Quickstart Guide

Acronyms / Jargon / Etc

Machines

LGC - Linux gateway computer, main node used to access the LCU and REALTA

LCU - Linux control unit, the telescope's brain, used to configure observations, accessibly only via the LGC on the ilofar user

ucc1 - Main recording machine

Paths

REALTA:/mnt/ucc1_recording2/data/observations/YYYY_MM_DD - recording drive path

REALTA:/mnt/ucc1_recording2/data/scripts - recording scripts locations

ilofar@LGC:~/scheduling/schedules - input schedule file folder

Concepts

Software level (swlevel)

- 0 - Station is shutdown
- 1 - Station has basic power
- 2 - Minimum observing mode, hardware can generate antenna correlations
- 3 - Station is fully setup and can beam-form
- 4+ are not needed in a quick-start guide

RCU Modes

- 3: LBA, 0 - 100MHz
- 5: HBAlo, 100 - 200MHz
- 7: HBAhi, 200 - 250MHz

Data products

- BST: typically 1 second cadence data
- XST: Antenna cross-correlations for all-sky imaging
- CEP/REALTA Data: The 3Gbps data stream containing 5.12us data

Preparing Observation Schedules

Scheduling Observations

Observations are typically scheduled using scripts that execute the quicksched.py script found on the LGC. This is a python3.6+ script that takes in a file with schedule entries in a specified format, and then produces two files to observe the specified sources with the HBAs in mode 5 from subbands 12 to 499.

> The telescope typically takes 3 minutes in order to be configured before the first (and any `STOPSTART`) observation begins to produce real data. Additionally it is not uncommon for station handover to occur late. Ensure you consider these points when scheduling high priority observations.

Each entry roughly follows the format

```
YYYY-MM-DDTHH:MM - YYYY-MM-DDTHH:MM :<TAB>SourceName [RightAscentionRadians, DeclinationRadians, 'COORDINATEBASIS']
```

These can be chained together to form a schedule that looks like this.

```
2021-09-20T19:30 - 2021-09-20T20:29 :	J1931+4229 [5.110297162889553, 5.110297162889553, 'J2000']
2021-09-20T20:30 - 2021-09-20T20:49 :	B1951+32_PSR [5.20531221260202, 0.5738302241353217, 'J2000']
```

After running `python3 quicksched.py --no-write my_schedule.txt`, it will notify you of any issues with the input, or give you a summary of the observations taking place.

Entry Modifiers

`STOPSTART`: In the case that other observations are scheduled using other software, and you need to leave a gap, you can add `STOPSTART` to the end of the line after the gap takes place. Whenever this keyword is detected, the station shuts down after the previous observation, then starts up 3 minutes before the given scheduled observation is meant to begin. As an example, these observations will run from 19:27 - 20:29, bring the station down to swlevel 0 for an hour, bring the station up at 21:30, and finally run again from 21:33 - 21:59 before returning the station to software level 0.

```
2021-09-20T19:30 - 2021-09-20T19:59 :□J1931+4229 [5.110297162889553, 5.110297162889553, 'J2000']
2021-09-20T20:00 - 2021-09-20T20:29 :□B1951+32_PSR [5.20531221260202, 0.5738302241353217, 'J2000']
2021-09-20T21:33 - 2021-09-20T21:59 :□J1931+4229 [5.110297162889553, 5.110297162889553, 'J2000']
STOPSTART
```

`[IGNORE_ELEVATION]`: In the case that a source will be below the default minimum elevation (typically between 10 and 15 degrees depending on setup), the `quicksched.py` script will reject the observation. If you intend to observe the source at such a low elevation, you can include `[IGNORE_ELEVATION]` to highlight that you intend to observe this source at a low elevation and the script will proceed without highlighting the elevation of the source.

```
2021-09-20T19:30 - 2021-09-20T19:59 :□J1931+4229 [5.110297162889553, 5.110297162889553, 'J2000']
2021-09-20T21:33 - 2021-09-20T21:59 :□J1931+4229 [5.110297162889553, 5.110297162889553, 'J2000']
[IGNORE_ELEVATION]
```

`[MODEN]`: Where N is replaced with an observing mode (3-7), this will change the observing antenna and frequencies from the mode 5 default to the specified mode as needed.

> Further validation is needed for mode 6, as it sometimes fails to change the clock mode.

```
2021-09-20T19:30 - 2021-09-20T19:59 :□J1931+4229 [5.110297162889553, 5.110297162889553, 'J2000']
[MODE3] # LBA Obs
2021-09-20T20:00 - 2021-09-20T20:29 :□J1931+4229 [5.110297162889553, 5.110297162889553, 'J2000'] #
HBALo Obs
2021-09-20T20:30 - 2021-09-20T20:59 :□J1931+4229 [5.110297162889553, 5.110297162889553, 'J2000']
[MODE7] # HBAhi obs
```

`OR(TOGGLE_FILE)`: Where "TOGGLE_FILE" is replaced with a file name of choice, this modifier can be used to switch between two observations during schedule execution based on the existence of a file. By default, the first entry on a line will be performed if the file does not exist, while the second will be executed if it does exist. Two files need to be created to perform a toggle: one in the observation folder, and another in the recording folder. Typically, the file name is prefixed by "NOT_" to make it clear that it is disabling the first observation, and enabling the second.

The file must be created before the end of the last observation, otherwise the default observation will be performed.

For I-LOFAR observers, there is a `toggle_obs.sh` script that will create the input filename in the correct directories for you.

After an observation is switched, the file is removed. As a result, it is not recommended to use the same file name for multiple switches, unless the observer wants to re-create the same file multiple times.

```
2022-07-19T17:00 - 2022-07-19T17:59 :FRB20200120E [2.6090272489593733, 1.2012325539587194, 'J2000']
OR(NOT_M81) [Sun357]
2022-07-19T18:00 - 2022-07-19T18:59 :[Sun357] OR(NOT_SUN) J0939+45 [2.528618475878951,
0.7897614865274342, 'J2000']
```

Preparing Observation Schedules With `prepquick.py`

The `prepquick.py` script can be used to generate a number of candidate observation windows, based on transit time and observation length, or on source elevation. Each file must start with a start time, which can then be followed by a number of lines with source names, coordinates and information regarding the observation length.

Any information on a line after a '#' is not parsed.

> The script does not attempt to realign overlapping observations, you must manually fix these issues before quicksched.py will accept them.

Defining Sources

Sources can be defined by either a set of coordinates in a common basis (J2000, SUN, JUPITER, etc.) or by a name recognised by the CDS, which will be queried by an astropy.coordinates.SkyCoord.from_name() call.

Most solar entries will be automatically converted to mode 357 entries.

Here's an example set of inputs and outputs in both forms, with the time specifiers removed for clarity:

```
#input.txt
Sun
Moon
PSR B1951+32
J0054+66 [0.23561944901923448, 1.1664616585719019, 'J2000']
TVLM 513-46

#output.txt (self rearranged by start time)
[Sun357]
```

```
J0054+66_PSR [0.23561944901923448, 1.1664616585719019, 'J2000']
MOON [0, 0, 'MOON']
TVLM_513-46 [3.931949476576734, 0.3985272105537257, 'J2000']
B1951+32_PSR [5.205312212078421, 0.5738302241353217, 'J2000']
```

Observing by Elevation

You can get the observing window for a source while it is above a certain elevation by appending a tilde and elevation in degrees. The elevation will be overruled by the value of the `--cutoff` flag if it is set too low.

```
# input.txt
2023-04-05T00:00:00.000
Sun ~ 15
J0139+3336 ~ 45
Jupiter ~ 20

# output.txt
2023-04-05T07:40 - 2023-04-05T17:24 :	[Sun357]
2023-04-05T08:30 - 2023-04-05T17:14 :	JUPITER [0, 0, 'JUPITER']
2023-04-05T09:25 - 2023-04-05T17:09 :	J0139+3336_PSR [0.4361308670963172, 0.586720056619953, 'J2000']
```

Observing by Observation Lengths

Otherwise, you can use an observation length to get a candidate schedule for a source by appending a tilde and an observation window in minutes. Note that due to the necessary time needed between beam switches, a minute will be subtracted from this time.

```
# input.txt
2023-04-05T00:00
Sun @ 75
J0139+3336 @ 120
Jupiter @ 30

# output.txt
2023-04-05T12:00 - 2023-04-05T13:14 :	[Sun357]
2023-04-05T12:20 - 2023-04-05T14:19 :	J0139+3336_PSR [0.4361308670963172, 0.586720056619953, 'J2000']
2023-04-05T12:40 - 2023-04-05T13:09 :	JUPITER [0, 0, 'JUPITER']
```

Times

A starting line containing a ISOT-formatted date/time string is required in order to determine the source transits. One or more of these lines can be included in the file, and all following sources will be parsed using the last provided timestamp.

The script will determine the next transit after a given time (be it in 5 minutes or 23 hours), and produce the scheduling line candidate based on that. Here is an example that combines each of the described properties above, alongside using multiple different transit times.

```
# input.txt
2023-05-30T06:00
Sun ~ 15
Moon @ 20
PSR B1951+32 @ 20
J0054+66 [0.23561944901923448, 1.1664616585719019, 'J2000'] @ 20
TVLM 513-46 ~ 55

# New day, new observations
2023-06-01T18:00:00.000
Sun ~ 45
J0139+3336 ~ 45
Jupiter ~ 20

2023-06-03T11:00:00.000
Sun ~ 45



# output.txt
2023-05-30T06:10 - 2023-05-30T18:44 :□[Sun357]
# Note the overlap of this observation, this needs manual tweaking
2023-05-30T08:45 - 2023-05-30T09:04 :□J0054+66_PSR [0.23561944901923448, 1.1664616585719019, 'J2000']
2023-05-30T20:15 - 2023-05-30T20:34 :□MOON [0, 0, 'MOON']
2023-05-30T21:25 - 2023-05-31T00:29 :□TVLM_513-46 [3.931949476576734, 0.3985272105537257, 'J2000']


2023-05-31T03:40 - 2023-05-31T03:59 :□B1951+32_PSR [5.205312212078421, 0.5738302241353217, 'J2000']


2023-06-02T05:05 - 2023-06-02T14:39 :□JUPITER [0, 0, 'JUPITER']
2023-06-02T05:35 - 2023-06-02T13:19 :□J0139+3336_PSR [0.4361308670963172, 0.5867200566l9953, 'J2000']
# Note that this sun entry is the day after the time stamp provided
2023-06-02T09:35 - 2023-06-02T15:19 :□[Sun357]
```

2023-06-03T09:35 - 2023-06-03T15:19 :□[Sun357]

Queuing an Observation Schedule

> Working in the scheduling directory is strongly discouraged. It will very likely lead to frustration and spam for any other users on the ilofar account. It is advised to prepare schedules on your local machine or in the temp folder.

Once you have used the previous two guides to build a schedule file, you can then send it to be queued for observations. The schedule should be moved into directory at `/home/ilofar/scheduling/schedules` by whatever is determined the preferred method. On the `ilofar` user, the folder can be quickly accessed using the `sched` alias.

Once the file is in the folder, the monitoring service should notify you and pick up the file. It will be validated with the `quicksched.py` script, and then either moved to one of the `prepared` or `failed` directories, depending on the output. In the case of failure, the logs should be piped to you, or they will be accessible in the `logs` folder inside the scheduling directory. The file can be modified in the failed directory, and moved back one directory once you believe the issue is resolved.

A second monitoring process check for any files in the `prepared` directory initially when the files are moved in to the directory, then every 4 hours afterwards. If a schedule starts within the next 24 hours, it will be queued for execution, after which the `interrupt.sh` script will be needed to exit the opened shells, screens and tmux's before submitting a replacement schedule.

```
dmckenna@americano:~/git/scheduling$ rsync -avhP ./sched_2023-06-06.txt
~/scheduling/schedules/tmp/sched_2023-06-06.txt
dmckenna@americano:~/git/scheduling$ ssh ilofar@lgc


ilofar@LGC:~$ sched
ilofar@LGC:~/scheduling/schedules$ cat ./tmp/sched_2023-06-06.txt
2023-06-06T07:00 - 2023-06-06T13:29 :	[Sun357]
2023-06-06T13:30 - 2023-06-06T13:59 :	Crab_PSR [1.4596732110242794, 0.3842252837309499, 'J2000']
ilofar@LGC:~/scheduling/schedules$ mv ./tmp/sched_2023-06-06.txt ./
ilofar@LGC:~/scheduling/schedules$
Message from ilofar@LGC on (none) at 15:16 ...
Attempting to parse new input schedule at /home/ilofar/scheduling/schedules/sched_2023-06-06.txt
EOF
```

```
Message from ilofar@LGC on (none) at 15:16 ...
/home/ilofar/scheduling/schedules/sched_2023-06-06.txt: Success; moving schedule to the
/home/ilofar/scheduling/schedules/prepared/ directory.
EOF
<ENTER>
ilofar@LGC:~/scheduling/schedules$
```

Interrupting Scheduled Observations

The interruptObs.sh script operates in one of two modes: focused and global. In focused mode, it will take an input schedule, determine the relevant launched instances for that schedule, and specifically shut down all related screens, tmuxes and processes. In global mode, the script will kill anything related to our scheduling scripts.

See date(1) (-d input) for further details on accepted time stamps. We also support "next" in focused mode, which will parse the input file, find the next gap between observations, and perform the switch then to minimise downtime.

Focused Mode

In focused mode, you must provide at least an interruption time (in a format accepted by date), and a reference schedule to analyse. You can optionally provide a third parameter with a replacement schedule if you do not want to manually staged and run another schedule afterwards. These are all valid examples for killing a specified "./schedule/my_sched.txt" schedule file.

```
ilofar@LGC:~/workdir$ interruptObs.sh 2023-06-01T07:00:00 ./myschedule.txt
ilofar@LGC:~/workdir$ interruptObs.sh now ./myschedule.txt
ilofar@LGC:~/workdir$ interruptObs.sh now ~/scheduling/schedules/staged/sched_2023-06-01.txt
ilofar@LGC:~/workdir$ interruptObs.sh next ~/scheduling/schedules/staged/sched_2023-06-01.txt
schedule_replacement.txt
```

Global Mode

In global mode, you just need to specify a time and the "KILLALL" keyword as the schedule. After this, it will find any running scripts, screens, tmuxes, etc, on both the recording node, LCU, and local node that are associated with scheduling and kill them. This should typically only be used in the case that something has gone horribly wrong.

```
ilofar@LGC:~/workdir$ interruptObs.sh 2023-06-01T07:00:00 KILLALL
ilofar@LGC:~/workdir$ interruptObs.sh now KILLALL
```

Processing Pulsar Data

`CDMT` (Stokes I) Method

The GPUs in `ucc4` are typically used to reduce the majority of the pulsar data produced by the station into mode 5, 8x channelised, 16x downsampled Sigproc Stokes I filterbanks. This is achieved through using a `python3` script, `cdmtProc.py`, to produce a `bash` script with the correct flags needed to call `CDMT` with maximum performance, while not overutilising the GPU's VRAM.

The script needs to be provided a path to the recording folder, and optionally an extension to add to the file name. The resulting `bash` script can then be executed. Once data is analysed, the `compress.sh` and `transfer.sh` scripts can be used to compress the raw data using `zstandard`, then transfer it to the DIAS archive.

```
obs@ucc4:~$ tmux new
obs@ucc4:~$ dckrgpu


root@b02cf5b93fbc:/home/user# cd /mnt/ucc4_data2/data/David/
root@b02cf5b93fbc:/mnt/ucc4_data2/data/David# mkdir rrat_2023_05_30
root@b02cf5b93fbc:/mnt/ucc4_data2/data/David# cd rrat_2023_05_30
root@b02cf5b93fbc:/mnt/ucc4_data2/data/David/rrat_2023_05_30# cp ../cdmtProc.py ./
root@b02cf5b93fbc:/mnt/ucc4_data2/data/David/rrat_2023_05_30# python3 cdmtProc.py -i
/mnt/ucc1_recording2/data/rrats/2023_05_30/ --extra 1
root@b02cf5b93fbc:/mnt/ucc4_data2/data/David/rrat_2023_05_30# ll
total 92
drwxr-xr-x   2 root root  4096 May 30 15:48 ./
drwxr-xr-x 129 1000 1000 69632 May 30 15:47 ../
-rw-r--r--   1 root root 14501 May 30 15:47 cdmtProc.py
-rw-r--r--   1 root root   886 May 30 15:48 cdmtProc_1.sh
root@b02cf5b93fbc:/mnt/ucc4_data2/data/David/rrat_2023_05_30# bash cdmtProc_1.sh


<data is processed>


root@b02cf5b93fbc:/mnt/ucc4_data2/data/David/rrat_2023_05_30# bash ../compress.sh; bash ../transfer.sh
```

Single Pulse, Periodic Emission Searches

Two scripts in the `/mnt/ucc4_data2/data/David/` directory can be used to setup single pulse searches.

The `generateHeimdall.py` script will open a number of matplotlib interfaces with bandpasses for RFI flagging before writing a list of narrow and wide DM searches for single pulses data. When flagging the data, each click on the bandpass will flag a subband worth of data (8 channels), and a range of subbands can be flagged by clicking somewhere, moving the mouse to the end of the range, then pressing "a" with the window active. The output script will be in the `cands/` subdirectory of the input folder.

The candidate files are then used to generate plots of each pulse that fits certain criteria (typically SNR > 7.5, width < 2**10 samples). This process is automated with the `frb_v4.py` and `generateCands.py` scripts.

```
dmckenna@ucc2:/mnt/ucc2_data1/data/dmckenna/working_cands$ mkdir rrat_2023_05_30
dmckenna@ucc2:/mnt/ucc2_data1/data/dmckenna/working_cands$ cd rrat_2023_05_30
dmckenna@ucc2:/mnt/ucc2_data1/data/dmckenna/working_cands/rrat_2023_05_30$ cp ../frb_v4.py .
dmckenna@ucc2:/mnt/ucc2_data1/data/dmckenna/working_cands/rrat_2023_05_30$ for i in {1..4}; do
python3.8 ../generateCands.py -i /mnt/ucc4_data2/data/David/rrat_2023_05_30/ -o proc_ucc4_${i}.sh; bash
proc_ucc4_${i}.sh; sleep 3600; done
```

The `generatePrepfolds.py` script will generate a list of `rfifind` and `prepfold` commands for both set targets, and targets within 1 degree of the original pointing. The output script will be in the `folds/` subdirectory of the input folder.

Single pulse candidates and folded profiles can then be copied locally, with folded profiles converted from `ps` files to `png` using the `grabFolds.sh` script.

`udpPacketManager`/`digifil` (Stokes IQUV) Method

For full Stokes IQUV data, we use `udpPacketManager` to convert the raw CEP data into a `DADA` data block, which is then processed with `digifil`. The script for automating this need to be modified to contain a dispersion measure for coherent dedispersion, and a downsampling fact for the data. Afterwards, the script can be run, pointed at a specific directory, and it will handle the rest for you.

```
obs@ucc1:~$ tmux new
obs@ucc1:~$ bash /mnt/ucc1_recording2/data/rrats/4pol/enter_docker.sh


root@106316e58e36:/home/user# cd /mnt/ucc1_recording2/data/rrats/2023_05_30/
root@106316e58e36:/mnt/ucc1_recording2/data/rrats/2023_05_30# mkdir tmp_processing
root@106316e58e36:/mnt/ucc1_recording2/data/rrats/2023_05_30# cd tmp_processing
root@106316e58e36:/mnt/ucc1_recording2/data/rrats/2023_05_30/tmp_processing# cp ../../4pol/* ./
root@106316e58e36:/mnt/ucc1_recording2/data/rrats/2023_05_30/tmp_processing# bash 4pol_generate.sh
```

```

# Alternatively,
root@106316e58e36:/home/user# cd /mnt/ucc1_recording2/data/rrats/prcoessing/
root@106316e58e36:/mnt/ucc1_recording2/data/rrats/processing# mkdir 2023_05_30
root@106316e58e36:/mnt/ucc1_recording2/data/rrats/processing# cd 2023_05_30
root@106316e58e36:/mnt/ucc1_recording2/data/rrats/processing/2023_05_30# cp ../../4pol/* ./
root@106316e58e36:/mnt/ucc1_recording2/data/rrats/processing/2023_05_30# bash 4pol_generate.sh
/mnt/ucc1_recording2/data/rrats/2023_05_30/
```

Processing Solar Data (Basic)

UPM Method

Solar data is typically processed using udpPacketManager, to produce a Stokes IV output, downsampled by a factor of 256 to 1.3ms, using a set of automated scripts.

The `process_data.sh` script can be set to process any subdirectories with data on a specific path, or will automatically find the newest directory of observations, and process any Solar observations.

The series of commands needed to setup a processing run can be found in the code block below.

```
obs@ucc1:~$ cdrs
obs@ucc1:/mnt/ucc1_recording2/data/sun$ mkdir 20230530_sun357
obs@ucc1:/mnt/ucc1_recording2/data/sun$ cd 20230530_sun357
obs@ucc1:/mnt/ucc1_recording2/data/sun/20230530_sun357$ cp ../templates/* ./
obs@ucc1:/mnt/ucc1_recording2/data/sun/20230530_sun357$ bash enter_docker.sh

root@2639bfd271d6:/home/user# cd $LAST_DIR
root@2639bfd271d6:/mnt/ucc1_recording2/data/sun/20230530_sun357# bash process_data.sh

# Alternatively
root@2639bfd271d6:/home/user# cd $LAST_DIR
root@2639bfd271d6:/mnt/ucc1_recording2/data/sun/20230530_sun357# bash process_data.sh
/mnt/ucc1_recording2/data/rrats/2023_05_30/

# After processing
root@2639bfd271d6:/mnt/ucc1_recording2/data/sun/20230530_sun357# exit
obs@ucc1:/mnt/ucc1_recording2/data/sun/20230530_sun357$ bash transfer.sh
```

Processing Other Data (EMPTY)

UPM Method

Bash Aliases

LGC

```
$ # Access the LCU while in local mode
$ lcu

$ # Access the Recording node (ucc1)
$ ucc1

$ # Access the scheduling directory
$ sched

$ # Test a schedule file
$ testsched my_input_file.txt
```

UCC1

```
$ # Access recording drive 1 (old)
$ cdr
$ # Access the active recording drive
$ cdr2

$ # View the remaining storage on the recording drives
$ dfr

$ # See the disk usage of the current subdirectories
$ duds

$ # Access the recorded observations directory
$ cdo
$ # Access the last recorded observation
$ cdol

$ # Access the RRATs processing directory (old recording directory)
$ cdrr
$ # Access the latest RRAT directory
$ cdrrl

$ # Access the Solar processing directory
$ cdrs
$ # Access the latest Solar directory
$ cdrsl
```