

K(NR)RATOS

THE LOFAR SOFTWARE IN NURADIORECO

Current status
and future development

CURRENT STATUS OF DEVELOPMENT



CURRENT STATUS OF DEVELOPMENT



CURRENT STATUS OF DEVELOPMENT



CURRENT STATUS OF DEVELOPMENT

- Reader for TBB data
- RFI cleaner
- Galactic calibration
- Finding pulses per station
- Beamforming direction fitting



CURRENT STATUS OF DEVELOPMENT

- Reader for TBB data
  - Supports LBA INNER and OUTER
  - Also returns calibration delays
  - LORA parameters in HybridShower
- RFI cleaner
- Galactic calibration
- Finding pulses per station
- Beamforming direction fitting



CURRENT STATUS OF DEVELOPMENT

- Reader for TBB data
- RFI cleaner
  - Saves dirty channels in Station
  - Converts NRR channel numbers to TBB channel numbers
- Galactic calibration
- Finding pulses per station
- Beamforming direction fitting



CURRENT STATUS OF DEVELOPMENT

- Reader for TBB data
- RFI cleaner
- Galactic calibration
  - Absolute & relative calibration
  - Uses channel rotation to determine coefficients
- Finding pulses per station
- Beamforming direction fitting



CURRENT STATUS OF DEVELOPMENT

- Reader for TBB data
- RFI cleaner
- Galactic calibration
- Finding pulses per station
  - Looks for significant pulses in Station
  - Saves triggered status in Station
  - Saves SNR in Channel
- Beamforming direction fitting



CURRENT STATUS OF DEVELOPMENT

- Reader for TBB data
- RFI cleaner
- Galactic calibration
- Finding pulses per station
- Beamforming direction fitting
  - Iteratively improve arrival direction using significant pulses
  - Save best fit direction in Station



We also contribute to other modules in NRR

- Improvements to **channelGalacticNoiseAdder** (Karen)
- Added interpolation module called **readCoREASInterpolator** (Tiepolo & Ruben)
- Channel group ID feature to distinguish our antenna's
- Improvements to saving/reading **.nur** files and logging
- (LOFAR detector description)

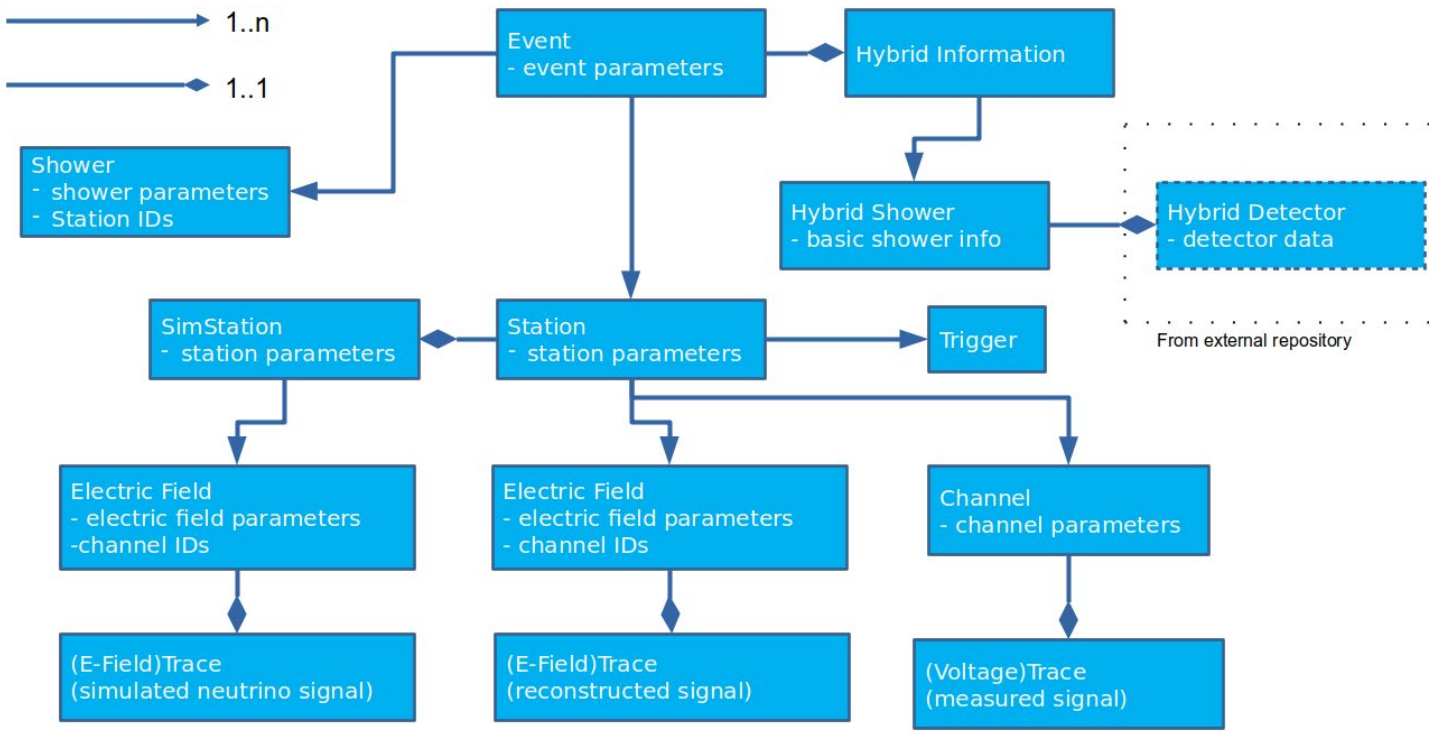
EVERYTHING IS THERE TO GET STARTED WITH ANALYSING LOFAR DATA IN NRR

What are **you** waiting for?

Buckle ‘up, it’s going to be a wild ride

AN INCOMPLETE INTRODUCTION TO NURADIORECO (NRR)

Data in NRR is stored in Event objects during processing



Quantities derived from data are stored in Shower objects

- Parameters, both input and reconstructed, are both stored in Shower objects.
- There are several types of Shower types:
  - HybridShower: here we store LORA data
  - RadioShower: here we store parameters reconstructed from radio
  - SimShower: here we store parameters of simulated showers

Modules manipulate the Event data

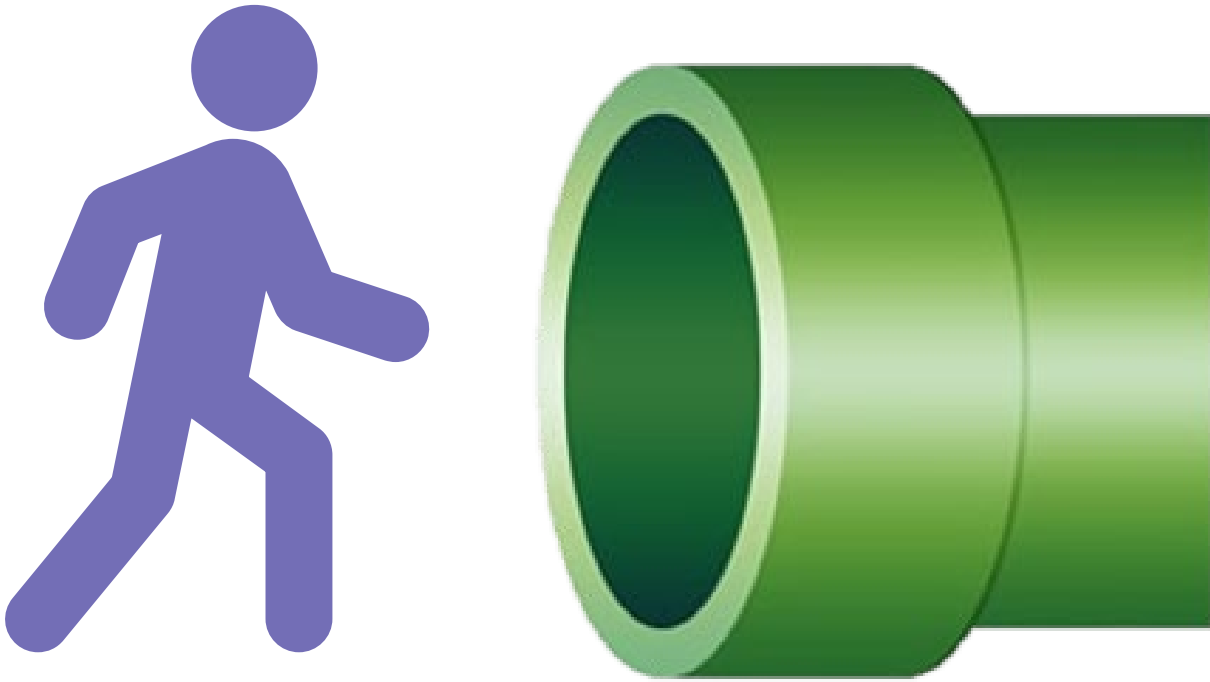
- Interaction with Event data is done through modules.
- A module is a class which implements 4 functions:
  1. **__init__()**
  2. **begin()**
  3. **run()** -> decorated using **@register_run()**
  4. **end()**
- Each module should also define it’s own logger.

Events are stored on disk in the custom NUR format

- NRR provides modules to read and write Events to disk.
- The file type is custom, called NUR files.
  - They are a binary file type, not unline HDF5.
  - They store Event parameters in their headers, allowing easy file browsing.
  - Multiple Events can be stored in one file.

The LOFAR antenna layout is stored in JSON and read in by a Detector object

- Events only store (raw) data and do not store information about your detector.
- It is the Detector which is responsible to store this physical information:
  - Location of Stations and Channels
  - Orientation of Channels
  - Antenna types (related to antenna pattern)



A WALK THROUGH THE PIPELINE

Every scripts should start with setting up the logger

```
# Setup general logging level
import logging
from NuRadioReco.utilities.logging import setup_logger
logger = setup_logger(level=logging.WARNING)
```

~~Every scripts should start with setting up the logger~~

```
# Setup general logging level
import logging
logger = logging.getLogger(“NuRadioReco.my_script”)
```

The Detector description is part of the NRR repo

```
# Load the LOFAR detector description
detector = NuRadioReco.detector.detector.Detector(
        'LOFAR/LOFAR.json’,
        source='json’,
        antenna_by_depth=False
)
```

The Detector time is set automatically during read-in

```
from NuRadioReco.modules.io.LOFAR.readLOFARData import LOFAR_event_id_to_unix

# Load the LOFAR detector description
detector = NuRadioReco.detector.detector.Detector(
        'LOFAR/LOFAR.json’,
        source='json’,
        antenna_by_depth=False
)

detector.update(Time(LOFAR_event_id_to_unix(event_id), format='unix'))
```

The Detector time is set automatically during read-in

```
from NuRadioReco.modules.io.LOFAR.readLOFARData import LOFAR_event_id_to_unix

# Load the LOFAR detector description
detector = NuRadioReco.detector.detector.Detector(
        'LOFAR/LOFAR.json’,
        source='json’,
        antenna_by_depth=False
)

detector.update(Time(LOFAR_event_id_to_unix(event_id), format='unix'))
```

The reader is configured to work out of the box on COMA

```
reader = readLOFARData.readLOFARData(
        restricted_station_set=station_set,
        tbb_directory='/mnt/coma/tbbfiles’,
        json_directory='/mnt/coma/jsonfiles’,
        metadata_directory='/mnt/coma/metadata’
)

reader.begin(event_id)
for event in reader.run(detector):
        print(event.get_id())
```

The RFI filter uses the reader to retrieve the TBB file paths

```
# Apply RFI cleaning and bandpass filters
rfi_filter = stationRFIFilter.stationRFIFilter()
rfi_filter.begin(reader=reader, rfi_cleaning_trace_length=8192)
rfi_filter.run(event)
rfi_filter.end()
```

The begin() function is called in the init function – might change

```
# Calibrate the traces based on Galactic noise
calibrator = stationGalacticCalibrator.stationGalacticCalibrator()
calibrator.run(event, detector)
calibrator.end()
```

Mapping calibration delays to channel names is on the to-do

```
# Apply the calibration delay as phase shifts
for station in event.get_stations():
        station_name = f'CS{station.get_id():03}’
        station_calibration_delays = dict(
           zip(
                map(int, reader.get_stations()[station_name]['metadata'][-2]),
                reader.get_stations()[station_name]['metadata'][-1]
           )
        )
        for channel in station.iter_channels():
                channel.apply_time_shift(
                        -1 * station_calibration_delays[channel.get_id()] * units.s
                )
```

The structure of applying modules is always the same

```
# Find stations with significant pulse and corresponding good channels
pulse_finder = stationPulseFinder.stationPulseFinder()
pulse_finder.begin()
pulse_finder.run(event, detector)
pulse_finder.end()

# Update the direction by fitting it to beamformed data
fitter = beamformingDirectionFitter_LOFAR.beamformingDirectionFitter()
fitter.begin(30)
fitter.run(event, detector)
fitter.end()
```

Saving an Event is straightforward...

```
eventWriter = NuRadioReco.modules.io.eventWriter.eventWriter()
eventWriter.begin(
        nur_save_path + f'{event_id}_filtered.nur’
)
eventWriter.run(event, detector)
eventWriter.end()
```

...as is loading it back in

```
eventReader = NuRadioReco.modules.io.eventReader.eventReader()
eventReader.begin(
        nur_save_path + f'{event_id}_filtered.nur’,
        read_detector = True
)

detector = eventReader.get_detector()
detector.update(Time(LOFAR_event_id_to_unix(event_id), format='unix’))

eventReader.run(event, detector)
eventReader.end()
```

SOME EXAMPLES TO GET YOU GOING WITH THE NRR FRAMEWORK

Check out the examples on Gitlab: Software development/example_scripts

Plotting the positions of a station

```
detector = NuRadioReco.detector.detector.Detector(
        'LOFAR/LOFAR.json’,
        source='json', antenna_by_depth=False
)
detector.update(Time(LOFAR_event_id_to_unix(127417834), format='unix’))

fig, ax = plt.subplots(1, 1, figsize=(10, 10))

for ind, station_id in enumerate(detector.get_station_ids()):
        station_pos = [detector.get_absolute_position(station_id) +
                           detector.get_relative_position(station_id, channel_id)
                       for channel_id in detector.get_channel_ids(station_id)]
        station_pos = np.array(station_pos)
        ax.scatter(station_pos[:, 0], station_pos[:, 1])
```

Adding Galactic noise to a simulated trace

```
channelGalacticNoiseAdder = channelGalacticNoiseAdder.channelGalacticNoiseAdder()
channelGalacticNoiseAdder.begin(
        skymodel='gsm2008’
)

for station in event.get_stations():
        channelGalacticNoiseAdder.run(
                event, station, det, passband=[30 * units.MHz, 80 * units.MHz]
        )
```

Interpolating HDF5 files

Converting CoREAS simulations to HDF5 files

- A Python script called “coreas_to_hdf5.py” is available in the CORSIKA **run** directory.
  - By default, this scripts needs to be run with a Python environment which contains **radiotools** (and **Numpy =< 1.20**).
    - The **Numpy** limitation is related to the use of **np.float** and such, which is now deprecated.
  - It takes in the path to the REAS file, and uses that to find all the other necessary files.
    - Problem 1: the directory structure we usually use does not work with this.
    - Problem 2: the **CorsikaParameterFile** variable is not filled in most of our REAS files.
- Solution: have a wrapper script that can

  a) modify the REAS file to have the parameter field filled, and

  b) deal with our directory stucture using e.g. symlinks.

Interpolating HDF5 files

```
reader = readCoREASInterpolator(
    lowfreq= 30.0 * units.MHz,
    highfreq= 80.0 * units.MHz,
    kind='fluence’
)
reader.begin(“/path/to/hdf5”)
event = reader.run(
    detector,
    core_xy=np.array([1.0, 2.5])
)
reader.end()
```

See also **HowToContribute.md** in the software repository

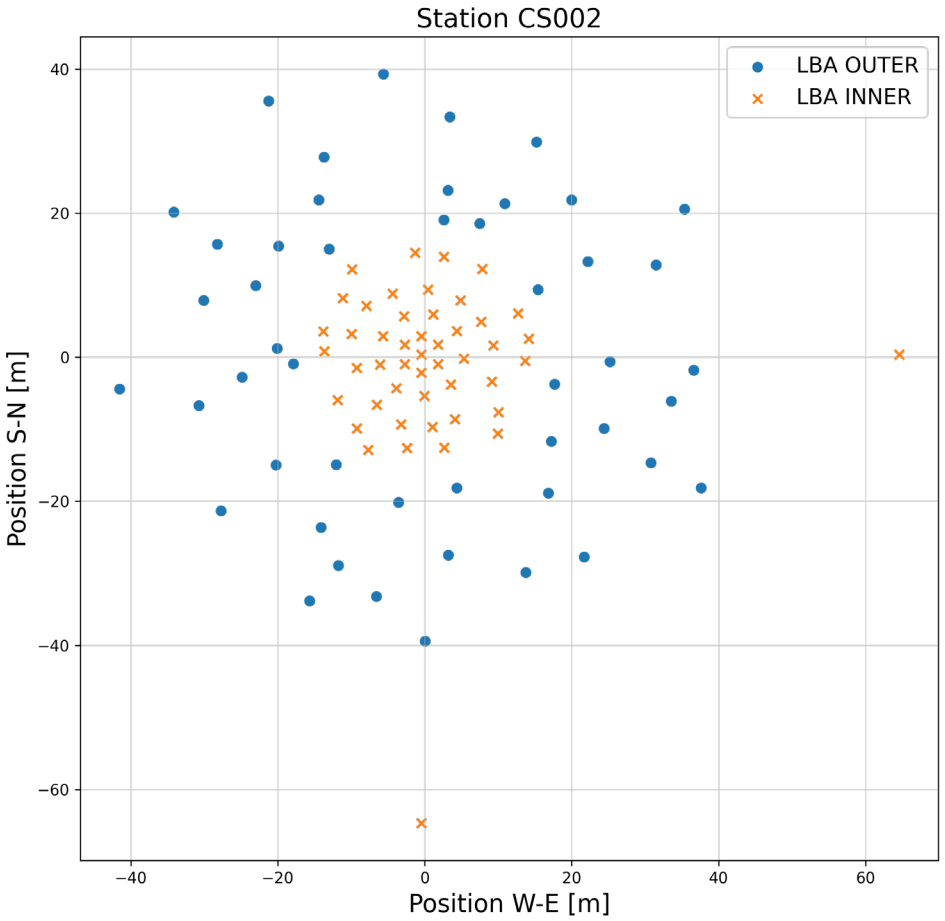
CONTRIBUTING TO THE LOFAR BRANCH OF NRR

The branch structure of the LOFAR group

- Our “main” branch is called **lofar/development**
  - On this branch we can work faster*
  - We try to merge into **develop** regularly
- If you work on a new feature for LOFAR, branch of from **lofar/development**
  - You can then open a PR like usual to merge back into our main branch
- If you need a change to the core framework, merge this into **develop** via a separate PR
  - Merging **lofar/development** into **develop** should only affect LOFAR modules

Some of the conventions we use

- Channel names:
  - ❖ For LBA OUTER, we follow the ASTRON naming
  - ❖ For LBA INNER, we changed the **4th** digit into a ‘9’ for every channel
  - ❖ We have functions to map one to the other
- Channel characteristics:
  - ❖ Our JSON file contains a field ‘ant_type’ which denotes whether it is X or Y
  - ❖ The ‘ant_mode’ denotes INNER/OUTER



Some of the conventions we use

- Channel group ID
  - ❖ Should be as antenna index
  - ❖ In the JSON file, they are the integer value of the even channel ID
    - Eg. channels “001000000” and “001000001”, have group ID 1000000
  - ❖ In the Channel object, they are converted to strings and prepended with an “a”
    - Eg. Channels 1000000 and 1000001, have group ID “a1000000”

STILL ON OUR TO-DO LIST

Ride the waves with us!

Some smoll® and smaller tasks

- Bounds in **beamformingDirectionFitter**
- Implement planeware direction fitting
- Integrating LORA into the framework
- Xmax reconstruction
- Database for events
- **Documentation**

- Calibration delays as dictionary
- Other metadata from **readLOFARData** also as dictionary
- Eliminate **mini_beamformer()**
- Add plotting functions along the pipeline
- LOFAR tab for **EventBrowser**
- **Unit tests**
- …

- Consistent directory structure for CoREAS simulations, to easily convert them to HDF5.
- Directories to store HDF5, as well as NUR files.
- How to store results together with raw TBB data?
- Unit test ?
- Where to store (custom) scripts?
- How to run the pipeline consistently?

Setting up a pipeline on COMA

“THE RADIO ANTENNA IS A BRIDGE BETWEEN REALMS. A WINDOW TO THE UNKNOWN. I WILL NOT WIELD THIS GIFT LIGHTLY. I SHALL LISTEN TO THE COSMOS, ABSORB ITS SECRETS, AND EXPAND MY UNDERSTANDING. PERHAPS I CAN FIND THE ANSWERS I’VE SOUGHT FOR SO LONG.”

Copilot, imagining to be Kratos who discovered a radio antenna amidst the desolate ruins of an ancient temple

