Caracal Documentation

Release latest

Paolo Serra

Jan 15, 2021
## Contents

1 Installation & Run 3
  1.1 Usage and publication policy ................................................... 3
  1.2 Requirements ............................................................................ 3
  1.3 Manual installation ..................................................................... 3
  1.4 Installation with the caratekit.sh script ........................................ 4
  1.5 Run .............................................................................................. 4
  1.6 Known and new issues ................................................................. 5

2 Manual 7
  2.1 Introduction ................................................................................. 7
  2.2 Configuration file ......................................................................... 9
  2.3 Data products ............................................................................... 10
  2.4 Data reduction ........................................................................... 11
  2.5 Workers parameters .................................................................... 21
  2.6 Software used by CARACal ......................................................... 96
  2.7 Caratekit utility ........................................................................... 97

3 Acknowledgements 103
  3.1 Team ......................................................................................... 103
  3.2 Support and Funding ................................................................. 104
  3.3 CARACal publication policy ....................................................... 104
  3.4 Logos ......................................................................................... 105
CARACal stands for Containerized Automated Radio Astronomy Calibration and 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 > 3.5 (nearly all functionality available for > 2.6.0-dist, see here for further information)
  - 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. Any name as ${name} occurring in the description below can be chosen arbitrarily. If it symbolises directories or files, those directories or files should exist and the user should have write access.

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

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

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

(continues on next page)
pip install -U pip setuptools wheel

# 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 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}. Any name as ${name} occurring in the description below can be chosen arbitrarily. If it symbolises directories or files, those directories or files should exist and the user should have write access.

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
```

If using Docker run CARACal with:
caracal -c ${your-configuration-file}

If using Singularity run CARACal with:

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

For more detailed installation instructions, trouble-shooting tips and a full user manual please see cara-
cal.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.
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 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](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:

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

2.4. Data reduction
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.

2.4. Data reduction
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:

- the msdir directory where to find/write .MS files (general: msdir) with the only exception of read-only input .MS files, which can be located at rawdatadir (see below);
- if necessary, the rawdatadir directory where to find the read-only input .MS files, which facilitates the processing of large files on shared machines without having to move them (general: rawdatadir);
- 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 directories msdir, input and output 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: Optionally, the input files can be untar-ed (getdata: untar).
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-calibration 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: KGBKAKGB results in:

- delay calibration (K);
- gain calibration (G) applying the initial 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, respectively.

The `crosscal` worker produces .HTML plots of the various calibration terms for later, interactive inspection. Furthermore, the `inspect` worker produces .PNG plots of the calibrators’ 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 ensemble 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: uvlind*). 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 *uvlind* 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 amplitude 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 = lgc1
Label for output files (calibration tables, images).

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 ‘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”).

tile_size

    int, optional, default = 128

    Size of tile (time bins) to process. Can be used to reduce memory footprint.

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).

**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 boundary), ‘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

**b_fillgaps**

*int, optional, default = 70*

Fill flagged channels in the bandpass solutions by interpolation.

**b_smoothwindow**

*int, optional, default = 1*

Size of the mean running window for smoothing of the bandpass (in channels). A size of 1 means no smoothing.

**spw_k**

*str, optional, default = ‘ ‘*

Only use this subset(s) of the band to compute ‘K’ gains. Default uses full band

**spw_g**

*str, optional, default = ‘ ‘*

Only use this subset(s) of the band to compute ‘GF’ gains. Default uses full band

**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

tfeq cutoff
- 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

2.5. Workers parameters
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 immediately 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

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

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

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 boundaries), ‘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

Normalise average solution amplitude to 1.0

b_fillgaps

Fill flagged channels in the bandpass solutions by interpolation.

b_smoothwindow

Size of the mean running window for smoothing of the bandpass (in channels). A size of 1 means no smoothing.

spw_k

Only use this subset(s) of the band to compute ‘K’ gains. Default uses full band
**spw_g**

*str, optional, default = ' '*

Only use this subset(s) of the band to compute ‘GF’ gains. Default uses full band

**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", timfreq"}, 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

**npi**

*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 channels 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 butcher’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 worker.

cabs

list of map, optional, default = ‘ ’

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

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.

**shared_mem**

`int, optional, default = 400`

Shared memory for tasks in units of GBs. Does not work with singularity.

**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).

decov_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.

decov_rmsfactor
float, optional, default = 0.0
Set the minor-cycle stopping-threshold to X*[residual RMS], where X is this parameter value.

decov_peakfactor
Caracal Documentation, Release latest

float, optional, default = 0.25
Set the minor-cycle stopping-threshold to X*{peak residual}, where X is this parameter value.

defcon_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.

defcon_gain
float, optional, default = 0.1
Gain setting for the deconvolution loops.

defcon_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.

defcon_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_colname_postcal

str, optional, default = SUBDD_DATA

Data column to use for imaging after dd-calibration. Defaults to ‘SUBDD_DATA’, so as to not overwrite the corrected data. If data size increase is a concern, switch to ‘CORRECTED_DATA’

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
Caracal Documentation, Release latest

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**

2.5. Workers parameters
int, optional, default = 10
Time solution interval in timeslot units for G-Jones matrix.

\texttt{dd\_g\_chan\_int}
int, optional, default = 0
Frequency solution interval in channel units for G-Jones matrix.

\texttt{dd\_dd\_timeslots\_int}
int, optional, default = 100
Time solution interval in timeslot units for DE-Jones matrix.

\texttt{dd\_dd\_chan\_int}
int, optional, default = 100
Frequency solution interval in channel units for DE-Jones matrix.

\texttt{dist\_nworker}
int, optional, default = 0
Number of processes.

\texttt{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.

\texttt{image\_ws\_clean}
WSClean imaging parameters for ddcal worker.

enable
bool, optional, default = True
Enable WSClean imaging of the DD-calibrated data.

\texttt{img\_ws\_npix}
int, optional, default = 1800
Number of pixels in output image.

\texttt{img\_ws\_padding}
float, optional, default = 1.3
Padding in WSClean.

\texttt{img\_ws\_mgain}
float, optional, default = 0.90
Gain for the major cycle during image CLEANing.
**img_ws_cell**

*float, optional, default = 2.*


**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 = 0.0*

Briggs robust value.

**img_ws_uvtaper**

*str, optional, default = 0*


**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 = 0, 5, 10, 20
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_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_within**

```plaintext
str, optional, default = ' '  
Give JS9 region file. Only sources within those regions will be included.
```

**dd_points_only**

```plaintext
bool, optional, default = False  
Select only ‘point’ sources.
```

**dd_num_sources**

```plaintext
int, optional, default = 0  
Select only N brightest sources.
```

**dd_num_workers**

```plaintext
int, optional, default = 0  
Explicitly set the number of worker threads. Default is 0, meaning it uses all threads.
```

**dd_mem_frac**

```plaintext
float, optional, default = 0.5  
Fraction of system RAM that can be used. Used when setting automatically the chunk size.
```

**report**

```plaintext
bool, optional, default = False  
(Re)generate a full HTML report at the end of this worker.
```

**cabs**

```plaintext
list of map, optional, default = ' '  
Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.
```

### 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**

```plaintext
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 ‘calfields’ 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
Caracal Documentation, Release latest

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 = ‘’


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.

2.5. Workers parameters 43
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 = ‘ ’

   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
   The mask to apply. This can be provided by the user, but CARACaI also provides an existing static mask for MeerKAT, specify ‘meerkat.rfimask.npy’ to use it.

   uvrange
str, optional, default = ‘ ‘
Select range in UV-distance (CASA-style range, e.g. ‘lower-upper’) for flagging. This is in units of metres. Leaving this parameter blank will select the entire range in UV-distance.

flag_manual

Manually flag subsets of data, using a syntax based on the CASA flagdata task

enable

bool, optional, default = False
Enable the ‘flag_manual’ segment.

rules

list of str, optional, default = ‘ ‘
Sequence of flagging rules, of the form “ms_pattern key:value key:value”

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.

readmode

2.5. Workers parameters
AOflagger read mode. If set to ‘indirect’, AOflagger temporarily writes a reordered .MS file to disc, which results in fast flagging but requires free disc space. If set to ‘memory’, AOflagger reads the .MS file into memory, which is even faster than ‘indirect’ but is impossible for large files. If set to ‘auto’, AOflagger will decide between the ‘memory’ mode and the ‘direct’ mode – the slowest mode – in which AOFlagger reads baselines by scanning the entire file for the data relevant for the currently required baseline.

tricolour

backend

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

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

Name of the Tricolour strategy file.

strat_narrow

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

tfcrop

usewindowstats

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

Dimension(s) along which to calculate a fit(fits) to the data. Options are ‘freq’, ‘time’, ‘freetime’, and ‘timefreq’. Note that the differences between ‘freetime’ and ‘timefreq’ are only apparent if RFI in one dimension is significantly 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”, “YX”, “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**
Caracal Documentation, Release latest

\[
\text{int, optional, default} = 5
\]

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**

\[
\text{bool, optional, default} = \text{True}
\]

Enable the ‘summary’ segment.

**report**

\[
\text{bool, optional, default} = \text{False}
\]

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

**cabs**

\[
\text{list of map, optional, default} = '\ ' '\
\]

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

### 2.5.4 general

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

**title**

\[
\text{str, optional, default} = '\ ' '\
\]

An optional project title.

**msdir**

\[
\text{str, optional, default} = \text{msdir}
\]

Location where CARACal will write and expect to find .MS files. The only exception is that of read-only input .MS files, which can be located in rawdatadir (see below).

**rawdatadir**

\[
\text{str, optional, default} = '\ ' '\
\]

If set to an empty string this parameter is ignored. If not set to an empty string, this is the directory where CARACal expects to find the input .MS files. This directory and the input .MS files within it can be reado-only, which makes it possible to work on large data without moving them within a shared machine. Any .MS file further created by CARACal is still written to msdir (see above).
input

\[ \text{str, optional, default = input} \]
Location where CARACal expects to find various input files (e.g., RFI flagging strategy files).

output

\[ \text{str, optional, default = output} \]
Location where CARACal writes output products.

prefix

\[ \text{str, optional, default = caracal} \]
Prefix for CARACal output products.

prep_workspace

\[ \text{bool, optional, default = True} \]
Initialise the pipeline by copying input files (i.e., those that are MeerKAT specific, flagging strategies, beam model, etc.).

init_notebooks

\[ \text{list of str, optional, default = std-progress-report, project-logs} \]
Install standard radiopadre notebooks, given by list of basenames.

report_notebooks

\[ \text{list of str, optional, default = detailed-final-report} \]
Like init_notebooks, but will also be automatically rendered to HTML when report=True in a worker.

final_report

\[ \text{bool, optional, default = False} \]
Render report_notebooks to HTML at the end of each pipeline run.

cabs

\[ \text{list of map, optional, default = ‘ ’} \]
Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.
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.

extension

str, optional, default = ms

Extension of raw (input) visibility data

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 worker.

ignore_missing

bool, optional, default = False

Ignore missing matches/files in the dataid list and proceed with the files that were found. If none can be found, an exception will be raised regardless.

cabs

list of map, optional, default = ‘ ‘

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.
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

**field**

`str`, optional, default = `calibrators`

Fields that should be inspected. It can be set to ‘target’, ‘calibrators’ (i.e., all calibrators) or any comma-separated combination of ‘fcal’, ‘bpcal’, ‘gcal’, as defined in the obsconf worker. N$

**label_plot**

`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 each field

2.5. Workers parameters
plots_by_corr

- list of str, optional, default = ‘ ‘
  - Sequence of shadems plot specifications, made per each correlation

plots

- list of str, optional, default = ‘ ‘
  - Sequence of freeform shadems plot specifications

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 = diag
  - Label(s) specifying the correlations. Use the special values ‘diag’ and ‘all’ to select only diagonal (paralell hand) or all 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’.

real_imag

- Plot real vs imaginary parts of data.

enable

- bool, optional, default = False
  - Executed the real v/s imaginary data plotting.
col
  str, optional, default = corrected
  Data column to plot.

avgttime
  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.

  col
    str, optional, default = corrected
    Data column to plot.

  avgttime
    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 = False
    Executes plotting data amplitude as a function of uvwave.

  col
    str, optional, default = corrected
    Data column to plot.

  avgttime
    str, optional, default = 10
    Time to average for plotting, in seconds.
**avgchan**

\[str,\text{optional, default }= 10\]

Number of channels to average for plotting.

**amp_ant**

Plot data amplitude v/s antenna.

**enable**

\[bool,\text{optional, default }= \text{False}\]

Executes plotting data amplitude v/s antennas.

**col**

\[str,\text{optional, default }= \text{corrected}\]

Data column to plot.

**avgtime**

\[str,\text{optional, default }= 10\]

Time to average for plotting, in seconds.

**avgchan**

\[str,\text{optional, default }= 10\]

Number of channels to average for plotting.

**phase_uvwave**

Plot data phase v/s uvwave.

**enable**

\[bool,\text{optional, default }= \text{False}\]

Executes plotting data phase v/s uvwave.

**col**

\[str,\text{optional, default }= \text{corrected}\]

Data column to plot.

**avgtime**

\[str,\text{optional, default }= 10\]

Time to average for plotting, in seconds.

**avgchan**

\[str,\text{optional, default }= 10\]

Number of channels to average for plotting.
**amp_scan**

Plot data amplitude v/s scan number.

**enable**

```python
bool, optional, default = False
```

Executes plotting data amplitude v/s scan number.

**col**

```python
str, optional, default = corrected
```

Data column to plot.

**avgt ime**

```python
str, optional, default = 10
```

Time to average for plotting, in seconds.

**avgchan**

```python
str, optional, default = 10
```

Number of channels to average for plotting.

**amp_chan**

Plot Amplitude vs Channel data.

**enable**

```python
bool, optional, default = False
```

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

**col**

```python
str, optional, default = corrected
```

Data column to plot.

**avgt ime**

```python
str, optional, default = 10
```

Time to average for plotting, in seconds.

**avgchan**

```python
str, optional, default = 10
```

Number of channels to average for plotting.

**phase_chan**

Plot Phase vs Chan.

**enable**

```python
bool, optional, default = False
```

Executes the plotting of amplitude v/s phase for data.
\[ \text{col} \]
\[ \text{str, optional, default = corrected} \]
Data column to plot.

\[ \text{avgtime} \]
\[ \text{str, optional, default = 10} \]
Time to average for plotting, in seconds.

\[ \text{avgchan} \]
\[ \text{str, optional, default = 10} \]
Number of channels to average for plotting.

\[ \text{report} \]
\[ \text{bool, optional, default = False} \]
(Re)generate a full HTML report at the end of this worker.

\[ \text{cabs} \]
\[ \text{list of map, optional, default = ' '} \]
Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

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

\[ \text{enable} \]
\[ \text{bool} \]
Execute the line worker.

\[ \text{label_in} \]
\[ \text{str, optional, default = corr} \]
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).

\[ \text{line_name} \]
\[ \text{str, optional, default = HI} \]
Suffix to be used for the name of the output files (data cubes etc).
restfreq

str, optional, default = 1.420405752GHz
Spectral line rest frequency.

ncpu

int, optional, default = 0
Number of CPUs to use for distributed processing. If set to 0 all available CPUs are used. This parameter is currently only passed on to WSClean for line imaging.

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 ‘workename’ 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 ‘workename’ 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.

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

frame

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

veltype

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

changrid

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

Enable the ‘UVLIN’ segment.

fitorder

Polynomial order of the continuum fit.

fitspw

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.

exclude_known_sources

Exclude from the UVLIN fit the channels corresponding to known line sources listed in a catalogue. The catalogue file has the name given by the parameter ‘known_sources_cat’ below and is located in the ‘input’ directory specified in the ‘general’ worker. The resulting channel selection is combined with the one provided by the ‘fitspw’ parameter above. Some published catalogues are included in the CARACal repository and are ready for use. See ‘know_sources_cat’ below.

known_sources_cat
Catalogue of known line sources. The catalogue is in ASCII format, one row per source, with columns (1) source ID, (2) RA (hh:mm:ss.s), (3) Dec (dd:mm:ss.s), (4) Vmin (km/s, optical convention), (5) Vmax (km/s, optical convention), (6) line flux (Jy km/s). The HIPASS catalogue from Meyer et al. (2004), MNRAS, 350, 1195 is included in CARACal with the required format (file name hicat_caracal.txt).

**known_sources_radius**

*float, optional, default = 1.0*

Only line sources within this radius (in deg) from the pointing centre are excluded from the UVLIN fit.

**known_sources_flux**

*float, optional, default = 0.0*

Only line sources brighter than this flux (in Jy km/s) are excluded from the UVLIN fit (no primary beam correction included).

**known_sources_dv**

*float, optional, default = 30.*

Remove (add) this velocity buffer from (to) the Vmin (Vmax) values in the catalogue to avoid errors caused by anoaccounted-for Doppler shifts. This parameter is given in km/s.

**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.

**readmode**

*{"indirect", "memory", "auto"}, optional, default = auto*

AOFlagger read mode. If set to ‘indirect’, AOFlagger temporarily writes a reordered .MS file to disc, which results in fast flagging but requires free disc space. If set to ‘memory’, AOFlagger reads the .MS file into memory, which is even faster than ‘indirect’ but is impossible for large files. If set to ‘auto’, AOFlagger will decide between the ‘memory’ mode and the ‘direct’ mode – the slowest mode – in which AOFlagger reads baselines by scanning the entire file for the data relevant for the currently required baseline.
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).

predict_noise

Print to log-caracal.txt the expected natural noise level of the line cube (Stokes I, single channel) based on Tsys/eff and dish diameter below.

**enable**
bool, optional, default = False

Enable the ‘predict_noise’ segment.

tsys\text{eff}

float, optional, default = 20.5

Value of Tsys/eff in K.

diam

float, optional, default = 13.5

Dish diameter in m.

\textbf{make\_cube}

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

\textbf{enable}

bool, optional, default = True

Enable the ‘make\_cube’ segment.

\textbf{image\_with}

[“ws\text{clean}”, “casa”], optional, default = wsc\text{lean}

Choose whether to image with WSClean + SoFiA (‘ws\text{clean}’) or with CASA Clean (‘casa’).

\textbf{use\_mstr\text{transform}}

bool, optional, default = True

Image the .MS file(s) produced by the mstr\text{transform} section of this worker instead of the input .MS file(s).

\textbf{stokes}

str, optional, default = I

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

\textbf{spwid}

int, optional, default = 0

Spectral window to use.

\textbf{nchans}

int, optional, default = 0

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

\textbf{firstchan}

int, optional, default = 0

First channel of the line cube.

\textbf{bin\text{chans}}

int, optional, default = 1

Integer binning of channels.
npix

\textit{seq, optional, default = 900, 900}

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

\textbf{cell}

\textit{float, optional, default = 2}

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

\textbf{padding}

\textit{float, optional, default = 1.2}

Images have initial size padding*\textit{npix}, and are later trimmed to the image size set via the ‘npix’ parameter.

\textbf{weight}

\{“natural”, “uniform”, “briggs”\}, \textit{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.

\textbf{robust}

\textit{float, optional, default = 0}

Robust parameter in case of Briggs weighting.

\textbf{taper}

\textit{float, optional, default = 0}

Gaussian taper FWHM in arcsec. Zero means no tapering.

\textbf{niter}

\textit{int, optional, default = 1000000}

Maximum number of clean iterations to perform.

\textbf{gain}

\textit{float, optional, default = 0.1}

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

\textbf{wscl_mgain}

\textit{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.

\textbf{wscl_sofia_niter}

\textit{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.
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
str, optional, default = ' '
Comma-separated integer scales for multiscale cleaning in pixels. If set to an empty string
WSClean selects the scales automatically. These include the 0 scale, a scale calculated based
on the beam size, and all scales obtained increasing the scale by a factor of 2 until the image
size is reached.
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.

wscl_nrdeconvsubimg

int, optional, default = 0

Speed-up deconvolution by splitting each channel into a number of subimages, which are deconvolved in parallel. This parameter sets the number of subimages as follows. If set to 1 no parallel deconvolution is performed. If set to 0 the number of subimages is the same as the number of CPUs used by the line worker (see “ncpu” parameter above). If set to a number > 1, the number of subimages is greater than or equal to the one requested by the user.

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.

pb_type

{“gauss”, “mauch”}, optional, default = gauss

Choose between a Gaussian (gauss) primary beam model or the MeerKAT Mauch et al. (2020) model (mauch).

dish_size
\[ \text{float, optional, default} = 13.5 \]

Dish diameter in meters. Only used in the Gaussian primary beam model.

**cutoff**

\[ \text{float, optional, default} = 0.1 \]

Primary beam pixels below this value are set to NaN.

**freq_to_vel**

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

**enable**

\[ \text{bool, optional, default} = \text{False} \]

Enable the ‘freq_to_vel’ segment.

**reverse**

\[ \text{bool, optional, default} = \text{False} \]

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

**sofia**

Run SoFiA source-finder on the final HI cubes to produce a detection mask, moment images and catalogues. Note that these settings are not used to make clean masks.

**enable**

\[ \text{bool, optional, default} = \text{True} \]

Enable the ‘sofia’ segment.

**flag**

\[ \text{bool, optional, default} = \text{False} \]

Use flag regions?

**flagregion**

\[ \text{list of int, optional, default} = 0, 0, 0, 0, 0, 0 \]

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

**rmsMode**

\[ \text{str, optional, default} = \text{mad} \]

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

**thr**

\[ \text{float, optional, default} = 4.0 \]

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 worker.

cabs

`list of map, optional, default = ‘ ‘`

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

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.
[“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**

\[ \text{int, optional, default } = 101 \]

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**

\[ \text{bool, optional, default } = \text{False} \]

Execute segment ‘merge_with_extended’.

**extended_source_map**

\[ \text{str, optional, default } = \text{extended_mask.fits} \]

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

**mask_method**

\[ \{ \text{"thresh", "sofia"}, \text{optional, default } = \text{thresh} \}\]

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

**thr_lev**

\[ \text{float, optional, default } = 8e-2 \]

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**

\[ \text{bool, optional, default } = \text{False} \]

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

**cabs**

\[ \text{list of map, optional, default } = \text{' '} \]

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

### 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**

\[ \text{bool} \]

Execute the mosaic worker.
mosaic_type

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

target_images

list of str, optional, default = ‘ ’
List of .FITS images/cubes to be mosaicked. Their file names must end with “image.fits” and must include the path relative to the current working directory. These images/cubes MUST be located within (sub-directories of) the current working directory.

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.

pb_type

{"gaussian", "mauchian"}, optional, default = gaussian
If no continuum pb.fits are already in place, user needs to choose whether a rudimentary primary beam is created (‘gaussian’) or one that follows the model of Mauch et al. (2020), relevant for MeerKAT data (‘mauchian’).

dish_diameter

float, optional, default = 13.5
If ‘pb_type’ has been set to ‘gaussian’, user needs to specify the dish diameter (in units of m).

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 primary beams can be created.

round_cdelt3

int, optional, default = 0
If mosaic_type is “spectral”, round CDELT3 in the header of the input cubes to the number of decimal digits given by this parameter (0 means do not round). This is useful when the CDELT3 values of the input cubes are not identical (which would make the mosaicking algorithm crash) but the differences are small and can be ignored. Note that the CDELT3 values of the input cubes are overwritten with the common, rounded value (if the rounding is sufficient to find a common value).

report

bool, optional, default = False
(Re)generate a full HTML report at the end of this worker.

cabs

list of map, optional, default = ‘ ‘
Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

2.5.10 obsconf

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

Get observation information.

enable

 bool

   Enable segment ‘obsinfo’.

listobs

 bool, optional, default = True

 Run the CASA ‘listobs’ task to write the observation information to an TXT file. This is not executed if the TXT file already exists.

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. This is not executed if the JSON file already exists.

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’. This is not executed if the elevation plots already exist.

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, optional, default = auto*

The reference antenna, which can be identified by an antenna name or number. Default is ‘auto’. i.e. refant is selected automatically based on the parameters ‘maxdist’ and ‘minbase’ below.

maxdist

*float, optional, default = 1000*

If ‘refant’ is set to ‘auto’ above, this sets the maximum distance the reference antenna should be from the array centre. Units are in ‘m’.

minbase

*float, optional, default = 150*

If ‘refant’ is set to ‘auto’ above, this sets the minimum baseline length required for a reference antenna. Units are in ‘m’.

report

*bool, optional, default = False*

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

list of map, optional, default = ‘ ‘

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

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’.

tol

float, optional, default = 360.0

Tolerance (in arcseconds) for matching calibrator to database names.

tol_diff

float, optional, default = 1.0

Tolerance (in arcseconds) for checking if coordinates differ from those in the database so much that rephasing is required. Please keep the value less than ‘tol’.
fixuvw

Fix the UVW coordinates through the CASA task ‘fixvis’.

enable

bool, optional, default = False

Enable the ‘fixvis’ segment.

fixcalcoords

Fix the coordinates of the bandpass calibrators.

enable

bool, optional, default = False

Enable fixing calibrator coordinate fixing 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 exist 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 worker.

cabs

```
list of map, optional, default = ‘ ‘
```

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

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 = 0

Number of CPUs to use for distributed processing. If set to 0 all available CPUs are used. This parameter
is passed on to the following software in the selfcal worker, WSClean for imaging, Cubical and MeqTrees
for calibration, PyBDSF for source finding.

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_mfs_weighting**

*bool, optional, default = false*

Enables MF weighting. Default is enabled.

**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.
Caracal Documentation, Release latest

**img_nmiter**

```plaintext
int, optional, default = 0
Number of major cycles.
```

**img_cleanborder**

```plaintext
float, optional, default = 1.3
Clean border.
```

**img_nchans**

```plaintext
int, optional, default = 3
Number of channels in output image.
```

**img_joinchans**

```plaintext
bool, optional, default = True
Join channels to create MFS image.
```

**img_specfit_nrcoeff**

```plaintext
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. Use 0 to disable spectral fitting. Note that spectral fitting is required if you want to do subsequent continuum subtraction using crystalball.
```

**img_stokes**

```plaintext
{“I”}, optional, default = I
Stokes image to create. For this first release of CARACal, the only option is ‘I’.
```

**img_multiscale**

```plaintext
bool, optional, default = False
Switch on multiscale cleaning.
```

**img_multiscale_scales**

```plaintext
str, optional, default = ‘
Comma-separated integer scales for multiscale cleaning in pixels. If set to an empty string WSClean selects the scales automatically. These include the 0 scale, a scale calculated based on the beam size, and all scales obtained increasing the scale by a factor of 2 until the image size is reached.
```
**img_nrdeconvsubimg**

*int, optional, default = 0*

Speed-up deconvolution by splitting the image into a number of subimages, which are deconvolved in parallel. This parameter sets the number of subimages as follows. If set to 1 no parallel deconvolution is performed. If set to 0 the number of subimages is the same as the number of CPUs used by the selfcal worker (see “ncpu” parameter above). If set to a number > 1 , the number of subimages is greater than or equal to the one requested by the user.

**img_nwlayers_factor**

*int, optional, default = 3*

Use automatic calculation of the number of w-layers, but multiple that number by the given factor. This can e.g. be useful for increasing w-accuracy. In practice, if there are more cores available than the number of w-layers asked for then the number of w-layers used will equal the number of cores available.

**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**
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.
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
Method used to create the clean mask. The possible values are 1) ‘wsclean’ to use WS-Clean’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 sofi_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 masking method 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.

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.

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).

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’.

Number of threads used by wsclean; has a default value of ‘0’. If specified in the configuration file, will overrule the value set by ncpu, which is the global default for both cubical and wsclean.

Specifies a fixed amount of memory in gigabytes.

Source-finding parameters.

Enable the ‘extract_sources’ segment.
Set the source finder to be used. Options are ‘pybds’ (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 = 0.05
```

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 worker.

**cabs**

list of map, optional, default = ‘ ’

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

**2.5.13 transform**

Split, average and/or calibrate the data.

**enable**

bool

Execute the transform worker.

**field**

str, optional, default = target

Fields to be split off from the input MS (see split_field below) or whose phase centre should be changed (see changecentre below). 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’).

**label_in**

str, optional, default = ‘ ’

Label of the input dataset. If split_field/changecentre below are enabled, label_in must be a single value. Else, if concat below is enabled, label_in must include at least two comma-separated values.

**label_out**

str, optional, default = corr

Label of the output dataset.

**rewind_flags**

Rewind flags to specified version. Only works with a single-valued label_in (i.e., it does not loop over multiple comma-separated values in label_in).

**enable**
```python
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.

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.

scan

str, optional, default = ‘’

Select the scan(s), e.g. ‘0,2,5’. Setting this to ‘’ means that all scans 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.

### concat

For each input .MS and target, concatenate together all .MS files with the different labels included in `label_in` (comma-separated values). The MS files are concatenated along the frequency axis. This only works if the frequency ranges covered by the MS files are contiguous and if the number of channels is the same for all MS files.

**enable**

```
bool, optional, default = False
```

Enable the ‘concat’ segment.

### 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 worker.

cabs

list of map, optional, default = ‘’

Specifies non-default image versions and/or tags for Stimela cabs. Running with scissors: use with extreme caution.

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:

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

Follow the instructions.

Finally, type:

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

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

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

or

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

### 2.7.2 Updating caratekit.sh

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

```bash
$ 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 \( \text{workspace} \) of a parent directory to a caratekit test directory.
- A name \( \text{caracal_testdir} \) of the caracal test directory

Installation/upgrade with Docker as containerisation technology:

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

Installation/upgrade with Singularity as containerisation technology:

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

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:

```
$\text{workspace}$

|--$\text{caracal_testdir}$
  |--\text{caracal} (local caracal copy)
  |   |--\text{caracal_venv} (virtualenv)
  |   |--\text{home} (local home directory)
  |   |--\text{report} (report directory)
  |       |--\text{install}
  |           |--\text{install.sh.txt} (template shell script)
  |           |--\text{install-sysinfo.txt} (system information file)
  |               (stimela-singularity)
```

caracal is a local copy of CARACal (release branch \textbf{currently master branch}), which is installed using the virtualenv caracal\_venv. home is a replacement of the \( \text{HOME} \) 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 \text{caratekit.sh} runs. The example run is called \text{install} (see switch \(-rp\)). This is reflected in a sub-directory named \text{install} in the report directory. This \text{caratekit.sh} run creates two report files, \text{install.sh.txt}, a bash-script re-tracing the installation steps (not documenting the creation of the reports), and \text{install-sysinfo.txt}, a file containing information about the system and the installed software of the machine that is being used. Generally, a \text{caratekit.sh} run can generate multiple report sub-directories, each of which can contain up to four files and one directory (see next section \textit{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 \( \text{workspace} \) as has been used to install \text{caratekit.sh} - The same target directory \( \text{carate_target} \) that caratekit.sh has been installed in.

The user chooses:

- The name \( \text{project} \) of the data reduction project
- The location \( \text{configfile}.yml \) of a CARACal configuration file. Templates can be found in the directory \( \text{workspace}/\text{caracal_testdir}/\text{caracal/carcal/sample_configurations} \). A choice to start with is the file \text{minimalConfig.yml}. 

• The name \( \text{${rawdata}$} \) of a directory containing the measurement sets (which have to have the suffix \( \text{.ms} \)) 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 \( \text{${rawdata}$} \) file are meant to be processed by the pipeline. A copy of the \( \text{${configfile}.yml} \) configuration file will be modified accordingly before the data reduction process is started. This can be changed by adding the switch \( --\text{copy-config-data or --cc} \) 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 explicitly the ids (the names of the measurement sets removing the suffix \( \text{.ms} \)) using the switch \( --\text{copy-data-id ARG -ci ARG} \), 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 \( \text{${configfile}.yml} \) 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 \( \text{${rawdata}$} \) directory. This can be overridden by using the caratekit.sh \( -\text{kc} \) switch. A (partial) data reduction is then conducted following the command Docker:

```
$ caratekit.sh -ws \( \text{${workspace}$} \) -cd -di -ct \( \text{${caracal_testdir}$} \) -rp \( \text{${project}$} \) -cs $ \rightarrow \text{${configfile}.yml} \) -td \( \text{${rawdata}$} \)
```

Singularity:

```
$ caratekit.sh -ws \( \text{${workspace}$} \) -cd -si -ct \( \text{${caracal_testdir}$} \) -rp \( \text{${project}$} \) -cs $ \rightarrow \text{${rawdata}$} 
```

After that, if everything has gone well, the directory tree of the \( \text{${project}$} \) has the following structure:

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

With above settings, caratekit.sh copies the measurement sets found in \( \text{${rawdata}$} \) into the newly created directory \( \text{${project}$/msdir} \) (see switch \( -\text{md} \) for moving test data instead), and the CARACal configu-
ration 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}.yaml.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)
```

(continues on next page)
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)

```bash
$ 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)

```bash
$ 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:

```plaintext
${workspace}
    ${caracal_testdir}
        ${project}
            ${configfile}_a.yml
            ${configfile}_b.yml
            ${configfile}_c.yml
        report (report directory)
            install
            ${project}
```

(continues on next page)
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}
```
3.1 Team

The CARACal team includes the following individuals (listed alphabetically), who’ve all contributed in a valuable and significant way to various modules of CARACal at various times during its development.

- Lexy Andati
- Erwin de Blok
- Benjamin Hugo
- Gyula Jozsa
- Peter Kamphuis
- Dane Kleiner
- Filippo Maccagni
- Sphesihle Makhathini
- Daniel Molnar
- Simon Perkins
- Athaneseus Ramaila
- Mpati Ramatsoku
- Paolo Serra
- Oleg Smirnov
- Kshitij Thorat
- Sarah White
3.2 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-Universität 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 679627, 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 1113121) as part of the ISARP Joint Research Scheme.
- At RUB this work is partly supported by BMBF project 05A17PC2

3.3 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.

3.3.1 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 679627 “FORNAX”, MAECI Grant Number ZA18GR02, DST-NRF Grant Number 1113121 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”


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

### 3.3.2 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 express 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.

### 3.3.3 BibTeX entries:

```
@inproceedings{Jozsa2020,  
Address = {San Francisco},  
Booktitle = {ADASS XXIX},  
booktitle = {ADASS XXIX},  
year = 2020,  
volume = {527},  
series = {ASP Conf. Ser.},  
Pages = {635-638},  
Title = {MeerKATHI - an end-to-end data reduction pipeline for MeerKAT and other radio telescopes},  
}
```

### 3.4 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.

#### 3.4.1 Rectangular logos

Rectangular CARACal logo, full resolution svg

Rectangular CARACal logo, high resolution
3.4.2 Rectangular logos

Square CARACal logo, full resolution svg

Square CARACal logo, very low resolution

Square CARACal logo, tiny resolution