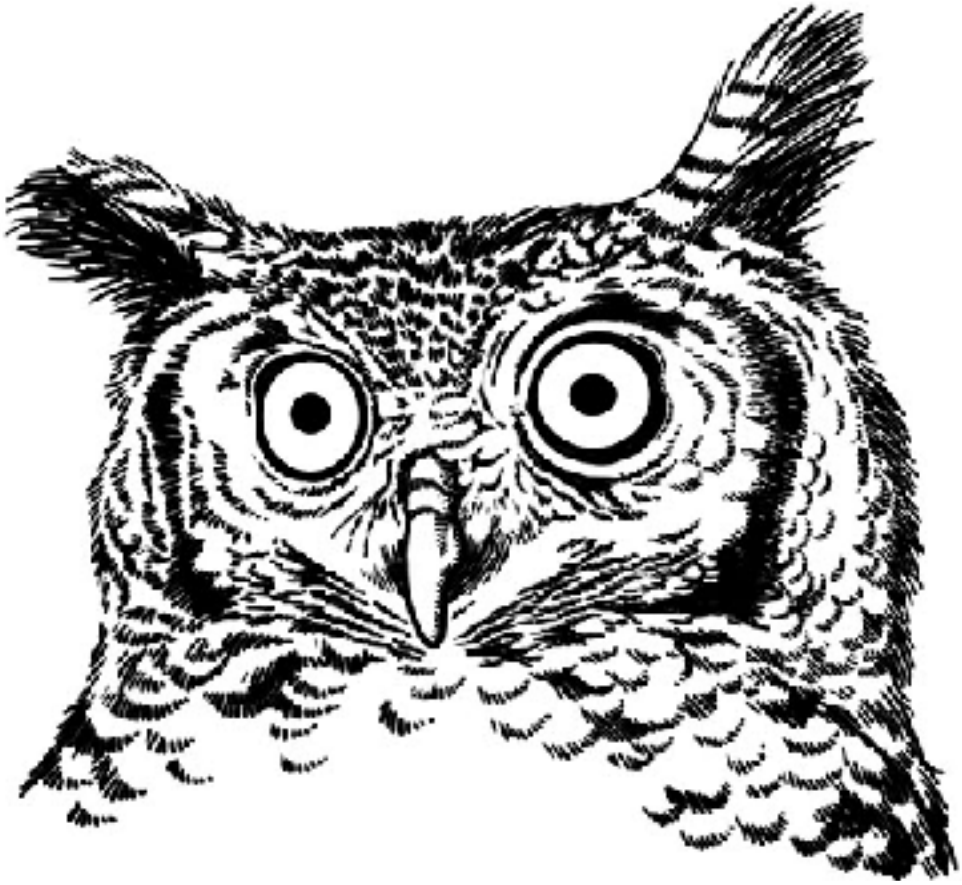
1st Edition



RTS

IN A NUTSHELL

A Desktop Quick Reference

RTS Processing Pipeline in a Nutshell

Installation

The RTS processing pipeline should be stored in the users local bin path:

    $HOME/bin/rtsprocess.py

The local environment should be set up as per the sample .bashrc shown at the end of this document - this ensures that paths are set correctly and the appropriate modules are loaded.

A default global sky catalogue should be made available in $RTS_CAT_PATH with the name rtscat.txt. Alternate catalogues can also be stored here and accessed via the pipeline.

The Basic RTS workflow

The typical work flow involves three stages:

- Download the visibility data from the archive
- Calibrate the visibility data
- Image the visibility data

The RTS typically works on a per-snapshot basis and identifies snapshots by their obsid (sometimes referred to as a gpsid). To determine which snapshots to download, refer to the MWA archive (http://mwa-metadata01.pawsey.org.au).

Within a snapshot, the RTS processes coarse channels (1.28 MHz channels) on separate CPU or GPU nodes. Thus when processing the entire band a total of 24 + 1 nodes is required (the additional node is for the master).

Working with slurm

slurm is the queuing system on the galaxy cluster at pawsey.

sbatch script - used to submit a job to the queue.

sbatch —dependency=afterany:jobid script - used to submit a job to the queue, the job will only run after the job with the specified jobid is complete. This is useful for setting up a download-calibrate-image workflow where each step much complete before the next step is initiated.

squeue - used to list jobs submitted to the queue. This is useful to see if a job has started and to see what job id has been assigned to a submitted job.

scancel jobid - used to cancel a job.

Downloading data from the archive

The following is a sample shell script to start the download process (qload.sh):

```
#!/bin/bash -l
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
#SBATCH --time=12:00:00
#SBATCH --partition=workq
#SBATCH --account=mwasci
#SBATCH --export=NONE
```

```
source /scratch2/mwaeor/bpindor/MWA_Python/bin/activate
rm load.log
rtsprocess.py load.in
```

Note that loading data only requires a single core. The RTS load script (load.in) has the following format:

```
FetchData: 1166109584
Reflag: 1166109584 1.3
#
FetchData: 1166109704
Reflag: 1166109704 1.3
```

The FetchData command requests the pipeline to download the data for the specified obsid. The Reflag command decompresses the Cotter flag files for the specified obsid, checks the flagging occupancy for each channel and flags any channels that have increased levels of flagging. The value specified after the obsid indicates a factor above the median number visibilities per channel flagged above which a channel will be entirely flagged.

To initiate the download process simply submit to the queue as follows:

```
sbatch qload.sh
```

Calibration

The following is a sample shell script to start the calibration process (qcal.sh):

```
#!/bin/bash -l
#SBATCH --nodes=25
#SBATCH --ntasks-per-node=1
#SBATCH --time=12:00:00
#SBATCH --partition=gpuq
#SBATCH --account=mwasci
#SBATCH --export=NONE
rm cal.log
rtsprocess.py cal.in
```

Note that the calibration requires 25 nodes to process the entire observing band. Calibration can be performed in either GPU or CPU mode. If using the GPU then the gpuq queue must be used otherwise the workq queue must be used. The RTS calibration script (cal.in) has the following format:

```
UpdateCal : 1
nCalibrators : 1
#
TemplateBase: rts_
#
catextent: 20
calcadence: 64
calsrcs: 30
#
rtsbin: rts_gpu
# Clean up any previous calibration
Clean: 1165919616-1166109944
# Do the calibration
Calibrate: 1165919616-1166109944
```

The UpdateCal and nCalibrators parameters should be set to 1 - these indicate that the calibration solutions will be updated and that one calibrator sources will be used, respectively. It is important to note that the calibrator source used is actually a hybrid source that is created specifically for this obsid. The hybrid calibrator is created by extracting calsrcs number of sources from the global sky model that are within catextent x catextent degree region centred on the beam former pointing centre. The TemplateBase provides a prefix for the RTS configuration template files used to generate the actual RTS input files needed for calibration and/or imaging. calcadence

specifies the number of time stamps within the data set to use for the purpose of calibration. rtsbin specifies the RTS binary to use for calibration, for GPU processing this should be set to rts_gpu, for CPU processing this should be set to rts_cpu.

The Clean command cleans out any non-critical files that were left over from previous runs of the calibration/imaging process. The Calibrate command initiates the calibration process. Internally, this generates a local sky model (based on the parameters set in calsrcs and catextent), creates an RTS input file and then initiates the calibration process using the RTS. Both Clean and Calibrate can accept either a single obsid or a range of obsids. If a range is specified then each available obsid in this range that has been previously downloaded will be calibrated.

The calibration phase works best if there is a single, central, simple and dominant source within the beam former pointing. Any departure from this will result in a less ideal solution. Worst case scenarios include bright sources near the edge of the field or complex extended sources anywhere within the field of view - such sources often result in calibration failure.

To initiate the calibration process simply submit to the queue as follows:

```
sbatch qcal.sh
```

At the end of the calibration process there should be a BandpassCalibration_node*.dat file and a DI_JonesMatrices_node*.dat file for each available coarse channel in the data set.

Basic imaging

The following is a sample shell script to start the imaging process (qimage.sh):

```
#!/bin/bash -l
#SBATCH --nodes=25
#SBATCH --ntasks-per-node=1
#SBATCH --time=12:00:00
#SBATCH --partition=gpuq
#SBATCH --account=mwasci
#SBATCH --export=NONE
rm image.log
rtsprocess.py image.in
```

Note that the image process requires 25 nodes to process the entire observing band. Imaging can be performed in either GPU or CPU mode for natural weighting schemes but the uniform and robust weighting schemes are currently only supported in CPU mode. If using the GPU then the gpuq queue must be used otherwise the workq queue must be used. The RTS imaging script (image.in) has the following format:

```
Weighting: Natural
FieldSize: 8.0
nCalibrators : 1
TemplateBase: rts_
#
fscrunch: 1
imgcadence: 96
#
DoAccumulate: False
#
Minbaseline: 80
MakePSF: False
catextent: 6
nPeel: 0
nIonoCal: 0
imgsrcs: 100
UpdateCal: 0
#
rtsbin: rts_gpu
ImgCat: Regenerate
Image: 1165919616-1166109944(1.6504278,-17.9501) self uvceti
```

The TemplateBase provides a prefix for the RTS configuration template files used to generate the actual RTS input files needed for imaging. imgcadence specifies the number of time stamps at a time to use within the data set to use for the purpose of imaging (if imgcadence is a multiple of the number of integrations in a snapshot then that multiple set of image cubes will be generated. rtsbin specifies the RTS binary to use for imaging, for GPU processing this should be set to rts_gpu, for CPU processing this should be set to rts_cpu. Note that only natural weighting is supported by the GPU version of the RTS.

The Weighting parameter can be used to set the weighting scheme (Natural, Uniform or Robust). For Robust weighting the robustness can be set with the Robustness parameter. DoAccumulate specifies whether weights should be accumulated over the entire band (needed for uniform and robustness weighting if generating full band images). A separate Accumulate step is needed to generate the weighting files before imaging if using Uniform or Robust weighting.

fscrunch specifies the number of neighbouring channels to average together. If set to 1 then the data will be imaged using the finest available channel resolution (usually either 40 kHz or 10 kHz depending on the observing set up). fscrunch can not be set to a value large than the number of channels in a coarse channel (generally 32 for 40 kHz channels and 128 for 10 kHz channels).

Minbaseline is used to set the shortest baseline length (in wavelengths) to be used for imaging, Maxbaseline is used to set the longest baseline length (in wavelengths).

Peeling is enabled by setting the catextent (where the sources will be selected from a catextent x catextent degree region centred on the beam former pointing centre) and a local sky model of imgsrcs sources will be generated. From this catalogue, nPeel specifies the number of sources to directly subtract from the visibilities, nIonoCal specifies the number of sources to use for full ionospheric calibration (flux and position determined) and subtraction.

The ImgCat command specifies whether the local sky model should be regenerated or alternatively whether it should be copied from another specified obsid. This is useful if you want to maintain a consistent set of sky models for peeling.

The Image command performs the imaging. It can take a single obsid or a range of obsids (in which case all downloaded obsids in this range will be imaged). By default, the correlated image centre is used as the image centre, alternatively, the image centre can be explicitly specified within parenthesis immediately after the obsid. The second obsid specifies the calibrator to use. If set to “self” then a previously determined calibration solution for that obsid is used. If an obsid is specified then a calibration solution is copied from that obsid instead (useful if the field is complex but a simpler calibrator field is available). The final parameter indicates the directory in to which the image cubes will be

copied to (within this path each obsid will have its own directory within which the image cube is stored).

To initiate the imaging process simply submit to the queue as follows:

```
sbatch qimage.sh
```

At the end of the imaging process there should be an image cube for each of the requested obsids in the destination path provided.

Different weighting schemes (TODO)

Natural

Uniform

Robust

Image Data (TODO)

Describe image data

How to integrate

HealPix data

Other Helper Tools (TODO)

imint.py destination image-list — integrate a series of images

imstats.py image-list — determine basic statistics for a series of images (minimum, maximum, std.dev).

sflag.py index threshold — determine the median rms noise in Stokes V of the indexed obsid and flag any channel that has (channel noise > threshold * median).

collapseobsids.py index - create integrated full-Stokes images for the indexed obsid.

project.py index - reproject all channel images in the indexed obsid to the local template image.

Command Reference

FetchData: *obsid*

> Download GPU box files, metadata and Cotter flag files from the archive for the specified obsid

FetchMetadata: *obsid[-obsid2]*

> Regenerate metadata for the specified obsid(s).

Reflag: *obsid[-obsid2] threshold*

> Analyse the Cotter flag files for the specified obsid(s) and flag any channels that exceed a median flagging occupancy by more than a factor specified by the threshold.

PairReflag: *obsid1[-obsid2] obsid threshold*

> PairReflag performs the same function as Reflag but ensures that the obsid1[-obsid2] snapshots are flagged in the same manner as obsid. This is particularly useful for LST-matched difference imaging to ensure similar visibility coverage.

Clean: *obsid1[-obsid2]*

> Remove all temporary and/or generated files from the specified obsid(s) e.g. calibration files, log files, images, etc.

Calibrate: *obsid1[-obsid2]*

> Generate a local sky model for the specified obsid(s) and calibrate based on this sky model.

UVDump: *obsid1[-obsid2]*

> Output a uv-fits format data set for the specified obsid(s).

Accumulate: *obsid1[-obsid2]*

> Perform uv weighting accumulation for the specified obsid(s). This step is necessary if using non-natural weighting schemes and if the intention is to integrate images across the entire observing band.

Image: *obsid1[-obsid2][(rahrs,decdeg)] calobs|self destPath*

> Generate image cubes for the specified obsid(s). If no image centre is supplied (i.e. rahrs and decdeg) then the pointing centre will be used. calobs specifies the obsid to use for a previously generated calibration solution (via Calibrate), alternatively if self is specified then the imager will use the previously generated calibration solution for the obsid i.e. in-field calibration.

Parameter reference

Weighting: *Natural* | *Uniform* | *Robust* (Default: Natural)

Natural

Robustness: *robustness* (Default: +2.0)

+2.0

DoAccumulate: *True* | *False* (Default: False)

False

FieldSize: *degrees* (Default: 2.0)

2.0 degrees

NSide: *n* (Default: 2048)

2048

fscrunch: *nchan* (Default: 4)

4

nCalibrators: *ncal* (Default: 1)

1

nPeel: *npeel* (Default: 0)

0

nIonoCal: *ncal* (Default: 0)

0

UpdateCal: *nup* (Default: 0)

0

ShortTaper: *wavelengths* (Default: None)

None

LongTaper: *wavelengths* (Default: None)

None

MinBaseline: *wavelengths* (Default: None)

None

MaxBaseline: *wavelengths* (Default: None)

None

StartAt: *integration* (Default: 0)

0

EndAt: *integration* (Default: 0)

0

UseFlag: *True* | *False* (Default: True)

True

UseMeta: *True* | *False* (Default: True)

True

TemplateBase: *prefix* (Default: rts_)

rts_

CalCadence: *integrations* (Default: 64)

64

ImgCadence: *integrations* (Default: 96)

96

CatExtent: *degrees* (Default: 20)

20

rtsbin: *rts_cpu* | *rts_gpu* (Default: rts_gpu)

rts_gpu

ImgCat: *obsid* | *Regenerate* (Default: Regenerate)

Regenerate

CalCat: *obsid* | *Regenerate* (Default: Regenerate)

Regenerate

SrcCat: *cat_file* (Default: rtscat.txt)

rtscat.txt

CalSrcs: *sources* (Default: 25)

25

ImgSrcs: *sources* (Default: 300)

300

Regrid: *False* | *True* (Default: False)

False

RegridMethod: 0-3 (Default: 3)

3

MakePSF: *False* | *True* (Default: False)

False

MakeStokes: *False* | *True* (Default: True)

True

Sample .bashrc file for RTS set-up.

```
#
export HISTSIZE=10000
test -s ~/.alias && . ~/.alias || true
#
export MWA_OPS_DIR=/group/mwaops
export MWA_SCI_DIR=/group/mwasci
#
# switch to GCC/GNU enironment
module switch PrgEnv-cray PrgEnv-gnu
module unload gcc
# use gcc 4.8 for CUDA/RTS, use version 4.9 for boost
module load gcc/4.8.2
module load cray-libsci
module load cmake
module load scipy
module load lapack
module load cudatoolkit
module load astropy
module load cfitsio
module load boost
module load casacore
module load ephem
module load readline
module load psycopg2
module load gsl
module load matplotlib

# set path to CFITSIO library
export CFITSIO_DIR=/ivec/cle52/galaxy/devel/PrgEnv-gnu/5.2.25/cfitsio/3370
source ~/MIRRC.sh

# set path to RTS binary
export RTSDIR=$MWA_OPS_DIR/CODE/RTS
export RTSBIN=$RTSDIR/bin

# set path to path where global sky models can be found
export RTS_CAT_PATH="$HOME/catalog/"

# set up library paths
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$MWA_OPS_DIR/CODE/lib"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:$CRAY_LD_LIBRARY_PATH"
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:${MIRLIB}"

# set up binary paths
export PATH="${PATH}:${MWA_OPS_DIR}/CODE/bin:${RTSBIN}:${MIRBIN}"
export PATH="${PATH}:${MWA_SCI_DIR}/code/MWA_Tools/scripts"

# set up Python paths
export PYTHONPATH=${PYTHONPATH}:$MWA_SCI_DIR/code/MWA_Tools/
export PYTHONPATH=${PYTHONPATH}:$MWA_SCI_DIR/code/MWA_Tools/scripts
export PYTHONPATH=${PYTHONPATH}:$MWA_SCI_DIR/code/MWA_Tools/mwapy
export PYTHONPATH=${PYTHONPATH}:$MWA_SCI_DIR/code/MWA_Tools/configs

# env vars for psql client
export PGPASSWORD=BowTie
export PGHOST=mwa-metadata01.pawsey.org.au
export PGDATABASE=mwa
export PGUSER=mwa
```

Sample RTS calibration template: rts_cal.in

```
//---------//
FscrunchChan=4

SubBandIDs=

StorePixelMatrices=0

MaxFrequency=180
ImageOversampling=3

applyDIcalibration=1
doMWArxCorrections=1
doRFIflagging=1
useFastPrimaryBeamModels=1

CorrDumpTime=0.0
CorrDumpsPerCadence=0
NumberOfIntegrationBins=0
NumberOfIterations=

StartProcessingAt=0
StartIntegrationAt=0

// In correlator mode, Base File name is used to find correlator files.
BaseFilename=*_gpubox

doRawDataCorrections=1
ReadGpuboxDirect=1
UsePacketInput=0
UseThreadedVI=0

ArrayFile=array_file.txt
ArrayNumberOfStations=128

ChannelBandwidth=0.04
NumberOfChannels=32

ArrayPositionLat=-26.70331940
ArrayPositionLong=116.67081524

//time is needed to set lst, even if ObservationTimeBase is set.
ObservationTimeBase=2456519.20083
```

```
// --- observing stuff --- //
ReadAllFromSingleFile=
ObservationFrequencyBase=167.055

ObservationPointCentreHA=1.031
ObservationPointCentreDec=-25.93

ObservationImageCentreRA=
ObservationImageCentreDec=

// --- weighting stuff --- //
calBaselineMin=20.0
calShortBaselineTaper=50.0

// --- calibration stuff --- //
DoCalibration=
SourceCatalogueFile=catalogue.txt
NumberOfCalibrators=1
```

Sample RTS imaging template: rts_img.in

```
//---------//
//ImagePSF=1
FscrunchChan=4

SubBandIDs=
StorePixelMatrices=1

MaxFrequency=180
ImageOversampling=5

applyDIcalibration=1
doMWArxCorrections=1
doRFIflagging=0
useFastPrimaryBeamModels=1

CorrDumpTime=0.0

CorrDumpsPerCadence=0
NumberOfIntegrationBins=0
NumberOfIterations=

StartProcessingAt=0
StartIntegrationAt=0

// In correlator mode, Base File name is used to find correlator files.
ReadAllFromSingleFile=
BaseFilename=*_gpubox

doRawDataCorrections=1
ReadGpuboxDirect=1
UsePacketInput=0
UseThreadedVI=1

ArrayFile=array_file.txt
ArrayNumberOfStations=128

ChannelBandwidth=0.04
NumberOfChannels=32

ArrayPositionLat=-26.70331940
ArrayPositionLong=116.67081524

//time is needed to set lst, even if ObservationTimeBase is set.
ObservationTimeBase=2456519.20083
```

```
// --- observing stuff --- //
ObservationFrequencyBase=167.055
ObservationPointCentreHA=1.031
ObservationPointCentreDec=-25.93

ObservationImageCentreRA=
ObservationImageCentreDec=

// --- weighting stuff --- //
calBaselineMin=20.0
calShortBaselineTaper=50.0

// --- calibration stuff --- //
DoCalibration=
SourceCatalogueFile=catalogue_imaging.txt
NumberOfCalibrators=1
NumberOfSourcesToPeel=0
NumberOfIonoCalibrators=0
UpdateCalibratorAmplitudes=0
```

Sample u-v export RTS template: rts_uv.in (not tested!)

```
//---------//
FscrunchChan=4

SubBandIDs=

MaxFrequency=200
ImageOversampling=5
writeVisToUVFITS=1

applyDIcalibration=1
doMWArxCorrections=1
doRFIflagging=0
useFastPrimaryBeamModels=1

CorrDumpTime=0.0
CorrDumpsPerCadence=0
NumberOfIntegrationBins=0
NumberOfIterations=

StartProcessingAt=0
StartIntegrationAt=0

// In correlator mode, Base File name is used to find correlator files.
ReadAllFromSingleFile=
BaseFilename=*_gpubox

doRawDataCorrections=1

ReadGpuboxDirect=1
UsePacketInput=0
UseThreadedVI=1

ArrayFile=array_file.txt
ArrayNumberOfStations=128

ChannelBandwidth=0.04
NumberOfChannels=32

ArrayPositionLat=-26.70331940
ArrayPositionLong=116.67081524

//time is needed to set lst, even if ObservationTimeBase is set.
ObservationTimeBase=2456519.20083
```

```
// --- observing stuff --- //

ObservationFrequencyBase=167.055

ObservationPointCentreHA=1.031
ObservationPointCentreDec=-25.93

ObservationImageCentreRA=
ObservationImageCentreDec=

// --- weighting stuff --- //

calBaselineMin=20.0
calShortBaselineTaper=50.0

//imgLongBaselineTaper = 3000.0
imgBaselineMin = 100.0

// --- calibration stuff --- //

DoCalibration=

SourceCatalogueFile=catalogue_imaging.txt

NumberOfCalibrators=1
NumberOfSourcesToPeel=0
NumberOfIonoCalibrators=0
UpdateCalibratorAmplitudes=0
```