

CASA Primer

The Basics:

CASA (Common Astronomy Software Applications) is the data analysis software package for the next generation of radio telescopes, specifically ALMA and the EVLA. It can also be used to process archival and current VLA data. CASA performs end-to-end processing of the large datasets resulting from these new telescopes: from data calibration to image analysis. This document is intended to be a very brief introduction to the basic functions and organization of CASA. Much more detailed information is available in the CASA Cookbook, which you can download from the CASA homepage: <http://casa.nrao.edu>. The package is still under development, and full functionality is not yet available. Check with the CASA homepage to find out the latest status of the project.

Instructions for obtaining the most recent release of CASA are available from the CASA homepage. Read the README files for installation instructions for the most recent release.

Starting CASA:

On startup, CASA adopts current settings of environment variables, so set them before starting CASA. For example, PAGER regulates the help display. The cat option works well in both interactive and script mode. In bash, use `PAGER=cat`. In csh or tcsh, use `setenv PAGER cat`.

Navigate to your working directory in an xterm window. Start CASA by typing `casapy` on the Unix command line. CASA is built in iPython, and you will see a number of initialization messages, a list of available tasks, and, finally, a Python command line prompt:

```
CASA <1>:
```

CASA will also spawn a separate logger window, which logs messages from various tasks. You can annotate, search and filter the logger messages. Logger messages are saved in a file `casapy.log`, in your local directory. Whenever you start CASA, the previous log is renamed (based on the date and time), and a new `casapy.log` is opened.

In addition, your command line history is automatically maintained and stored as `ipython.log` in your local directory. This file can be edited and re-executed using `execfile '<filename>'`.

Exiting CASA:

Exit CASA by typing `quit`, `exit`, or `Ctrl-D`. If something has gone wrong, and you want to halt execution, `Ctrl-C` will usually cleanly abort the application.

Crashing CASA:

Simply restarting will get you going again if CASA crashes. Some spawned windows, such as the logger, may need to be closed separately. In Linux, you can find any remaining hidden processes with `ps -elf | grep casa`. (In MacOSX, use `ps -aux | grep casa`.) Kill them with the Unix commands `kill` or `killall`.

CASA Tasks:

CASA processes are organized into tasks and tools. Tools are the underlying functionality of CASA, intended for expert users participating in development. New users should ignore the tools. Tasks utilize a number of tools and accomplish a specific data analysis or calibration goal. To get information about tasks while in CASA, use:

- *tasklist*: lists the names of available tasks;
- *taskhelp*: adds a one-line description of each task;
- *startup*: repeats the task availability information that was given when you started CASA;
- *help <taskname>*: lists a short description of the task, the name of each parameter, a brief description of the parameter, the default value, an example, and allowed values, if applicable;
- *inp <taskname>*: lists the parameters of the task, with current values and a brief description;
- *inp*: same as above, but for the current task.

Data Storage:

CASA stores data in a Measurement Set (MS). Logically, it is a generalized description of data from any interferometer or single-dish telescope. Physically, it is several 'tables' in a directory on disk. 'Tables' in CASA are actually directories containing files that are sub-tables. If you create a MS called AM675.ms, the sub-tables are stored in the directory AM675.ms/.

Calibration solutions or images will also be written to disk as directories and sub-directories. The default is to store them in the working directory, which conveniently eliminates the need for the full path name. If you need to delete an image or a calibration table, be sure to use *rm -r*, so that all the subdirectories will be deleted. For example, to remove the AM675.ms table, type *!rm -r AM675.ms* from within CASA.

The MAIN data table is arranged so that each row is a single timestamp for a single spectral window and a single baseline. There are several columns. DATA holds the original visibility data, CORRECTED holds the calibrated data, MODEL holds the Fourier inversion of a particular model image, and IMAGING_WEIGHT holds the weights to be used in imaging. Occasionally, you will need to specify a column for a particular task, so it is useful to know about them.

Setting Parameters:

Each task has a set of parameters. If a task is successfully executed, then the successful parameter set is stored in a `<taskname>.last` file in the working directory.

Other ways to save and restore parameters are available:

- `tget <taskname>`: retrieves parameters from `<taskname>.last`;
- `default('<taskname>')`: restores the default parameters;
- `saveinputs(<taskname>,<filename>)`: saves the current set of parameters into `<filename>`;
- `execfile '<filename>'`: retrieves a set of parameters and runs the task.

Parameters are set using the Python `<parameter>=<value>` syntax. Some parameters have sub-parameters: an `inp` listing will show these on a gray background. If a gray parameter is set to a value that allows sub-parameters, then the sub-parameters are listed in the next instance of `inp`. Otherwise, the sub-parameters remain hidden. CASA uses multiple font colors and highlighting to give more information about the parameters and their current values. Pay particular attention if the text color is red: the value of the parameter is invalid, and the task will not run.

CASA has two distinct ways of running tasks, one sets parameters globally (so that all tasks will use the same values), and the other does not. This is helpful when writing scripts that you may wish to use more than once. To set global parameters, set each parameter, and then invoke the task:

```
CASA <1>: default('plotxy')
CASA <2>: vis='ngc5921.ms'
CASA <3>: xaxis='channel'
CASA <4>: yaxis='amp'
CASA <5>: go
```

To set parameters for a single instance of a single task, call it as a function:

```
CASA <1>: plotxy(vis='ngc5921.ms',xaxis='channel',yaxis='amp')
```

Other tasks will revert to global values of the parameters (these may be the defaults).

When a task is invoked in this way, unspecified values use the defaults, not the global values!

The `go` command may also be used to invoke a task without changing the current task. For example, you may be setting parameters for `plotxy`, and wish to verify an antenna name. Typing `go listobs` will list the observations in the logger, but will not change the current task to `listobs`. Typing `inp` after this will give the inputs list for `plotxy`, since the current task has not been changed.

Capturing Return Values:

Some tasks return information to the interface. For example, the *imstat* task returns a Python 'dictionary' with the image statistics in it. To catch these return statistics, you must assign a variable to the task call. For example:

`xstat=imstat('ngc5921.clean.image')`. To see what's in the 'dictionary', type the variable name (in this case, *xstat*) at the command line.

Python Basics:

Typing *help* at the CASA prompt with no arguments will bring up the native Python help, which has information about Python commands and keywords. Hitting <RETURN> at the Python help> prompt will return you to CASA.

Command-line Completion and Recall:

Hitting the <TAB> key completes unambiguous commands or variable names and shows you a list of the possible completions, if there are more than one. It also completes filenames from the working directory if no CASA or Python commands match. Once you have typed enough of a command to make it unique, <TAB> will complete it. This will help you avoid errors due to typos. You can also use up and down arrow command line recall in the CASA interface. Begin typing a command, and then use the up arrow; you will navigate back and edit commands matching what you typed.

Setting Variables:

Set Python variables using <parameter>=<value> syntax. Python assigns the type dynamically as you set the value, and will allow you to assign an integer value to a string parameter. However, CASA will check types before running a task, and alert you if you have made a mistake. Python variable names are case sensitive. Also note that misspelling a variable assignment will not be noticed by the interface. For example, you might wish to set *correlation*='RR', but instead type *corellation*='RR'. You will find *correlation* unset, and a new variable, *corellation* will exist, with value RR. Using command-line completion and recall helps avoid these errors.

Lists and Ranges:

The Python *range* function give a task a list of indices. For example, *iflist=range(4,8)* will set *iflist* = [4, 5, 6, 7].

Indexes:

Python indices are 0-based. The first element in the list *antlist* is *antlist[0]*. CASA also uses 0-based indexing. Field or antenna IDs, for example, start at 0. The first field in a measurement set has *FIELD_ID*=0

Indentation:

Python uses indentation to determine the level of nesting in loops. Be careful when cutting and pasting, you may get an error (or an erroneous result).

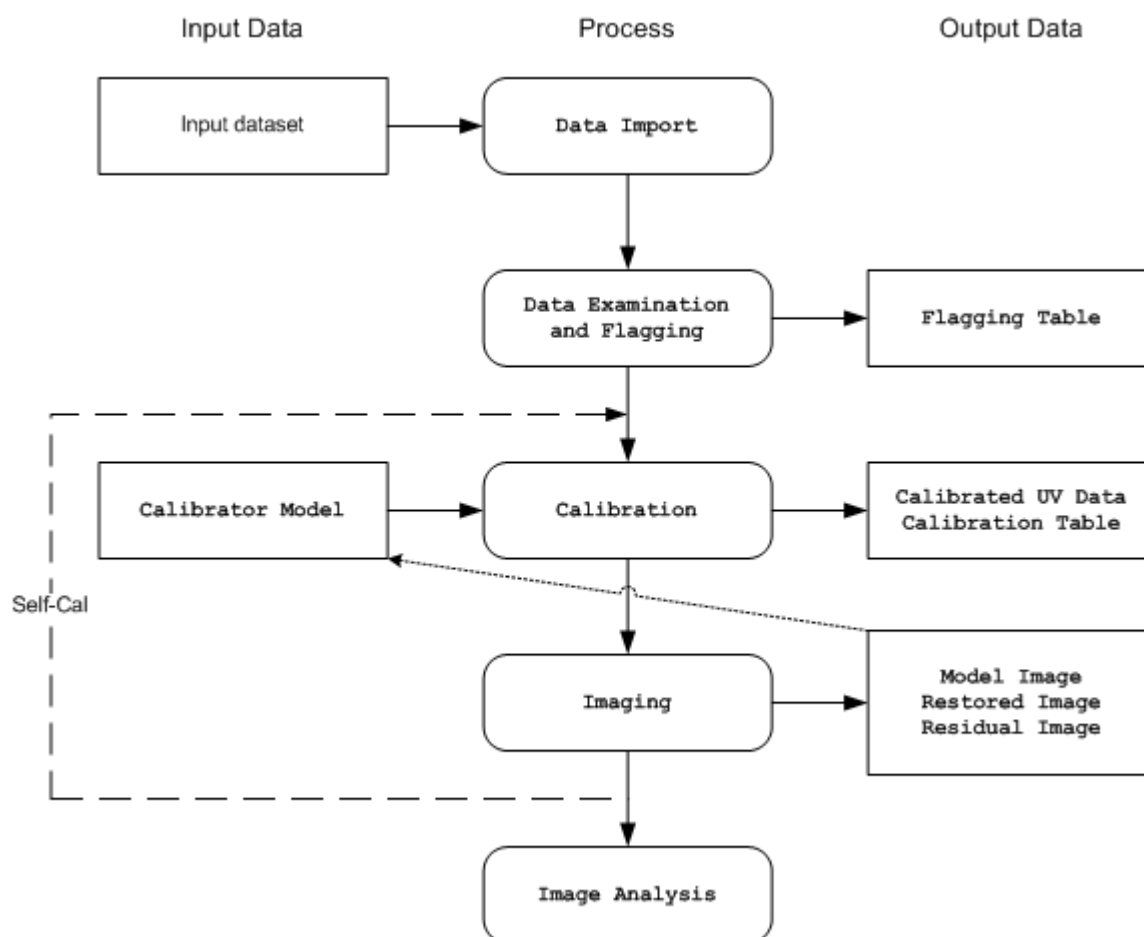
System shell access:

To access system commands from CASA, precede your command with a bang (!). For example, to remove the file *mydata.ms*, type *!rm -r mydata.ms* This is inconsistent. Some commands (*ls*, *pwd*, *less*) can be executed without the '!'. The *cp* command must use the '!', and the *cd* command must NOT use the '!'. Note: to access a Unix environment variable from CASA, you will need to use \$\$ instead of \$.

Running Scripts:

You may wish to run scripts which contain series of commands for data calibration, etc. To execute the script contained in the text file `myscript.py`, type `execfile('myscript.py')`.

End-to-end: An overview of the process from Loading to Imaging:



Importing Data and Images:

Data is initially loaded into CASA using one of three import tasks, `importuvfits`, `importvla`, `importasdm`. These import UVFITS, VLA and ALMA data, respectively. Two other related tasks are available: `importfits` can be used to import a FITS image into a CASA image format table; and `concat` can be used to concatenate a second measurement set into a given MS.

Examining, Editing and Flagging:

There are several tasks to list, plot and/or flag data in a MS:

- `listobs`: summarizes the contents of a MS;
- `flagmanager`: saves and manages versions of the flagging entries in the MS;
- `flagautocorr`: non-interactively flags auto-correlations;

- plotxy: interactively plots and flags visibility data in 2-D;
- flagdata: non-interactively flags and unflags specified data;
- viewer: displays MS data, with some editing capabilities.

Calibration:

During calibration, the user specifies a set of calibrations to pre-apply before solving for a particular effect (gain or bandpass or polarization). The user (NOT the task!) specifies a calibration table to store the solutions. Take care in naming the table for future use. As the calibration proceeds, the user accumulates the calibrations in a new cumulative table. Finally, the calibration is applied to the dataset, and calibrated data are written into the CORRECTED column. Calibration tasks are:

- setjy: computes the model visibilities for a specified source given a flux density. setjy 'knows' about standard calibrator sources;
- bandpass: solves for frequency-dependent complex gains;
- gaincal: solves for time-dependent complex gains;
- fluxscale: bootstraps the flux density scale from standard calibrators;
- polcal: polarization calibration;
- accum: accumulates incremental calibration solutions into a cumulative table;
- smoothcal: smooths calibration solutions derived from one or more sources;
- applycal: applies calculated calibration solutions;
- clearcal: re-initializes calibrated visibility data in a given MS;
- listcal: lists calibration solutions;
- plotcal: plots (and optionally flags) calibration solutions;
- uvcontsub: carry out uv-plane continuum subtraction for spectral-line data
- split: write out a new (calibrated) MS for specified sources

Imaging:

At some point you want to actually see the model of the sky you have created from your data. The main tasks involved are:

- clean: calculates a deconvolved image based on the visibility data;
- feather: combines a single dish and synthesis image in the Fourier plane;
- makemask: makes a mask image from a cleanbox, a file or list;
- ft: Fourier transforms the model (or component list) and fills the MODEL column;
- deconvolve: deconvolves an input image from a provided PSF.

Analysis:

Of course, you will want to measure things! Several tasks are available to help with this:

- imhead: summarize and manipulate the 'header' information in a CASA image;
- imcontsub: perform continuum subtraction on a spectral-line image cube;
- immath: perform mathematical operations on or between images;
- immoments: compute the moments of an image cube;
- imstat: calculate statistics on an image or part of an image;
- regridimage: regrid an image onto the coordinate system of another image;
- viewer: there are useful statistics and plotting capabilities in the viewer.

Exporting:

You can export your data from CASA using either `exportuvfits` or `exportfits`, which save the data in UVFITS or FITS formats, respectively.

Further Information:

About CASA: The CASA Cookbook, available from the CASA homepage:

<http://casa.nrao.edu> Other documentation is there as well.

About Interferometry: Synthesis Imaging in Radio Astronomy II (1999, ASP Conference Series Vol 180, eds. Taylor, Carilli and Perley)

About Python: see the vast internet. Googling 'Python manual' will turn up a number of options.