

Archived Pages

Old pages that no longer apply to the station setup

- [\(Archived\) Handling Combined Solar/KBT/Beamformed Observations](#)
- [\(Archived\) Handling KBT/Solar357 Observations \(ScheduleKBT.py Method\)](#)
- [\(Archived\) Executing a Schedule and Processing Data](#)

(Archived) Handling Combined Solar/KBT/Beamformed Observations

Scheduling Observations

Observations are typically scheduled using the [quicksched.py](#) script found [here](#). 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 typical beamformed sources using HBAs in mode 5 from subbands 12 to 499, and Solar entries using the KBT backend to observe in "mode 357".

By default, the script wakes up 3 minutes before the first (and any **STOPSTART**) observations to configure the telescope into software level 3. Ensure you consider this time when scheduling observations.

Each entry roughly follows the format

```
# Generic Beamformed Target
YYYY-MM-DDTHH:MM - YYYY-MM-DDTHH:MM <TAB> SourceName [RightAscensionRadians, DeclinationRadians,
COORDINATEBASIS]

# Solar Mode 357 Entry
YYYY-MM-DDTHH:MM - YYYY-MM-DDTHH:MM <TAB> [Sun357]
```

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

```
2021-09-20T12:30 - 2021-09-20T19:29 :[Sun357]
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']]
```

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, shut down for an hour, then run again from 21:33 - 21:59 before returning the station to software level 0.

```
2021-09-20T19:30 - 2021-09-20T19:59 -[J]1931+4229 [5.110297162889553, 5.110297162889553, 'j2000']
2021-09-20T20:00 - 2021-09-20T20:29 -[B]1951+32_PSR [5.20531221260202, 0.5738302241353217, 'j2000']
2021-09-20T21:33 - 2021-09-20T21:59 -[J]1931+4229 [5.110297162889553, 5.110297162889553, 'j2000']
STOPSTART
```

After running `python quicksched.py my_schedule.txt`, two files will be created in your directory with the prefixes `lcu_script_YYYYMMDD.sh` and `lcc_script_YYYYMMDD.sh`, which will be used to perform and record the observation. In the case that a Solar KBT observation is also requested, a command to execute "VerifySolarKBT.py" will also be generated, which can be run on the LGC.

Running Observations

Configuring the LCU

In order to perform your observation, you will need to transfer the `lcu_script_YYYYMMDD.sh` script to the LCU. In the case that handover has already been performed, you can simply transfer the script to the station (preferably a sub-directory in `~/local_scripts/`) and run it in a screen as a normal bash script. Starting from the LGC on the ilofar account, you can use the `lcu` alias to open an SSH connect to the station, then you should navigate to the `~/local_scripts/` folder and either use, or create, your own sub-folder for your observations.

```
ilofar@lgc ~ $ lcu
user1@lcu ~ $ cd local_scripts/
user1@lcu ~/local_scripts $ mkdir my_dir; cd my_dir
```

You may need to find a copy of `sleepuntil.sh` in the parent folder if you are creating a new directory. Afterwards, you can open a screen and execute the script.

```
lcu ~/local_scripts/dmckenna $ screen -S dmckennaObs
lcu ~/local_scripts/dmckenna $ bash lcu_script_YYYYMMDD.sh
< Control a, b to exit >
```

Scheduling before local mode

However if we do not yet have the station and the observation will start before you are available for the day, there is a script on the LGC at `~/David/transfer.sh` that can take an input and transfer it to the station, then launch a screen and execute the script unattended.

The `transfer.sh` script depends on the `sleepuntil.sh` script located in the same folder, if you make a copy of `transfer.sh` to modify the target directory, be sure to copy `sleepuntil.sh` to the same folder to ensure it can run.

The `transfer.sh` script takes 3 inputs: the path of the script to transfer, the date to transfer (as YYYYMMDD) and the time to start trying to transfer (HHMMSS). As an example, to transfer a script for handover occurring at 2022-02-02T08:00:00 the following command could be used to start trying to transfer file 30 minutes prior to handover. In the case that we receive the station early, this will allow for the file to be transferred and station to be configured as soon as possible.

```
llofar@LGC ~/David $ tmux new -s transferLCuScript
< enter tmux shell>
llofar@LGC ~/David $ bash transfer.sh lcu_script_name.sh 20220202 073000
< Control d, b to exit >
```

Preparing the LGC

In the case that Solar/KBT observations are requested, a script, `VerifySolarKBT.py` can be run on the LGC to validate that the KBT system has initialised correctly and BST files are being generated.

The required command will be produced based on your input schedule. As an example, an observation that runs from 2022-02-22T12:22:00 to 2022-02-22T16:22:00, will produce a command with one entry, which can then be ran from the standard "obs_schedule" folder on the LGC. As a reminder on how to find and execute the script,

```
llofar@LGC:~$ cd Scripts/Python/obs_schedule_py3/
llofar@LGC:~/Scripts/Python/obs_schedule_py3$ python3 VerifySolarKBT.py -d 2022-02-22T12:22:00,2022-02-22T16:22:00
```

This script will start up 1 minute after the set start time, and send the standard "KBT OK" or "KBT FAILED" emails to the observers mailing list, and produce end-of-observation plots after the observation has ended.

Recording on UCC1

Given `ucc1` should always be available, the script can be transferred to the recording drive and executed as needed. We currently perform all recording on the `ucc1_recording2` drive which can be easily accessed via the `ucc2` alias on the `ucc1` account. After that, you can get the required scripts in the scripts folder, and make a sub-directory for the given source category and perform the observation.

```
# Connect to ucc1
lofar@LGC ~ $ ucc1

# Navigate to the recording directory
obs@ucc1 ~ $ cdr2

# Make your recording folder
obs@ucc1 /mnt/ucc1_recording2/data $ mkdir rrats/2022_02_02

# Copy the recording scripts to the folder
obs@ucc1 /mnt/ucc1_recording2/data $ cp ./scripts/* ./rrats/2022_02_02/

# Enter the folder
obs@ucc1 /mnt/ucc1_recording2/data $ cd rrats/2022_02_02

# Add your ucc_script_YYYYMMDD.sh script to the folder via rsync, scp, nano + copy paste, etc

# Open a new tmux shell
obs@ucc1 /mnt/ucc1_recording2/data/rrats/2022_02_02 $ tmux new -s recording
<attach to tmux shell>

# Run the observing script
obs@ucc1 /mnt/ucc1_recording2/data/rrats/2022_02_02 $ bash ucc_script_YYYYMMDD.sh

# Close the tmux shell after all observation have been completed.
```

Processing Observations

These instructions assume you are in the standard processing Docker container. This can be accessed using the following command, after which you can return to your previous directory with the `$LAST_DIR` variable.

```
obs@uccN ~ $ docker --rm -it -e TERM -v /mnt:/mnt --gpus all --env LAST_DIR=$(pwd) pulsar-gpu-dsp2021
root@AAAAAAAA /home/soft $ cd $LAST_DIR
```

Standard Observations (TODO)

Rough overview: use `lofar_udp_extractor` to process the input to the desired output format.

```
$ lofar_udp_extractor -i /path/to/input/file/udp_1613%d_2022-02-02T02:22:22.000.zst -p "$PROCESSING_MODE" -
o "$OUTPUT_FILENAME" -a "$HEADER_PARAMETERS" -d "0.0.COORDS" -c "ANT_SBB:SBB"
```

Channelising Voltages with Digifil (TODO)

Rough overview: Produce a dada-compatible output with `lofar_udp_extractor`, copy a header to the same directory, process it with `digifil`

```
$ lofar_udp_extractor -i /path/to/input/file/udp_1613%d_2022-02-02T02:22:22.000.zst -p 10 -o  
"$INPUT_FILENAME".dada  
$ cp generic.hdr "$INPUT_FILENAME".hdr  
$ digifil -F "$NUM_CHANS":1 -d "$N_POL_OUT" -I 0 -c -b 32 -B 512 -t "$T_DOWNSAMPLE_FAC" -o  
"$OUTPUT_FILENAME".fil "$INPUT_FILENAME".dada
```

Pulsar Observations

Pulsar observations are currently a Rube Goldberg machine to generate coherently dedisperse filterbanks with a factor of 8 channelisation via `CDMT`, wrapped with a python script called `cdmtProc.py`. This guide assumes that you are observing a source in mode 5, using subbands 12:499, using the standard recording scripts to produce an output with 8x channelisation and 655us sampling.

This process assumes that your source is available in `psrcat`, or has an entry in the `dmFallback.txt` file in `/mnt/ucc3_data1/data/hdrs/` folder, and that a fake sigproc header has been generated and provided in `/mnt/ucc3_data1/data/hdrs/mode5/`. If the source is available in `psrcat`, the command to generate a fake header will be generated in the output from `cdmtProc.py`, but you will need to transfer this header to `ucc3` manually.

The latest version of `cdmtProc.py` is found at `/mnt/ucc4_data2/data/David/`, and is run using the following syntax, where the parent folder is the main folder used for recording, such as `/mnt/ucc1_recording2/data/rats/2022_02_22/`. Afterwards, a script produces an output `cdmtProc.sh` which can be run to generate both 32-bit and 8-bit filterbank files which can then be processed as needed.

```
root@AAAAAAA /mnt/ucc4_data2/data/my_processing_dir # python ../cdmtProc.py -i  
/mnt/ucc1_recording2/data/rats/to/parent/folder/  
root@AAAAAAA /mnt/ucc4_data2/data/my_processing_dir # bash cdmtProc.sh
```

Solar Observations

Please see the [section on processing data on this page for processing Solar observations](#) with `udpPacketManager` and the automated scripts.

(Archived) Handling KBT/Solar357 Observations (ScheduleKBT.py Method)

Running The Solar Observations

Kaira Background Task (KBT) is the scheduling software used to observe the Sun in mode 357 with I-LOFAR. It is currently run through a python script that handles all telescope side operations and currently has work-in-progress support to launch a beamformed data recording process on `rcu357_1beam_datastream -r` to simplify Solar observations.

Currently, all scripts related to KAIRA are stored in the `~/Scripts/kbt` folder on the `lofar`, while the scheduling script is stored in `~/Scripts/Python/obs_schedule`. In almost all cases, the only reason to enter the KBT folder will be to modify the RCU configuration in the case of downed LBA antenna or hardware failures (N.B., the `rcu` and `rcu` are not automatically synced, you will need to manually transfer these files to the `rcu` prior to taking observations for the changes to be made).

Once you are in the `~/Scripts/Python/obs_schedule` folder, you will need to run the Python 2 script `ScheduleSolarKBT_DMCK_wip.py`. Most of the flags are not needed for standard observations, and in most cases the command can be run with only the `-s` and `-t` parameters to have the station observe while the Sun is above 10 degrees elevation on the current day, starting either with the Sun reaching 10 degrees, or immediately if the Sun is already passed that altitude.

Unless you have used the `-u` flag, the script will give you a prompt to confirm the start and stop times of the observation when you initially run the command. The observation will not start until the prompt has been accepted.

Such a command would look like this:

```
$ python ScheduleSolarKBT_DMCK_wip.py -e rcu357_1beam_datastream -r
```

In the case you wish to **schedule an observation**, you will need to add the `-d` flag, and optionally start (`-t`) and stop (`-p`) times.

```
$ python ScheduleSolarKBT_DMCK_wip.py -e rcu357_1beam_datastream -r -d YYYY-MM-DD -t HH:MM -p HH:MM
$ python ScheduleSolarKBT_DMCK_wip.py -e rcu357_1beam_datastream -r -d 2022-02-22 -t 09:30 -p 14:55
```

Additionally, if you have to split up the observations, the `-u` flag may be of use, as it will allow the script to run without any input to confirm the start/end times. For example, if there are intermediate observations at 11:00 - 11:30 and 14:15 - 14:45, a `schedule_kbt.sh` script could be written like this

```
python ScheduleSolarKBT_DMCK_wip.py -u -e rcu357_1beam_datastream -r -d 2022-02-22 -p 10:55
python ScheduleSolarKBT_DMCK_wip.py -u -e rcu357_1beam_datastream -r -d 2022-02-22 -t 11:31 -p 14:10
python ScheduleSolarKBT_DMCK_wip.py -u -e rcu357_1beam_datastream -r -d 2022-02-22 -t 14:46
```

Processing the Solar Observations

When run with the `-r` flag, the `ScheduleKBT` script will launch a recording process on `ucc1` to record the raw voltage correlations from the station. Once the observations for local mode have been completed, you will need to process these voltages in order to convert them into Stokes parameters and down-sample them in order to save on space.

Do not start processing solar observation until after the station has been handed back to ASTRON. The process is entirely disk limited, and running the processing script while other data is still being recorded will cause packet loss.

This is performed by using software within a Docker container on `ucc1`. Scripts are copied to each Solar recording directory to simplify the process. To start, you will need to connect to `ucc1` and then go to your solar observations.

```
obs@ucc1~$ cd / # cd to the recording drive
obs@ucc1:/mnt/ucc1_recording/data$ cd sun/YYYYMMDD_sun357 # Enter the solar data folder for a given observation
```

Afterwards, you will want to enter a `tmux` shell and launch the Docker container to get access to the software you will need, your current directory will be stored in `$LAST_DIR` to make getting back to it easier.

```
obs@ucc1:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357$ tmux new -s solarProcessing
```



```
< enter tmux shell >
```

```
obs@ucc1:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 $ bash enter_docker.sh
```

```
< Docker container with processing software will load >
```

```
root@aaaaaaa:/home/user $ cd $LAST_DIR # Your previous directory was saved in $LAST_DIR, go back to to
root@aaaaaaa:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 #
```

You will then be able to run the processing script, which will hand processing the data, then compressing the outputs. The results can then be transferred wherever they will need to be sent to and the container and shell can be closed.

```
root@aaaaaaa:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 # bash process_data.sh # Run the processing
script
```

```
< allow processing to complete, it may take a few seconds for the initial output to appear in ./output_files/ >
```

```
< exit the container, tmux shells, notify someone to transfer the data to DIAS >
```

```
root@aaaaaaa:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 # exit
```

```
obs@ucc1:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 $ exit
```

```
obs@ucc1 ~ $ exit
```

Command Flow Summary

```
llofar@LGC~ $ cd ~/Scripts/Python/obs_schedule
```

```
llofar@LGC ~/Scripts/Python/obs_schedule $ tmux new -s KBT
```

```
< enter tmux shell >
```

```
llofar@LGC ~/Scripts/Python/obs_schedule $ python ScheduleSolarKBT_DMCK_wip.py
```

```
[TTT]-d YYYY-MM-DD \ # Date for scheduled observation
```

```
-r \ # -r will enable beamforming on ucc1
```

```
-e rcu357_1beam_datastream \ # Name of experiment (no need to change)
```

```
-t HH:MM \ # Start Time (not always needed)
```

```
-p HH:MM \ # End Time (not always needed)
```

```
# Optional
```

```
-u \ # Unattended mode, does not require confirmation of times to start script
```

< Control+b, d to exit tmux shell once things are running as intended >

< Allow all observations to complete, return later, exit the tmux shell with `exit` >

```
llofar@LGC~ $ ucc1
```

< ucc1 is an alias to connect to ucc1 >

```
obs@ucc1~ $ cdr2 # cd to the recording drive
```

```
obs@ucc1:/mnt/ucc1_recording/data $ cd sun/YYYYMMDD_sun357 # Enter the solar data folder for a given observation
```

```
obs@ucc1:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 $ tmux new -s solarProcessing
```

< enter tmux shell >

```
obs@ucc1:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 $ bash enter_docker.sh
```

< Docker container with processing software will load >

```
root@aaaaaaa:/home/user $ cd $LAST_DIR # Your previous directory was saved in $LAST_DIR, go back to to
```

```
root@aaaaaaa:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 # bash process_data.sh # Run the processing script
```

< allow processing to complete, it may take a few seconds for the initial output to appear in ./output_files/ >

< exit the container, tmux shells, notify someone to transfer the data to DIAS >

```
root@aaaaaaa:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 # exit
```

```
obs@ucc1:/mnt/ucc1_recording/data/sun/YYYYMMDD_sun357 $ exit
```

```
obs@ucc1 ~ $ exit
```

(Archived) Executing a Schedule and Processing Data

Running Observations

Automatic Configuration on the LGC

The observing process has been simplified to only require the use of a schedule file and a single command on the `LGC`. The current working directory can be accessed via the `ilofar` used on the LGC and running the alias `lsched` in the console. The current directory is

```
ilofar@LGC:~/Scripts/Python/sched_bf/tmp_dockenv$
```

Once in the directory, you can create your schedule file as described above, and execute the `scheduleAndRun.sh` with an input file name to automate transferring the scripts to the `LCU` and `DCU`, and setup the beamforming and recording processes as needed. This command does not need to be run in a `screen/tmux` and will exit early if there is an issue with the input schedule file. Issues can be debugged using the `quicksched.py` script and the `--no-write` flag.

```
bash scheduleAndRun.sh sched_YYYY-MM-DD.txt

# Run into issues? Run the quicksched.py script separately to debug the file
python3 quicksched.py sched_YYYY-MM-DD.txt --no-write
```

Manually Configuring the LCU

In order to perform your observation, you will need to transfer the `lcu_script_YYYYMMDD.sh` script to the LCU. In the case that handover has already been performed, you can simply transfer the script to the station (preferably a sub-directory in `~/local_scripts`) and run it in a screen as a normal bash script. Starting from the LGC on the `ilofar` account, you can use the `lcu` alias to open an SSH connect to the station, then you should navigate to the `~/local_scripts` folder and either use, or create, your own sub-folder for your observations.

```
llofar@lgc ~ $ lcu
user1@lcu ~ $ cd local_scripts/
user1@lcu ~/local_scripts $ mkdir my_dir; cd my_dir
```

You may need to find a copy of `sleepuntil.sh` in the parent folder if you are creating a new directory. Afterwards, you can open a screen and execute the script.

```
lcu ~/local_scripts/dmckenna $ screen -S dmckennaObs
lcu ~/local_scripts/dmckenna $ bash lcu_script_YYYYMMDD.sh
< Control a, b to exit >
```

Scheduling before local mode

However if we do not yet have the station and the observation will start before you are available for the day, there is a script on the LGC at `~/David/transfer.sh` that can take an input and transfer it to the station, then launch a screen and execute the script unattended.

The `transfer.sh` script depends on the `sleepuntil.sh` script located in the same folder, if you make a copy of `transfer.sh` to modify the target directory, be sure to copy `sleepuntil.sh` to the same folder to ensure it can run.

The `transfer.sh` script takes 3 inputs: the path of the script to transfer, the date to transfer (as YYYYMMDD) and the time to start trying to transfer (HHMMSS). As an example, to transfer a script for handover occurring at 2022-02-02T08:00:00 the following command could be used to start trying to transfer file 30 minutes prior to handover. In the case that we receive the station early, this will allow for the file to be transferred and station to be configured as soon as possible.

```
llofar@LGC ~/David $ tmux new -s transferLCuScript
< enter tmux shell>
llofar@LGC ~/David $ bash transfer.sh lcu_script_name.sh 20220202 073000
< Control d, b to exit >
```

Recording on UCC1

Given `ucc1` should always be available, the script can be transferred to the recording drive and executed as needed. We currently perform all recording on the `ucc1_recording2` drive which can be easily accessed via the `cd2` alias on the `obs` account. After that, you can get the required scripts in the scripts folder, and make a sub-directory for the given source category and perform the observation.

```

# Connect to ucc1
ilofar@LGC ~ $ ucc1

# Navigate to the recording directory
obs@ucc1 ~ $ cd r2

# Make your recording folder
obs@ucc1 /mnt/ucc1_recording2/data $ mkdir rrats/2022_02_02

# Copy the recording scripts to the folder
obs@ucc1 /mnt/ucc1_recording2/data $ cp ./scripts/* ./rrats/2022_02_02/

# Enter the folder
obs@ucc1 /mnt/ucc1_recording2/data $ cd rrats/2022_02_02

# Add your ucc_script_YYYYMMDD.sh script to the folder via rsync, scp, nano + copypaste, etc

# Open a new tmux shell
obs@ucc1 /mnt/ucc1_recording2/data/rrats/2022_02_02 $ tmux new -s recording
<attach to tmux shell>

# Run the observing script
obs@ucc1 /mnt/ucc1_recording2/data/rrats/2022_02_02 $ bash ucc_script_YYYYMMDD.sh

# Close the tmux shell after all observation have been completed.

```

Processing Observations (TODO)

These instructions assume you are in the standard processing Docker container. This can be accessed using the following command, after which you can return to your previous directory with the `$LAST_DIR` variable.

```

obs@uccN ~ $ docker run --rm -it -e TERM -v /mnt:/mnt --gpus all --env LAST_DIR=$(pwd) pulsar-gpu-dsp2021
root@AAAAA /home/soft $ cd $LAST_DIR

```

Standard Observations (TODO)

Rough overview: use `lofar_udp_extractor` to process the input to the desired output format.

```

$ lofar_udp_extractor -i /path/to/input/file/udp_1613%d_2022-02-02T02:22:22.000.zst -p "$PROCESSING_MODE" -
o "$OUTPUT_FILENAME" -a "$HEADER_PARAMETERS" -d "0,0,COORDS" -c "ANT,SBB:SBB"

```

Channelising Voltages with Digifil (TODO)

Rough overview: Produce a dada-compatible output with `lofar_udp_extractor`, copy a header to the same directory, process it with `digifil`

```
$ lofar_udp_extractor -i /path/to/input/file/udp_1613%d_2022-02-02T02:22:22.000.zst -p 10 -o  
"$INPUT_FILENAME".dada  
$ cp generic.hdr "$INPUT_FILENAME".hdr  
$ digifil -F "$NUM_CHANS":1 -d "$N_POL_OUT" -i 0 -c -b 32 -B 512 -t "$T_DOWNSAMPLE_FAC" -o  
"$OUTPUT_FILENAME".fil "$INPUT_FILENAME".dada
```

Pulsar Observations

Pulsar observations are currently a Rube Goldberg machine to generate coherently dedisperse filterbanks with a factor of 8 channelisation via `CDMT`, wrapped with a python script called `cdmtProc.py`. This guide assumes that you are observing a source in mode 5, using subbands 12:499, using the standard recording scripts to produce an output with 8x channelisation and 655us sampling.

This process assumes that your source is available in `psrcat`, or has an entry in the `dmFallback.txt` file in `/mnt/ucc3_data1/data/hdrs/` folder, and that a fake sigproc header has been generated and provided in `/mnt/ucc3_data1/data/hdrs/mode5r/`. If the source is available in `psrcat`, the command to generate a fake header will be generated in the output from `cdmtProc.py`, but you will need to transfer this header to `ucc3` manually.

The latest version of `cdmtProc.py` is found at `/mnt/ucc4_data2/data/David/`, and is run using the following syntax, where the parent folder is the main folder used for recording, such as `/mnt/ucc1_recording2/data/mats/2022_02_22/`. Afterwards, a script produces an output `cdmtProc.sh` which can be run to generate both 32-bit and 8-bit filterbank files which can then be processed as needed.

```
root@AAAAAAA /mnt/ucc4_data2/data/my_processing_dir # python ../cdmtProc.py -i  
/mnt/ucc1_recording2/data/route/to/parent/folder/  
root@AAAAAAA /mnt/ucc4_data2/data/my_processing_dir # bash cdmtProc.sh
```