.4074929e+10   4096    6104.233
           3      LSRK   4.4166215e+10   4096    6104.233
  23 OrionS_ps      01:54:01    4 x       30.0s
       0    05:35:13.5 -05.24.08.2
           0      LSRK   4.5489354e+10   4096    6104.233
           1      LSRK   4.5300785e+10   4096    6104.233
           2      LSRK   4.4074929e+10   4096    6104.233
           3      LSRK   4.4166215e+10   4096    6104.233
  24 OrionS_psr     02:01:47    4 x       30.0s
       0    05:15:13.5 -05.24.08.2
          12      LSRK   4.3962126e+10   4096   6104.2336
          13      LSRK    4.264542e+10   4096   6104.2336
          14      LSRK    4.159498e+10   4096   6104.2336
          15      LSRK   4.3422823e+10   4096   6104.2336
  25 OrionS_ps      02:04:27    4 x       30.0s
       0    05:35:13.5 -05.24.08.2
          12      LSRK   4.3962126e+10   4096   6104.2336
          13      LSRK    4.264542e+10   4096   6104.2336
          14      LSRK    4.159498e+10   4096   6104.2336
```

```
         15      LSRK   4.3422823e+10   4096   6104.2336
 26 OrionS_psr     02:07:10    4 x       30.0s
      0    05:15:13.5 -05.24.08.2
         12      LSRK   4.3962126e+10   4096   6104.2336
         13      LSRK    4.264542e+10   4096   6104.2336
         14      LSRK    4.159498e+10   4096   6104.2336
         15      LSRK   4.3422823e+10   4096   6104.2336
 27 OrionS_ps      02:09:51    4 x       30.0s
      0    05:35:13.5 -05.24.08.2
         12      LSRK   4.3962126e+10   4096   6104.2336
         13      LSRK    4.264542e+10   4096   6104.2336
         14      LSRK    4.159498e+10   4096   6104.2336
         15      LSRK   4.3422823e+10   4096   6104.2336
```

8.2.3 Scantable Manipulation

Within ASAP, data is stored in a `scantable`, which holds all of the observational information and provides functionality to manipulate the data and information. The building block of a `scantable` is an integration which is a single row of a scantable. Each row contains just one spectrum for each beam, IF and polarization.

Once you have a `scantable` in ASAP, you can select a subset of the data based on scan numbers, sources, or types of scan; note that each of these selections returns a new 'scantable' with all of the underlying functionality:

```
  CASA <5>: scan27=scans.get_scan(27)                 # Get the 27th scan
  CASA <6>: scans20to24=scans.get_scan(range(20,25))  # Get scans 20 - 24
  CASA <7>: scans_on=scans.get_scan('*_ps')           # Get ps scans on source
  CASA <8>: scansOrion=scans.get_scan('Ori*')         # Get all Orion scans
```

To copy a scantable, do:

```
  CASA <15>: ss=scans.copy()
```

8.2.3.1 Data Selection

In addition to the basic data selection above, data can be selected based on IF, beam, polarization, scan number as well as values such as Tsys. To make a selection you create a `selector` object which you then define with various selection functions, e.g.,

```
  sel = sd.selector()      # initialize a selector object
                           # sel.<TAB> will list all options
  sel.set_ifs(0)           # select only the first IF of the data
  scans.set_selection(sel) # apply the selection to the data
  print scans              # shows just the first IF
```

8.2.3.2 State Information

Some properties of a scantable apply to all of the data, such as example, spectral units, frequency frame, or Doppler type. This information can be set using the `scantable` _set_xxxx_ methods. These are currently:

```
CASA <1>: sd.scantable.set_<TAB>
sd.scantable.set_dirframe    sd.scantable.set_fluxunit    sd.scantable.set_restfreqs
sd.scantable.set_doppler     sd.scantable.set_freqframe   sd.scantable.set_selection
sd.scantable.set_feedtype    sd.scantable.set_instrument  sd.scantable.set_unit
```

For example, `sd.scantable.set_fluxunit` sets the default units that describe the flux axis:

```
  scans.set_fluxunit('K')  # Set the flux unit for data to Kelvin
```

Choices are `'K'` or `'Jy'`. Note: the scantable.set_fluxunit function only changes the **name** of the current fluxunit. To change fluxunits, use `scantable.convert_flux` as described in § 8.2.4.2 instead (currently you need to do some gymnastics for GBT or non-AT telescopes).

Use `sd.scantable.set_unit` to set the units to be used on the spectral axis:

```
  scans.set_unit('GHz')    # Use GHZ as the spectral axis for plots
```

The choices for the units are `'km/s'`, `'channel'`, or `'*Hz'` (e.g. `'GHz'`, `'MHz'`, `'kHz'`, `'Hz'`). This does the proper conversion using the current frame and doppler reference as can be seen when the spectrum is plotted.

You can use `sd.scantable.set_freqframe` to set the frame in which the freqency (spectral) axis is defined:

```
CASA <2>: help(sd.scantable.set_freqframe)
Help on method set_freqframe in module asap.scantable:

set_freqframe(self, frame=None) unbound asap.scantable.scantable method
    Set the frame type of the Spectral Axis.
    Parameters:
        frame:   an optional frame type, default 'LSRK'. Valid frames are:
                 'REST', 'TOPO', 'LSRD', 'LSRK', 'BARY',
                 'GEO', 'GALACTO', 'LGROUP', 'CMB'
    Examples:
        scan.set_freqframe('BARY')
```

The most useful choices here are `frame = 'LSRK'` (the default for the function) and `frame = 'TOPO'` (what the GBT actually observes in). Note that the `'REST'` option is not yet available. The doppler frame is set with `sd.scantable.set_doppler`:

```
CASA <3>: help(sd.scantable.set_doppler)
Help on method set_doppler in module asap.scantable:

set_doppler(self, doppler='RADIO') unbound asap.scantable.scantable method
    Set the doppler for all following operations on this scantable.
    Parameters:
        doppler:    One of 'RADIO', 'OPTICAL', 'Z', 'BETA', 'GAMMA'
```

Finally, there are a number of functions to query the state of the scantable. These can be found in the usual way:

```
CASA <4>: sd.scantable.get<TAB>
sd.scantable.get_abcissa      sd.scantable.get_restfreqs     sd.scantable.getbeamnos
sd.scantable.get_azimuth      sd.scantable.get_scan          sd.scantable.getcycle
sd.scantable.get_column_names sd.scantable.get_selection     sd.scantable.getif
sd.scantable.get_direction    sd.scantable.get_sourcename    sd.scantable.getifnos
sd.scantable.get_elevation    sd.scantable.get_time          sd.scantable.getpol
sd.scantable.get_fit          sd.scantable.get_tsys          sd.scantable.getpolnos
sd.scantable.get_fluxunit     sd.scantable.get_unit          sd.scantable.getscan
sd.scantable.get_parangle     sd.scantable.getbeam           sd.scantable.getscannos
```

These include functions to get the current values of the states mentioned above, as well as as methods to query the number of scans, IFs, and polarizations in the scantable, and their designations. See the inline help for the individual functions for more information.

8.2.3.3 Masks

Several functions (fitting, baseline subtraction, statistics, etc) may be run on a range of channels (or velocity/frequency ranges). You can create masks of this type using the `create_mask` function:

```
  # spave = an averaged spectrum
  spave.set_unit('channel')
  rmsmask=spave.create_mask([5000,7000])   # create a region over channels 5000-7000
  rms=spave.stats(stat='rms',mask=rmsmask) # get rms of line free region

  rmsmask=spave.create_mask([3000,4000],invert=True) # choose the region
                                                     # *excluding* the specified channels
```

The mask is stored in a simple Python variable (a list) and so may be manipulated using an Python facilities.

8.2.3.4 Scantable Management

`scantables` can be listed via:

```
  CASA <33>: sd.list_scans()
  The user created scantables are:
  ['scans20to24', 's', 'scan27']
```

As every `scantable` will consume memory, if you will not use it any longer, you can explicitly remove it via:

```
  del <scantable name>
```

8.2.3.5 Scantable Mathematics

It is possible to do simple mathematics directly on `scantables` from the CASA command line using the $+, -, *, /$ operators as well as their cousins $+=, -=, *=, /=$

```
CASA <10>: scan2=scan1+2.0 # add 2.0 to data
CASA <11>: scan *= 1.05    # scale spectrum by 1.05
```

NOTE: mathematics between two scantables is not currently available in ASAP.

8.2.3.6 Scantable Save and Export

ASAP can save scantables in a variety of formats, suitable for reading into other packages. The formats are:

- ASAP – This is the internal format used for ASAP. It is the only format that allows the user to restore the data, fits, etc, without loosing any information. As mentioned before, the ASAP scantable is a CASA Table (memory-based table). This function just converts it to a disk-based table. You can access this with the CASA `browsetable` task or any other CASA table tasks.
- SDFITS – The Single Dish FITS format. This format was designed for interchange between packages but few packages can actually read it.
- ASCII – A simple text based format suitable for the user to process using Python or other means.
- MeasurementSet (V2: CASA format) – Saves the data in a MeasurementSet. All CASA tasks which use an MS should work on this.

```
scans.save('output_filename','format'), e.g.,
CASA <19>: scans.save('FLS3a_calfs','MS2')
```

8.2.4 Calibration

For some observatories, the calibration happens transparently as the input data contains the Tsys measurements taken during the observations. The nominal 'Tsys' values may be in Kelvin or Jansky. The user may wish to apply a Tsys correction or apply gain-elevation and opacity corrections.

8.2.4.1 Tsys scaling

If the nominal Tsys measurement at the telescope is wrong due to incorrect calibration, the `scale` function allows it to be corrected.

```
scans.scale(1.05,tsys=True) # by default only the spectra are scaled
                            # (and not the corresponding Tsys) unless tsys=True
```

8.2.4.2 Flux and Temperature Unit Conversion

To convert measurements in Kelvin to Jansky (and vice versa), the `convert_flux` function may be used. This converts and scales the data to the selected units. The user may need to supply the aperture efficiency, telescope diameter or the Jy/K factor

```
scans.convert_flux(eta=0.48, d=35.) # Unknown telescope
scans.convert_flux(jypk=15) # Unknown telecope (alternative)
scans.convert_flux() # known telescope (mostly AT telescopes)
scans.convert_flux(eta=0.48) # if telescope diameter known
```

8.2.4.3 Gain-Elevation and Atmospheric Optical Depth Corrections

At higher frequencies, it is important to make corrections for atmospheric opacity and gain-elevation effects. **NOTE:** Currently, the MS to scantable conversion does not adequately populate the azimuth and elevation in the `scantable`. As a result, one must calculate these via:

```
scans.recalc_azel()
Computed azimuth/elevation using
Position: [882590, -4.92487e+06, 3.94373e+06]
 Time: 01:48:38 Direction: 05:35:13.5 -05.24.08.2
   => azel: 154.696 43.1847 (deg)
 Time: 01:48:38 Direction: 05:35:13.5 -05.24.08.2
   => azel: 154.696 43.1847 (deg)
 Time: 01:48:38 Direction: 05:35:13.5 -05.24.08.2
   => azel: 154.696 43.1847 (deg)
 Time: 01:48:38 Direction: 05:35:13.5 -05.24.08.2
   => azel: 154.696 43.1847 (deg)
 Time: 01:48:38 Direction: 05:35:13.5 -05.24.08.2
   => azel: 154.696 43.1847 (deg)
 ...
```

Once you have the correct Az/El, you can correct for a *known* opacity by:

```
scans.opacity(tau=0.09)  # Opacity from which the correction factor:
                         # exp(tau*zenith-distance)
```

8.2.4.4 Calibration of GBT data

Data from the GBT is uncalibrated and comes as sets of integrations representing the different phases within a calibration cycle (e.g., on source, calibration on, on source, calibration off, on reference, calibration on; on reference, calibration off). Currently, there are a number of routines emulating the standard GBT calibration (in GBTIDL):

- calps - calibrate position switched data
- calfs - calibrate frequency switched data

- calnod - calibration nod (beam switch) data

All these routines calibrate the spectral data to antenna temperature adopting the GBT calibration method as described in the GBTIDL calibration document available at:

- `http://wwwlocal.gb.nrao.edu/GBT/DA/gbtidl/gbtidl_calibration.pdf`

There are two basic steps:

First: determine system temperature using a noise tube calibrator (sd.dototalpower())

For each integration, the system temperature is calculated from CAL noise on/off data as:

$T_{sys} = T_{cal}$ x $\frac{<ref_{caloff}>}{<ref_{calon} - ref_{caloff}>} + \frac{T_{cal}}{2}$

`ref` refers to reference data and the spectral data are averaged across the bandpass. Note that the central 80% of the spectra are used for the calculation.

Second, determine antenna temperature (sd.dosigref())

The antenna temperature for each channel is calculated as:

$T_a(\nu) = T_{sys}$ x $\frac{sig(\nu) - ref(\nu)}{ref(\nu)}$

where $sig = \frac{1}{2}(sig_{calon} + sig_{caloff})$, $ref = \frac{1}{2}(sig_{calon} + sig_{caloff})$.

Each calibration routine may be used as:

```
scans=sd.scantable('inputdata',False)         # create a scantable called 'scans'
calibrated_scans = sd.calps(scans,[scanlist]) # calibrate scantable with position-switched
                                              # scheme
```

Note: For calps and calnod, the scanlist must be scan pairs in correct order as these routines only do miminum checking.

8.2.5 Averaging

One can average polarizations in a scantable using the `sd.scantable.average_pol` function:

```
averaged_scan = scans.average_pol(mask,weight)

where:
  Parameters:
      mask:       An optional mask defining the region, where the
                  averaging will be applied. The output will have all
                  specified points masked.
      weight:     Weighting scheme. 'none' (default), 'var' (1/var(spec)
                  weighted), or 'tsys' (1/Tsys**2 weighted)

  Example:

spave = stave.average_pol(weight='tsys')
```

One can also average scans over time using `sd.average_time`:

```
  sd.average_time(scantable,mask,scanav,weight,align)

  where:

    Parameters:
        one scan or comma separated  scans
        mask:     an optional mask (only used for 'var' and 'tsys' weighting)
        scanav:   True averages each scan separately.
                  False (default) averages all scans together,
        weight:   Weighting scheme.
                    'none'     (mean no weight)
                    'var'      (1/var(spec) weighted)
                    'tsys'     (1/Tsys**2 weighted)
                    'tint'     (integration time weighted)
                    'tintsys'  (Tint/Tsys**2)
                    'median'   ( median averaging)
        align:    align the spectra in velocity before averaging. It takes
                  the time of the first spectrum in the first scantable
                  as reference time.
    Example:

  stave = sd.average_time(scans,weight='tintsys')
```

Note that alignment of the velocity frame should be done before averaging if the time spanned by the scantable is long enough. This is done through the `align=True` option in `sd.average_time`, or explicity through the `sd.scantable.freq_align` function, e.g.

```
CASA <62>: sc = sd.scantable('orions_scan20to23_if0to3.asap',False)
CASA <63>: sc.freq_align()
Aligned at reference Epoch 2006/01/19/01:49:23 (UTC) in frame LSRK
CASA <64>: av = sd.average_times(sc)
```

The time averaging can also be applied to multiple scantables. This might have been taken on different days, for example. The `sd.average_time` function takes multiple scantables as input. However, if taken at significantly different times (different days for example) then `sd.scantable.freq_align` must be used to align the velocity scales to the same time, e.g.

```
CASA <65>: sc1 = sd.scantable('orions_scan21_if0to3.asap',False)
CASA <66>: sc2 = sd.scantable('orions_scan23_if0to3.asap',False)
CASA <67>: sc1.freq_align()
Aligned at reference Epoch 2006/01/19/01:49:23 (UTC) in frame LSRK
CASA <68>: sc2.freq_align(reftime='2006/01/19/01:49:23')
Aligned at reference Epoch 2006/01/19/01:54:46 (UTC) in frame LSRK
CASA <69>: scav = sd.average_times(sc1,sc2)
```

8.2.6 Spectral Smoothing

Smoothing on data can be done as follows:

```
scantable.smooth(kernel,    # type of smoothing: 'hanning' (default), 'gaussian', 'boxcar'
        width,              # width in pixls (ignored for hanning); FWHM for gaussian.
        insitu)             # if False (default), do smoothing in-situ; otherwise,
                            # make new scantable

Example:
# spave is an averaged spectrum
spave.smooth('boxcar',5)    # do a 5 pixel boxcar smooth on the spectrum
sd.plotter.plot(spave)      # should see smoothed spectrum
```

8.2.7 Baseline Fitting

The function `sd.scantable.poly_baseline` carries out a baseline fit, given an mask of channels (if desired):

```
msk=scans.create_mask([100,400],[600,900])
scans.poly_baseline(msk,order=1)
```

This will fit a first order polynomial to the selected channels and subtract this polynomial from the full spectrum.

The `auto_poly_baseline` function can be used to automatically baseline your data without having to specify channel ranges for the line free data. It automatically figures out the line-free emission and fits a polynomial baseline to that data. The user can use masks to fix the range of channels or velocity range for the fit as well as mark the band edge as invalid:

```
scans.auto_poly_baseline(mask,edge,order,threshold,chan_avg_limit,plot,insitu):

  Parameters:
      mask:       an optional mask retreived from scantable
      edge:       an optional number of channel to drop at
                  the edge of spectrum. If only one value is
                  specified, the same number will be dropped from
                  both sides of the spectrum. Default is to keep
                  all channels. Nested tuples represent individual
                  edge selection for different IFs (a number of spectral
                  channels can be different)
      order:      the order of the polynomial (default is 0)
      threshold:  the threshold used by line finder. It is better to
                  keep it large as only strong lines affect the
                  baseline solution.
      chan_avg_limit:
                  a maximum number of consequtive spectral channels to
                  average during the search of weak and broad lines.
                  The default is no averaging (and no search for weak
                  lines). If such lines can affect the fitted baseline
                  (e.g. a high order polynomial is fitted), increase this
                  parameter (usually values up to 8 are reasonable). Most
                  users of this method should find the default value
                  sufficient.
```

```
    plot:       plot the fit and the residual. In this each
                indivual fit has to be approved, by typing 'y'
                or 'n'
    insitu:     if False a new scantable is returned.
                Otherwise, the scaling is done in-situ
                The default is taken from .asaprc (False)

  Example:
scans.auto_poly_baseline(order=2,threshold=5)
```

8.2.8 Line Fitting

Multi-component Gaussian fitting is available. This is done by creating a fitting object, specifying fit parameters and finally fitting the data. Fitting can be done on a `scantable` selection or an entire `scantable` using the `auto_fit` function.

```
#spave is an averaged spectrum
f=sd.fitter()                        # create fitter object
msk=spave.create_mask([3928,4255])   # create mask region around line
f.set_function(gauss=1)              # set a single gaussian component
f.set_scan(spave,msk)                # set the scantable and region
                                     #
                                     # Automatically guess start values
f.fit()                              # fit
f.plot(residual=True)                # plot residual
f.get_parameters()                   # retrieve fit parameters
#   0: peak = 0.786 K , centre = 4091.236 channel, FWHM = 70.586 channel
#      area = 59.473 K channel
f.store_fit('orions_hc3n_fit.txt')   # store fit
                                     #
                                     # To specify initial guess:
f.set_function(gauss=1)              # set a single gaussian component
f.set_gauss_parameters(0.4,4100,200\ # set initial guesses for Gaussian
      ,component=0)                  #   for first component (0)
                                     #   (peak,center,fwhm)
                                     #
                                     # For multiple components set
                                     # initial guesses for each, e.g.
f.set_function(gauss=2)              # set two gaussian components
f.set_gauss_parameters(0.4,4100,200\ # set initial guesses for Gaussian
      ,component=0)                  #   for first component (0)
f.set_gauss_parameters(0.1,4200,100\ # set initial guesses for Gaussian
      ,component=1)                  #   for second component (1)
```

8.2.9 Plotting

The ASAP plotter uses the same Python matplotlib library as in CASA (for x-y plots). It is accessed via the:

```
 sd.plotter<TAB>        # see all functions (omitted here)
sd.plotter.plot(scans) # the workhorse function
sd.plotter.set<TAB>
sd.plotter.set_abcissa     sd.plotter.set_legend      sd.plotter.set_range
sd.plotter.set_colors      sd.plotter.set_linestyles  sd.plotter.set_selection
sd.plotter.set_colours     sd.plotter.set_mask        sd.plotter.set_stacking
sd.plotter.set_font        sd.plotter.set_mode        sd.plotter.set_title
sd.plotter.set_histogram   sd.plotter.set_ordinate
sd.plotter.set_layout      sd.plotter.set_panelling
```

Spectra can be plotted at any time, and it will attempt to do the correct layout depending on whether it is a set of scans or a single scan.

The details of the plotter display (matplotlib) are detailed in the earlier section.

8.2.10 Single Dish Spectral Analysis Use Case With ASAP Toolkit

Below is a script that illustrates how to reduce single dish data using ASAP within CASA. First a summary of the dataset is given and then the script.

```
#           MeasurementSet Name:  /home/rohir3/jmcmulli/SD/OrionS_rawACSmod      MS Version 2
#
# Project: AGBT06A_018_01
# Observation: GBT(1 antennas)
#
#Data records: 256       Total integration time = 1523.13 seconds
#   Observed from   01:45:58   to   02:11:21
#
#Fields: 4
#  ID   Name          Right Ascension  Declination   Epoch
#  0    OrionS        05:15:13.45      -05.24.08.20  J2000
#  1    OrionS        05:35:13.45      -05.24.08.20  J2000
#  2    OrionS        05:15:13.45      -05.24.08.20  J2000
#  3    OrionS        05:35:13.45      -05.24.08.20  J2000
#
#Spectral Windows:  (8 unique spectral windows and 1 unique polarization setups)
#  SpwID  #Chans Frame Ch1(MHz)    Resoln(kHz) TotBW(kHz)  Ref(MHz)    Corrs
#  0        8192 LSRK  45464.3506  6.10423298  50005.8766  45489.3536  RR  LL HC3N
#  1        8192 LSRK  45275.7825  6.10423298  50005.8766  45300.7854  RR  LL HN15CO
#  2        8192 LSRK  44049.9264  6.10423298  50005.8766  44074.9293  RR  LL CH3OH
#  3        8192 LSRK  44141.2121  6.10423298  50005.8766  44166.2151  RR  LL HCCC15N
#  12       8192 LSRK  43937.1232  6.10423356  50005.8813  43962.1261  RR  LL HNCO
#  13       8192 LSRK  42620.4173  6.10423356  50005.8813  42645.4203  RR  LL H15NCO
#  14       8192 LSRK  41569.9768  6.10423356  50005.8813  41594.9797  RR  LL HNC18O
#  15       8192 LSRK  43397.8198  6.10423356  50005.8813  43422.8227  RR  LL SiO

# Scans: 21-24  Setup 1 HC3N et al
# Scans: 25-28  Setup 2 SiO et al

casapath=os.environ['AIPSPATH']
```

```
#ASAP script                            # COMMENTS
#-------------------------------------- -----------------------------------------------
import asap as sd                       #import ASAP package into CASA
                                        #Orion-S (SiO line reduction only)
                                        #Notes:
                                        #scan numbers (zero-based) as compared to GBTIDL

                                        #changes made to get to OrionS_rawACSmod
                                        #modifications to label sig/ref positions
os.environ['AIPSPATH']=casapath         #set this environment variable back - ASAP changes it


s=sd.scantable('OrionS_rawACSmod',False)#load the data without averaging
```

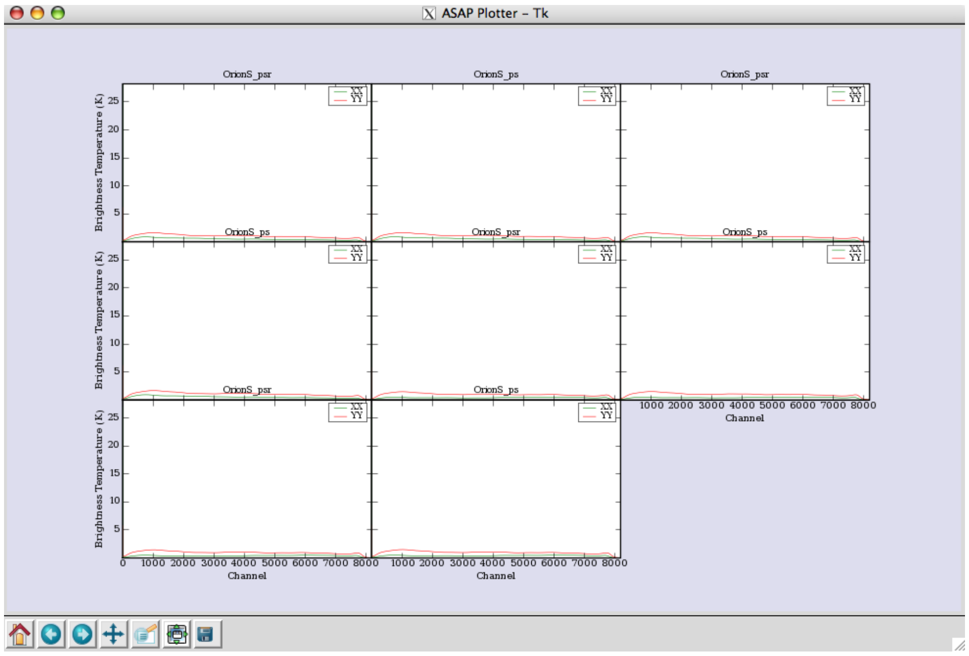


Figure 8.1: Multi-panel display of the scantable. There are two plots per scan indicating the _psr (reference position data) and the _ps (source data).

```
s.summary()                             #summary info
s.set_fluxunit('K')                     # make 'K' default unit
scal=sd.calps(s,[20,21,22,23])          # Calibrate HC3N scans


scal.recalc_azel()                      # recalculate az/el to
scal.opacity(0.09)                      # do opacity correction
sel=sd.selector()                       # Prepare a selection
```

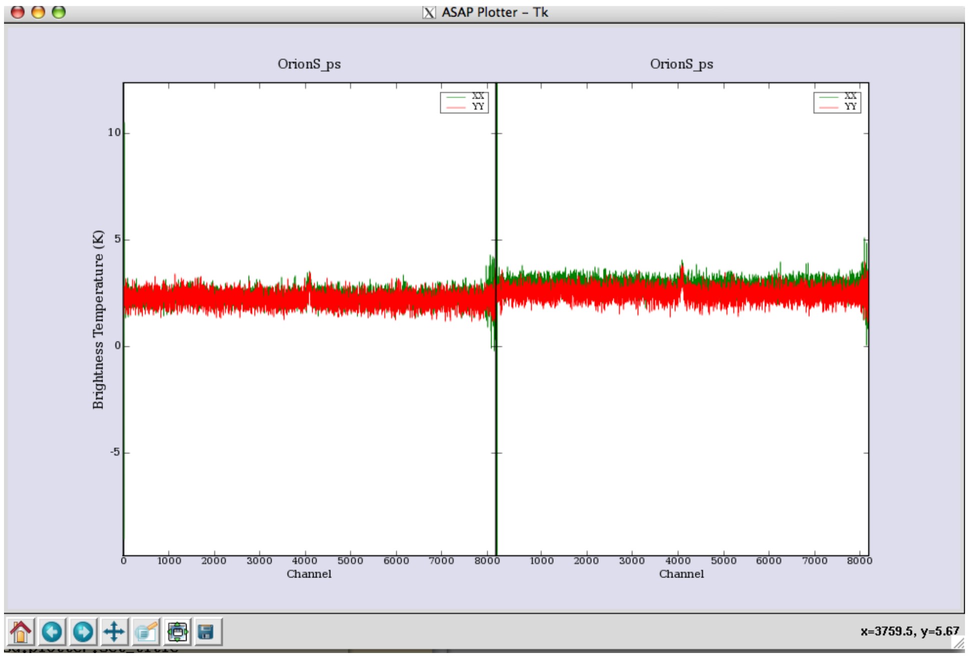


Figure 8.2: Two panel plot of the calibrated spectra. The GBT data has a separate scan for the SOURCE and REFERENCE positions so scans 20,21,22 and 23 result in these two spectra.

```
sel.set_ifs(0)                          # select HC3N IF
scal.set_selection(sel)                 # get this IF
stave=sd.average_time(scal,weight='tintsys')    # average in time
spave=stave.average_pol(weight='tsys')  # average polarizations;Tsys-weighted (1/Tsys**2) average
sd.plotter.plot(spave)                  # plot

spave.smooth('boxcar',5)                # boxcar 5
spave.auto_poly_baseline(order=2)       # baseline fit order=2
sd.plotter.plot(spave)                  # plot

spave.set_unit('GHz')
sd.plotter.plot(spave)
sd.plotter.set_histogram(hist=True)       # draw spectrum using histogram
sd.plotter.axhline(color='r',linewidth=2) # zline
sd.plotter.save('orions_hc3n_reduced.eps')# save postscript spectrum


spave.set_unit('channel')
rmsmask=spave.create_mask([5000,7000])  # get rms of line free regions
rms=spave.stats(stat='rms',mask=rmsmask)#  rms
                                        #-----------------------------------------------
                                        #Scan[0] (OrionS_ps) Time[2006/01/19/01:52:05]:
                                        # IF[0] = 0.048
                                        #-----------------------------------------------
                                        # LINE
```

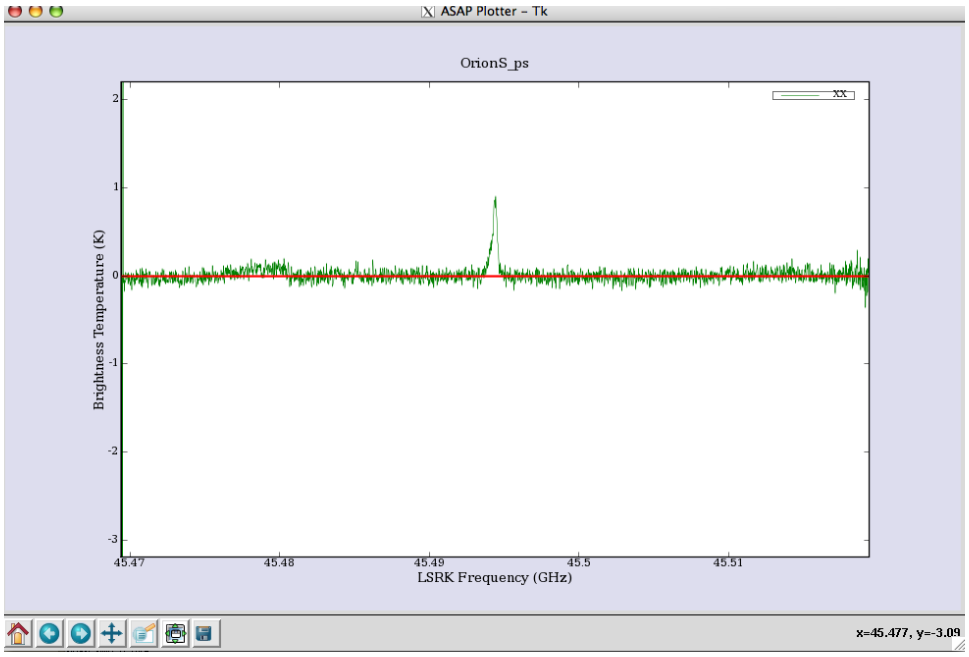


Figure 8.3: Calibrated spectrum with a line at zero (using histograms).

```
linemask=spave.create_mask([3900,4200])
max=spave.stats('max',linemask)         #  IF[0] = 0.918
sum=spave.stats('sum',linemask)         #  IF[0] = 64.994
median=spave.stats('median',linemask)   #  IF[0] = 0.091
mean=spave.stats('mean',linemask)       #  IF[0] = 0.210
```

```
                                        # Fitting
spave.set_unit('channel')               # set units to channel
sd.plotter.plot(spave)                  # plot spectrum
f=sd.fitter()
msk=spave.create_mask([3928,4255])      # create region around line
f.set_function(gauss=1)                 # set a single gaussian component
f.set_scan(spave,msk)                   # set the data and region for the fitter
f.fit()                                 # fit
f.plot(residual=True)                   # plot residual

f.get_parameters()                      # retrieve fit parameters
#   0: peak = 0.786 K , centre = 4091.236 channel, FWHM = 70.586 channel
#      area = 59.473 K channel
```

```
f.store_fit('orions_hc3n_fit.txt')      # store fit

# Save the spectrum
spave.save('orions_hc3n_reduced','ASCII',True)  # save the spectrum
```

8.3 Single Dish Imaging

Single dish imaging is supported within CASA using standard tasks and tools. The data must be in the MeasurementSet format. Once there, you can use the `sdgrid` task or the `im` (imager) tool to create images:

Tool example:

```
  scans.save('outputms','MS2')                     # Save your data from ASAP into an MS

  im.open('outputms')                              # open the data set
  im.selectvis(nchan=901,start=30,step=1,          # choose a subset of the dataa
     spwid=0,field=0)                              # (just the key emission channels)
  dir='J2000 17:18:29 +59.31.23'                   # set map center
  im.defineimage(nx=150,cellx='1.5arcmin',         # define image parameters
     phasecenter=dir,mode='channel',start=30,      # (note it assumes symmetry if ny,celly
     nchan=901,step=1)                             #  aren't specified)

  im.setoptions(ftmachine='sd',cache=1000000000)  # choose SD gridding
  im.setsdoptions(convsupport=4)                   # use this many pixels to support the
                                                   # gridding function used
                                                   # (default=prolate spheroidal wave function)
  im.makeimage(type='singledish',                  # make the image
     image='FLS3a_HI.image')
```

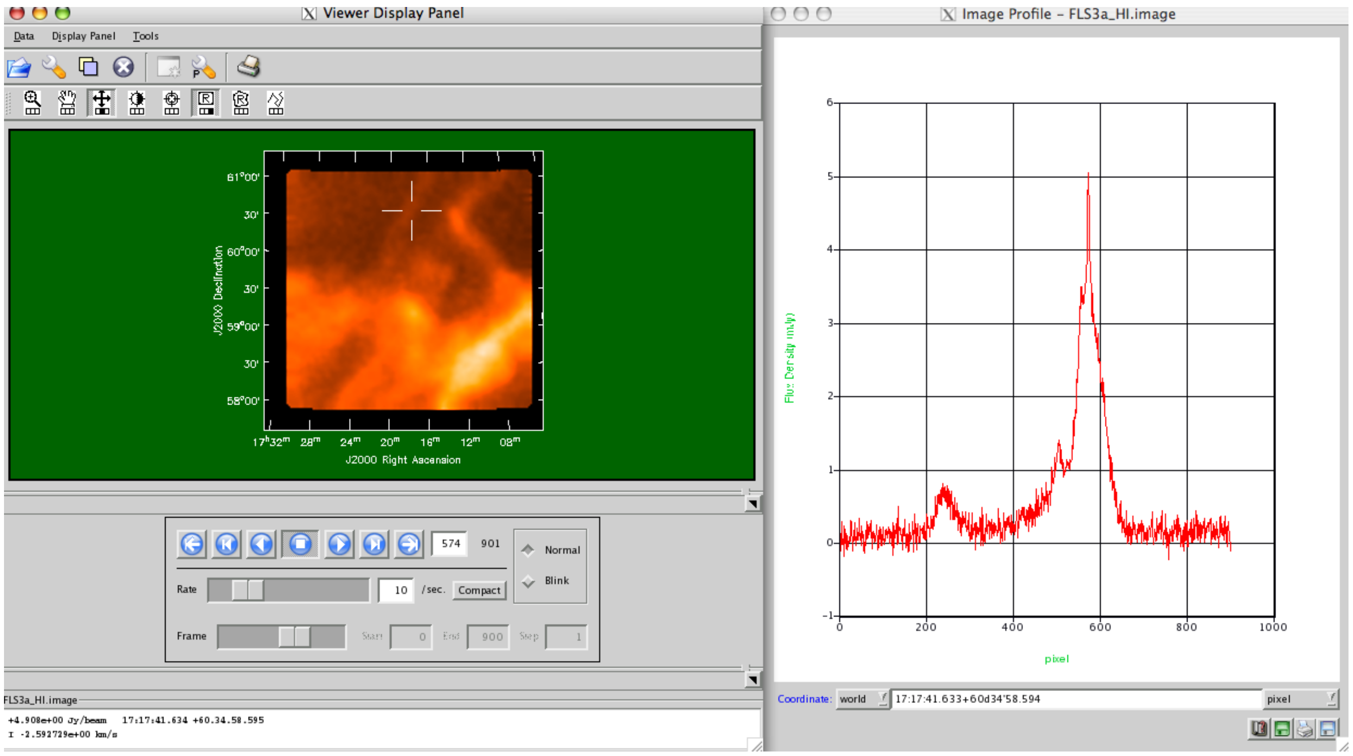
8.3.1 Single Dish Imaging Use Case With ASAP Toolkit

Again, the data summary and then the script is given below.

```
# Project: AGBT02A_007_01
# Observation: GBT(1 antennas)
#
#   Telescope Observation Date    Observer       Project
#   GBT       [                   4.57539e+09, 4.5754e+09]Lockman        AGBT02A_007_01
#   GBT       [                   4.57574e+09, 4.57575e+09]Lockman        AGBT02A_007_02
#   GBT       [                   4.5831e+09, 4.58313e+09]Lockman        AGBT02A_031_12
#
# Thu Feb 1 23:15:15 2007    NORMAL ms::summary:
# Data records: 76860       Total integration time = 7.74277e+06 seconds
#    Observed from   22:05:41   to   12:51:56
```

```
#
# Thu Feb 1 23:15:15 2007    NORMAL ms::summary:
# Fields: 2
#   ID   Name          Right Ascension  Declination   Epoch
#   0    FLS3a         17:18:00.00      +59.30.00.00  J2000
#   1    FLS3b         17:18:00.00      +59.30.00.00  J2000
#
# Thu Feb 1 23:15:15 2007    NORMAL ms::summary:
# Spectral Windows:  (2 unique spectral windows and 1 unique polarization setups)
#   SpwID  #Chans Frame Ch1(MHz)    Resoln(kHz) TotBW(kHz)  Ref(MHz)    Corrs
#   0        1024 LSRK  1421.89269  2.44140625  2500        1420.64269  XX  YY
#   1        1024 LSRK  1419.39269  2.44140625  2500        1418.14269  XX  YY


# FLS3 data calibration
# this is calibration part of FLS3 data
#
casapath=os.environ['AIPSPATH']
import asap as sd
os.environ['AIPSPATH']=casapath

print '--Import--'

s=sd.scantable('FLS3_all_newcal_SP',false)         # read in MeasurementSet

print '--Split--'

# splitting the data for each field
s0=s.get_scan('FLS3a*')                            # split the data for the field of interest
s0.save('FLS3a_HI.asap')                           # save this scantable to disk (asap format)
del s0                                             # free up memory from scantable

print '--Calibrate--'
s=sd.scantable('FLS3a_HI.asap')                    # read in scantable from disk (FLS3a)
s.set_fluxunit('K')                                # set the brightness units to Kelvin
scanns = s.getscannos()                            # get a list of scan numbers
sn=list(scanns)                                    # convert it to a list
print "No. scans to be processed:", len(scanns)

res=sd.calfs(s,sn)                                 # calibrate all scans listed using frequency
                                                   # switched calibration method

print '--Save calibrated data--'
res.save('FLS3a_calfs', 'MS2')                     # Save the dataset as a MeasurementSet

print '--Image data--'

im.open('FLS3a_calfs')                             # open the data set
im.selectvis(nchan=901,start=30,step=1,            # choose a subset of the dataa
spwid=0,field=0)                                   # (just the key emission channels)
dir='J2000 17:18:29 +59.31.23'                     # set map center
```

```
im.defineimage(nx=150,cellx='1.5arcmin',           # define image parameters
phasecenter=dir,mode='channel',start=30,           # (note it assumes symmetry if ny,celly
nchan=901,step=1)                                  #  aren't specified)

im.setoptions(ftmachine='sd',cache=1000000000)     # choose SD gridding
im.setsdoptions(convsupport=4)                     # use this many pixels to support the
                                                   # gridding function used
                                                   # (default=prolate spheroidal wave function)
im.makeimage(type='singledish',image='FLS3a_HI.image') # make the image
```



```
+4.908e+00 Jy/beam   17:17:41.634 +60.34.58.595
I -2.592729e+00 km/s
```

```
17:17:41.633+60d34'58.594
```

Figure 8.4: FLS3a HI emission. The display illustrates the visualization of the data cube (left) and the profile display of the cube at the cursor location (right); the Tools menu of the Viewer Display Panel has a Spectral Profile button which brings up this display. By default, it grabs the left-mouse button. Pressing down the button and moving in the display will show the profile variations.

8.4 Known Issues, Problems, Deficiencies and Features

The Single-Dish calibration and analysis package within CASA is still very much under development. Not surprisingly, there are a number of issues with ASAP that are known and are under repair. Some of these are non-obvious "features" of the way ASAP or `sd` is implemented, or limitations of the current Python tasking environment. Some are functions that have yet to be implemented. These currently include:

1. `sd.plotter`

   Currently you can get hardcopy only after making a viewed plot. Ideally, ASAP should allow you to choose the device for plotting when you set up the plotter.

   Multi-panel plotting is poor. Currently you can only add things (like lines, text, etc.) to the first panel. Also, `sd.plotter.set_range()` sets the same range for multiple panels, while we would like it to be able to set the range for each independently, including the default ranges.

   The appearance of the plots need to be made a lot better. In principle matplotlib can make "publication quality" figures, but in practice you have to do alot of work to make it do that, and our plots are not good.

   The sd.plotter object remembers things throughout the session and thus can easily get confused. For example you have to reset the range `sd.plotter.set_range()` if you have ever set it manually. This is not always the expected behavior but is a consequence of having `sd.plotter` be its own object that you feed data and commands to.

   Eventually we would like the capability to interactively set things using the plots, like select frequency ranges, identify lines, start fitting.

2. `sd.selector`

   The selector object only allows one selection of each type. It would be nice to be able to make a union of selections (without resorting to query) for the `set_name` - note that the others like scans and IFs work off lists which is fine. Should make `set_name` work off lists of names.

3. `sd.scantable`

   There is no useful inline help on the scantable constructor when you do `help sd.scantable`, nor in help sd.

   The inline help for scantable.summary claims that there is a verbose parameter, but there is not. The scantable.verbosesummary asaprc parameter (e.g. in `sd.rcParams`) does nothing.

   GBT data has incorrect fluxunit (`'Jy'`, should be `'K'`), freqframe (`'LSRK'`, is really `'TOPO'`) and reference frequency (set to that of the first IF only).

   You cannot set the rest frequencies for GBT data. THIS IS THE MOST SERIOUS BUG RIGHT NOW.

   The `sd.scantable.freq_align` does not yet work correctly.

   Need to add to scantable.stats: `'maxord', 'minord'` - the ordinate (channel, vel, freq) of the max/min

4. `sd` general issues

   There should be a `sdhelp` equivalent of `toolhelp` and `tasklist` for the sd tools and tasks.

   The current output of ASAP is verbose, and is controlled by setting `sd.rcParams['verbose']=False` (or `True`). At the least we should make some of the output less cryptic.

   Strip off leading and trailing whitespace on string parameters.