Caracal Documentation

Release v1.0.0

Paolo Serra

May 12, 2020

Contents

CARACal is a pipeline for radio interferometry data reduction. It works on data from any radio interferometer as long as they are in "measurement set" format.

A few sections of this documentation are still preliminary. Please bear with us.

CHAPTER 1

Installation & Run

1.1 Usage and publication policy

When using CARACal please be aware of and adhere to the CARACal publication policy.

1.2 Requirements

- Python 3.5 or higher.
- Container technology of your choice. It can be one of the following:
  - Docker;
  - Singularity > 2.6.0-dist;
  - Podman **(currently not fully supported)**.

1.3 Manual installation

We strongly recommend and describe an installation using a Python3 virtual environment. Only try outside a virtual environment if you know what you are doing.

Choose the name of the virtual environment `${caracal-venv}`. Then:

```
python3 -m venv ${caracal-venv}
# OR, if the command above does not work
#virtualenv -p python3 ${caracal-venv}

source ${caracal-venv}/bin/activate
pip install -U pip setuptools wheel
```

(continues on next page)

(continued from previous page)

```
# CARACal stable release
pip install -U caracal
# OR CARACal developer version
#pip install -U git+https://github.com/ska-sa/caracal.git#egg=caracal
```

(Ignore any error messages concerning pyregion.)

In case you are *not* carrying out a fresh installation, remove earlier Stimela images with:

```
stimela clean -ac
```

If using Singularity, choose a pull folder `${singularity_pull_folder}`, where the Singularity images are stored and define an environment variable by adding this in the rc file of your shell (e.g. .bashrc):

```
export SINGULARITY_PULLFOLDER=${WORKSPACE_ROOT}/singularity_images
```

and run:

```
stimela pull -s
```

If using Podman (currently not fully supported):

```
stimela pull -p
```

1.4 Installation with the caratekit.sh script

Download the installation script caratekit.sh . Choose the parent directory `${workspace}` and the name of the CARACal directory `${caracal_dir}`.

If using Docker:

```
caratekit.sh -ws ${workspace} -cr -di -ct ${caracal_dir} -rp install -f -kh
```

If using Singularity:

```
caratekit.sh -ws ${workspace} -cr -si -ct ${caracal_testdir} -rp install -f -kh
```

1.5 Run

If you installed CARACal manually, activate the virtual environment with:

```
source ${caracal-venv}/bin/activate
```

If you installed CARACal with the caratekit.sh script, activate the virtual environment with:

```
source ${workspace}/${caracal_dir}/caracal_venv/bin/activate
```

Run CARACal with:

```
caracal - c ${your-configuration-file}
```

For more detailed installation instructions, trouble-shooting tips and a full user manual please see caracal.readthedocs.io.

1.6 Known and new issues

We encourage users who experience problems installing or running CARACal to check for known issues or open a new issue at our GitHub issues page. When opening a new issue, please include your installation type (e.g., Docker, Singularity), software version (both CARACal and Stimela), CARACal configuration file, and CARACal log files.

CHAPTER 2

Manual

2.1 Introduction

2.1.1 What is CARACal?

CARACal is a pipeline to reduce radio interferometry continuum and spectral line data in total intensity. It works on data from any radio interferometer as long as they are in "measurement set" format.

CARACal is essentially a collection of Python/Stimela scripts. Stimela is a platform-independent radio interferometry scripting framework based on Python and container technology (e.g., Docker, Singularity). Stimela allows users to execute tasks from many different data reduction packages in Python without having to install those packages individually (e.g., CASA, MeqTrees, AOflagger, SoFiA, etc.). Using Stimela, the different software packages are available through a unified scheme. CARACal consists of a sequence of Stimela scripts, which it links and runs sequentially.

Within CARACal — and throughout this documentation — the individual Stimela scripts are called "workers". Each CARACal worker corresponds to a specific section of the data reduction process (e.g., flagging, cross-calibration). Each worker executes several tasks from the interferometry packages included in Stimela (e.g., the cross-calibration worker can calibrate delays, bandpass, gains and flux scale).

In practice, users tell CARACal what to do — and how to do it — via a YAML configuration file. The configuration file has one section for each run of a worker (some workers, e.g., the flagging one, might need to be run multiple times). By editing the configuration file users control the workers' options, deciding which tasks to run and with what settings. An explanation of the configuration file syntax is given in the *Configuration file* section of this manual.

Users will not have to touch anything but the configuration file. They can check what has happened through a variety of data products, including images, diagnostic plots and log files. A list of all CARACal data products is available at the *Data products* section of this manual.

In the rest of this Introduction we give the complete list and a brief description of each worker. A more comprehensive description is available in the *Data reduction* section of this manual, which follows the flow of a typical data reduction process. The full list of parameters available for the individual workers through the configuration file can be found at the *Workers parameters* section of this manual or following the links below.

2.1.2 List and Brief description of CARACal workers

The following workers are available in CARACal. Typically, they are executed in the same order in which they are given below. Only the first three workers (general, getdata and obsconf) should always be executed. All other workers are optional.

general

This worker sets the name of various input/output directories and the prefix used for the output data products (e.g., diagnostic plots, images, etc.).

getdata

This worker sets the name of the files to be processed and whether any conversion to .MS format is necessary.

obsconf

This worker collects basic information on the content of the .MS files to be processed (e.g., target and calibrators' name, channelisation, etc.). The worker can also extract this information automatically from the .MS metadata.

transform

This worker splits the calibrators (in preparation for cross-calibration) or the targets (in preparation for imaging) to new .MS files. Time and frequency averaging is available, as well as phase rotation to a new phase centre. Crosscalibration can be applied on the fly while splitting.

prep

This worker prepares the data for calibration and imaging. For example, it can recalculate UVW coordinates, add spectral weights based on Tsys measurements, and flag a "legacy" flag version.

flag

This worker flags the data and returns statistics on the flags. As all other workers, it can be run multiple times within a single CARACal run as explained at *Configuration file* (though this feature is not necessarily useful for all workers). It can flag data based on, e.g., channel-, antenna- and time selection, or using automated algorithms that run on autocorrelations (to catch antennas with clear problems) or crosscorrelations. It can also unflag all data.

crosscal

This worker cross-calibrates the data. Users can design their own calibration strategy including delay, bandpass, gains and flux scale calibration, self-calibration of the secondary, and flagging. The calibration is applied to the calibrators' visibilities for later inspection. Numerous settings are available for users to decide how to calibrate. Gain plots are produced.

inspect

This worker plot the visibilities for diagnostic purpose. Several different kinds of plots can be made.

mask

This worker creates an a-priori clean mask based on NVSS or SUMSS, to be used during the continuum imaging/self-calibration loop. It can also merge the resulting mask with a mask based on an existing image.

selfcal

This worker performs continuum imaging and standard (i.e., direction-independent) self-calibration. Automated convergence of the calibration procedure is optionally available. This worker can also interpolate and transfer sky model and calibration tables to another .MS (e.g., from a coarse- to a fine-channel .MS file).

line

This worker creates spectral-line cubes and images. It can subtract the continuum via both model and UVLIN-like subtraction, Doppler correct, flag solar RFI, perform automated iterative cleaning with 3D clean masks, and, finally, run a spectral-line source finder.

mosaic

This worker mosaics continuum images or line cubes using a Gaussian primary beam with FWHM = 1.02 lambda / antenna_diameter out to a cutoff level.

2.2 Configuration file

Users settings are passed to CARACal through a configuration file consisting of a sequence of blocks — each corresponding to the run of a CARACal worker. The workers are run following the order in which they appear in the configuration file. For reference see *List and Brief description of CARACal workers*.

The following workers must always run be and, therefore, must always appear in the configuration file: *general*, *getdata* and *obsconf*. All other workers are optional.

Within each worker's block of the configuration file, the worker's parameters are arranged in a nested structure following the YAML syntax rules (see https://yaml.readthedocs.io). As an example, a block of the config file may look like:

```
worker_name:
  enable: true
  parameter_1: value_1
  parameter_2:
    parameter_2_1: value_2_1
    parameter_2_2: value_2_2
  parameter_3: value_3
  ...
```

The complete list of all workers' parameters is available at *Workers parameters*, where the parameters' nesting is also illustrated.

Workers can be executed more than once in a single run of CARACal. This could be useful, for example, if a user wants to flag the data both before and after cross-calibration. To indicate a new run of a worker the worker name must be followed by "__<suffix>" in the configuration file (note the double underscore). The first run of a worker can also have a "__<suffix>" but does not have to. In the example above, the flag worker must thus appear twice in the configuration file:

```
flag__beforecrosscal:
  enable: true
  parameter_1: value_1A
  ...

[other workers]

flag__aftercrosscal:
  enable: true
  parameter_1: value_1B
  ...
```

Most parameters are optional and do not need to be included in the configuration file. Their default values are set to work in as many cases as possible. A few parameters are compulsory. The pages at *Workers parameters* indicate whether a parameter is optional, its data type, allowed values (if applicable) and default value.

CARACal comes with a set of sample configuration files. These are available at https://github.com/caracal-pipeline/caracal/tree/master/caracal/sample_configurations and include, for example:

- minimalConfig.yml, which includes as few parameters as possible and performs a basic data reduction including both continuum and spectral line imaging;
- meerkat-continuum-defaults.yml, which is optimised for the reduction of data taken with the MeerKAT telescope for the purpose of total-intensity continuum imaging.

Users could take these sample configuration files as a starting point for their work.

2.3 Data products

The following data products are written by CARACal.

2.3.1 Diagnostic Plots

Diagnostic plots are located in the output/diagnostic_plots directory, where the cross and self calibration plots are separated into their respective subdirectory.

2.3.2 Calibration Tables

Calibration tables produced by the pipeline in the cross calibration process can be found in the output/caltables directory, while the self calibration products can be found in the output/continuum/selfcal_products directory.

2.3.3 Continuum Images

Continuum images are located in the output/continuum directory, labeled as image_N where N is the number in the self calibration process.

2.3.4 Spectral-line Cubes

Spectral-line cubes are located in the output/cubes directory, labeled as cube_N where N is the Nth cube produced from the clean + mask process.

2.3.5 Spectral-line Moment Images

Spectral-line moment images are located (when source finding is enabled) in the respective cube_N directory. For example, the moment maps of cube_3 are located in output/cubes/cube_3.

2.3.6 Mask Files

Mask files produced by the pipeline are located in the output/masking directory. Self-provided masks to be used by the pipeline also need to be located in this directory.

2.3.7 Log Files

A copy of the pipeline log is located in the output directory. All logs are located in the output/logs directory, with the timestamp YYYYMM-HHMM. A copy of the config file is located in the output/cfgFiles directory, with the same timestamp.

2.3.8 Reports

HTML Reports on the observation(s) and sources are located in the output/reports directory.

2.4 Data reduction

2.4.1 CARACal workflow

A CARACal run involves running a sequence of CARACal workers following the order in which they are listed in the user configuration file. For reference see *List and Brief description of CARACal workers*.

Users can design their own CARACal workflow with some level of flexibility. The exact workflow (i.e., the exact sequence of CARACal workers) will depend on science goals but also on I/O constraints. Below we provide two examples.

Recommended workflow

The currently recommended CARACal workflow minimises the data volume increase during the course of a full CARACal run, and allows to treat the input .MS file(s) as read-only.

This workflow is implemented in all sample configuration files located at https://github.com/caracal-pipeline/caracal/tree/master/caracal/sample_configurations .

general
Compulsory worker to set up data/input/output directories.
getdata
Compulsory worker to specify the input .MS files.
obsconf
Compulsory worker to set up target/calibrators names.

transform

Split calibrators-only .MS files, one per input .MS file.

prep
Prepare the calibrators-only .MS files for processing.

flag
Flag the calibrators-only .MS files.

crosscal
Derive the cross-calibration tables and apply them to the calibrators.

inspect
Inspect the calibrated calibrator's visibilities to check the quality of the cross-calibration.

transform
Split target-only .MS files, one per input .MS file and target, applying the cross-calibration on the fly.

prep
Prepare the target-only .MS files for processing.

flag
Flag the target-only .MS files.

transform
Average the target-only .MS files in frequency for continuum imaging.

flag
Flag line channels in the averaged target-only .MS files.

selfcal
Make a continuum image of each target, self-calibrate, and transfer both gains and continuum model to the full-frequency-resolution target-only .MS files.

line
Subtract the continuum, Doppler correct and make the line cube and moment images from the target-only .MS files.

mosaic
Mosaic the continuum images of the targets.

mosaic
Mosaic the line cubes.

Note that this workflow includes multiple runs of several workers. These require a "__<suffix>" in the configuration file, as described at the page *Configuration file*.

The data volume budget of this workflow is:

- input .MS:
  - DATA
- target-only .MS:

  - DATA
  - CORRECTED_DATA
  - MODEL
- calibrators-only .MS (negligible data volume):
  - DATA
  - CORRECTED_DATA
  - MODEL
- target-only frequency-averaged .MS (negligible data volume):
  - DATA
  - CORRECTED_DATA
  - MODEL

Simple workflow

Simpler workflows than the recommended one are possible. For example, the workflow below employs less worker runs. However, it modifies the input .MS file(s), results into a larger data volume increase, and runs the self-calibration loop on larger .MS files.

general
Compulsory worker to set up data/input/output directories.
getdata
Compulsory worker to specify the input .MS files.
obsconf
Compulsory worker to set up target/calibrators names.

prep
Prepare the input .MS files for processing.
flag
Flag the calibrators in the input .MS files.
crosscal
Derive the cross-calibration tables and apply them to target and calibrators.
inspect
Inspect the calibrated calibrator's visibilities to check the quality of the cross-calibration.

transform
Split cross-calibrated target-only .MS files, one per input .MS file and target.
flag
Flag the target-only .MS files.
selfcal
Make a continuum image of each target, self-calibrate, and transfer both gains and continuum model to the full-frequency-resolution target-only .MS files.

line
Subtract the continuum, Doppler correct and make the line cube and moment images from the target-only .MS files.

mosaic
Mosaic the continuum images of the targets.
mosaic
Mosaic the line cubes.

The data volume budget of this workflow is:

- input .MS
  - DATA
  - CORRECTED_DATA
  - MODEL
- target-only .MS
  - DATA
  - CORRECTED_DATA
  - MODEL

Typically, this is 50% larger than for the *Recommended workflow* above.

2.4.2 Set up

[relevant workers: *general*, *getdata*, *obsconf* **]**

Directories and input file names

A run of CARACal must always start by setting up a number of directory and file names. This is done through the *general* and *getdata* workers.

In the *general* worker users give:

- if necessary, the *rawdatadir* directory where to find the files that should be converted to .MS format (*general: rawdatadir*);
- the *msdir* directory where to find/write .MS files (*general: msdir*);
- the *output* directory where to write all output data products (*general: output*);
- the *input* directory where to find various input files, such as AOflagger strategies etc. (*general: input*);
- the prefix for the output data products (*general: prefix*).

If they do not exist yet, the above directories can be created by setting *general: prep_workspace* to *true*. This also copies files from the caracal/data/meerkat_files directory to the *input* directory set above.

In the *getdata* worker users give the name of the .MS files to be processed (*getdata: dataid*). Furthermore, the following optional steps are available:

- convert from HDF5/MVF format to .MS (*getdata: mvftoms*); this step includes the following additional conversion options:

  - create a .MS.TAR file,
  - convert only visibilities in a selected channel range,
  - discard cross-polarisation products;
- untar an existing .MS.TAR file (*getdata: untar*);
- EXPERIMENTAL – virtually concatenate all .MS input files (*getdata: combine*); optionally users can delete an existing concatenate file and tar/untar the concatenated file.

Metadata

Finally, before starting the actual data processing users need to provide some info about the content of the input .MS files through the *obsconf* worker. Target and calibrators name can be set by editing the relevant parameters of this worker:

- *obsconf: target*
- *obsconf: bpcal*
- *obsconf: fcal*
- *obsconf: gcal*

In fact, CARACal can automatically extract most of these info from the .MS files themselves. To do so users should enable the *obsconf: obsinfo* parameter. This writes a .JSON and a .TXT file to disc. Once the .JSON file is on disc, the above parameters can be read automatically from that file. Users can, however, set any of the above manually.

Special attention should be paid to the choice of a reference antenna. Users should carefully select a good reference antenna for calibration with the *obsconf: refant* parameter.

Additional info

The *obsconf* worker can also produce a plot showing the elevation track of all targets and calibrators, and return the time range during which the data were taken with the Sun below the horizon.

2.4.3 Flagging and flag versions

[relevant workers: *flag*, *crosscal*, *selfcal*, *line***]**

Several CARACal workers can flag visibilities. The most obvious one is the *flag* worker. However, the *crosscal* and *selfcal* workers can also flag based on both gains and visibilities; and the *line* worker can flag solar RFI and potential continuum subtraction errors.

If necessary, users can navigate through the different flagging steps thanks to the flag versions saved to disk by CARACal. Most workers can indeed rewind the flags to a specified version, which allows users to correct errors or repeat certain processing steps without too much effort.

Here we explain the CARACal's management of flag versions, and describe the *flag* worker. The flagging done by the other workers mentioned above is described elsewhere in the *Data reduction* section of this CARACal *Manual*.

Management of flag versions

Every run of a CARACal worker that can result in new flags saves two flag version to disc: one before and one after the worker run. These flag versions are called:

- *<prefix>_<worker_name>_before*

- *<prefix>_<worker_name>_after*

where <prefix> is set by *general: prefix*, and the worker name is taken from the *Configuration file*.

Furthermore, the *prep* worker (which does not flag) saves a flag version called *caracal_legacy* for the input .MS (unless that flag version exists already); and the *transform* worker (which does not flag either) saves a flag version called *caracal_legacy* for the .MS file that it creates. A typical *CARACal workflow* starts with one of these two workers. Thus, .MS files processed by CARACal should always have *caracal_legacy* as first item of the time-ordered list of flag versions.

Flag versions are stored following the order in which they were created. This makes it possible to rewind flags to a specified state. All CARACal workers where this operation is useful have indeed a *rewind_flags* section. When rewinding flags to a certain version, all versions saved after that are deleted. The exact usage of flags rewinding is explained in the *Workers parameters* pages.

The flag worker

The *flag* worker can run on the input .MS files given in *getdata: dataid* or on .MS files created by CARACal at various stages of the pipeline (e.g., by the *transform* worker). The name of the .MS files to be flagged (if other than the input files) is based on the name of the input .MS files and a label set by the *flag: label_in* parameter in this worker. As an example, if the .MS files were created by the *transform* worker then *flag: label_in* should be the same as *transform: label_out*.

The *flag* worker cannot flag both calibrators and target in one go. To flag both, two separate *flag* blocks are required in the *Configuration file*, as show in *CARACal workflow*. In each block the user can set what to flag through the *flag: field* parameter.

The *flag* worker allows users to flag the data in a variety of ways. Unless otherwise stated below, flagging is done with the CASA task FLAGDATA. Follow the links below for a detailed documentation of the individual flagging modes.

- Unflag all data.
- Flag on autocorrelations to catch antennas with obvious problems using the custom program POLITSIYAKAT (*flag: flag_autopowerspec*). Individual scans are compared to the median of all scans per field and channel; and individual antennas are compared to the median of all antennas per scan, field and channel. Both methods have their own flagging threshold, which users can tune. Users can also set which column and which fields to flag.
- Flag all autocorrelations (*flag: flag_autocorr*).
- Flag specific portions of the beginning and/or end of each scan (*flag: flag_quack*). As in the CASA task FLAGDATA, users can set the time interval that should be flagged and the quackmode.
- Flag shadowed antennas (*flag: flag_shadow*). Users can tune the amount of shadowing allowed before flagging an antenna. For observations obtained with a MeerKAT subarray it is possible to include offline antennas in the shadowing calculation.
- Flag selected channel ranges (*flag: flag_spw*).
- Flag selected time ranges (*flag: flag_time*).
- Flag selected antennas (*flag: flag_antennas*). Within this task, users can limit the flagging of selected antennas to a selected timerange.
- Flag selected scans (*flag: flag_scan*).
- Flag according to a static mask of bad frequency ranges using the custom program RFIMASKER (*flag: flag_mask*). The mask file should be located in the *input* directory set by *general: input*. Users can decide to limit the flagging to a selected UV range. This could be useful to flag short baselines only.
- Flag RFI choosing between the available algorithms and strategies (*flag: flag_rfi*). Possible choices include AOFlagger, Tricolour, CASA tfcrop. The requested AOFlagger or tricolour strategy file should be located in

the *input* directory set by *general: input*. CARACal comes with a number of strategy files, which are located in https://github.com/caracal-pipeline/caracal/tree/master/caracal/data/meerkat_files and are copied to the *input* directory by the *general* worker. However, users can copy their own strategy file to the same *input* directory and use it within CARACal.

Finally, a summary of the flags can be obtained with *flag: summary*. The summary is available at the relevant log file (see *Data products*).

2.4.4 Cross-calibration

[relevant workers: *crosscal*, *inspect*]

Cross-calibration runs largely on CASA tasks. Using these tasks, CARACal allows users to solve for delays, bandpass, gains and flux scale in several different ways. The *crosscal* worker operates within the framework that .MS files include a primary (bandpass and flux) calibrator and, optionally, a secondary (gains) calibrator.

Just as a classic, simple example, it is possible to solve for:

1. time-independent antenna delays and normalised bandpass based on the observation of the primary calibrator;
2. time-dependent antenna flux scale based on the observation of the primary calibrator;
3. time-dependent antenna gains based on the observation of the secondary calibrator;
4. time-dependent antenna flux scale at fine time resolution obtained by scaling the gains from step 3 above to the gains from step 2 above.

However, CARACal allows users to take less traditional cross-calibtation steps, too, such as self-calibration on the secondary calibrator, or delay calibration on the secondary, and to flag the calibrated visibilities in between calibration steps.

Flexible cross-calibration strategies

CARACal allows for powerful and sophisticated cross-calibration strategies thanks to the flexibility provided by the parameters *crosscal: primary: order* and *crosscal: secondary: order*. These allow users to build their favourite sequence of calibration/imaging/flagging steps choosing among:

- K = delay calibration with CASA GAINCAL
- B = bandpass calibration with CASA BANDPASS
- G = gain amplitude and/or phase calibration with CASA GAINCAL
- F = gain amplitude and/or phase calibration with CASA GAINCAL, followed by bootstrapping of the flux scale from the primary calibrator with CASA FLUXSCALE (secondary calibrator only)
- I = imaging with WSCLEAN (secondary calibrator only)
- A = flagging with CASA FLAGDATA using the tfcrop algorithm

Each of these steps may have its own settings with respect to gain type (e.g., each G could be amplitude-only, phase-only, or both amplitude and phase), solution interval, normalisation, data combination at boundaries (e.g., scan, SPW), imaging and flagging settings.

For example, *crosscal: primary: order*: KGBAKGB results in:

- delay calibration (K);
- gain calibration (G) applying the intial K on the fly;
- bandpass calibration (B) applying the initial K and G on the fly;

- flagging of the visibilities with the initial K, G and B applied;
- final K calibration applying the initial G and B on the fly;
- final G calibration applying the final K and initial B on the fly;
- final B calibration applying the final K and G on the fly.

In this example, it would be possible to set different solution intervals for the initial and final G through the *crosscal: primary: solint* parameter, which is a sequence containing one entry per element in *crosscal: primary: order*. In case the solution interval is not relevant (A and I steps) users can give an empty string ''. The same applies to the calibration parameters *crosscal: primary: calmode* and *crosscal: primary: combine*.

An example for the secondary is *crosscal: secondary: order*: FIG, which results in:

- gain calibration and bootstrapping of the flux scale;
- imaging;
- gain calibration.

Note that in this example no bootstrapping of the flux scale is necessary after the second gain calibration G because the gains are now self-calibrated on a I sky model which, following the initial F, is already on the correct flux scale.

We refer to the *crosscal* page for a complete description of all cross-calibration parameters.

Apply the cross-calibration and diagnostic plots

CARACal can apply the cross calibration tables to all calibrators (useful for diagnostics). It can also apply it to the target, although this can also be done by the *transform* worker on the fly while splitting the target from the input .MS file. When applyin the calibration, the *crosscal* worker adopts the following interpolation rules:

- Delay calibration: applied to primary, secondary, target with nearest, linear, linear interpolation, respectively.
- Bandpass calibration: applied to primary, secondary, target with nearest, linear, linear interpolation, respectively.
- Gain calibration before bootstrapping the flux scale: applied to primary, secondary, target with linear, linear, linear interpolation, respectively.
- Gain calibration after bootstrapping the flux scale: applied to primary, secondary, target with linear, nearest, linear interpolation, repsectively.

The *crosscal* worker produces .HTML plots of the various calibration terms for later, interactive inspection. Furthermore, the *inspect* worker produces .PNG plots of the caibrators' calibrated visibilities to check the quality of the calibration. A variety of standard plots are produced, such as phase-vs-uvdistance and real-vs-imaginary. Furthermore, users can define their own plots as described in the *inspect* page.

We strongly recommend that users inspect the .HTML and .PNG plots produced by the crosscal and inspect workers to ensure that the quality of the cross-calibration is adequate to their science goals.

2.4.5 Continuum imaging and self-calibration

[relevant workers: *transform*, *flag*, *selfcal*, *mask***]**

Split, average and flag target visibilities

Following cross-calibration, CARACal creates a new .MS file which contains the cross-calibrated target visibilities only. This is done by the *transform* worker. In case the cross-calibration tables have not been applied to the target by the *crosscal* worker, *transform* can do so on the fly while splitting using the CASA task MSTRANSFORM.

Optionally, the *transform* worker can average in time and/or frequency while splitting. Depending on the science goals, it might be useful to run this worker more than once. E.g., the first time to create a frequency-averaged dataset for continuum imaging and self-calibration, and the second time to create a narrow-band dataset for spectral-line work. The possibility of running this worker multiple times within a single CARACal run allows users to design the best *CARACal workflow* for their project.

Before self-calibrating it might also be good to flag the target's visibilities. (Typically the target is not flagged before applying the cross-calibration.) This can be done with the *flag* worker (which was probably already run on the calibrators' visibilities before cross-calibration) setting *flag: field* to target.

Image the continuum and self-calibrate

Having cross-calibrated, split, optionally averaged and flagged the target, it is now possible to iteratively image the radio continuum emission and self-calibrate the visibilities. The resulting gain tables and continuum model can also be transferred to another .MS file (particularly useful for spectral line work). All this can be done with the *selfcal* worker.

Several parameters allow users to set up both the imaging and self-calibration according to their needs. Imaging is done with WSclean, and the parameters of this imaging software are available in the *selfcal* worker. Calibration is done with either Cubical or MeqTrees, and also in this case the *selfcal* worker includes the parameters available in those packages.

Additional parameters allow users to decide how many calibration iterations to perform through the parameter *selfcal: cal_niter*. For a value N, the code will create N+1 images following the sequence image1, selfcal1, image2, selfcal2, ... imageN, selfcalN, imageN+1.

Optionally, users can enable *selfcal: aimfast*, which at each new iteration compares the new continuum image with the previous one and decides whether the image has improved significantly. In case it has not, no further iterations are performed. In this case therefore *selfcal: cal_niter* is the maximum number of iterations.

While imaging it is usually convenient to identify where to clean. Within CARACal this can be done in several different ways through the parameter *selfcal: image: clean_mask_method*:

- with WSclean automated masking method, which cleans blindly down to a masking threshold, defines the clean mask as the ensamble of all cleaned pixels, and then re-cleans them down to a deeper clean cutoff;
- with SoFiA, which makes a clean mask for the Nth imaging run from the (N-1)th image;
- with a clean mask made by the *mask* worker or supplied by the user.

Several parameters allow users to control the calibration step in the *selfcal* worker. Users can set the time and frequency solution intervals. Gain phase and amplitude can both be solved for, each with its own time and frequency solution interval (more standard phase-only self-calibration is also possible). We refer to the *selfcal* page for a full description of all available modes and parameters.

Gain and model transfer

If the self-cal loop was executed on a frequency-averaged .MS file, it might be necessary to transfer the resulting gains and continuum model back to the original, full-frequency-resolution .MS file. This is done with *selfcal: transfer_apply_gains* (using Cubical) and *selfcal: transfer_model* (using Crystalball), respectively. The latter allows users to limit the model transfer to the N brightest sources, to sources in a region, or to point sources only. Be aware that the model transfer step can be very time consuming for large .MS files.

2.4.6 Spectral line imaging

[relevant workers: *line*]

Spectral line imaging runs on a combination of custom software, CASA, WSclean, SunBlocker and SoFiA in order to subtract the continuum, Doppler correct, flag solar RFI, create cleaned spectral line cubes and moment images. It can run on the input .MS files or on .MS files created by CARACal at various stages of the pipeline (e.g., by the *transform* worker). In the latter case the name of the .MS files to be imaged is based on the name of the input .MS files and on *line: label_in*.

The input .MS files may contain several targets. In this case, CARACal makes one HI cube per target, using all available visibilities for that target from all input .MS files. This worker does not mosaic line cubes made for different targets. That is done by the *mosaic* worker.

Continuum subtraction

Continuum subtraction is usually necessary before imaging the spectral line of interest. CARACal can do this using two standard methods, which can be run sequentially within a single CARACal run: *i)* subtraction of the continuum model visibilities from the field visibilities (*line: subtractmodelcol*); and *ii)* fitting and subtracting polynomials from the individual real and imaginary visibility spectra (parameter *line: mstransform: uvlin*). A third standard method currently NOT implemented in CARACal consists of fitting and subtracting polynomials from individual image spectra in the data cube. This may be implemented in the future.

In practice, the first method consists of subtracting the MODEL_DATA column from the CORRECTED_DATA column of the .MS files. CARACal writes the resulting visibilities in the CORRECTED_DATA column itself.

Users should therefore be aware that the CORRECTED_DATA column gets overwritten.

Yes, that is dangerous, but it can be undone with *line: addmodelcol*.

The MODEL_DATA column contains the continuum model to be subtracted. Within CARACal, the MODEL_DATA column should have been filled in with the continuum model resulting from the continuum imaging and self-calibration done by the *selfcal* worker.

When running the second continuum subtraction method, which uses the CASA task MSTRANSFORM, users can select the order of the fit, the channels that should be included in the fit, and the column that should be considered. This method writes a new file with an "mst" suffix appended to the file name. Subsequent steps of this worker can be instructed to run on the file written by MSTANSFORM.

We have found that CASA MSTRANSFORM produces bad, partly-unflagged visibility spectra when the only unflagged channels of those spectra are not included in the *uvlin* fit. These bad spectra look like RFI in the cube. In order to flag them, users can run the *line: flag_mst_errors* step.

Doppler correction

Doppler correction is performed with the CASA task MSTRANSFORM (parameter *line: mstransform: doppler*) in the same run of this task used to perform continuum subtraction (see above). Users can select the telescope (choosing from a list of available names, see parameter *line: mstransform: telescope*), as well the regridding mode (channel, frequency or velocity), velocity type and output frame. Users can also set the frequency (or velocity, ...) grid they want to Doppler correct to, but they can also let CARACal find the optimal one given the input files. In the latter case, visibilities will be regridded to the widest Doppler-corrected spectral interval common to all input .MS files, at the worse Doppler-corrected spectral resolution of them all.

Solar RFI flagging

CARACal flags solar RFI using SunBlocker (*line: sunblocker*). The main idea of SunBlocker is that, because solar RFI is broadband, averaging visibilities in frequency should enhance its detectability. However, the phase of solar RFI changes rapidly with frequency, leading to vectorial averages with very low amplitude. In order to enhance the detectability of the solar RFI SunBlocker performs a scalar average. It does so in uv cells of the visibility plane (i.e.,

on gridded visibilities). Once that is done, UV cells with anomalously high (scalar) average ampliude are flagged. This method has been shown to work well on continuum-subtracted data. Users have control over some of the SunBlocker settings, such as flagging threshold and gridding. It is also possible to run this task on day-time data only based on the output of *obsconf: obsinfo: vampirisms*.

Imaging

Spectral line imaging is done with WSclean or CASA. The former has been used a lot more and is therefore more tested within CARACal.

WSclean produces a set of individual .FITS images per channel (i.e., dirty image, psf, clean model, restored image), and when that is done CARACal stacks them all together in .FITS image cubes. Cleaning is done iteratively by making a first cube using WSClean algorithms for a blind clean, then making a clean mask with SoFiA and running WSClean again with that clean mask, and so on. Users can let CARACal continue iterating until the noise in the residual cube converges (up to a maximum number of iterations) or perform a fixed number of iterations ignoring noise convergence. Users also have full control of all WSCLEAN imaging parameters.

Several additional steps are available and can be run once the .FITS image cubes are ready. This includes removing the trivial Stokes axis from the cubes, converting the frequency axis from frequency to velocity, and creating a primary beam cube on the same WCS grid of the image cube. At the moment the primary beam cube is calculated assuming a Gaussian primary beam with FWHM = 1.02 * lightspeed / frequency / dishdiameter. Primary beam cubes can be used subsequently in the *mosaic* worker.

As a final step, CARACal can run SoFiA in order to make a line detection mask and the corresponding moment images. Users have control over several (but not all) SoFiA settings.

Diagnostics

As a useful diagnostic, CARACal can run SHARPENER, which extracts 1D spectra at the position of bright continuum sources in the field. This is helpful to assess the quality of the continuum subtraction. It can also create one last flagging summary.

2.4.7 Mosaicking

[relevant workers: *mosaic*]

A single CARACal run can image multiple target fields distributed in an arbitrary manner over multiple input .MS files. If adjacent, these fields could be mosaicked together both in continuum and spectral line using the *mosaic* worker. This worker can take existing primary beam images or make its own. At the moment these are simple Gaussian primary beams. More realistic primary beam are likely to be implemented in the future.

This section of the CARACal *Manual* is under development and currently lacks detailed information on mosaicking. For complete information on the modes and parameters please see the *mosaic* worker page.

2.5 Workers parameters

2.5.1 crosscal

Carry out Cross calibration of the data (delay, bandpass and gain calibration).

enable

bool

Execute the crosscal worker.

label_in

str

Label of the .MS file(s) to work on.

label_cal

str, *optional*, *default = 1gc1*

Label for output files (calibration tables, images).

rewind_flags

Rewind flags to specified version.

enable

bool, *optional*, *default = True*

Enable the rewind_flags segement.

mode

{"reset_worker", "rewind_to_version"}, *optional*, *default = reset_worker*

If mode = 'reset_worker' rewind to the flag version before this worker if it exists, or continue if it does not exist; if mode = 'rewind_to_version' rewind to the flag version given by 'version' below.

version

str, *optional*, *default = auto*

Flag version to rewind to. If 'auto' it will rewind to the version prefix_workername_before, where 'prefix' is set in the 'general' worker, and 'workername' is the name of this worker including the suffix '__X' if it is a repeated instance of this worker in the configuration file. Note that all flag versions that had been saved after this version will be deleted.

overwrite_flagvers

bool, *optional*, *default = False*

Allow Caracal to overwrite existing flag versions. Not recommended. Only enable this if you know what you are doing.

uvrange

str, *optional*, *default = >50*

Select what UV range should be used throughout this worker following the CASA notation (e.g., ">100"). The default units are metres but other units can be used (e.g., ">0.5klambda").

set_model

Fill in the MODEL column of the .MS file(s) for the field selected by "field" below in preparation for crosscalibration. This can use CASA setjy for point-source models, or MeqTrees for available local sky models.

enable

bool, *optional*, *default = True*

Execute the set_model segment.

meerkat_skymodel

bool, *optional*, *default = False*

Use the MeerkAT local sky model (lsm) of the calibrator field instead of a point source model. At the moment a MeerKAT lsm is only available for the calibrator PKS 1934-638. For the calibrator 0408-6545 a model is available but is not well tested yet and we do not recommend using it.

no_verify

bool, *optional*, *default = False*

Enables setting standard manually [???].

field

str, *optional*, *default = fcal*

Set the field to execute the set_model segment on. Specify either the field number, field name or field specification as per obsconf worker (e.g., "fcal").

threads

int, *optional*, *default = 8*

Number of threads used by MeqTrees if meerkat_skymodel above is enabled.

primary

Calibrating on the bandpass/flux calibrator field.

reuse_existing_gains

bool, *optional*, *default = False*

Reuse gain tables if they exist. Note that this does not check whether the existing tables were obtained with the same Caracal settings. Use with caution.

order

str, *optional*, *default = KGB*

Order in which to solve for gains for this field. E.g, if order is set to 'KGB', then we solve for delays, then gains and finally bandpass. The full options are 1) K - delay calibration, 2) G - gain calibration (decide whether to solve for amplitude, phase or both with 'calmode' below), 3) B - bandpass calibration, 4) A - automatic flagging with CASA tfcrop (existing gains will be applied first).

b_solnorm

bool, *optional*, *default = False*

Normalise average solution amplitude to 1.0

calmode

list of str, *optional*, *default = a, ap, ap*

For each step in 'order' above, set whether to solve for phase ('p'), amplitude ('a') or both ('ap'). This is actually only relevant when solving for the gains, i.e., for the G steps in 'order' above. However, users should include an entry (even just an empty string) for all steps in 'order'.

solint

list of str, *optional*, *default = 120s, 120s, inf*

For each step in 'order' above, set the solution interval. Set to 'inf' to obtain a single solution (see also 'combine' below). Include time units, e.g., '120s' or '2min'.

combine

list of str, *optional*, *default = '', '', scan*

For each step in 'order' above, set along what axis the data should be combined before solving. Options are '' (i.e., no data combination; solutions break at obs, scan, field, and spw boundarie), 'obs', 'scan', 'spw', 'field'. To combine along multiple axes use comma-separated axis names in a single string, e.g., 'obs,scan'. This setting is only relevant for the steps of type K, G and B included in 'order' above. For A steps this setting is ignored and an empty string may be used.

b_fillgaps

int, *optional*, *default = 70*

Fill flagged solution channels by interpolation.

plotgains

bool, *optional*, *default = True*

Plot gains with ragavi-gains. The .html plots are located in <output>/diagnostic_plots/crosscal/.

flag

Flagging settings used for all "A" (= auto-flagging) steps included in "order" above. These steps include applying the existing gains and flagging the corrected data.

col

{"corrected", "residual"}, *optional*, *default = corrected*

Data column to flag on

usewindowstats

{“none”, “sum”, “std”, “both”}, *optional*, *default = std*

Calculate additional flags using sliding window statistics

combinescans

bool, *optional*, *default = False*

Accumulate data across scans depending on the value of ntime

flagdimension

{“freq”, “time”, “freqtime”, “timefreq”}, *optional*, *default = freqtime*

Dimensions along which to calculate fits (freq/time/freqtime/timefreq)

timecutoff

float, *optional*, *default = 4.0*

Flagging thresholds in units of deviation from the fit

freqcutoff

float, *optional*, *default = 3.0*

Flagging thresholds in units of deviation from the fit

correlation

str, *optional*, *default = ‘ ‘*

Correlation

secondary

Calibrating on the gain calibrator field.

reuse_existing_gains

bool, *optional*, *default = False*

Reuse gain tables if they exist. Note that this does not check whether the existing tables were obtained with the same Caracal settings. Use with caution.

apply

str, *optional*, *default = B*

Calibration terms solved for in the primary segment that should be applied to the secondary calibrator before solving for the terms in ‘order’ below.

order

str, *optional*, *default = KGAF*

Order of the calibration/flagging/imaging steps for the secondary calibrator. E.g, if order is set to ‘KGAF’, we solve for delays, then for gains, after that the existing gains (KG) are applied before flagging the calibrated data, and finally, we solve for gains and transfer the flux scale from the primary step. The full options are 1) K - delay calibration; 2) G - gain calibration (set whether to solve for amplitude, phase or both with ‘calmode’ below); 3) F - same as G, but imedietly followed by a fluxscale. Note that a G table must exist from the primary step for this work; 4) B - bandpass calibration; 5) A - automatic flagging with CASA tfcrop (existing gains will be applied first); 6) I - imaging with WSClean using the settings in ‘image’ below, which fills the MODEL column of the .MS file(s) with a sky model and, therefore, enables self-calibration with a subsequent G step.

calmode

list of str, *optional*, *default = a, ap, None, ap*

For each step in 'order' above, set whether to solve for phase ('p'), amplitude ('a') or both ('ap'). This is actually only relevant when solving for the gains, i.e., for the G steps in 'order' above. However, users should include an entry (even just an empty string) for all steps in 'order'.

solint

list of str, *optional*, *default = 120s, inf, None 120s*

For each step in 'order' above, set the solution interval. Set to 'inf' to obtain a single solution (see also 'combine' below). Include time units, e.g., '120s' or '2min'.

combine

list of str, *optional*, *default = '', '', None, ''*

For each step in 'order' above, set along what axis the data should be combined before solving. Options are '' (i.e., no data combination; solutions break at obs, scan, field, and spw boundarie), 'obs', 'scan', 'spw', 'field'. To combine along multiple axes use comma-separated axis names in a single string, e.g., 'obs,scan'. This setting is only relevant for the steps of type K, G and B included in 'order' above. For A steps this setting is ignored and an empty string may be used.

B_solnorm

bool, *optional*, *default = False*

Normalise average solution amplitude to 1.0

plotgains

bool, *optional*, *default = True*

Plot gains with ragavi-gains. The .html plots are located in <output>/diagnostic_plots/crosscal/.

flag

Flagging settings used for all "A" (= auto-flagging) steps included in "order" above. These steps include applying the existing gains and flagging the corrected data.

col

{"corrected", "residual"}, *optional*, *default = corrected*

Data column to flag on

usewindowstats

{"none", "sum", "std", "both"}, *optional*, *default = std*

Calculate additional flags using sliding window statistics

combinescans

bool, *optional*, *default = False*

Accumulate data across scans depending on the value of ntime

flagdimension

{"freq", "time", "freqtime", "timefreq"}, *optional*, *default = freqtime*

Dimensions along which to calculate fits (freq/time/freqtime/timefreq)

timecutoff

float, *optional*, *default = 4.0*

Flagging thresholds in units of deviation from the fit

freqcutoff

float, *optional*, *default = 3.0*

Flagging thresholds in units of deviation from the fit

correlation

str, *optional*, *default = ‘ ‘*

Correlation

image

Image settings for imaging secondary calibrator

npix

int, *optional*, *default = 4096*

Number of pixels in output image

padding

float, *optional*, *default = 1.5*

Padding in WSclean

mgain

float, *optional*, *default = 0.85*

Image CLEANing gain

cell

float, *optional*, *default = 0.5*

Image pixel size (arcsec)

weight

{“briggs”, “uniform”, “natural”}, *optional*, *default = briggs -1.0*

Image weighting type. If Briggs, set the img robust parameter

auto_mask

float, *optional*, *default = 10*

Auto masking threshold

auto_threshold

float, *optional*, *default = 1.5*

Auto clean threshold

col

str, *optional*, *default = CORRECTED_DATA*

Column to image

local_rms

bool, *optional*, *default = True*

switch on local rms measurement for cleaning

rms_window

int, *optional*, *default = 150*

switch on local rms measurement for cleaning

niter

int, *optional*, *default = 120000*

Number of cleaning iterations

nchans

int, *optional*, *default = 7*

Number of channesls in output image

fit_spectral_pol

int, *optional*, *default = 2*

Number of spectral polynomial terms to fit to each clean component. This is equal to the order of the polynomial plus 1.

apply_cal

Apply calibration

applyto

list of str, *optional*, *default = bpcal, gcal, target*

Fields to apply calibration to

calmode

{"=", "calflag", "calflagstrict", "trial", "flagonly", "flagonlystrict"}, *optional*, *default = calflag*

Calibration mode, the default being "calflag" - calibrates and applies flags from solutions. See CASA documentation for info on other modes.

summary

Prints out the buther's bill, i.e. data flagging summary at the end of cross calibration process.

enable

bool, *optional*, *default = True*

Execute printing flagging summary.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.2 ddcal

Perform direction-dependent calibration on the data (SHARED-RISK DEVELOPMENT MODE).

enable

bool

Execute the ddcal worker (i.e. carry out DD-calibration).

label_in

str, *optional*, *default = corr*

Label of the .MS files to process. By default uses the 'corr' label_in for self-calibrated dataset.

use_pb

bool, *optional*, *default = False*

Enable primary beam usage in making the DD-corrected DDFacet image. Note that this is EXPERIMENTAL and currently only available for MeerKAT data.

image_dd

Imaging parameters for DD calibration with DDFacet.

enable

bool, *optional*, *default = True*

Enable the 'image_dd' segment.

npix

int, *optional*, *default = 8000*

Number of pixels in the image. Note that DDFacet has its own super-special scheme to decide the actual number of pixels, so this is only an approximation.

use_mask

bool, *optional*, *default = True*

Enable clean mask for DDFacet initial imaging. Note that this doubles the imaging time since it runs DDFacet twice – once to get a preliminary image to make a mask with (mask is made by the cleanmask tool), and once to get the final image with masking. Previous WSClean masks cannot be used because pixel numbers might be different.

mask_sigma

float, *optional*, *default = 10.0*

The number of standard deviations (i.e. sigma_rms) to use when clipping the initial image for masking.

mask_boxes

int, *optional*, *default = 9*

Divide the initial image (for making the mask) into this number of boxes, then perform sigma clipping in each of these boxes.

mask_niter

int, *optional*, *default = 20*

The number of sigma-clipping iterations to perform on the image, for masking, or set to 0 to clip until convergence is achieved.

mask_overlap

float, *optional*, *default = 0.3*

Overlap region for the boxes, given as a fraction of the number of boxes.

mask_tol

float, *optional*, *default = 0.75*

Tolerance for dilating the mask. Dilation will stop when the percentage difference between dilations is smaller than this value.

cell

float, *optional*, *default = 1.3*

Pixel size in arcsec.

facets_nfacets

int, *optional*, *default = 24*

Number of facets to use, and is the same as the Facets-NFacets parameter of DDFacet.

weight_col

{"WEIGHT_SPECTRUM", "WEIGHT", "IMAGING_WEIGHT"}, *optional*, *default = WEIGHT*

Read data weights from the specified column. Options are WEIGHT_SPECTRUM, WEIGHT, and (for rarer occasions) IMAGING_WEIGHT.

weight_mode

{"Natural", "Uniform", "Robust", "Briggs"}, *optional*, *default = Briggs*

UV weighting mode. Options are 'Natural', 'Uniform', 'Robust', and 'Briggs'.

weight_robust

float, *optional*, *default = -0.4*

Briggs robustness parameter, from -2 (more uniform) to 2 (more natural).

deconv_maxminoriter

int, *optional*, *default = 100000*

Number of clean iterations.

freq_nband

int, *optional*, *default = 10*

Number of frequency bands for gridding.

freq_ndegridband

int, *optional*, *default = 15*

Number of frequency bands for degridding. 0 means degrid each channel.

deconv_rmsfactor

float, *optional*, *default = 0.0*

Set the minor-cycle stopping-threshold to X*{residual RMS}, where X is this parameter value.

deconv_peakfactor

float, *optional*, *default = 0.25*

Set the minor-cycle stopping-threshold to X*{peak residual}, where X is this parameter value.

deconv_mode

{"HMP", "Hogbom", "SSD", "GAClean"}, *optional*, *default = Hogbom*

The deconvolution algorithm to use. Options are 'HMP' (Hybrid Matching Pursuit, aka multi-scale/multifrequency), 'Hogbom' (Hogbom's CLEAN algorithm), 'SSD' (SubSpace Deconvolution algorithm), and 'GAClean' (Genetic Algorithm Clean). Please direct queries to DDFacet Developers for further details.

deconv_gain

float, *optional*, *default = 0.1*

Gain setting for the deconvolution loops.

deconv_fluxthr

float, *optional*, *default = 1.0e-6*

Absolute flux-density threshold at which deconvolution is stopped, in units of Jy. Relevant for HMP and Hogbom modes.

deconv_allownegative

bool, *optional*, *default = True*

Allow negative components for cleaning (valid for HMP and Hogbom modes).

hogbom_polyfitorder

int, *optional*, *default = 6*

Order of the polynomial to be used for frequency fitting.

parallel_ncpu

int, *optional*, *default = 0*

Number of processes / threads to use in parallel mode. 0 = use all of those available. 1 = disable parallelism.

predict_colname

str, *optional*, *default = MODEL_DATA*

MS column to write the predicted visibilities corresponding to the model. Setting '' will disable this parameter.

log_memory

bool, *optional*, *default = True*

Log the memory usage by DDFacet.

cache_reset

bool, *optional*, *default = True*

Reset all caches (including PSF and dirty image). Change from default at your own risk.

log_boring

bool, *optional*, *default = True*

Enable progress bars and other pretty console output. Doesn't seem to work. But who knows, try it out.

data_colname

str, *optional*, *default = CORRECTED_DATA*

Data column to use for initial imaging. Defaults to 'CORRECTED_DATA', the assumption being that self-calibration has already been done on the measurement set.

data_chunkhours

float, *optional*, *default = 0.05*

Chunk data into time bins of X hours to conserve memory, where X is this parameter.

output_mode

{"Dirty", "Clean", "Predict", "PSF"}, *optional*, *default = Clean*

Output mode of DDFacet. Options are 'Dirty', 'Clean', 'Predict', and 'PSF'. This setting defaults to 'Clean', since that is what we want to do in this worker.

calibrate_dd

Direction-dependent calibration parameters.

enable

bool, *optional*, *default = True*

Enable the 'calibrate_dd' segment.

sigma

float, *optional*, *default = 4.5*

Sigma threshold to use in detecting outlier regions in images, via CATDagger (which is enabled by setting 'de_sources_mode', below, to 'auto'). The default value of 4.5 works well, but a lower value may be needed for some images.

min_dist_from_phcentre

int, *optional*, *default = 1300*

The radius (in number of pixels), from the centre of the image, out to which sources will not be tagged for DD-calibration. (This is because sources close to the phase centre may not have been cleaned deeply enough.) The default is kept at 1300 (which roughly corresponds to 30').

dist_ncpu

int, *optional*, *default = 1*

The number of cpus for distributed computing.

de_sources_mode

str, *optional*, *default = manual*

Mode in which sources are tagged for DD calibration. Options are 'auto' (which uses CATDagger), and 'manual' (for which one needs to provide a list of sources). Use 'auto' with caution and at your own risk.

de_target_manual

list of str, *optional*, *default = ' '*

List of target fieldnames for carrying out DD calibration. The remaining fields will not undergo DD calibration.

de_sources_manual

list of str, *optional*, *default = ' '*

List of sources per target to tag for DD calibration, in the same order as the 'de_target_manual' list. Use ';' to separate different sources per target.

sol_min_bl

float, *optional*, *default = 100*

The minimum baseline length to solve for.

madmax_enable

bool, *optional*, *default = true*

Enable madmax flagging in CubiCal.

madmax_thr

list of int, *optional*, *default = 0, 10*

Threshold for MAD flagging per baseline (specified in number of standard deviations). Residuals exceeding mad-thr*MAD/1.428 will be flagged. MAD is computed per baseline. This can be specified as a list e.g. N1,N2,N3,... The first value is used to flag residuals before a solution starts (use 0 to disable), the next value is used when the residuals are first recomputed during the solution several iterations later (see -chi-int), etc. A final pass may be done at the end of the solution. The last value in the list is reused if necessary. Using a list with gradually-decreasing values may be sensible.

madmax_global_thr

list of int, *optional*, *default = 0, 12*

Threshold for global median MAD (MMAD) flagging. MMAD is computed as the median of the per-baseline MADs. Residuals exceeding S*MMAD/1.428 will be flagged.

madmax_estimate

{"corr", "all", "diag", "offdiag"}, *optional*, *default = corr*

MAD estimation mode. Use 'corr' for a separate estimate per baseline and correlation. Otherwise, a single estimate per baseline is computed using 'all' correlations, or only the 'diag' or 'offdiag' correlations.

dd_data_col

str, *optional*, *default = CORRECTED_DATA*

Column to calibrate, with the assumption that you have already run the selfcal worker.

dd_out_data_col

str, *optional*, *default = SUBDD_DATA*

Output data column. Note that the ddcal worker is currently hardcoded for this being set to 'SUBDD_DATA'.

dd_weight_col

str, *optional*, *default = WEIGHT*

Column to read weights from, and apply them by default. Specify an empty string to disable this parameter.

dd_sol_stall_quorum

float, *optional*, *default = 0.95*

Minimum percentage of solutions that must have stalled before terminating the solver.

dd_g_type

str, *optional*, *default = complex-2x2*

Gain matrix type for the G-Jones matrix. Keep this set to 'complex-2x2', because DD-calibration fails otherwise.

dd_g_clip_high

float, *optional*, *default = 1.5*

Amplitude clipping – flag solutions with any amplitudes above this value for G-Jones matrix.

dd_g_clip_low

float, *optional*, *default = 0.5*

Amplitude clipping – flag solutions with any amplitudes below this value for G-Jones matrix.

dd_g_update_type

str, *optional*, *default = phase-diag*

Determines update type. This does not change the Jones solver type, but restricts the update rule to pin the solutions within a certain subspace.

dd_g_max_prior_error

float, *optional*, *default = 0.35*

Flag solution intervals where the prior error estimate is above this value for G-Jones matrix.

dd_g_max_post_error

float, *optional*, *default = 0.35*

Flag solution intervals where the posterior variance estimate is above this value for G-Jones matrix.

dd_dd_max_prior_error

float, *optional*, *default = 0.35*

Flag solution intervals where the prior error estimate is above this value for DE term.

dd_dd_max_post_error

float, *optional*, *default = 0.35*

Flag solution intervals where the posterior variance estimate is above this value for DE term.

dd_g_timeslots_int

int, *optional*, *default = 10*

Time solution interval in timeslot units for G-Jones matrix.

dd_g_chan_int

int, *optional*, *default = 0*

Frequency solution interval in channel units for G-Jones matrix.

dd_dd_timeslots_int

int, *optional*, *default = 100*

Time solution interval in timeslot units for DE-Jones matrix.

dd_dd_chan_int

int, *optional*, *default = 100*

Frequency solution interval in channel units for DE-Jones matrix.

dist_nworker

int, *optional*, *default = 25*

Number of processes.

copy_data

Copy DD-calibrated data to CORRECTED_DATA column. THIS IS DANGEROUS - only if you want to go ahead with line imaging.

enable

bool, *optional*, *default = True*

Enable copying of DD-calibrated data to CORRECTED_DATA column.

image_wsclean

WSClean imaging paramaters for ddcal worker.

enable

bool, *optional*, *default = True*

Enable WSClean imaging of the DD-calibrated data.

img_ws_npix

int, *optional*, *default = 1800*

Number of pixels in output image.

img_ws_padding

float, *optional*, *default = 1.3*

Padding in WSClean.

img_ws_mgain

float, *optional*, *default = 0.90*

Gain for the major cycle during image CLEANing.

img_ws_cell

float, *optional*, *default = 2.*

Image pixel size (in arcsec).

img_ws_weight

{"briggs", "uniform", "natural"}, *optional*, *default = briggs*

Image weighting type. Options are 'briggs', 'uniform', and 'natural'. If 'briggs', set the img_ws_robust parameter below.

img_ws_robust

float, *optional*, *default = 0.*

Briggs robust value.

img_ws_uvtaper

str, *optional*, *default = 0*

Taper for imaging (in arcsec).

img_ws_niter

int, *optional*, *default = 1000000*

Number of cleaning iterations.

img_ws_nmiter

int, *optional*, *default = 0*

Number of major cycles.

img_ws_cleanborder

float, *optional*, *default = 1.3*

Clean border.

img_ws_nchans

int, *optional*, *default = 3*

Number of channels in output image.

img_ws_joinchans

bool, *optional*, *default = True*

Join channels to create MFS image.

img_ws_specfit_nrcoeff

int, *optional*, *default = 2*

Number of spectral polynomial terms to fit to each clean component. This is equal to the order of the polynomial plus 1.

img_ws_stokes

{"I"}, *optional*, *default = I*

Stokes image to create. For this first release of CARACal, the only option is 'I'.

img_ws_auto_mask

float, *optional*, *default = 7*

Auto-masking threshold, given as the number of sigma_rms.

img_ws_auto_thr

float, *optional*, *default = 0.5*

Auto-clean threshold, given as the number of sigma_rms.

img_ws_col

str, *optional*, *default = CORRECTED_DATA*

Column to image.

img_ws_fits_mask

str, *optional*, *default = catalog_mask.fits*

Filename of fits mask (in output/masking folder).

img_ws_multi_scale

bool, *optional*, *default = False*

Switch on multiscale cleaning.

img_ws_multi_scale_scales

list of int, *optional*, *default = 10, 20, 30*

Scales for multiscale cleaning, in pixels.

img_ws_local_rms

bool, *optional*, *default = False*

Switch on local-rms measurement for cleaning.

transfer_model_dd

Repredict WSClean model to the highest channel resolution.

enable

bool, *optional*, *default = False*

Enable the 'transfer_model_dd' segment.

dd_model

str, *optional*, *default = auto*

Name of the sky-model file. (Currently the only supported format is that of WSClean component lists.) When set to 'auto', the pipeline builds the file name from the input parameters of the selfcal loop. The file is assumed to be in the 'output' directory.

dd_spectra

bool, *optional*, *default = True*

Model sources as non-flat spectra. The spectral coefficients and reference frequency must be present in the sky model.

dd_row_chunks

int, *optional*, *default = 0*

Number of rows of input .MS that are processed in a single chunk.

dd_model_chunks

int, *optional*, *default = 0*

Number of sky-model components that are processed in a single chunk.

dd_exp-sign-convention

str, *optional*, *default = casa*

Sign convention to use for the complex exponential. The default, 'casa', specifies the e^(2.pi.I) convention, while 'thompson' specifies the e^(-2.pi.I) convention in The White Book and Fourier analysis literature.

dd_within

str, *optional*, *default = ' '*

Give JS9 region file. Only sources within those regions will be included.

dd_points_only

bool, *optional*, *default = False*

Select only 'point' sources.

dd_num_sources

int, *optional*, *default = 0*

Select only N brightest sources.

dd_num_workers

int, *optional*, *default = 0*

Explicitly set the number of worker threads. Default is 0, meaning it uses all threads.

dd_mem_frac

float, *optional*, *default = 0.5*

Fraction of system RAM that can be used. Used when setting automatically the chunk size.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.3 flag

Flagging of the data. The selected flagging steps are executed in the same order in which they are given below.

enable

bool

Execute the flag worker.

field

{"target", "calibrators"}, *optional*, *default = calibrators*

Fields that should be flagged. It can be set to either 'target' or 'calibrators' (i.e., all calibrators) as defined in the obsconf worker. Note that this selection is ignored – i.e., all fields in the selected .MS file(s) are flagged – in the flagging step 'flag_mask' (see below). If a user wants to only flag a subset of the calibrators the selection can be further refined using 'calibrator_fields' below. The value of 'field' is also used to compose the name of the .MS file(s) that should be flagged, as explained in 'label_in' below.

label_in

str, *optional*, *default = ' '*

This label is added to the input .MS file(s) name, given in the getdata worker, to define the name of the .MS file(s) that should be flagged. These are <input>_<label>.ms if 'field' (see above) is set to 'calibrators', or <input>-<target>_<label>.ms if 'field' is set to 'target' (with one .MS file for each target in the input .MS). If empty, the original .MS is flagged with the field selection explained in 'field' above.

calfields

str, *optional*, *default = auto*

If 'field' above is set to 'calibrators', users can specify here what subset of calibrators to process. This should be a comma-separated list of 'xcal' ,'bpcal', 'gcal' and/or 'fcal', which were all set by the obsconf worker. Alternatively, 'auto' selects all calibrators.

rewind_flags

Rewind flags to specified version.

enable

bool, *optional*, *default = True*

Enable the 'rewind_flags' segment.

mode

{"reset_worker", "rewind_to_version"}, *optional*, *default = reset_worker*

If mode = 'reset_worker' rewind to the flag version before this worker if it exists, or continue if it does not exist; if mode = 'rewind_to_version' rewind to the flag version given by 'version' below.

version

str, *optional*, *default = auto*

Flag version to rewind to. If set to 'auto' it will rewind to the version prefix_workername_before, where 'prefix' is set in the 'general' worker, and 'workername' is the name of this worker including the suffix '__X' if it is a repeated instance of this worker in the configuration file. Note that all flag versions that had been saved after this version will be deleted.

overwrite_flagvers

bool, *optional*, *default = False*

Allow CARACal to overwrite existing flag versions. Not recommended. Only enable this if you know what you are doing.

unflag

Unflag all visibilities for the selected field(s).

enable

bool, *optional*, *default = False*

Enable the 'unflag' segment.

flag_autopowerspec

Flags antennas based on drifts in the scan average of the auto-correlation spectra per field. This doesn't strictly require any calibration. It is also not field-structure dependent, since it is just based on the DC of the field. Compares scan to median power of scans per field per channel. Also compares antenna to median of the array per scan per field per channel. This should catch any antenna with severe temperature problems.

enable

bool, *optional*, *default = False*

Enable the 'flag_autopowerspec' segment.

scan_thr

int, *optional*, *default = 3*

Threshold for flagging (in sigma) above the rest of the scans per field per channel.

ant_group_thr

int, *optional*, *default = 5*

Threshold for flagging (in sigma) above array median-power spectra per scan per field per channel.

col

str, *optional*, *default = DATA*

Data column to flag.

threads

int, *optional*, *default = 8*

Number of threads to use.

flag_autocorr

Flag auto-correlations, through the FLAGDATA task in CASA.

enable

bool, *optional*, *default = False*

Enable the 'flag_autocorr' segment.

flag_quack

Do quack flagging, i.e. flag the beginning and/or end chunks of each scan. Again, this is done through FLAGDATA.

enable

bool, *optional*, *default = False*

Enable the 'flag_quack' segment.

interval

float, *optional*, *default = 8.*

Time interval (in seconds) to flag.

mode

{"beg", "endb", "tail", "end"}, *optional*, *default = beg*

Quack flagging mode. Options are 'beg' (which flags the beginning of the scan), 'endb' (which flags the end of the scan), 'tail' (which flags everything but the first specified seconds of the scan), and 'end' (which flags all but the last specified seconds of the scan).

flag_elevation

Use CASA FLAGDATA to flag antennas with pointing elevation outside the selected range.

enable

bool, *optional*, *default = False*

Enable the 'flag_elevation' segment.

low

float, *optional*, *default = 0*

Lower elevation limit. Antennas pointing at an elevation below this value are flagged.

high

float, *optional*, *default = 90*

Upper elevation limit. Antennas pointing at an elevation above this value are flagged.

flag_shadow

Use CASA FLAGDATA to flag shadowed antennas.

enable

bool, *optional*, *default = False*

Enable the 'flag_shadow' segment.

tol

float, *optional*, *default = 0.*

Amount of shadow allowed (in metres). A positive number allows antennas to overlap in projection. A negative number forces antennas apart in projection.

full_mk64

bool, *optional*, *default = False*

Consider all MeerKAT-64 antennas in the shadowing calculation, even if only a subarray is used.

flag_spw

Use CASA FLAGDATA to flag spectral windows/channels.

enable

bool, *optional*, *default = False*

Enable the 'flag_spw' segment.

chans

str, *optional*, *default = *:856~880MHz , *:1658~1800MHz, *:1419.8~1421.3MHz*

Channels to flag. Given as "spectral window index:start channel ~ end channel" e.g. "*:856~880MHz". End channels are not inclusive.

ensure_valid

bool, *optional*, *default = True*

Check whether the channel selection returns any data. If it does not, FLAGDATA is not executed (preventing the pipeline from crashing). This check only works with the following spw formats (multiple, comma-separated selections allowed), "*:firstchan~lastchan"; "firstspw~lastspw:firstchan~lastchan"; "spw:firstchan~lastchan"; "firstchan~lastchan". Channels are assumed to be in frequency (Hz, kHz, MHz, GHz allowed; if no units are given it assumes Hz).

flag_time

Use CASA FLAGDATA to flag a specified timerange in the data.

enable

bool, *optional*, *default = False*

Enable the 'flag_time' segment.

timerange

str, *optional*, *default = ' '*

Timerange to flag. Required in the format 'YYYY/MM/DD/HH:MM:SS~YYYY/MM/DD/HH:MM:SS'.

ensure_valid

bool, *optional*, *default = False*

Check whether the timerange is in the MS being considered. This stops the pipeline from crashing when multiple datasets are being processed.

flag_scan

Use CASA FLAGDATA to flag bad scans.

enable

bool, *optional*, *default = False*

Enable the 'flag_scan' segment.

scans

str, *optional*, *default = 0*

Use CASA FLAGDATA syntax for selecting scans to flag.

flag_antennas

Flag bad antennas. Or just the ones you have sworn a vendetta against.

enable

bool, *optional*, *default = False*

Enable the 'flag_antennas' segment.

antennas

str, *optional*, *default = 0*

Use CASA FLAGDATA syntax for selecting antennas to flag.

timerange

str, *optional*, *default = ' '*

Timerange to flag. Required in the format 'YYYY/MM/DD/HH:MM:SS~YYYY/MM/DD/HH:MM:SS'.

ensure_valid

bool, *optional*, *default = False*

Check whether the timerange is in the MS being considered. This stops the pipeline from crashing when multiple datasets are being processed.

flag_mask

Apply a static mask to flag known RFI.

enable

bool, *optional*, *default = False*

Enable the 'flag_mask' segment.

mask

str, *optional*, *default = ' '*

The mask to apply. This can be provided by the user, but CARACal provides an existing static mask ('labelled_rfimask.pickle.npy') that is specific to MeerKAT data.

uvrange

str, *optional*, *default = ‘ ‘*

Select range in UV-distance (CASA-style range, e.g. ‘lower~upper’) for flagging. This is in default units of metres but, e.g., ‘0~1000km’ and ‘0-500klambda’ can also be specified. Leaving this parameter blank will select the entire range in UV-distance.

flag_rfi

Flag RFI using AOFlagger, Tricolour, or CASA FLAGDATA with tfcrop.

enable

bool, *optional*, *default = False*

Enable the ‘flag_rfi’ segment.

flagger

{“aoflagger”, “tricolour”, “tfcrop”}, *optional*, *default = aoflagger*

Choose flagger for automatic flagging. Options are ‘aoflagger’, ‘tricolour’ and ‘tfcrop’.

col

str, *optional*, *default = DATA*

Specify which column to flag.

aoflagger

strategy

str, *optional*, *default = firstpass_Q.rfis*

The AOFlagger strategy file to use.

ensure_valid

bool, *optional*, *default = True*

Ensure that the selected AOFlagger strategy is compatible with the type of correlations present in the input .MS file(s). E.g., attempts to flag on Stokes V for an .MS with XX and YY only will result in an error and CARACal exiting. The rules are 1. XY,YX must be present in order to flag on Stokes V,U (or on XY,YX), and 2. XX,YY must be present in order to flag on Stokes I,Q (or on XX,YY). Disable this parameter only if you know what you are doing.

tricolour

backend

{“numpy”, “zarr-disk”}, *optional*, *default = numpy*

Visibility and flag data is re-ordered from an MS-row ordering into time-frequency windows ordered by baseline. Options are ‘numpy’ (if only a few scans worth of visibility data need to be re-ordered) and ‘zarr-disk’ (for larger data sizes, where re-ordering on disk, rather than in memory, is necessary).

mode

{“auto”, “manual”}, *optional*, *default = auto*

If mode is set to ‘manual’, Tricolour uses the flagging strategy set via ‘strategy’ below. If mode is set to ‘auto’, it uses the strategy in ‘strategy_narrowband’ in case of small bandwidth of the .MS file(s).

strategy

str, *optional*, *default = mk_rfi_flagging_calibrator_fields_firstpass.yaml*

Name of the Tricolour strategy file.

strat_narrow

str, *optional*, *default = calibrator_mild_flagging.yaml*

Name of the Tricolour strategy file to be used for an MS with narrow bandwidth, if mode = 'auto' (see above).

tfcrop

usewindowstats

{"none", "sum", "std", "both"}, *optional*, *default = std*

Calculate additional flags using sliding-window statistics. Options are 'none', 'sum', 'std', and 'both'. See usewindowstats, within documentation for CASA FLAGDATA, for further details.

combinescans

bool, *optional*, *default = False*

Accumulate data across scans, depending on the value set for the 'ntime' parameter.

flagdimension

{"freq", "time", "freqtime", "timefreq"}, *optional*, *default = freqtime*

Dimension(s) along which to calculate a fit(/fits) to the data. Options are 'freq', 'time', 'freqtime', and 'timefreq'. Note that the differences between 'freqtime' and 'timefreq' are only apparent if RFI in one dimension is signifcantly stronger than in the other.

timecutoff

float, *optional*, *default = 4.0*

Flagging threshold, in units of standard deviation (i.e. sigma) from the fit along the time axis.

freqcutoff

float, *optional*, *default = 3.0*

Flagging threshold, in units of standard deviation (i.e. sigma) from the fit along the frequency axis.

correlation

str, *optional*, *default = ' '*

Specify the correlation(s) to be considered for flagging with 'tfcrop'. E.g. 'XX,YY', or leave as '' to select all correlations.

inspect

Use the diagnostic products of RFInder to inspect the presence of RFI.

enable

bool, *optional*, *default = False*

Enable the 'inspect' segment.

telescope

{"meerkat", "apertif", "wsrt"}, *optional*, *default = meerkat*

Name of the telescope. Options are 'meerkat', 'apertif', and 'wsrt'.

field

str, *optional*, *default = target*

Field over which to determine flag statistics. Options are 'gcal', 'bpcal', and 'target'.

polarization

{"xx", "XX", "yy", "YY", "xy", "XY", "yx", "YX", "q", "Q"}, *optional*, *default = q*

Select the polarization, e.g. 'xx', 'yy', 'xy', 'yx', 'q' (and also each of these in upper case).

spw_enable

bool, *optional*, *default = True*

Enable averaging/rebinning in frequency.

spw_width

int, *optional*, *default = 10*

Frequency width of rebinned output table (in units of MHz).

time_enable

bool, *optional*, *default = True*

Enable averaging/rebinning in time.

time_step

int, *optional*, *default = 5*

Time steps (in units of minutes).

summary

Use CASA FLAGDATA in 'summary' mode to write flagging summary at the end of the pre-calibration flagging.

enable

bool, *optional*, *default = True*

Enable the 'summary' segment.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.4 general

General pipeline information, including data IDs, and prefixes for output files.

title

str, *optional*, *default* = ‘ ‘

Execute the ‘general’ worker.

rawdatadir

str, *optional*, *default* = ‘ ‘

Location where CARACal (over-)writes HDF5 files and JSON info files downloaded by the getdata worker.

msdir

str, *optional*, *default* = *msdir*

Location where CARACal will write and expect to find measurement set (.MS) files.

input

str, *optional*, *default* = *input*

Location where CARACal expects to find various input files (e.g., RFI flagging strategy files).

output

str, *optional*, *default* = *output*

Location where CARACal writes output products.

prefix

str, *optional*, *default* = *caracal*

Prefix for CARACal output products.

prep_workspace

bool, *optional*, *default* = *True*

Initialise the pipeline by copying input files (i.e. those that are MeerKAT specific, flagging strategies, beam model, etc.).

init_notebooks

list of str, *optional*, *default* = *std-progress-report, project-logs*

Install standard radiopadre notebooks, given by list of basenames.

report_notebooks

list of str, *optional*, *default = detailed-final-report*

Like init_notebooks, but will also be automatically rendered to HTML when report=True in a worker

final_report

bool, *optional*, *default = True*

Render report_notebooks to HTML at the end of each pipeline run

2.5.5 getdata

Download and/or convert/unarchive data so that it is in the measurement set (MS) format for further processing.

dataid

list of str

Basename of MS. For MeerKAT data to be downloaded by CARACal, this should be the data ID of the observation.

mvftoms

For MeerKAT HDFS files only – Convert HDF5/MVF files in data_path (specified for the general worker) to MS files. The latter are written to msdir (also as specified for the general worker), and an MS.TAR file is created.

enable

bool, *optional*, *default = False*

Enable the 'mvftoms' segment.

tar

bool, *optional*, *default = False*

Create a tarball of the converted MS.

chanrange

str, *optional*, *default = all*

Only extract channels in this range (0-based, inclusive; comma-separated string). It is also possible to specify 'all'.

fullpol

bool, *optional*, *default = False*

Extract all four correlations instead of only the XX and YY correlations.

untar

Unarchive MS from an archive file.

enable

bool, *optional*, *default = False*

Enable the 'untar' segment.

tar_options

str, *optional*, *default = -xvf*

The tar options to pass to the 'tar' command.

combine

Virtually concatenate MSs and proceed with the combined MS.

enable

bool, *optional*, *default = False*

Enable the 'combine' segment.

reset

bool, *optional*, *default = False*

Delete concatenated MS if it exists. Otherwise, proceed with the existing MS.

tar

Create a tarball of the concatenated MS.

enable

bool, *optional*, *default = False*

Enable the 'tar' segment.

tar_options

str, *optional*, *default = -cvf*

The tar options to pass to the 'tar' command.

untar

Unarchive MS from an archive file.

enable

bool, *optional*, *default = False*

Enable the 'untar' segment.

tar_options

str, *optional*, *default = -xvf*

The tar options to pass to the 'tar' command.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.6 inspect

Diagnostic plots for data

enable

bool

Execute the inspect worker (i.e. diagnostic plotting of the first-pass cross-calibrated data).

label_in

str, *optional*, *default = ‘ ‘*

Label of the input dataset

label_cal

str, *optional*, *default = ‘ ‘*

Label for output products (plots etc.) for this step.

dirname

str, *optional*, *default = ‘ ‘*

Subdirectory (under diagnostic plots) where the plots are to go.

shadems

Direct list of shadems plots.

enable

bool, *optional*, *default = True*

Execute series of “extended” shadems plots.

default_column

str, *optional*, *default = CORRECTED_DATA*

Data column to plot.

plots_by_field

list of str, *optional*, *default = ‘ ‘*

Sequence of shadems plot specifications, made per field

plots_by_corr

list of str, *optional*, *default = ‘ ‘*

Sequence of shadems plot specifications, made per correlation

ignore_errors

bool, *optional*, *default = True*

Don’t halt the pipeline for shadems plotting errors.

standard_plotter

{“plotms”, “shadems”, “ragavi_vis”, “none”}, *optional*, *default = ragavi_vis*

Application to use for making “standard” plots. Use “none” to disable.

correlation

str, *optional*, *default = XX, YY*

Label specifying the correlations.

num_cores

int, *optional*, *default = 8*

number of CPUs to use

mem_limit

str, *optional*, *default = 8GB*

Amount of memory (RAM) to use

uvrange

str, *optional*, *default = ‘ ‘*

Set the U-V range for data selection, e.g. ‘>50’.

fields

list of str, *optional*, *default = bpcal, gcal*

Fields to plot. Specify by field id, index or keys like, gcal, bpcal.

real_imag

Plot real vs imaginary parts of data.

enable

bool, *optional*, *default = True*

Executed the real v/s imaginary data plotting.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like, gcal, bpcal.

col

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchan

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_phase

Plot Amplitude vs Phase for data.

enable

bool, *optional*, *default = False*

Executes the plotting of amplitude v/s phase for data.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

col

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchan

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_uvwave

Plot data amplitude v/s uvwave.

enable

bool, *optional*, *default = True*

Executes plotting data amplitude as a function of uvwave.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

col

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchan

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_ant

Plot data amplitde v/s antenna.

enable

bool, *optional*, *default = False*

Executes plotting data amplitude v/s antennas.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

col

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchan

str, *optional*, *default = 10*

Number of channels to average for plotting.

phase_uvwave

Plot data phase v/s uvwave.

enable

bool, *optional*, *default = True*

Executes plotting data phase v/s uvwave.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

col

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchan

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_scan

Plot data amplitude v/s scan number.

enable

bool, *optional*, *default = False*

Executes plotting data amplitude v/s scan number.

fields

list of str, *optional*, *default = ‘ ‘*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

col

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchan

str, *optional*, *default = 10*

Number of channels to average for plotting.

amp_chan

Plot Amplitude vs Channel data.

enable

bool, *optional*, *default = True*

Executes the plotting of amplitude v/s phase for data.

fields

list of str, *optional*, *default = ' '*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

col

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchan

str, *optional*, *default = 10*

Number of channels to average for plotting.

phase_chan

Plot Phase vs Chan.

enable

bool, *optional*, *default = True*

Executes the plotting of amplitude v/s phase for data.

fields

list of str, *optional*, *default = ' '*

Fields to plot. Specify by field id, index or keys like: gcal, bpcal.

col

str, *optional*, *default = corrected*

Data column to plot.

avgtime

str, *optional*, *default = 10*

Time to average for plotting, in seconds.

avgchan

str, *optional*, *default = 10*

Number of channels to average for plotting.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.7 line

Process visibilities for spectral line work and create line cubes and images.

enable

bool

Execute the line worker.

label_in

str, *optional*, *default = corr*

Label defining the name of the .MS files to be processed. The .MS file names are composed using the .MS names set by dataid in the getdata worker, followed by the target ID (one file per target), followed by this label. This is the format used by CARACal whenever it writes an .MS file to disk (e.g., in the transform worker).

line_name

str, *optional*, *default = HI*

Suffix to be used for the name of the output files (data cubes etc).

restfreq

str, *optional*, *default = 1.420405752GHz*

Spectral line rest frequency.

rewind_flags

Rewind flags of the input .MS file(s) to specified version. Note that this is not applied to the .MS file(s) you might be running "transfer_apply_gains" on.

enable

bool, *optional*, *default = True*

Enable the 'rewind_flags' segment.

mode

{"reset_worker", "rewind_to_version"}, *optional*, *default = reset_worker*

If set to 'reset_worker', rewind to the flag version before this worker if it exists, or continue if it does not exist; if set to 'rewind_to_version', rewind to the flag version given by 'version' and 'mstransform_version' below.

version

str, *optional*, *default = auto*

Flag version to restore. This is applied to the .MS file(s) identified by "label" above. Set to "null" to skip this rewinding step. If 'auto' it will rewind to the version prefix_workername_before, where 'prefix' is set in the 'general' worker, and 'workername' is

the name of this worker including the suffix '__X' if it is a repeated instance of this worker in the configuration file. Note that all flag versions saved after this version will be deleted.

mstransform_version

str, *optional*, *default = auto*

Flag version to restore. This is applied to the .MS file(s) identified by "label" above plus the "_mst" suffix. Set to "null" to skip this rewind step. If 'auto' it will rewind to the version prefix_workername_before, where 'prefix' is set in the 'general' worker, and 'workername' is the name of this worker including the suffix '__X' if it is a repeated instance of this worker in the configuration file. Note that all flag versions saved after this version will be deleted.

overwrite_flagvers

bool, *optional*, *default = False*

Allow CARACal to overwrite existing flag versions. Not recommended. Only enable this if you know what you are doing.

subtractmodelcol

Replace the CORRECTED_DATA column of the .MS file(s) with the difference CORRECTED_DATA - MODEL_DATA. This is useful for continuum subtraction as it subtracts the continuum clean model written to MODEL_DATA. WARNING! The CORRECTED_DATA column is overwritten. To undo this operation enable the addmodelcol segment in this worker.

enable

bool, *optional*, *default = True*

Enable the 'subtractmodelcol' segment.

addmodelcol

Replace the CORRECTED_DATA column of the .MS file(s) with the sum CORRECTED_DATA + MODEL_DATA. This is useful to undo the operation performed by subtractmodelcol in this worker. WARNING! The CORRECTED_DATA column is overwritten.

enable

bool, *optional*, *default = False*

Enable the 'addmodelcol' segment.

mstransform

Perform Doppler-tracking corrections and/or UVLIN continuum subtraction with CASA mstransform. For each input .MS file, this produces an output .MS file whose name is the same as that of the input .MS file plus the suffix "_mst".

enable

bool, *optional*, *default = True*

Enable the 'mstransform' segment.

col

str, *optional*, *default = corrected*

Which column of the .MS file(s) to process.

doppler

Include the Doppler-tracking correction in the run of CASA mstransform.

enable

bool, *optional*, *default = True*

Enable the 'doppler' (i.e. Doppler correction) segment.

telescope

{"askap", "atca", "gmrt", "meerkat", "vla", "wsrt"}

Name of the telescope used to take the data. This is used to set the telescope's geographical coordinates when calculating the Doppler correction. Default is 'meerkat'. Current options are askap, atca, gmrt, meerkat, vla, wsrt.

mode

{"frequency"}, *optional*, *default = frequency*

Regridding mode (channel/velocity/frequency/channel_b). IMPORTANT! Currently, only frequency mode is supported. Other modes will throw an error.

frame

{"", "topo", "geo", "lsrk", "lsrd", "bary", "galacto", "lgroup", "cmb", "source"}, *optional*, *default = bary*

Output reference frame. Current options are '', topo, geo, lsrk, lsrd, bary, galacto, lgroup, cmb, and source.

veltype

{"radio", "optical"}, *optional*, *default = radio*

Velocity used when regridding if mode = velocity. Current options are radio,and optical.

changrid

str, *optional*, *default = auto*

Output channel grid for Doppler correction. Default is 'auto', and the pipeline will calculate the appropriate channel grid. If not 'auto' then it must be in the format 'nchan,chan0,chanw' where nchan is an integer, and chan0 and chanw must include units appropriate for the chosen mode (see parameter 'mode' above).

uvlin

Include UVLIN-like continuum subtraction in the run of CASA mstransform.

enable

bool, *optional*, *default = True*

Enable the 'UVLIN' segment.

fitspw

str, *optional*, *default = ' '*

Selection of line-free channels using CASA syntax (e.g. '0:0~100;150:300'). If set to null, a fit to all unflagged visibilities will be performed.

fitorder

int, *optional*, *default = 1*

Polynomial order of the continuum fit.

obsinfo

bool, *optional*, *default = True*

Create obsinfo.txt and obsinfo.json per .MS file created by CASA mstransform.

flag_mst_errors

Run AOFlagger to flag any faulty visibilities produced by CASA mstransform.

enable

bool, *optional*, *default = False*

Enable the 'flag_mst_errors' segment.

strategy

str, *optional*, *default = postmst.rfis*

AOFlagger strategy file.

sunblocker

Use sunblocker to grid the visibilities and flag UV cells affected by solar RFI. See description of sunblocker on github repository gigjozsa/sunblocker in method phazer of module sunblocker.py.

enable

bool, *optional*, *default = False*

Enable the 'sunblocker' segment.

use_mstransform

bool, *optional*, *default = True*

Run sunblocker on the .MS file(s) produced by the mstransform section of this worker instead of the input .MS file(s).

imsize

int, *optional*, *default = 900*

Image size (pixels). Use the same as in the make_cube section. This is used to set up the gridding of the visibilities.

cell

float, *optional*, *default = 2.*

Pixel size (arcsec). Use the same as in the make_cube section. This is used to set up the gridding of the visibilities.

thr

float, *optional*, *default = 4.*

Flag UV cells whose visibility deviates by more than this threshold from the average visibility on the UV plane. The threshold is in units of the rms dispersion of all visibilities.

vampirisms

bool, *optional*, *default = False*

Apply the flags to data taken during day time only. Note that all data are used when calculating which UV cells to flag.

uvmin

float, *optional*, *default = 0.*

Minimum uvdistance to be analysed (in wavelengths, lambda).

uvmax

float, *optional*, *default = 2000*

Maximum uvdistance to be analysed (in wavelengths, lambda).

make_cube

Make a line cube using either WSClean + SoFiA (optional for clean masks) or CASA Clean.

enable

bool, *optional*, *default = True*

Enable the 'make_cube' segment.

image_with

{"wsclean", "casa"}, *optional*, *default = wsclean*

Choose whether to image with WSClean + SoFiA ('wsclean') or with CASA Clean ('casa').

use_mstransform

bool, *optional*, *default = True*

Image the .MS file(s) produced by the mstransform section of this worker instead of the input .MS file(s).

stokes

str, *optional*, *default = I*

Polarizations in output cube (I,Q,U,V,XX,YY,XY,YX,RR,LL,RL,LR and combinations).

spwid

int, *optional*, *default = 0*

Spectral window to use.

nchans

int, *optional*, *default = 0*

Number of channels of the line cube, where 0 means all channels.

firstchan

int, *optional*, *default = 0*

First channel of the line cube.

binchans

int, *optional*, *default = 1*

Integer binning of channels.

npix

seq, *optional*, *default = 900 , 900*

Image size in pixels. List of integers (width and height) or a single integer for square images.

cell

float, *optional*, *default = 2*

Pixel size. The default unit is arcsec, but other units can be specified, e.g., 'scale 20asec'.

padding

float, *optional*, *default = 1.2*

Images have initial size padding*npix, and are later trimmed to the image size set via the 'npix' parameter.

weight

{"natural", "uniform", "briggs"}, *optional*, *default = briggs*

Options for the type of weighting to be used are natural, uniform, or briggs. When using Briggs weighting, the additional robust parameter has to be specified.

robust

float, *optional*, *default = 0*

Robust parameter in case of Briggs weighting.

taper

float, *optional*, *default = 0*

Gaussian taper FWHM in arcsec. Zero means no tapering.

niter

int, *optional*, *default = 1000000*

Maximum number of clean iterations to perform.

gain

float, *optional*, *default = 0.1*

Fraction of the peak that is cleaned in each minor iteration.

wscl_mgain

float, *optional*, *default = 1.0*

Gain value for major iterations in WSClean. I.e., the maximum fraction of the image peak that is cleaned in each major iteration. A value of 1 means that all cleaning happens in the image plane and no major cycle is performed.

wscl_sofia_niter

int, *optional*, *default = 2*

Maximum number of WSClean + SoFiA iterations. The initial cleaning is done with WSClean automasking or with a user-provided clean mask. Subsequent iterations use a SoFiA clean mask. A value of 1 means that WSClean is only executed once and SoFiA is not used.

wscl_sofia_converge

float, *optional*, *default = 1.1*

Stop the WSClean + SoFiA iterations if the cube RMS has dropped by a factor < wscl_sofia_converge when comparing the last two iterations (considering only channels that were cleaned). If set to 0 then the maximum number of iterations is performed regardless of the change in RMS.

wscl_removeintermediate

bool, *optional*, *default = False*

If set to true, WSClean + SoFiA intermediate-cubes are deleted from the output directory. If set to false, WSClean + SoFiA intermediate-cubes are retained in the output directory.

wscl_user_clean_mask

str, *optional*, *default = ' '*

User-provided WSClean clean-mask for the first WSClean + SoFiA iteration (i.e. give the filename of the clean-mask, which is to be located in the output/masking folder).

wscl_auto_mask

float, *optional*, *default = 10*

Cleaning threshold used only during the first iteration of WSClean. This is given as the number of sigma_rms to be cleaned down to, where sigma_rms is the noise level estimated by WSClean from the residual image before the start of every major deconvolution iteration. WSClean will clean blindly down to this threshold (wscl_auto_mask), before switching to the auto-threshold set via wscl_auto_threshold.

wscl_auto_thr

float, *optional*, *default = 0.5*

Cleaning threshold used for subsequent iterations of WSClean. This is given as the number of sigma_rms to be cleaned down to, where sigma_rms is the noise level estimated by WSClean from the residual image before the start of every major deconvolution iteration.

wscl_make_cube

bool, *optional*, *default = True*

If set to true, the output of WSClean is a data cube. If set to false, the output is one .FITS image per spectral channel.

wscl_noupdatemod

bool, *optional*, *default = True*

If set to true, WSClean will not store the line clean model in MODEL_DATA.

wscl_multiscale

bool, *optional*, *default = False*

Switch on WSClean multiscale cleaning.

wscl_multiscale_scales

list of int, *optional*, *default = 0, 10, 20, 30*

List of scales for WSClean multiscale, in units of pixels. Only used is wscl_multi_scale is set to True.

wscl_multiscale_bias

float, *optional*, *default = 0.6*

Parameter to set the bias during multiscale cleaning, where a lower bias will give preference to larger angular scales.

casa_thr

str, *optional*, *default = 10mJy*

Flux-density level to stop CASA cleaning. It must include units, e.g. '1.0mJy'.

casa_port2fits

bool, *optional*, *default = False*

Port CASA output to fits files.

remove_stokes_axis

Remove the Stokes axis from the line cube.

enable

bool, *optional*, *default = False*

Enable the 'remove_stokes_axis' segment.

pb_cube

Make a primary-beam cube.

enable

bool, *optional*, *default = False*

Enable the 'pb_cube' segment.

apply_pb

bool, *optional*, *default = False*

Whether or not to apply the primary-beam correction to the image cube.

freq_to_vel

Convert the spectral axis' header keys of the line cube from frequency to velocity in the radio definition, v=c(1-obsfreq/restfreq). No change of spectra reference frame is performed.

enable

bool, *optional*, *default = False*

Enable the 'freq_to_vel' segment.

reverse

bool, *optional*, *default = False*

Perform the inverse transformation and change the cube's 3rd axis from radio velocity to frequency.

sofia

Run SoFiA source-finder to produce a detection mask, moment images and catalogues.

enable

bool, *optional*, *default = True*

Enable the 'sofia' segment.

flag

bool, *optional*, *default = False*

Use flag regions?

flagregion

list of int, *optional*, *default = 0, 0, 0, 0, 0, 0*

Pixel/channel range(s) to be flagged prior to source finding. Format is [[x1, x2, y1, y2, z1, z2], . . .].

rmsMode

str, *optional*, *default = mad*

Method to determine rms ('mad' for using median absolute deviation, 'std' for using standard deviation, 'negative' for using Gaussian fit to negative voxels).

thr

float, *optional*, *default = 4.0*

SoFiA source-finding threshold, in terms of the number of sigma_rms to go down to (i.e. the minimum signal-to-noise ratio).

merge

bool, *optional*, *default = False*

Merge pixels detected by any of the SoFiA source-finding algorithms into objects. If enabled, pixels with a separation of less than mergeX pixels in the X direction, mergeY pixels in the Y direction, and mergeZ channels in the Z direction will be merged and identified as a single object in the mask. Objects whose extent is smaller than minSizeX, minSizeY or minSizeZ will be removed from the mask.

mergeX

int, *optional*, *default = 2*

Merging radius (in pixels) in the X direction (RA axis).

mergeY

int, *optional*, *default = 2*

Merging radius (in pixels) in the Y direction (Dec axis).

mergeZ

int, *optional*, *default = 3*

Merging radius (in channels) in Z direction (spectral axis).

minSizeX

int, *optional*, *default = 3*

Minimum size (in pixels) in the X direction (RA axis).

minSizeY

int, *optional*, *default = 3*

Minimum size (in pixels) in the Y direction (Dec axis).

minSizeZ

int, *optional*, *default = 3*

Minimum size (in channels) in the Z direction (spectral axis).

cubelets

bool, *optional*, *default = True*

Create a cubelet for each detected emission-line object.

mom0

bool, *optional*, *default = True*

Create a moment-0 image of the field.

mom1

bool, *optional*, *default = True*

Create a moment-1 image of the field.

sharpener

Run sharpener to extract and plot the spectra of all continuum sources brighter than a given threshold.

enable

bool, *optional*, *default = False*

Enable the 'sharpener' segment.

catalog

{"NVSS", "PYBDSF"}, *optional*, *default = PYBDSF*

Type of catalogue to use. Options are PYBDSF and NVSS.

chans_per_plot

int, *optional*, *default = 50*

Number of channels to plot per detailed plot.

thr

float, *optional*, *default = 20*

Threshold to select sources in online catalogue (in units of mJy).

width

str, *optional*, *default = 1.0d*

Field-of-view of output catalogue (in units of degrees).

label

str, *optional*, *default = ‘ ‘*

Prefix label of plot names and titles.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.8 mask

Create .FITS mask from a catalog and (optionally) merge with an existing .FITS mask provided by the user. WARNING - At the moment this worker can only be executed on a single target at a time. Iterating over N targets is not done automatically.

enable

bool

Execute the mask worker.

label_in

str, *optional*, *default = corr*

Label of the .MS file that contains information about the target.

label_out

str, *optional*, *default = catalog_mask*

Prefix used for the name of the .FITS mask created by this worker. The full name consists of this prefix followed by the target name extracted by the observation_config worker. To use this output .FITS mask as a clean mask in the self_cal worker users should set relevant entry of cleanmask_method to label_out.

centre_coord

list of str, *optional*, *default = HH:MM:SS , DD:MM:SS*

Coordinates of the centre of the field-of-view (read from reference_dir by default).

mask_size

int, *optional*, *default = 1800*

Number of pixels in the mask. This must be the same as img_npix in the selfcal worker.

cell_size

float, *optional*, *default = 2.*

Size of pixels in the mask, in units of arcsec. This must be the same as img_cell in the selfcal worker.

extended_source_map

str, *optional*, *default = Fornaxa_vla.FITS*

Name of the input mask for particularly-extended sources in the field.

catalog_query

Query catalog to select field/sources for constructing the mask.

enable

bool, *optional*, *default = true*

Enable the 'query_catalog' segment.

catalog

{"NVSS", "SUMSS"}, *optional*, *default = SUMSS*

Name of catalog to query. Options are 'NVSS' and 'SUMSS'.

image_width

str, *optional*, *default = 1.2d*

Angular size of the region of sky that we want to mask (e.g. '1.2d', where 'd' indicates degrees). This should be kept larger than the dirty image.

nvss_thr

float, *optional*, *default = 10e-3*

Flux-density threshold for selecting sources in the radio map, corrected for the primary beam. Value given is in units of Jy, or is the minimum signal-to-noise ratio (i.e. number of sigma_rms), used for SoFiA source-finding.

pbcorr

Apply a primary-beam correction to the input image before extracting the mask.

enable

bool, *optional*, *default = true*

Enable the 'pb_correction' segment.

frequency

float, *optional*, *default = 1.420405752*

Since the primary-beam size changes with frequency, provide the central frequency of the considered dataset.

make_mask

Build mask from an existing image using SoFiA and/or a threshold cutoff.

enable

bool, *optional*, *default = true*

Enable the 'make_mask' segment.

mask_method

{"thresh", "sofia"}, *optional*, *default = sofia*

The tool to use for masking. Options are 'thresh' and 'sofia'.

input_image

{"pbcorr", "path_to_mask"}, *optional*, *default = pbcorr*

Input image where to create mask ???? what is this ???

thr_lev

int, *optional*, *default = 5*

Flux-density threshold for selecting sources in the SUMSS map, corrected for the primary beam. Value given is in units of Jy, or is the minimum signal-to-noise ratio (i.e. number of sigma_rms), used for SoFiA source-finding.

scale_noise_window

int, *optional*, *default = 101*

Size of the window over which SoFiA measures the local rms, in units of pixels.

merge_with_extended

Merge newly-determined mask components with the existing mask for the extended source.

enable

bool, *optional*, *default = False*

Execute segment 'merge_with_extended'.

extended_source_map

str, *optional*, *default = extended_mask.fits*

Name of the mask-image of the extended source to merge with the current mask-image.

mask_method

{"thresh", "sofia"}, *optional*, *default = thresh*

The tool to use for masking. Options are 'thresh' and 'sofia'.

thr_lev

float, *optional*, *default = 8e-2*

Flux-density threshold for selecting sources in the SUMSS map, corrected for the primary beam. Value given is in units of Jy, or is the minimum signal-to-noise ratio (i.e. number of sigma_rms), used for SoFiA source-finding.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.9 mosaic

Mosaic the 2D-images (or cubes) made with the selfcal/crosscal (or line) worker. If not available on disc, the primary beam is built by the mosaic worker, assuming a Gaussian shape with FWHM = 1.02 lambda/D.

enable

bool

Execute the mosaic worker.

mosaic_type

{"continuum", "spectral"}

Type of mosaic to be made, either continuum (2D) or spectral (3D).

target_images

seq, *optional*, *default = directory/first_image.fits, directory/second_image.fits*

List of images to be mosaicked, with suffix of image.fits being expected.

label_in

str, *optional*, *default = corr*

For autoselection of images, this needs to match the label/label_cal setting used for the selfcal/crosscal worker (when mosaicking continuum images) or the label setting used for the line worker (when mosaicking cubes).

line_name

str, *optional*, *default = HI*

Spectral mode only – If autoselection is used to find the final cubes, this needs to match the line_name parameter used for the line worker.

use_mfs

bool, *optional*, *default = False*

Continuum mode only – If the images to be mosaicked were created using MFS, in the selfcal or crosscal worker, then this needs to be indicated via this parameter.

name

str, *optional*, *default = ‘ ‘*

The prefix to be used for output files. Default is the pipeline prefix, as set for the general worker.

domontage

Re-grid the input images, and associated beams.

enable

bool, *optional*, *default = True*

Enable the ‘domontage’ (i.e. re-gridding) segment.

cutoff

float, *optional*, *default = 0.1*

The cutoff in the primary beam. It should be a number between 0 and 1.

dish_diameter

float, *optional*, *default = 13.5*

If no continuum pb.fits are already in place, user needs to specify the dish diameter (in units of m) so that rudimentary primary beams can be created.

ref_frequency

float, *optional*, *default = 1383685546.875*

If no continuum pb.fits are already in place, user needs to specify the reference frequency (in units of Hz) so that rudimentary primary beams can be created.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.10 obsconf

Set up some basic information about the observation(s).

obsinfo

Get observation information.

enable

bool

Execute the obsconf worker.

listobs

bool, *optional*, *default = True*

Run the CASA 'listobs' task to get observation information.

summary_json

bool, *optional*, *default = True*

Run the MSUtils summary function to get observation information written as a JSON file, which can then be used to automatically configure pipeline.

vampirisms

bool, *optional*, *default = False*

Return the time range over which observations were taken at night.

plotelev

Make Elevation vs Hour-angle plots for observed fields.

enable

bool, *optional*, *default = True*

Enable segment 'plot_elevation_tracks'.

plotter

{"plotms", "owlcat"}, *optional*, *default = owlcat*

The application to be used for making plots. Options are 'plotms' and 'owlcat'.

target

list of str, *optional*, *default = all*

The field name(s) of the target field(s), separated by commas if there are multiple target fields. Or set this parameter to 'all' to select all of the target fields.

gcal

list of str, *optional*, *default = all*

The field name(s) of the gain (amplitude/phase) calibrator field(s). Or set 'all' to select all of the gcal fields, 'longest' to select the gcal field observed for the longest time, or 'nearest' to select the gcal field closest to the target. Note that if multiple targets and gcals are present, then 'all' (for both the 'target' and 'gcal' parameters) means that each target will be paired with the closest gcal.

bpcal

list of str, *optional*, *default = longest*

The field name(s) of the bandpass calibrator field(s). Or set 'all' to select all of the bpcal fields, 'longest' to select the bpcal field observed for the longest time, or 'nearest' to select the bpcal field closest to the target.

fcal

list of str, *optional*, *default = longest*

The field name(s) of the fluxscale calibrator field(s). Or set 'all' to select all of the fcal fields, 'longest' to select the fcal field observed for the longest time, or 'nearest' to select the fcal field closest to the target.

xcal

list of str, *optional*, *default = longest*

The field name(s) of the crosshand phase-angle calibrator field(s). Or set 'all' to select all of the xcal fields, 'longest' to select the xcal field observed for the longest time, or 'nearest' to select the xcal field closest to the target. This calibrator must be linearly polarized and have a non-zero parallactic angle coverage at the time of observation in order to solve for the X-Y offsets in digitizers and the absolute polarization angle of the system. Successful calibration derotates U from V.

refant

str

The reference antenna, which can be identified by an antenna name or number.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.11 prep

Prepare the data for calibration and imaging.

enable

bool

Execute the prep worker (i.e. the data-preparation step).

label_in

str

If this label is an empty string this worker operates on the input .MS file(s) given in the getdata worker. If the label is not an empty string then it is added to the input .MS file(s) name (specified for the getdata worker) to define the name of the .MS file(s) to work on. These are named <input>_<label>.ms if 'field' (see below) is set to 'calibrators', or <input>-<target>_<label>.ms if 'field' is set to 'target' (with one .MS file for each target in the input .MS).

field

{"target", "calibrators"}, *optional*, *default = calibrators*

In combination with a non-empty 'label_in' (see above), 'field' defines which .MS file(s) to work on. This parameter is ignored if 'label_in' is empty. Options are 'target' and 'calibrators'.

fixuvw

Fix the UVW coordinates through the CASA task 'fixvis'.

enable

bool, *optional*, *default = False*

Enable the 'fixvis' segment.

clearcal

bool, *optional*, *default = False*

Clear out calibrated data and reset the previous predicted model.

manage_flags

Manage flags.

enable

bool, *optional*, *default = True*

Enable the 'manage_flags' segment.

mode

{"legacy", "restore"}, *optional*, *default = legacy*

Mode for managing flags. If set to 'legacy', save the current FLAG column as a 'caracal_legacy' flag version if a flag version with that name does not exisy yet; else restore the 'caracal_legacy' flag version and delete all flag versions created after it. If set to 'restore', restore flags from the flag version specified by 'version' below, and delete all flag versions created after that version.

version

str, *optional*, *default = auto*

Name of the flag version to restore. If set to 'auto', rewind to the version prefix_workername_before, where 'prefix' is set in the 'general' worker, and 'workername' is the name of this worker including the suffix '__X' if it is a repeated instance of this worker in the configuration file. Note that all flag versions saved after this version will be deleted.

specweights

How to initialize spectral weights.

enable

bool, *optional*, *default = False*

Enable the 'spectral_weights' segment.

mode

{"uniform", "estimate", "delete"}, *optional*, *default = uniform*

Mode for spectral weights. Options are 'uniform' (set all weights to unity), 'estimate' (estimate spectral weights from frequency-dependent SEFD/Tsys/Noise values, and see 'estimate' segment of this section), and 'delete' (delete WEIGHT_SPECTRUM column if it exists).

calculate

Calculate spectral weights from frequency-dependent SEFD/Tsys/Noise values.

statsfile

str, *optional*, *default = use_package_meerkat_spec*

File with SEFD/Tsys/Noise data. If data is from the MeerKAT telescope, you can specify 'use_package_meerkat_spec' to use package data.

weightcols

list of str, *optional*, *default = WEIGHT, WEIGHT_SPECTRUM*

Column names for spectral weights.

noisecols

list of str, *optional*, *default = SIGMA, SIGMA_SPECTRUM*

Column names for noise values.

apply

bool, *optional*, *default = True*

Write columns to file.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.12 selfcal

Perform self-calibration on the data.

enable

bool

Execute the selfcal worker.

label_in

str, *optional*, *default = corr*

Label of the .MS files to process.

rewind_flags

Rewind flags of the input .MS file(s) to specified version. Note that this is not applied to .MS file(s) you might be running "transfer_apply_gains" on.

enable

bool, *optional*, *default = True*

Enable segment rewind_flags.

mode

{"reset_worker", "rewind_to_version"}, *optional*, *default = reset_worker*

If mode = 'reset_worker' rewind to the flag version before this worker if it exists, or continue if it does not exist; if mode = 'rewind_to_version' rewind to the flag version given by 'version' and 'transfer_apply_gains_version' below.

version

str, *optional*, *default = auto*

Flag version to restore. This is applied to the .MS file(s) identified by "label" above. Set to "null" to skip this rewinding step. If 'auto' it will rewind to the version prefix_workername_before, where 'prefix' is set in the 'general' worker, and 'workername' is the name of this worker including the suffix '__X' if it is a repeated instance of this worker in the configuration file. Note that all flag versions saved after this version will be deleted.

transfer_apply_gains_version

str, *optional*, *default = auto*

Flag version to restore. This is applied to the .MS file(s) identified by "transfer_to_label" in the "transfer_apply_gains" section below. Set to "null" to skip this rewind step. If 'auto' it will rewind to the version prefix_workername_before, where 'prefix' is set in the 'general' worker, and 'workername' is the name of this worker including the suffix '__X' if it is a repeated instance of this worker in the configuration file. Note that all flag versions saved after this version will be deleted.

overwrite_flagvers

bool, *optional*, *default = False*

Allow CARACal to overwrite existing flag versions. Not recommended. Only enable this if you know what you are doing.

calibrate_with

{"meqtrees", "cubical"}, *optional*, *default = cubical*

Tool to use for calibration. Options are meqtrees and cubical.

spwid

int, *optional*, *default = 0*

Provide spectral window ID.

ncpu

int, *optional*, *default = 5*

Number of CPUs to use.

minuvw_m

int, *optional*, *default = 0*

Exclude baselines shorter than this value (given in metres) from the imaging and self-calibration loop.

img_npix

int, *optional*, *default = 1800*

Number of pixels in output image.

img_padding

float, *optional*, *default = 1.3*

Padding in WSClean.

img_gain

float, *optional*, *default = 0.10*

Fraction of the peak that is cleaned in each minor iteration.

img_mgain

float, *optional*, *default = 0.90*

Gain for major iterations in WSClean. I.e., maximum fraction of the image peak that is cleaned in each major iteration. A value of 1 means that all cleaning happens in the image plane and no major cycle is performed.

img_cell

float, *optional*, *default = 2.*

Image pixel size (in units of arcsec).

img_weight

{"briggs", "uniform", "natural"}, *optional*, *default = briggs*

Type of image weighting, where the options are 'briggs', 'uniform', and 'natural'. If 'briggs', set the 'img_robust' parameter.

img_robust

float, *optional*, *default = 0.*

Briggs robust value.

img_taper

str, *optional*, *default = 0.*

Gaussian taper for imaging (in units of arcsec).

img_maxuv_l

float, *optional*, *default = 0.*

Taper for imaging (in units of lambda).

img_transuv_l

float, *optional*, *default = 10.*

Transition length of tukey taper (taper-tukey in WSClean, in % of maxuv).

img_niter

int, *optional*, *default = 1000000*

Number of cleaning iterations.

img_nmiter

int, *optional*, *default = 0*

Number of major cycles.

img_cleanborder

float, *optional*, *default = 1.3*

Clean border.

img_nchans

int, *optional*, *default = 3*

Number of channels in output image.

img_joinchans

bool, *optional*, *default = True*

Join channels to create MFS image.

img_specfit_nrcoeff

int, *optional*, *default = 2*

Number of spectral polynomial terms to fit to each clean component. This is equal to the order of the polynomial plus 1.

img_stokes

{"I"}, *optional*, *default = I*

Stokes image to create. For this first release of CARACal, the only option is 'I'.

img_multiscale

bool, *optional*, *default = False*

Switch on multiscale cleaning.

img_multiscale_scales

list of int, *optional*, *default = 10, 20, 30*

Scales of multiscale [0,10,20,etc.] in pixels.

img_sofia_settings

SoFiA source finder settings used for the imaging iterations whose entry in 'image/cleanmask_method' below is 'sofia'. The resulting clean mask is located in <output>/masking.

kernels

list of float, *optional*, *default = 0., 3., 6., 9.*

FWHM of spatial Gaussian kernels in pixels.

pospix

bool, *optional*, *default = True*

Merges only positive pixels of sources in mask.

flag

bool, *optional*, *default = False*

Set whether flag regions are to be used (True) or not (False).

flagregion

list of str, *optional*, *default = ‘ ‘*

Pixel/channel range(s) to be flagged prior to source finding. Format is [[x1, x2, y1, y2, z1, z2], . . .].

inputmask

str, *optional*, *default = ‘ ‘*

User-provided input-mask that will be supplemented by the SoFiA mask, created through SoFiA source-finding.

fornax_special

bool, *optional*, *default = False*

Activate masking of Fornax A using SoFiA.

fornax_thr

list of float, *optional*, *default = 4.0*

SoFiA source-finding threshold, in terms of the number of sigma_rms to go down to (i.e. the minimum signal-to-noise ratio).

fornax_sofia

bool, *optional*, *default = False*

Use SoFiA for the mask of Fornax A, instead of that by Fomalont.

cal_niter

int, *optional*, *default = 2*

Number of self-calibration iterations to perform.

start_iter

int, *optional*, *default = 1*

Start selfcal iteration loop at this start value (1-indexed).

cal_gain_cliplow

float, *optional*, *default = 0.5*

Lower threshold for clipping on gain amplitude.

cal_gain_cliphigh

float, *optional*, *default = 2.*

Upper threshold for clipping on gain amplitude.

cal_timeslots_chunk

int, *optional*, *default = -1*

Chunk data up by this number of timeslots. This limits the amount of data processed at once. Smaller chunks allow for a smaller RAM footprint and greater parallelism but sets an upper limit on the time solution intervals that may be employed. 0 means 'use the full time-axis' but does not cross scan boundaries. -1 means 'use the largest solution interval'.

cal_model_mode

{"vis_only", "pybdsm_only", "pybdsm_vis"}, *optional*, *default = vis_only*

Mode for using a calibration model, based on visibilities and/or PyBDSM source-finding. Options are vis_only, pybdsm_only, and pybdsm_vis. 'vis_only' means that only MODEL_DATA will be used to to calibrate. 'pybdsm_only' means that PyBDSM-generated, tigger-format local sky models will be used. 'pybdsm_vis' is the same as the 'pybdsm_only' mode except for the last iteration of selfcal, where the PyBDSM-based model is complemented by MODEL_DATA. This third mode is only to be used with output_data set to 'CORR_RES' (below) and is very tricky. Therefore, user discretion is advised.

cal_bjones

bool, *optional*, *default = False*

Enable calculation of the B-Jones matrix, for bandpass calibration.

cal_cubical

Parameters that only apply when using CubiCal for the calibration.

max_prior_error

float, *optional*, *default = 0.3*

Flag solution intervals where the prior variance estimate is above this value.

max_post_error

float, *optional*, *default = 0.3*

Flag solution intervals where the posterior variance estimate is above this value.

chan_chunk

int, *optional*, *default = -1*

Chunk data up by this number of channels. This limits the amount of data processed at once. Smaller chunks allow for a smaller RAM footprint and greater parallelism but sets an upper limit on the frequency solution intervals that may be employed. 0 means 'use the full frequency-axis' but does not cross SPW boundaries. -1 means 'use the largest solution interval'.

weight_col

str, *optional*, *default = WEIGHT*

Column with weights for use in CubiCal.

shared_mem

str, *optional*, *default = 100Gb*

Set the amount of shared memory for CubiCal.

flag_madmax

bool, *optional*, *default = True*

Flags based on maximum of mad in CubiCal.

madmax_flag_thr

list of int, *optional*, *default = 0, 10*

Threshold for madmax flagging in CubiCal, where the provided list works exactly as described in CubiCal readthedocs for the parameter –madmax-threshold.

solterm_niter

list of int, *optional*, *default = 50, 50, 50*

Number of iterations per Jones term for CubiCal. Always a 3 digit array with iterations for 'G,B,GA' even when B or GA are not used.

overwrite

bool, *optional*, *default = True*

Allow CubiCal to overwrite the existing gain_tables and other CubiCal output for self-calibration that were produced in a previous run of the selfcal worker with the same prefix.

dist_max_chunks

int, *optional*, *default = 4*

Maximum number of time/freq data-chunks to load into memory simultaneously. If set to 0, then as many data-chunks as possible will be loaded.

ragavi_plot

Use ragavi to plot diagnostic plots for self-calibration.

enable

bool, *optional*, *default = False*

Enable the plotting of diagnostics, using ragavi.

gaintype

list of str, *optional*, *default = G*

List of gain solution types. Options are 'F' (flux-calibration solutions), 'B' (bandpass-calibration solutions), 'K' (delay-calibration solutions), 'G' (gain-calibration solutions), and 'D' (D-Jones leakage-calibration solutions).

field

list of int, *optional*, *default = 0*

Fields to plot. Specify by field ID.

cal_meqtrees

Parameters that only apply when using MeqTrees for the calibration.

two_step

bool, *optional*, *default = False*

Trigger a two-step calibration process in MeqTrees where the phase-only calibration is applied before continuing with amplitude + phase-calibration. Aimfast is turned on to determine the solution sizes automatically.

aimfast

Quality assessment parameter.

enable

bool, *optional*, *default = False*

Enable the 'aimfast' segment.

tol

float, *optional*, *default = 0.02*

Relative change in weighted mean of metrics (specified via convergence_criteria below) from aimfast.

convergence_criteria

list of str, *optional*, *default = DR*

The residual statistic to check convergence against. Every metric/criterion listed will be combined into a weighted mean. Options are 'DR' (dynamic range), 'MEAN' (mean of the residual flux), 'STDDev' (standard deviation), 'SKEW' (skewness, 3rd-moment), and 'KURT' (kurtosis, 4th-moment). However, note that when cal_model_mode = 'vis_only', 'DR' is no longer an option.

area_factor

int, *optional*, *default = 6*

A multiplicative factor that sets the total area over which the metrics are calculated, where total_area = psf_size*area_factor. This area is centred on the position of peak flux-density in the image.

radius

float, *optional*, *default = 0.6*

Cross-matching radius (in units of arcsec), for comparing source properties in a catalogue before and after an iteration of self-calibration.

normality_model

{"normaltest", "shapiro"}, *optional*, *default = normaltest*

The type of normality test, to use for testing how well the residual image is modelled by a normal distribution. Options are 'normaltest' (i.e. D'Agostino) and 'shapiro'.

plot

bool, *optional*, *default = True*

Generate html plots for comparing catalogues and residuals.

image

Imaging parameters.

enable

bool, *optional*, *default = True*

Enable the 'image' segment.

col

list of str, *optional*, *default = DATA, CORRECTED_DATA*

Column(s) to image.

clean_cutoff

list of float, *optional*, *default = 0.5, 0.5*

Cleaning threshold to be used by WSClean. This is given as the number of sigma_rms to be cleaned down to, where sigma_rms is the noise level estimated by WSClean from the residual image before the start of every major deconvolution iteration.

cleanmask_method

list of str, *optional*, *default = wsclean, wsclean*

Method used to create the clean mask. The possible values are 1) 'wsclean' to use WSClean's auto-masking (threshold set by clean_mask_threshold below); 2) 'sofia' to create a clean mask using SoFiA (threshold set by clean_mask_threshold below, and additional settings in sofia_settings, do not use if output_data = CORR_RES); 3) a prefix string to use an existing .FITS mask located in output/masking and called prefix_target.fits, where the name of the target is set automatically by the pipeline. The latter .FITS mask could be the one created by the masking worker, in which case the prefix set here should correspond to label_out in the masking worker. Note that this third maskingm ethod can be used on multiple targets in a single pipeline run as long as they all have a corresponding prefix_target.fits mask in output/masking.

cleanmask_thr

list of float, *optional*, *default = 10.0, 6.0*

Threshold used to create the clean mask when clean_mask_method = 'wsclean' or 'sofia'. This is given as the number of sigma_rms to be cleaned down to, where sigma_rms is the (local) noise level.

cleanmask_localrms

list of bool, *optional*, *default = False, False*

Use a local-rms measurement when creating a clean mask with clean_mask_method = 'wsclean' or 'sofia'. If clean_mask_method = 'wsclean', this local-rms setting is also used for the clean_threshold above. Otherwise it is only used to define the clean mask, and clean_threshold is in terms of the global noise (rather than the local noise).

cleanmask_localrms_window

list of int, *optional*, *default = 31, 31*

Width of the window used to measure the local rms when creating the clean mask. The window width is in pixels for clean_mask_method = 'sofia', and in PSFs for clean_mask_method = 'wsclean'.

extract_sources

Source-finding parameters.

enable

bool, *optional*, *default = False*

Enable the 'extract_sources' segment.

sourcefinder

{"pybdsm", "sofia"}, *optional*, *default = pybdsm*

Set the source finder to be used. Options are 'pybdsm' (i.e. pybdsf) and 'sofia'.

local_rms

bool, *optional*, *default = False*

Use a local-rms estimate when applying the source-finding detection threshold.

spi

bool, *optional*, *default = False*

Extract the spectral index for the fitted sources.

thr_pix

list of int, *optional*, *default = 5*

Pixel threshold to be used for the source finder. I.e. the minimum number of contiguous pixels for emission to be classed as a 'source'.

thr_isl

list of int, *optional*, *default = 3*

Threshold to be used by the source finder to set the island boundary, given in the number of sigma above the mean. This determines the extent of the island used for fitting.

detection_image

bool, *optional*, *default = False*

Constrain the PyBDSM source-finding to only find sources included in the clean model.

calibrate

Calibration parameters.

enable

bool, *optional*, *default = True*

Enable the 'calibrate' segment.

model

list of str, *optional*, *default = 1,2*

Model number to use, or a combination of models. E.g. '1+2' to use the first and second models for calibration.

output_data

list of str, *optional*, *default = CORR_DATA*

Data to output after calibration. Options are 'CORR_DATA', 'CORR_RES' or 'CORRECTED_DATA', where CORR_DATA and CORRECTED_DATA are synonyms.

gain_matrix_type

list of str, *optional*, *default = GainDiagPhase, GainDiag*

Gain matrix type. 'GainDiagPhase' = phase-only calibration, 'GainDiagAmp' = amplitude only, 'GainDiag' = Amplitude + Phase, 'Gain2x2' = Amplitude + Phase taking non-diagonal terms into account, 'Fslope' = delay selfcal (for which solution intervals should be set to at least twice the values you would use for GainDiagPhase). Note that Fslope does not work with MeqTrees.

gsols_timeslots

list of int, *optional*, *default = 1*

G-Jones time solution interval. The parameter cal_timeslots_chunk above should be a multiple of Gsols_time. 0 entails using a single solution for the full time of the observations.

gsols_chan

list of int, *optional*, *default = 0*

G-Jones frequency solution interval. The parameter chan_chunk in calibrate section should a multiple of Gsols_channel. 0 entails using a single solution for the full bandwidth.

bsols_timeslots

list of int, *optional*, *default = 0*

B-Jones solutions for individual calibration steps in time.

bsols_chan

list of int, *optional*, *default = 2*

B-Jones solutions for individual calibration steps in frequency.

gasols_timeslots

list of int, *optional*, *default = -1*

Time intervals for amplitude calibration in CubiCal. 0 indicates average all. -1 defaults to Gsols_timeslots. If different from Gsols_timeslots, a second matrix is used and applied.

gasols_chan

list of int, *optional*, *default = -1*

Channel intervals for amplitude calibration in CubiCal. 0 indicates average all. -1 defaults to Gsols_channel. If different from Gsols_channels, a second matrix is used and applied.

restore_model

Take the modelled source(s) and restore it(/them) to the final, calibrated residual image.

enable

bool, *optional*, *default = False*

Enable the 'restore_model' segment.

model

str, *optional*, *default = 1+2*

Model number to use, or a combination of models. E.g. '1+2' to use the first and second models.

clean_model

str, *optional*, *default = 3*

Clean model number to use, or combination of clean models. E.g. '1+2' to use the first and second clean models.

flagging_summary

Output the flagging summary.

enable

bool, *optional*, *default = False*

Enable the 'flagging_summary' segment.

transfer_apply_gains

Interpolate gains over the high frequency-resolution data.

enable

bool, *optional*, *default = False*

Enable the 'transfer_apply_gains' segment.

transfer_to_label

str, *optional*, *default = corr*

Label of cross-calibrated .MS file to which to transfer and apply the selfcal gains.

interpolate

To interpolate the gains or not to interpolate the gains. That is indeed the question.

enable

bool, *optional*, *default = True*

Enable gain interpolation.

timeslots_int

int, *optional*, *default = -1*

Solution interval in time (units of timeslots/integration time) for transferring gains. -1 means use the solution interval from the calibration that is applied.

chan_int

int, *optional*, *default = -1*

Solution interval in frequency (units of channels) for transferring gains. -1 means use the solution interval from the calibration that is applied.

timeslots_chunk

int, *optional*, *default = -1*

Time chunk in units of timeslots for transferring gains with CubiCal. -1 means use the solution interval from the calibration that is applied.

chan_chunk

int, *optional*, *default = -1*

Frequency chunk in units of channels for transferring gains with CubiCal. '0' means the whole spw. -1 means use the solution interval from the calibration that is applied.

transfer_model

Transfer the model from the last WSClean imaging run to the MODEL_DATA column of another .MS .

enable

bool, *optional*, *default = True*

Enable the 'transfer_model' segment.

transfer_to_label

str, *optional*, *default = corr*

Label of the .MS file to which to transfer the model.

model

str, *optional*, *default = auto*

Name of the sky model file. (Currently the only supported format is that of WSClean component lists.) When 'auto', the pipeline builds the file name from the input parameters of the selfcal loop. The file is assumed to be in the 'output' directory.

row_chunks

int, *optional*, *default = 0*

Number of rows of the input .MS that are processed in a single chunk.

model_chunks

int, *optional*, *default = 0*

Number of sky model components that are processed in a single chunk.

within

str, *optional*, *default = ' '*

Give the JS9 region file. Only sources within those regions will be included.

points_only

bool, *optional*, *default = False*

Select only 'point' sources.

num_sources

int, *optional*, *default = 0*

Select only N brightest sources.

num_workers

int, *optional*, *default = 0*

Explicitly set the number of worker threads. Default is 0, meaning it uses all threads.

mem_frac

float, *optional*, *default = 0.5*

Fraction of system RAM that can be used. Used when setting automatically the chunk size.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.5.13 transform

Split, average and/or calibrate the data.

enable

bool

Execute the transform worker.

label_in

str, *optional*, *default = ‘ ‘*

Label of the input dataset.

label_out

str, *optional*, *default = corr*

Label of the output dataset.

rewind_flags

Rewind flags to specified version.

enable

bool, *optional*, *default = False*

Enable the ‘rewind_flags’ segment.

version

str, *optional*, *default = ‘ ‘*

Flag version to restore. Note that all flag versions saved after this version will be deleted.

split_field

Make new MS of targets or calibrators.

enable

bool, *optional*, *default = True*

Enable the 'split_field' segment.

field

str, *optional*, *default = target*

Fields to be split off from the input MS. Options are (separately) 'target', 'calibrators', 'bpcal', 'gcal' and 'fcal'. Also valid is any combination of 'bpcal', 'gcal' and 'fcal' in a comma-separated string (e.g. 'bpcal, fcal').

time_avg

str, *optional*, *default = ' '*

Time averaging to apply to the data, in units of seconds. If this parameter is instead set to '' or '0s' then no time averaging is applied.

chan_avg

int, *optional*, *default = 1*

Frequency averaging to apply to the data, given as the number of channels per frequency bin. If this parameter is set to '', '0', or '1', then no frequency averaging is applied.

col

str, *optional*, *default = corrected*

Column to be split, where the default is 'corrected'.

correlation

str, *optional*, *default = ' '*

Select the correlations, e.g. 'XX', 'YY'. Setting this to '' means that all correlations are selected.

create_specweights

bool, *optional*, *default = True*

Create a WEIGHT_SPECTRUM column in the output MS.

spw

str, *optional*, *default = ' '*

Select spectral windows and channels, following the same syntax as for the 'mstransform' task in CASA. Setting this to '' means that all spectral windows and channels are selected.

otfcal

Apply on-the-fly (OTF) calibration.

enable

bool, *optional*, *default = False*

Enable the 'otfcal' segment.

callib

str, *optional*, *default = ‘ ‘*

Name of the callib file to be used, if user has their own.

label_cal

str, *optional*, *default = 1gc1*

Label of the calibration tables to be used.

changecentre

Change the phase centre.

enable

bool, *optional*, *default = False*

Enable the ‘changecentre’ segment.

ra

str, *optional*, *default = 0h0m0.0s*

J2000 RA of the new phase centre, in the format XXhXXmXX.XXs .

dec

str, *optional*, *default = 0d0m0.0s*

J2000 Dec of the new phase centre, in the format XXdXXmXX.XXs .

obsinfo

Get observation information.

enable

bool, *optional*, *default = True*

Enable the ‘obsinfo’ segment.

listobs

bool, *optional*, *default = True*

Run the CASA ‘listobs’ task to get observation information.

summary_json

bool, *optional*, *default = True*

Run the MSUtils summary function to get observation information written as a JSON file.

report

bool, *optional*, *default = False*

(Re)generate a full HTML report at the end of this segment.

2.6 Software used by CARACal

As described in the *Introduction*, CARACal executes tasks from many different data reduction software packages thanks to the unifying scheme provided by

- Stimela

The software packages used by CARACal include:

- Aimfast
- AOFlagger
- Casa
- Crystalball
- Cubical
- MeqTrees
- Montage
- Ragavi
- PyBDSF
- RFInder
- ShadeMS
- SHARPener
- SoFiA
- Sunblocker
- Tricolour
- WSclean

We refer to the webpages linked above for references and manuals.

2.7 Caratekit utility

`caratekit.sh` is a shell script originally intended to test CARACal. It has no dependencies other than an existing bash shell installed on your computer system (most UNIX systems use bash as a default). To learn about the full range of options to use caratekit.sh type `caratekit.sh -h` or `caratekit -v` once it is installed. Here, we restrict the description to two standard use-cases: the installation of CARACal using `caratekit.sh` and three methods to consecutively reduce data with `caratekit.sh`. The advantage of a data reduction using `caratekit.sh` is that `caratekit.sh` automatically creates some system information that can be used to analyse potential issues and that can be linked in the CARACal issue tracker.

2.7.1 Installing `caratekit.sh`

Choose:

- A workspace directory `${workspace}` to install `caratekit.sh`

- A target directory `${carate_target}` to install `caratekit.sh` in. By default, this is `/usr/local/bin`. Some of those locations require a sudo password. If you don't have one, choose a directory to which you have write access.

Then type:

```
$ cd ${workspace}
$ git clone https://github.com/ska-sa/caracal.git
$ caracal/caratekit.sh -i
```

Follow the instructions.

Finally, type:

```
$ rm -rf ./caracal
```

To get a full display of the potential switches and usage of `caratekit.sh` type:

```
$ caratekit.sh -h
```

or

```
$ caratekit.sh -v
```

2.7.2 Updating `caratekit.sh`

We assume that you have chosen to make `caratekit.sh` visible in the `${PATH}`. Type:

```
$ caratekit.sh -i
```

and follow the instructions.

2.7.3 Installing and upgrading CARACal using `caratekit.sh` for further use with `caratekit.sh`

The syntax is the same for upgrading or installing CARACal. The user chooses:

- The location `${workspace}` of a parent directory to a `caratekit` test directory.
- A name `${caracal_testdir}` of the caracal test directory Installation/upgrade with Docker as containerisation technology:

```
$ caratekit.sh -ws ${workspace} -cr -di -ct ${caracal_testdir} -rp install -f
```

Installation/upgrade with Singularity as containerisation technology:

```
$ caratekit.sh -ws ${workspace} -cr -si -ct ${caracal_testdir} -rp install -f
```

Do not use -cr until the release. This installs the current release version of CARACal. Alternatively, the switch `-cr` can be omitted, to install the current master development branch (newest features but no guarantees). This creates the following directory structure:

```
${workspace}
└──${caracal_testdir}
   ├── caracal (local caracal copy)
```

(continues on next page)

(continued from previous page)

```
├── caracal_venv (virtualenv)
├── home (local home directory)
├── report (report directory)
│   └── install
│       ├── install.sh.txt (template shell script)
│       └── install-sysinfo.txt (system information file)
└── (stimela-singularity)
```

caracal is a local copy of CARACal (release branch **currently master branch**), which is installed using the `virtualenv caracal_venv`. `home` is a replacement of the `${HOME}` directory, in which currently only one hidden directory, `.stimela` is stored. This can be ignored in nearly all cases, but it is essential to have. The report directory contains automatically generated reports about `caratekit.sh` runs. The example run is called `install` (see switch `-rp`). This is reflected in a sub-directory named `install` in the report directory. This `caratekit.sh` run creates two report files, `install.sh.txt`, a bash-script re-tracing the installation steps (not documenting the creation of the reports), and `install-sysinfo.txt`, a file containing information about the system and the installed software of the machine that is being used. Generally, a `caratekit.sh` run can generate multiple report sub-directories, each of which can contain up to four files and one directory (see next section *Data reduction using caratekit.sh*).

2.7.4 Data reduction using caratekit.sh

Multiple variants are possible, here we present three.

The user uses the same - Workspace directory `${workspace}` as has been used to install `caratekit.sh` - The same target directory `${carate_target}` that `caratekit.sh` has been installed in.

The user chooses:

- The name `${project}` of the data reduction project
- The location `${configfile}.yml` of a CARACal configuration file. Templates can be found in the directory `${workspace}/${caracal_testdir}/caracal/caracal/sample_configurations`. A choice to start with is the file `minimalConfig.yml`.
- The name `${rawdata}` of a directory containing the measurement sets (which have to have the suffix `.ms`) that are supposed to be processed in the data reduction.

Specifying the data to be reduced

`caratekit.sh` assumes by default, that any measurement sets in the `${rawdata}` file are meant to be processed by the pipeline. A copy of the `${configfile}.yml` configuration file will be modified accordingly before the data reduction process is started. This can be changed by adding the switch `--copy-config-data` or `-cc` to the `caratekit.sh` calls below. In that case, only measurement sets with the data ids specified in the configuration file will be processed. Alternatively, the user can specify explicityly the ids (the names of the measurement sets removing the suffix `.ms`) using the switch `--copy-data-id ARG -ci ARG`, where `ARG` is a comma- separated list of the ids to use.

Start and single CARACal run

If the user assumes to run CARACal only once but also at the beginning of any other data reduction process the user edits the file `${configfile}.yml` following the CARACal description. Notice that using `caratekit.sh` the default is that the contents of the parameter `dataid` will be replaced to reflect the measurement sets found in the `${rawdata}` directory. This can be overridden by using the `caratekit.sh -kc` switch. A (partial) data reduction is then conducted following the command Docker:

```
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project} -cs $
↪{configfile}.yml -td ${rawdata}
```

Singularity:

```
$ caratekit.sh -ws ${workspace} -cd -si -ct ${caracal_testdir} -rp ${project} -cs $
↪{configfile}.yml -td ${rawdata}
```

After that, if everything has gone well, the directory tree of the `${project}` has the following structure:

```
${workspace}
└──${caracal_testdir}
    ├── caracal (local caracal copy)
    ├── caracal_venv (virtualenv)
    ├── home (local home directory)
    ├── ${project} (CARACal project directory)
    │   ├── input
    │   ├── ${configfile}.yml
    │   ├── msdir
    │   ├── output
    │   └── stimela_parameter_files
    ├── report (report directory)
    │   └── install
    │   │   ├── install.sh.txt (template shell script)
    │   │   └── install-sysinfo.txt (system information file)
    │   └── ${project}
    │       ├── ${project}-${configfile}-log-caracal.txt
    │       ├── ${project}-${configfile}.yml.txt
    │       ├── ${project}.sh.txt
    │       └── ${project}-sysinfo.txt
    └── (stimela-singularity)
```

With above settings, `caratekit.sh` copies the measurement sets found in `${rawdata}` into the newly created directory `${project}/msdir` (see switch `-md` for moving test data instead), and the CARACal configuration file `${configfile}` into the directory `${project}`, to then start CARACal using the `${configfile}` (`caracal -c ${configfile}`). It also creates a new sub-directory `${project}` to report. Apart from the shell script `${project}.sh.txt` and the system info file `${project}-sysinfo.txt`, this sub-directory contains a copy `${project}-${configfile}-log-caracal.txt` of the CARACal logfile and a copy `${project}-${configfile}.yml.txt` of the CARACal configuration file. Should the data reduction process be interrupted by an error, a further sub-directory `${project}-badlogs` to `${caracal_testdir}/report/${project}` is created containing all logfiles indicating an error (the logfiles are literally parsed for the expression "ERROR" and added if it is found).

Any bug report to the CARACal issue tracker* can be substantially improved by submitting the files in the specific `report` directory along with the issue.*

Follow-up steps to conduct a data reduction by method 1: renaming the project

If the data reduction is supposed to be conducted in several steps (e.g. giving the user the chance to inspect intermediate data products), there are two methods that can be used:

Using method 1, the user defines a set of consecutive configuration files, for simplicity `${configfile}_00`, `${configfile}_01`, ... , and a set of consecutive project names `${project}_00`, `${project}_01`, ... The individual names are the user's choice, but some logical structure is recommended.

By using the `-cf ${project}_jj -cf ${project}_ii` switches, the directory `${project}_ii` is

re-named into `${project}_jj` and a symbolic link `${project}_ii` is created pointing to directory `${project}_jj`. Then, the data reduction is re-started using the CARACal configuration file provided with switch `-cf`. The purpose of this is to maintain data reduction reports corresponding to the single data reduction steps, which would otherwise be overwritten.

E.g, invoking (Docker)

```
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project}_00 -cs $
↪{configfile}_00.yml -td ${rawdata}
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project}_01 -cf $
↪{project}_00 -cs ${configfile}_01.yml -td ${rawdata}
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project}_02 -cf $
↪{project}_01 -cs ${configfile}_02.yml -td ${rawdata}
...
```

or (Singularity)

```
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project}_00 -cs $
↪{configfile}_00.yml -td ${rawdata}
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project}_01 -cf $
↪{project}_00 -cs ${configfile}_01.yml -td ${rawdata}
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project}_02 -cf $
↪{project}_01 -cs ${configfile}_02.yml -td ${rawdata}
...
```

creates the directory structure:

```
${workspace}
└──${caracal_testdir}

    ├── ${project}_00 (link to ${project}_01)
    ├── ${project}_01 (link to ${project}_02)
    ├── ${project}_02
    │
    ├── report (report directory)
    │   ├── install
    │   │
    │   ├── ${project}_00
    │   │
    │   ├── ${project}_01
    │   │
    │   └── ${project}_02
    │
```

such that issues and reports can be tracked by the names.

Follow-up steps to a data reduction by method 2: using a middle name

Using method 2, the user defines a set of consecutive configuration files, for simplicity `${configfile}_a`, `${configfile}_b`, ... , and a set of consecutive medifixes (middle names), e.g. `00`, `01`, ... for report files. The medifixes will then be inserted in the report file names. If the user does not choose the same medifixes (as in the example), a set of report files with different names are created for each `caratekit.sh` call. The individual names are again the user's choice...

E.g, invoking

(Docker)

```
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project} -rm "00
↪" -cs ${configfile}_a.yml -td ${rawdata}
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project} -rm "01
↪" -cs ${configfile}_b.yml -td ${rawdata}
$ caratekit.sh -ws ${workspace} -cd -di -ct ${caracal_testdir} -rp ${project} -rm "02
↪" -cs ${configfile}_c.yml -td ${rawdata}
...
```

or (Singularity)

```
$ caratekit.sh -ws ${workspace} -cd -si -ct ${caracal_testdir} -rp ${project} -rm "00
↪" -cs ${configfile}_a.yml -td ${rawdata}
$ caratekit.sh -ws ${workspace} -cd -si -ct ${caracal_testdir} -rp ${project} -rm "01
↪" -cs ${configfile}_b.yml -td ${rawdata}
$ caratekit.sh -ws ${workspace} -cd -si -ct ${caracal_testdir} -rp ${project} -rm "02
↪" -cs ${configfile}_c.yml -td ${rawdata}
...
```

creates the directory structure:

```
${workspace}
└──${caracal_testdir}

    ├── ${project}
    │   ├── ${configfile}_a.yml
    │   ├── ${configfile}_b.yml
    │   ├── ${configfile}_c.yml
    │
    ├── report (report directory)
    │   ├── install
    │   ├── ${project}
    │   │   ├── ${project}-00.sh.txt
    │   │   ├── ${project}-00-sysinfo.txt
    │   │   ├── ${project}-00-${configfile}_a.yml.txt
    │   │   ├── ${project}-00-${configfile}_a-log-caracal.txt
    │   │   ├── ${project}-01.sh.txt
    │   │   ├── ${project}-01-sysinfo.txt
    │   │   ├── ${project}-01-${configfile}_b.yml.txt
    │   │   ├── ${project}-01-${configfile}_b-log-caracal.txt
    │   │
    │
```

such that issues and reports can be tracked, again, by the names. This method has the advantage that the ${project} directory keeps its name, but the disadvantage that the report directory might become large. We leave the choice to the user. Quickly installing CARACal using `caratekit.sh` ———————————————————— This method is described for the user who does not want to use `caratekit.sh` to reduce their data but want to use it only to install CARACal.

Download caratekit.sh . Choose the parent directory `${workspace}` and the name of the CARACal directory `${caracal_dir}`.

If using Docker:

```
$ caratekit.sh -ws ${workspace} -cr -di -ct ${caracal_dir} -rp install -f -kh
```

If using Singularity:

```
caratekit.sh -ws ${workspace} -cr -si -ct ${caracal_testdir} -rp install -f -kh
```

To run CARACal manually, activate the virtual environment with:

```
source ${workspace}/${caracal_dir}/caracal_venv/bin/activate
```

then type:

```
caracal - c ${your-configuration-file}
```

CHAPTER 3

Acknowledgements

3.1 Support and Funding

The CARACal team acknowledges support from the following institutes:

- South African Radio Astronomy Observatory (SARAO)
- Rhodes University
- Istituto Nazionale di Astrofisica (INAF) - Osservatorio Astronomico di Cagliari
- ASTRON
- Kapteyn Astronomical Institute
- Ruhr-Universitat Bochum

and from the following funding allocations:

- Starting Grant of the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant number 679629, project name FORNAX)
- Grant from the Italian Ministry of Foreign Affairs and International Cooperation (MAECI Grant Number ZA18GR02) and the South African Department of Science and Technology's National Research Foundation (DST-NRF Grant Number 113121) as part of the ISARP Joint Research Scheme.
- At RUB this work is partly supported by BMBF project 05A17PC2

3.2 CARACal publication policy

Please notice that users that have been using CARACal/MeerKATHI before its release should instead adhere to the Caracal publication Policy - V2.

3.2.1 Acknowledging CARACal

We request the usage of CARACal to be acknowledged:

In journal papers:

- using the following phrase: *"(Part of) the data published here have been reduced using the CARACal pipeline, partially supported by ERC Starting grant number 679629 "FORNAX", MAECI Grant Number ZA18GR02, DST-NRF Grant Number 113121 as part of the ISARP Joint Research Scheme, and BMBF project 05A17PC2 for D-MeerKAT. Information about CARACal can be obtained online under the URL: https://caracal.readthedocs.io"*
- and citing the following publication(s), bibtex see below: *Józsa, G. I. G., White, S. V., Thorat, K. , Smirnov, O. M., Serra, P., Ramatsoku, M., Ramaila, A. J. T., Perkins, S. J., Molnár, D. Cs., Makhathini, S., Maccagni, F. M., Kleiner, D., Kamphuis, P., Hugo, B. V., de Blok, W. J. G., Andati, L. A. L. 2020, ASPC, 527, 635*

In conference proceedings:

- citing above publication(s) and presenting the URL: https://caracal.readthedocs.io

On posters and on talk slides:

- presenting the *CARACal logo* and showing the URL: https://caracal.readthedocs.io

3.2.2 Sharing

We encourage users to send us a reference to their publication, to post pretty images and to share their data reduction strategies (a configuration file with an optional description) with other users. For that purpose we maintain the CARACal publications, pretty images, and data reduction strategies (PPD) table. Please submit any of the following, along with an expressive permission to make it publicly available on the internet, using the links below:

- A configuration (.yml) file of your CARACal data reduction
- An explanation of your CARACal data reduction strategy pursued in the configuration file (e.g. the data reduction section of your paper)
- A reference to your publication (partly) based on a CARACal data reduction
- A (link to a) pretty astronomical image (partly) based on your CARACal data reduction
- A (link to a) picture of you being happy about CARACal
- Anything else that you want to share

We will then make your contributions available to the community on the www in the PPD table.

3.2.3 BibTex entries:

@inproceedings{Jozsa2020, Address = {San Francisco}, Author = {G.~I.~G.~J'ozsa, S.~V.~White, K.~Thorat, O.~M.~Smirnov, P.~Serra, M.~Ramatsoku, A.~J.~T.~Ramaila, S.~J.~Perkins, D.~C.~Moln'{a}r, S.~Makhathini, F.~M.~Maccagni, D.~Kleiner, P.~Kamphuis, B.~V.~Hugo, W.~J.~G.~de~Blok, and L.~A.~L.~Andati}, Booktitle = {ADASS XXIX}, booktitle = {ADASS XXIX}, year = 2020, editor = {{Pizzo}, R. and {Deul}, E. and {Mol}, J.-D. and {de Plaa}, J. and

{Verkouter}, H}, volume = {527}, series = {ASP Conf. Ser.}, Pages = {635-638}, Title = {MeerKATHI - an end-to-end data reduction pipeline for MeerKAT and other radio telescopes}, }

3.3 Logos

Help us to make CARACal known to potential users by showing one of the logos below. We encourage users to use the rectangular logos with the software name, and to use high resolution, but also provide alternative, square logos that might be used as an alternative, and lower-resolution logos.

3.3.1 Rectangular logos

Rectangular CARACal logo, full resolution svg

Rectangular CARACal logo, high resolution

Rectangular CARACal logo, medium resolution

Rectangular CARACal logo, low resolution

3.3.2 Rectangular logos

Square CARACal logo, full resolution svg

Square CARACal logo, very low resolution

Square CARACal logo, tiny resolution