amstrax Documentation

Release 2.1.0

Joran Angevaare

Dec 15, 2023

SETUP AND BASICS

Github page: https://github.com/XAMS-nikhef/amstrax/

Built on top of:

- strax

CHAPTER

ONE

CONTENT

1.1 Amstrax

Processing XAMS(L) data with `amstrax`

Please notice that *amstrax* has two contexts:

- `amstrax.contexts.xams` for XAMS
- `amstrax.contexts.xams_little` for XAMSL data

We decided to use `xams_little` for XAMSL since it's otherwise too similar to `xams`.

1.1.1 Setting up amstrax

Installing *amstrax* is easy, just do:

```
pip install amstrax
```

`amstrax` is designed to be running on Nikhefs computing cluster with an SSH-tunnel to the XAMS DAQ.

To ensure this tunnel to work, there are three requirements for environment variables to be set. To ensure this is the case, please add the following sniplet (with the passwords you will need to get from a colleague to your `.bashrc`-file.

```
export DAQ_PASSWORD=<SECRET DAQ PASSWORD>
export MONGO_USER=<MONGO DATABASE USER>
export MONGO_PASSWORD=<MONGO DATABASE PASSWORD>
```

This additionally assumes you have have added your `.ssh/id_rsa.pub`-key on the DAQ machine (just add the output of `cat .ssh/id_rsa.pub` on stoomboot to the `.ssh/authorized_keys` on the DAQ machine). If you don't have an ssh key under `.ssh/id_rsa.pub` on stoomboot, google how to make one.

Straxen warnings

To make our live easier, we did not only include strax into the requirements but also straxen. This might give a few annoying warnings.

If you want to get rid of the utilix warning (which is totally irrelevant for us, you can do `touch ~/.xenon_config`).

1.2 Auto processing on stoomboot

This document describes the dataflow for XAMS and XAMSL data.

1.2.1 DAQ - `redax`

Important to note is that the data is collected with redax. `redax` runs on the "DAQ-machine". We assume that this collection of data is fully seperated from the processing (as it should be). Additionally, the DAQ host also hosts the `mongo database`. This is where we store information about the runs.

Workflow

Below we added a diagram of how the dataflow is organized. The data is read from the digitizers by redax which writes it to some local storage (`A`). From that local storage, one has to transfer this data to stoomboot, e.g. using `rsync` (step `B`) to some <`folder`> on `/data/xenon`. This folder may depend on which

As soon as the data is on stoomboot, one can auto process the data (more details below). To know which runs there are, we need to have `ssh tunnel` to the DAQ-host. We described the requirements for the ssh tunneling in the setup of this documentation.

Required user input

Right now, we did not automize step `B`. E.g. one can run this command to transfer a single run:

```
rsync -a -e ssh xams:/media/xams/Elements/xams/000932 /data/xenon/xamsl/live_data/.
```

You will notice that <`folder`> in the diagram above is `xamsl/live_data` in this example.

Processing on stoomboot (`C`)

To process the data from `xamsl/live_data` for a single run, one would have to do:

```
cd <amstrax_installation_folder>/amstrax/auto_processing
python amstraxer.py <run_id> --target raw_records_v1730 # Use --help to get more info
```

NB! This processed data to `./amstrax_data` and runs on the login node of stoomboot, you should not do this (only perhaps to test a few things).

Now, we'd like to do this a bit more automated and running on stoomboot, to this end do:

```
cd <amstrax_installation_folder>/amstrax/auto_processing
python submit_stbc.py <run_id> --target raw_records_v1730 # Use --help to get more info
```

NB! This processed data to `/data/xenon/xamsl/processing_stage` you may want to move the data somewhere else if it finished!

Automated submits

Now the real magic happens if you want to process all of it. To this end set in the rundoc

```
{'processing_status': 'pending'}
```

for all the runs you would want to process. Now, we can start making the data using:

```
cd <amstrax_installation_folder>/amstrax/auto_processing
python auto_processing.py --target desired_target
```

This will automatically submit jobs for where the processing status is `'pending'` for. It will update the runs-collection `'processing_status'` (also if it fails) so you can easily monitor the progress. ` **NB!** This processed data to `/data/xenon/xamsl/processing_stage` you may want to move the data somewhere else if it finished!

Automated submits - advanced

Let's assume all the above seemed trivial, let's go one step further.

First, let's automate step `A` by automatically transferring data as soon as it finished processing on the DAQ to stoomboot after which you will set the `'processing_status'` to `'pending'` for this run.

Next open a screen (on stoomboot): `screen -S auto_process` and do

```
cd <amstrax_installation_folder>/amstrax/auto_processing
python submit_stbc.py --run_id <run_id> --target raw_records_v1730 # Use --help to get␣
↪more info
```

exit the screen and enjoy data being automatically being processed for you. Keep in mind to not change the `'processing_status'` anymore, we will do a lot of useless jobs otherwise.

NB! This processed data to `/data/xenon/xamsl/processing_stage` you may want to move the data somewhere else if it finished!

1.2.2 What does auto processing NOT do?

There are a few things that we dit not automate, either because we think it's better if the user does it, or we do it for safety reasons (e.g. data loss due to assumed perfect code).

Auto processing:

- Does not do data management. The user is to a large degree responsible for this.
- Transfer data from the DAQ to stoomboot (see step `B`).

1.3 Writing documentation for amstrax

Keeping amstrax documented benifits new users as well as exciting.

First of all, information can be changed in `amstrax/doc`. For example this page is `amstrax/doc/writing_documentation.rst`. New pages can be included in `amstrax/doc`. If you add a new file here, don't forget to include it in `amstrax/doc/index.rst` (without the .rst, see the existing pages there).

1.3.1 Datastructure pages

In the documentation, arguably the most usefull pages are the pages of the plugin datastructure. These pages are autogenerated in `amstrax/docs/build_data_structure_doc.py`. If you want to add a new context there, you will have to add it in the `build_data_structure_doc` function. Also don't forget to add it to `amstrax/doc/index.rst`.

Testing changes to documentation and debugging

Sometimes the documentation will break and you need to test why it's not working anymore.

To do this go to `amstrax/docs` and do `bash make_docs.sh`. Hopefully this will get you started with debugging.

You might need to install a few dependencies but this is outside of the scope of this page. Just do a few conda/pip installs and hope for the best.

If you want to see the output of your build, navigate to `amstrax/docs/build/html` and open one of the `.html` files.

1.4 Amstrax - release procedure

Making releases is essential to maintaining the code in order to do this one needs to follow the following steps:

- Check which changes where made (which pull requests are merged)
- Update the `HISTORY.md` file
- Go to the `amstrax` folder and increment the version with `bumpversion patch` (or `bumpversion minor` if there are a lot of fundamental changes)
- Upload to master `git push`
- Add a tag `git push --tags; git push`
- Create a new release by clicking `Draft a new release`
- There should be a tag that you just created, select it and copy past the code that you just wrote in the `HISTORY.md`-file
- If all goes well, the new version should become available on PIPY in a few minutes

1.5 XAMSL data access

S. Di Pede, last update: 30.3.2022

By the time of writing this documentation, you can have access to the data if you activate the `amstrax_2021` environment on Stoomboot.

This documentation page uses the same structure idea of the STRAXEN documentation.

As a first step, check that you have access to the MongoDB on the xams-daq PC. Have a look at the AMSTRAX documentation and ask your colleagues for the credentials.

[1]:
```
import strax
import straxen
import amstrax
```

```
2022-03-30 14:40:05,669 - utilix - WARNING - Could not load a configuration file. You␣
↪can create one at /user/serenap/.xenon_config, or set a custom path using

export XENON_CONFIG=path/to/your/config
```

If the imports do not succeed please check that the location of the **init.py** file is here: `/data/xenon/xamsl/software/amstrax/amstrax/`

[2]:
```
import os.path
print(f'Amstrax location: {os.path.abspath(amstrax.__file__)}')
print(f'Strax location: {os.path.abspath(strax.__file__)}')
print(f'Straxen location: {os.path.abspath(straxen.__file__)}')
```

```
Amstrax location: /data/xenon/xamsl/software/amstrax/amstrax/__init__.py
Strax location: /data/xenon/xamsl/software/strax/strax/__init__.py
Straxen location: /data/xenon/xamsl/software/straxen/straxen/__init__.py
```

In `amstrax` we have two contexts for the two different detectors operated in the setup:

- XAMSL detector
- XAMS detector

and you can access the data just with the following commands:

[3]:
```
st = amstrax.contexts.xams_little()
st_xams = amstrax.contexts.xams()
```

```
[amstrax.rundb.RunDB, readonly: True, strax.storage.files.DataDirectory, readonly: True,␣
↪path: /data/xenon/xamsl/raw/, take_only: ('raw_records_v1724', 'raw_records_v1730',
↪'raw_records_aqmon'), strax.storage.files.DataDirectory, readonly: True, path: /data/
↪xenon/xamsl/processed/, strax.storage.files.DataDirectory, path: ./amstrax_data]
[amstrax.rundb.RunDB, readonly: True, strax.storage.files.DataDirectory, readonly: True,␣
↪path: /data/xenon/xams/raw/, take_only: ('raw_records_v1724', 'raw_records_v1730',
↪'raw_records_aqmon'), strax.storage.files.DataDirectory, readonly: True, path: /data/
↪xenon/xams/processed/, strax.storage.files.DataDirectory, path: ./amstrax_data]
```

The `contexts` method prints out the data storage directories:

- the run DataBase: `amstrax.rundb.RunDB`, storing the metadata
- two strax storage DataDirectory: `/data/xenon/xamsl/raw/` and `/data/xenon/xamsl/processed/`, storing, respectively, the raw data (raw_records) and all the other high-level processed data
- a general path where the data are stored in case they have not been processed yet in the above strax storage folders: `./amstrax_data`.

1.5.1 Run selection

In this documentation, we focus only on accessing the XAMSL data. To select the measurement of your interest please have a look at the details of XAMSL measurements.

To select *all* the available runs in the runDB (not only XAMSL measurements) you can use the strax method:

[4]:
```
runs = st.select_runs()
runs
```

```
Fetching run info from MongoDB: 100%|| 2574/2574 [00:00<00:00, 7309.73it/s]
```

```
Checking data availability:   0%|          | 0/2 [00:00<?, ?it/s]
```

[4]:
```
        name number                    mode                   start  \
0     000000      0                V1724_Run0 2020-11-23 08:36:02.740
1     000001      1                V1724_Run0 2020-11-24 07:51:24.798
2     000002      2                V1724_Run0 2020-11-24 12:55:19.577
3     000003      3                V1724_Run0 2020-11-24 13:08:10.073
4     000004      4                V1730_Run3 2020-11-24 13:54:39.023
...      ...    ...                       ...                    ...
2569  002569   2569  test_run_two_digitizers 2021-12-16 13:01:03.553
2570  002570   2570  test_run_two_digitizers 2021-12-16 13:02:03.861
2571  002571   2571  test_run_two_digitizers 2021-12-16 13:03:04.114
2572  002572   2572  test_run_two_digitizers 2021-12-16 13:04:04.404
2573  002573   2573  test_run_two_digitizers 2021-12-16 13:05:04.660

                        end tags               livetime processing_status  \
0    2020-11-23 08:36:50.846      0 days 00:00:48.106000               NaN
1    2020-11-24 07:51:36.827      0 days 00:00:12.029000               NaN
2    2020-11-24 12:55:28.594      0 days 00:00:09.017000               NaN
3    2020-11-24 13:08:19.093      0 days 00:00:09.020000               NaN
4    2020-11-24 13:55:00.072      0 days 00:00:21.049000               NaN
...                      ...  ...                    ...               ...
2569 2021-12-16 13:01:11.676      0 days 00:00:08.123000               NaN
2570 2021-12-16 13:02:11.974      0 days 00:00:08.113000               NaN
2571 2021-12-16 13:03:09.134      0 days 00:00:05.020000               NaN
2572 2021-12-16 13:04:09.532      0 days 00:00:05.128000           pending
2573 2021-12-16 13:05:09.685      0 days 00:00:05.025000           pending

      raw_records_v1730_available  raw_records_v1724_available
0                           False                        False
1                           False                        False
2                           False                        False
3                           False                        False
4                           False                        False
...                           ...                          ...
2569                        False                        False
2570                        False                        False
2571                        False                        False
2572                        False                        False
2573                        False                        False

[2574 rows x 10 columns]
```

1.5.2 Datastructure

The datastructure of XAMSL is different from the datastructure of XAMS. The two datastructures are kept separate to allow easier reprocessing and changes, therefore you need to keep in mind the right context and data-kind when you want to look into the data.

The available data kinds for XAMSL can be looked through the plugins class registry:

[5]:
```
plugins = st._plugin_class_registry

print(f"Which data kinds are available for XAMSL data?\n")

for datakind in plugins.keys():
    print(datakind)
```

```
Which data kinds are available for XAMSL data?

radon_records
radon_pulse_counts
radon_hits
radon_peaks
radon_lone_hits
radon_peak_basics
raw_records_v1724
raw_records_v1730
raw_records_aqmon
```

The XAMS data kinds are instead:

[6]:
```
plugins_xams = st_xams._plugin_class_registry

print(f"Which data kinds are available for XAMS data?\n")

for datakind_xams in plugins_xams.keys():
    print(datakind_xams)
```

```
Which data kinds are available for XAMS data?

records
pulse_counts
hits
peaks
lone_hits
peak_basics
raw_records_v1724
raw_records_v1730
raw_records_aqmon
```

You might have noticed that XAMSL plugins have the `radon_` keyword before the data-kind name except in the `raw_records` level.

Info about the datakind can be retrieved using:

[7]:
```
st.data_info('radon_records')
```

[7]:
```
             Field name        Data type  \
0                  time            int64
1                length            int32
2                    dt            int16
3               channel            int16
4          pulse_length            int32
5              record_i            int16
6                  area            int32
7       reduction_level            uint8
8              baseline          float32
9          baseline_rms          float32
10  amplitude_bit_shift            int16
11                 data  ('<i2', (110,))

                                          Comment
0                   Start time since unix epoch [ns]
1                  Length of the interval in samples
2                            Width of one sample [ns]
3                                 Channel/PMT number
4   Length of pulse to which the record belongs (w...
5                       Fragment number in the pulse
6                  Integral in ADC counts x samples
7   Level of data reduction applied (strax.Reducti...
8   Baseline in ADC counts. data = int(baseline) -...
9   Baseline RMS in ADC counts. data = baseline - ...
10  Multiply data by 2**(this number). Baseline is...
11  Waveform data in raw counts above integer part...
```

1.5.3 Data loading

You can load the data using the `st.get_array` function of the strax/straxen framework. Below, we load data for the measurement `000736`. You might notice that we only load `raw_records_v1730` because this was a measurement taken with the V1730 digitizer.

[8]:
```
rr = st.get_array('000736', 'raw_records_v1730',progress_bar=True)
rec = st.get_array('000736', 'radon_records',progress_bar=True)
pulse_count = st.get_array('000736', 'radon_pulse_counts',progress_bar=True)
peaks = st.get_array('000736', 'radon_peaks',progress_bar=True)
lone_hits = st.get_array('000736', 'radon_lone_hits',progress_bar=True)
peak_basics = st.get_array('000736', 'radon_peak_basics',progress_bar=True)
```

```
Loading raw_records_v1730: |          | 0.00 % [00:00<?]
```

```
Loading radon_records: |          | 0.00 % [00:00<?]
```

```
Loading radon_pulse_counts: |          | 0.00 % [00:00<?]
```

```
Loading radon_peaks: |          | 0.00 % [00:00<?]
```

```
Loading radon_lone_hits: |          | 0.00 % [00:00<?]
```

```
Loading radon_peak_basics: |          | 0.00 % [00:00<?]
```

1.5.4 Details about raw_records in XAMSL

As you can see in this wiki-page, you will find you raw data only in one of the `raw_records` type and only in specific channels. Let's take as an example the measurement `000282` and `000736`, which have been taken respectively with the V1724 and V1730 digitizers.

[9]:
```
rr_000736_v1724=st.get_array('000736', 'raw_records_v1724',progress_bar=True)
rr_000282_v1724=st.get_array('000282', 'raw_records_v1724',progress_bar=True)
rr_000282_v1730=st.get_array('000282', 'raw_records_v1730',progress_bar=True)
rec_000282=st.get_array('000282', 'radon_records',progress_bar=True)
```

```
Loading raw_records_v1724: |          | 0.00 % [00:00<?]
Loading raw_records_v1724: |          | 0.00 % [00:00<?]
Loading raw_records_v1730: |          | 0.00 % [00:00<?]
Loading radon_records: |          | 0.00 % [00:00<?]
```

We do expect to see data in: * channel 0, 1 in measurement `000736` of `raw_records_v1730` and their respective records * channel 2, 3 in measurement `000282` of `raw_records_v1724` and their respective records

All the other channels must be empty. Let's check.

[10]:
```
print(f'Measurement 000736, raw_records_v1730')
for ch in range(8):
    rr_ch=rr[rr['channel']==ch]
    print(f'Channel {ch}, data:')
    print(rr_ch['data'])

print(f'\nMeasurement 000736, raw_records_v1724')
for ch in range(8):
    rr_ch=rr_000736_v1724[rr_000736_v1724['channel']==ch]
    print(f'Channel {ch}, data:')
    print(rr_ch['data'])
```

```
Measurement 000736, raw_records_v1730
Channel 0, data:
[[8220 8222 8225 ... 7530 7654 7743]
 [7777 7808 7851 ... 8220 8222 8220]
 [8223 8220 8222 ... 8221 8222 8224]
 ...
 [8222 8221 8225 ... 8224 8219 8222]
 [8221 8227 8225 ... 8223 8225 8223]
 [8225 8223 8225 ...    0    0    0]]
Channel 1, data:
[[8199 8200 8200 ... 8012 8003 8025]
 [8077 8102 8098 ... 8199 8199 8203]
 [8203 8204 8205 ... 8199 8199 8202]
 ...
 [8199 8202 8200 ... 8201 8199 8198]
 [8202 8199 8201 ... 8201 8200 8201]
 [8199 8202 8199 ...    0    0    0]]
Channel 2, data:
[]
Channel 3, data:
[]
```

(continues on next page)

(continued from previous page)

```
Channel 4, data:
[]
Channel 5, data:
[]
Channel 6, data:
[]
Channel 7, data:
[]

Measurement 000736, raw_records_v1724
Channel 0, data:
[]
Channel 1, data:
[]
Channel 2, data:
[]
Channel 3, data:
[]
Channel 4, data:
[]
Channel 5, data:
[]
Channel 6, data:
[]
Channel 7, data:
[]
```

[11]:
```
print(f'Measurement 000282, raw_records_v1730')
for ch in range(8):
    rr_ch=rr_000282_v1730[rr_000282_v1730['channel']==ch]
    print(f'Channel {ch}, data:')
    print(rr_ch['data'])

print(f'\nMeasurement 000282, raw_records_v1724')
for ch in range(8):
    rr_ch=rr_000282_v1724[rr_000282_v1724['channel']==ch]
    print(f'Channel {ch}, data:')
    print(rr_ch['data'])
```

```
Measurement 000282, raw_records_v1730
Channel 0, data:
[]
Channel 1, data:
[]
Channel 2, data:
[]
Channel 3, data:
[]
Channel 4, data:
[]
Channel 5, data:
[]
```

(continues on next page)

(continued from previous page)

```
Channel 6, data:
[]
Channel 7, data:
[]

Measurement 000282, raw_records_v1724
Channel 0, data:
[]
Channel 1, data:
[]
Channel 2, data:
[[16082 16082 16082 ... 16077 16079 16081]
 [16076 16077 16078 ...     0     0     0]
 [16083 16083 16082 ...     0     0     0]
 ...
 [16080 16083 16081 ...     0     0     0]
 [16083 16081 16082 ...     0     0     0]
 [16077 16077 16078 ...     0     0     0]]
Channel 3, data:
[[16099 16099 16098 ...     0     0     0]
 [16085 16084 16086 ...     0     0     0]
 [16100 16098 16098 ... 15994 16020 15965]
 ...
 [16100 16095     0 ...     0     0     0]
 [16014 16033 15998 ...     0     0     0]
 [16094 16093 16093 ... 16078     0     0]]
Channel 4, data:
[]
Channel 5, data:
[]
Channel 6, data:
[]
Channel 7, data:
[]
```

The same rule stands for the `radon_records` of the specific measurement.

[12]:
```
print(f'Measurement 000736')
for ch in range(8):
    rr_ch=rec[rec['channel']==ch]
    print(f'Channel {ch}, data:')
    print(rr_ch['data'])

print(f'\nMeasurement 000282')
for ch in range(8):
    rr_ch=rec_000282[rec_000282['channel']==ch]
    print(f'Channel {ch}, data:')
    print(rr_ch['data'])
```

```
Measurement 000736
Channel 0, data:
[[  0   0   0 ... 692 568 479]
 [445 414 371 ...   0   0   0]
```

(continues on next page)

(continued from previous page)

```
 [  0   0   0 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
Channel 1, data:
[[  0   0   0 ... 188 197 175]
 [123  98 102 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 ...
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]]
Channel 2, data:
[]
Channel 3, data:
[]
Channel 4, data:
[]
Channel 5, data:
[]
Channel 6, data:
[]
Channel 7, data:
[]

Measurement 000282
Channel 0, data:
[]
Channel 1, data:
[]
Channel 2, data:
[[0 0 0 ... 5 3 1]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
Channel 3, data:
[[  0   0   0 ...   0   0   0]
 [  0   0   0 ...   0   0   0]
 [  0   0   0 ... 102  76 131]
 ...
 [  0   0   0 ...   0   0   0]
 [ 40  21  56 ...   0   0   0]
 [  0   0   0 ...  18   0   0]]
Channel 4, data:
[]
Channel 5, data:
[]
Channel 6, data:
```

(continues on next page)

(continued from previous page)

```
[]
Channel 7, data:
[]
```

1.5.5 Details of the XAMSL measurements

You can also load the details of the XAMSL measurements inside your working Jupyter Notebook during your analysis. The amstrax_files contain informations of the lab-logbook such as start/stop of the recirculation.

[13]:
```
import amstrax_files
```

This is the list of the files contained in the `amstrax_files` at the time of writing this documentation.

[14]:
```
print(amstrax_files.list_files())
```

```
['999999.tar', 'example.json', 'rundoc_999999.json', 'xamsl_measurements.csv']
```

Load the `xamsl_measurements` details file

[15]:
```
measurements = amstrax_files.get_file('xamsl_measurements.csv')
```

Select the range of measurements you are interested in e.g. the recirculation and purification effect investigation with the V1730 digitizer during stable HV conditions on the PMTs.

[16]:
```
moi = measurements.loc[(measurements['run'] >= 1588) & (measurements['run'] <= 1952)]
moi
```

[16]:

	run	pmt_bottom_v	pmt_top_v	notes
1219	1588	-560	-590	reci on/getter on-digi th 30ADC
1220	1589	-560	-590	reci on/getter on-digi th 30ADC
1221	1590	-560	-590	reci on/getter off-digi th 30ADC
1222	1591	-560	-590	reci on/getter off-digi th 30ADC
1223	1592	-560	-590	reci on/getter off-digi th 30ADC
...	...	...	...	...
1579	1948	-560	-590	reci on/getter off-digi th 30ADC
1580	1949	-560	-590	reci on/getter off-digi th 30ADC
1581	1950	-560	-590	reci on/getter off-digi th 30ADC
1582	1951	-560	-590	reci on/getter off-digi th 30ADC
1583	1952	-560	-590	reci on/getter on-digi th 30ADC

[365 rows x 4 columns]

[17]:
```
moi['notes']
```

[17]:
```
1219      reci on/getter on-digi th 30ADC
1220      reci on/getter on-digi th 30ADC
1221     reci on/getter off-digi th 30ADC
1222     reci on/getter off-digi th 30ADC
1223     reci on/getter off-digi th 30ADC
                    ...
1579     reci on/getter off-digi th 30ADC
1580     reci on/getter off-digi th 30ADC
1581     reci on/getter off-digi th 30ADC
```

(continues on next page)

(continued from previous page)

```
1582     reci on/getter off-digi th 30ADC
1583      reci on/getter on-digi th 30ADC
Name: notes, Length: 365, dtype: object
```

1.5.6 Data not available! -> You can process data by yourself!

If, for some reason, you receive a message that data are not available, you can try to make the data by yourself.

Specify the run number you want to make data of.

[ ]:
```
run_list = ['','',...]
```

And the target data-kind you need

[ ]:
```
target = '...'
```

Then just look over and use the strax function `st.make`.

[ ]:
```
from strax.utils import tqdm

for r in tqdm(run_list):
    print(r)
    st.make(r,target,progress_bar=True)
```

As we said before, these new processed data will be saved in the storage directory `./amstrax_data`. You don't need to remake the data every time as strax will know now where to find them.

1.5.7 Veel plezier met XAMSL data!

1.6 XAMS - amstrax datastructure

This page is an autogenerated reference for all the plugins in `amstrax.context.xams-context`.

Colors indicate data kinds. To load tables with different data kinds, you currently need more than one *get_df* (or *get_array*) commands.

1.6.1 event_info

Description

Provided by plugin: EventInfo

Data kind: events

(no plugin description)

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.2 corrected_areas

Description

Provided by plugin: CorrectedAreas

Data kind: events

Plugin which applies light collection efficiency maps and electron life time to the data.

Computes the cS1/cS2 for the main/alternative S1/S2 as well as the corrected life time. Note:

Please be aware that for both, the main and alternative S1, the area is corrected according to the xy-position of the main S2. There are now 3 components of cS2s: cs2_top, cS2_bottom and cs2. cs2_top and cs2_bottom are corrected by the corresponding maps, and cs2 is the sum of the two.

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.3 event_area_per_channel

Description

Provided by plugin: EventAreaPerChannel

Data kind: events

Simple plugin that provides area per channel for main and alternative S1/S2 in the event.

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.4 event_n_channel

Description

Provided by plugin: EventAreaPerChannel

Data kind: events

Simple plugin that provides area per channel for main and alternative S1/S2 in the event.

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.5 event_positions

Description

Provided by plugin: EventPositions

Data kind: events

> Computes the observed and corrected position for the main S1/S2 pairs in an event. For XENONnT data, it returns the FDC corrected positions of the default_reconstruction_algorithm. In case the fdc_map is given as a file (not through CMT), then the coordinate system should be given as (x, y, z), not (x, y, drift_time).

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.6 event_waveform

Description

Provided by plugin: EventWaveform

Data kind: events

Simple plugin that provides total (data) and top (data_top) waveforms for main and
alternative S1/S2 in the event.

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.7 event_basics

Description

Provided by plugin: EventBasics

Data kind: events

Computes the basic properties of the main/alternative S1/S2 within an event.

The main S1 and alternative S1 are given by the largest two S1-Peaks within the event. The main S2 is given by the largest S2-Peak within the event, while alternative S2 is selected as the largest S2 other than main S2 in the time window [main S1 time, main S1 time + max drift time].

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.8 peak_positions

Description

Provided by plugin: PeakPositions

Data kind: peaks

(no plugin description)

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.9 events

Description

Provided by plugin: Events

Data kind: events

(no plugin description)

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.10 peak_basics

Description

Provided by plugin: PeakBasics

Data kind: peaks

(no plugin description)

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.11 peaks

Description

Provided by plugin: Peaks

Data kind: peaks

(no plugin description)

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.12 peak_basics_ext

Description

Provided by plugin: PeakBasicsEXT

Data kind: peaks_ext

(no plugin description)

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.13 records

Description

Provided by plugin: PulseProcessing

Data kind: records

> Get the specific raw_records of the measurements (raw_records_v1724 or raw_records_v1730) and split the raw_records into: - records - pulse_counts Apply basic pulse processing:
>
> 1. Flip the pulse if it is necessary (only for the PMT pulse)
> 2. Calculate the baseline and integrate the waveform
> 3. Find hits

4. Filter the record and cut outside the hit bounds

pulse_counts holds some average information for the individual PMT channels for each chunk of raw_records. This includes e.g. number of recorded pulses, lone_pulses (pulses which do not overlap with any other pulse), or mean values of baseline and baseline rms channel.

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.14 pulse_counts

Description

Provided by plugin: PulseProcessing

Data kind: pulse_counts

Get the specific raw_records of the measurements (raw_records_v1724 or raw_records_v1730) and split the raw_records into: - records - pulse_counts Apply basic pulse processing:

1. Flip the pulse if it is necessary (only for the PMT pulse)
2. Calculate the baseline and integrate the waveform
3. Find hits
4. Filter the record and cut outside the hit bounds

pulse_counts holds some average information for the individual PMT channels for each chunk of raw_records. This includes e.g. number of recorded pulses, lone_pulses (pulses which do not overlap with any other pulse), or mean values of baseline and baseline rms channel.

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.15 peaks_ext

Description

Provided by plugin: PeaksEXT

Data kind: peaks_ext

(no plugin description)

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.16 records_ext

Description

Provided by plugin: PulseProcessingEXT

Data kind: records_ext

> Plugin which performs the pulse processing steps: 1. Baseline subtraction 2. Pulse splitting 3. Pulse merging 4. Pulse counting 5. Pulse length and area calculation

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.17 raw_records_ext

Description

Provided by plugin: DAQReader

Data kind: raw_records_ext

Read the XENONnT DAQ-live_data from redax and split it to the appropriate raw_record data-
types based on the channel-map.

Does nothing whatsoever to the live_data; not even baselining.

Provides:

- raw_records: (tpc)raw_records.

- raw_records_ext: (external)raw_records.

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.6.18 raw_records

Description

Provided by plugin: DAQReader

Data kind: raw_records

Read the XENONnT DAQ-live_data from redax and split it to the appropriate raw_record data-
types based on the channel-map.

Does nothing whatsoever to the live_data; not even baselining.

Provides:

- raw_records: (tpc)raw_records.
- raw_records_ext: (external)raw_records.

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.7 XAMSL - amstrax datastructure

This page is an autogenerated reference for all the plugins in `amstrax.context.xams_little-context`.

Colors indicate data kinds. To load tables with different data kinds, you currently need more than one *get_df* (or *get_array*) commands.

1.7.1 raw_records_v1724

Description

Provided by plugin: DAQReader

Data kind: raw_records_v1724

> Read the XAMS DAQ-live_data from redax and split it to the appropriate raw_record data-types based on the channel-map. Does nothing whatsoever to the live_data; not even baselining. Provides:
>
> - raw_records_v1724, sampled from the V1724 digitizer with sampling resolution = 10ns
> - raw_records_v1730, sampled from the V1730 digitizer with sampling resolution = 2ns
> - raw_records_aqmon, actually empty unless we need some strax-deadtime

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.7.2 raw_records_v1730

Description

Provided by plugin: DAQReader

Data kind: raw_records_v1730

> Read the XAMS DAQ-live_data from redax and split it to the appropriate raw_record data-types based on the channel-map. Does nothing whatsoever to the live_data; not even baselining. Provides:
>
> - raw_records_v1724, sampled from the V1724 digitizer with sampling resolution = 10ns
> - raw_records_v1730, sampled from the V1730 digitizer with sampling resolution = 2ns
> - raw_records_aqmon, actually empty unless we need some strax-deadtime

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.7.3 raw_records_aqmon

Description

Provided by plugin: DAQReader

Data kind: raw_records_aqmon

Read the XAMS DAQ-live_data from redax and split it to the appropriate raw_record data-types based on the channel-map. Does nothing whatsoever to the live_data; not even baselining. Provides:

- raw_records_v1724, sampled from the V1724 digitizer with sampling resolution = 10ns
- raw_records_v1730, sampled from the V1730 digitizer with sampling resolution = 2ns
- raw_records_aqmon, actually empty unless we need some strax-deadtime

Columns provided

Dependencies

Configuration options

These are all options that affect this data type. This also includes options taken by dependencies of this datatype, because changing any of those options affect this data indirectly.

1.8 amstrax package

1.8.1 Subpackages

amstrax.auto_processing package

Submodules

amstrax.auto_processing.amstraxer module

Process a single run with amstrax

amstrax.auto_processing.amstraxer.**main**(*args*)

amstrax.auto_processing.amstraxer.**parse_args**()

amstrax.auto_processing.amstraxer.**to_dict_tuple**(*res: dict*)

Convert list configs to tuple configs

amstrax.auto_processing.auto_processing module

amstrax.auto_processing.auto_processing.**parse_args**()

amstrax.auto_processing.copy_live module

amstrax.auto_processing.copy_live.**copy_data**(*run_id*, *live_data_path*, *location*, *hostname*, *production*, *ssh_host*)

Copy data to the specified location using rsync.

amstrax.auto_processing.copy_live.**get_rundocs**(*runsdb*, *args*)

Retrieve run documents from MongoDB collection based on specific criteria.

amstrax.auto_processing.copy_live.**handle_runs**(*rundocs*, *args*)

amstrax.auto_processing.copy_live.**main**(*args*)

Main function to automate copying of new runs.

amstrax.auto_processing.copy_live.**parse_args**()

amstrax.auto_processing.process_run module

amstrax.auto_processing.submit_stbc module

amstrax.auto_processing.submit_stbc.**parse_args**()

amstrax.auto_processing.submit_stbc.**submit_job**(*run_id*, *target*, *context*, *detector*, *job_folder='jobs'*, *log_folder='logs'*, *script='amstraxer_easy'*)

This script will save data to

Module contents

amstrax.plugins package

Submodules

amstrax.plugins.daqreader module

class amstrax.plugins.daqreader.**DAQReader**

Bases: Plugin

Read the XENONnT DAQ-live_data from redax and split it to the appropriate raw_record data- types based on the channel-map.

Does nothing whatsoever to the live_data; not even baselining.

Provides:

- raw_records: (tpc)raw_records.
- raw_records_ext: (external)raw_records.

chunk_target_size_mb = 50

compressor = 'lz4'

compute(*chunk_i*)

data_kind: Union[str, immutabledict, dict] = immutabledict({'raw_records_ext': 'raw_records_ext', 'raw_records': 'raw_records'})

depends_on: Tuple = ()

infer_dtype()

Return dtype of computed data; used only if no dtype attribute defined

input_timeout = 300

is_ready(*chunk_i*)

Return whether the chunk chunk_i is ready for reading. Returns True by default; override if you make an online input plugin.

parallel = 'process'

provides: Tuple[str, ...] = ('raw_records_ext', 'raw_records')

rechunk_on_save = immutabledict({'raw_records': False, 'raw_records_ext': False})

setup()

Hook if plugin wants to do something on initialization

source_finished()

Return whether all chunks the plugin wants to read have been written. Only called for online input plugins.

takes_config = immutabledict({'record_length': <strax.config.Option object>, 'max_digitizer_sampling_time': <strax.config.Option object>, 'run_start_time': <strax.config.Option object>, 'daq_chunk_duration': <strax.config.Option object>, 'daq_overlap_chunk_duration': <strax.config.Option object>, 'daq_compressor': <strax.config.Option object>, 'readout_threads': <strax.config.Option object>, 'daq_input_dir': <strax.config.Option object>, 'safe_break_in_pulses': <strax.config.Option object>, 'channel_map': <strax.config.Option object>})

amstrax.plugins.daqreader.**split_channel_ranges**(*records*, *channel_ranges*)

Return numba.List of record arrays in channel_ranges.

channel_ranges is a list of tuples specifying the channel ranges for each subdetector.

amstrax.plugins.event_processing module

amstrax.plugins.pax_interface module

amstrax.plugins.peak_processing module

amstrax.plugins.pulse_processing module

class amstrax.plugins.pulse_processing.**PulseProcessing**

Bases: Plugin

Get the specific raw_records of the measurements (raw_records_v1724 or raw_records_v1730) and split the raw_records into: - records - pulse_counts Apply basic pulse processing:

1. Flip the pulse if it is necessary (only for the PMT pulse)
2. Calculate the baseline and integrate the waveform
3. Find hits
4. Filter the record and cut outside the hit bounds

pulse_counts holds some average information for the individual PMT channels for each chunk of raw_records. This includes e.g. number of recorded pulses, lone_pulses (pulses which do not overlap with any other pulse), or mean values of baseline and baseline rms channel.

compressor = 'zstd'

compute(*raw_records*, *start*, *end*)

config: Dict

data_kind: Union[str, immutabledict, dict] = {'pulse_counts': 'pulse_counts', 'records': 'records'}

depends_on: tuple = 'raw_records'

deps: Dict

dtype: Union[tuple, dtype, immutabledict, dict]

infer_dtype()

Return dtype of computed data; used only if no dtype attribute defined

input_buffer: Dict[str, Chunk]

parallel = 'process'

provides: tuple = ('records', 'pulse_counts')

rechunk_on_save = immutabledict({'records': False, 'pulse_counts': True})

run_i: int

run_id: str

takes_config = immutabledict({'baseline_samples': <strax.config.Option object>, 'pmt_pulse_filter': <strax.config.Option object>, 'save_outside_hits': <strax.config.Option object>, 'n_tpc_pmts': <strax.config.Option object>, 'check_raw_record_overlaps': <strax.config.Option object>, 'allow_sloppy_chunking': <strax.config.Option object>, 'hit_min_amplitude': <strax.config.Option object>})

amstrax.plugins.pulse_processing.**baseline_per_channel**(*records*, *baseline_samples=40*, *flip=False*, *allow_sloppy_chunking=False*, *fallback_baseline=16000*)

Determine baseline as the average of the first baseline_samples of each pulse in each channel. Subtract the pulse data from int(baseline), and store the baseline mean and rms.

Parameters
baseline_samples – number of samples at start of pulse to average

to determine the baseline. :param flip: If true, flip sign of data (only for PMT data) :param allow_sloppy_chunking: Allow use of the fallback_baseline in case the 0th fragment of a pulse is missing :param fallback_baseline: Fallback baseline (ADC counts)

Assumes records are sorted in time (or at least by channel, then time).

Assumes record_i information is accurate – so don't cut pulses before baselining them!

amstrax.plugins.pulse_processing.**channel_split**(*rr*, *first_other_ch*)

Return

amstrax.plugins.pulse_processing.**check_overlaps**(*records*, *n_channels*)

Raise a ValueError if any of the pulses in records overlap Assumes records is already sorted by time.

amstrax.plugins.pulse_processing.**mask_and_not**(*x*, *mask*)

amstrax.plugins.pulse_processing.**pulse_count_dtype**(*n_channels*)

amstrax.plugins.pulse_processing_alt_baseline module

Module contents

1.8.2 Submodules

1.8.3 amstrax.SiPMdata module

1.8.4 amstrax.common module

amstrax.common.**get_config_defaults**(*st*, *exclude=['raw_records', 'records']*, *include=[]*)

amstrax.common.**get_elife**(*run_id*)

Return electron lifetime for run_id in ns

amstrax.common.**print_versions**(*modules=('strax', 'amstrax')*, *print_output=True*, *include_python=True*, *return_string=False*, *include_git=True*)

Print versions of modules installed. :param modules: Modules to print, should be str, tuple or list. E.g.

print_versions(modules=('numpy', 'dddm',))

Parameters

- **return_string** – optional. Instead of printing the message, return a string
- **include_git** – Include the current branch and latest commit hash

Returns
optional, the message that would have been printed

amstrax.common.**select_channels**(*arr*, *channel_list*)

Select only the values in arr that have arr['channel'] in channel_list

1.8.5 amstrax.contexts module

amstrax.contexts.**context_for_daq_reader**(*st: Context*, *run_id: str*, *detector: str = 'xams'*, *runs_col_kwargs: Optional[dict] = None*, *run_doc: Optional[dict] = None*, *check_exists=True*)

Given a context and run_id, change the options such that we can process the live data.

IMPORTANT: After setting the context, we specify the location of the live-data for a single run. This means you CANNOT re-use this context! Therefore, if you want to process data, you should start a new context if you want to process another run starting from the live data

Parameters

- **st** – Context to change
- **run_id** – the run_id of the run that should be processed
- **runs_col_kwargs** – Optional options (kwargs) for starting the run-collection, see *get_mongo_collection*
- **run_doc** – Optional document associated with this run-id.

Returns

Context ready to start processing <run_id> with from the live-data

amstrax.contexts.**xams**(*output_folder='./strax_data'*, *init_rundb=True*, *mongo_kwargs={'mongo_collname': 'runs_gas', 'mongo_dbname': 'run', 'runid_field': 'number'}*, **args*, ***kwargs*)

amstrax.contexts.**xams_led**(***kwargs*)

1.8.6 amstrax.hitfinder_thresholds module

amstrax.hitfinder_thresholds.**hit_min_amplitude**(*model*, *n_tpc_pmts=16*)

Return hitfinder height threshold to use in processing.

Parameters

model – Model name (str), or int to use a uniform threshold,

or array/tuple or thresholds to use. :param threshold: value of the threshold to be applied in ADC counts.

1.8.7 amstrax.itp_map module

1.8.8 amstrax.rundb module

class amstrax.rundb.**RunDB**(*mongo_dbname='admin'*, *mongo_collname='runs_gas'*, *runid_field='name'*, *local_only=True*, *new_data_path=None*, *reader_ini_name_is_mode=False*, *readonly=True*, **args*, ***kwargs*)

Bases: `StorageFrontend`

Frontend that searches RunDB MongoDB for data.

Loads appropriate backends ranging from Files to S3.

backends: `list`

find_several(*keys: List[DataKey]*, ***kwargs*)

Return list with backend keys or False for several data keys.

Options are as for find()

hosts = {'dali': '^dali.*rcc.*'}

provide_run_metadata = True

run_metadata(*run_id*, *projection=None*)

Return run metadata dictionary, or raise RunMetadataNotAvailable

amstrax.rundb.**get_mongo_client**(*daq_host: str = ''*, *daq_user: str = ''*, *secret_serving_port: Dict[str, Any] = {}*) → MongoClient

Get a MongoDB client.

amstrax.rundb.**get_mongo_collection**(*detector: str = 'xams'*, *runcolname: str = 'run'*, ***link_kwargs*) → Collection

Get a specific MongoDB collection based on the detector.

1.8.9 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX

H

I

M

P

R

S

T

X