reprox Documentation

Release 0.2.2

J. R. Angevaare

Jul 12, 2022

SETUP AND BASICS

1 Content **3**
- 1.1 Setting up reprox . . . 3
- 1.2 Reprocessing on dali . . . 3
- 1.3 Advanced usage . . . 5
- 1.4 reprox package . . . 7

2 Indices and tables **13**

Python Module Index **15**

Index **17**

Github page: https://github.com/XENONnT/reprox

CHAPTER

ONE

CONTENT

1.1 Setting up reprox

One can either use a frozen installation:

Install using: `pip install reprox`

or, as recommended, use the developer installation:

```
git clone git@github.com:XENONnT/reprox.git
pip install -e reprox
```

1.2 Reprocessing on dali

Process data in so far available on dali with the current container

1.2.1 Logic

There are several (sequential) steps with (associated scripts):

- Step 1. Find runs to process (`reprox-find-data`)
- Step 2. Process the runs that were found (`reprox-start-jobs`)
- Step 3. Move the data that was processed to the desired folder (`reprox-move-folders`)

One can also run these three steps from one file (`reprox-reprocess`), which runs all three in order.

The best place to start is by going over these files and do `reprox-find-data --help` to see which options there are. Most are discussed below.

1.2.2 Running step by step

Below, we show how these three steps are done. This can also be done in one command skip to single command.

Step 0 - Activation and test installation

You only have to do it once, to prevent confusion we will go over it step by step.

First, activate a container (NB! the singularity containers do not work as they cannot communicate with the job submission of dali).

```
source /cvmfs/xenon.opensciencegrid.org/releases/nT/development/setup.sh
git clone git@github.com:XENONnT/reprox.git
pip install -e reprox --user
```

test that the installation is complete and successful

```
reprox-find-data --help
```

Trouble-shooting

Now, the commands above may sometimes not work as expected due to permission errors on the containers. If there is an error, you could see `reprox-find-data: command not found`. If this is the case, simply navigate to the `bin` folder of `reprox` and run the commands as below:

```
cd repox/bin
python reprox-find-data --help
```

The other `reprox` scripts are similarly located in the `bin` folder. If you had to change this once, you have to do `python <script>` for all the scripts listed below.

Step 1 - finding data to (re)process on dali

Now we have to know which data to process, this can be done with the following command. Determine which data to process:

```
reprox-find-data \
    --package cutax \
    --context xenonnt_v6
    --target event_info event_pattern_fit cuts_basic \
    --cmt-version global_v6
```

The `--package` and `--context` arguments specify where to load the context from (`straxen/cutax`) and which context to use. In this example, we use `xenonnt_v6`. The `--target` argument specifies which datatypes to produce. This can be a list as in the example above. We will check if the datatypes can be produced for this given context. Since some context may use a global CMT version that is only valid for a range of runs, the `--cmt-version` is specified separately and tells the script to only process runs that are valid in this `cmt_version`. This can be disabled using `--cmt-version False` (for example, you know that the CMT version is always valid for the datatypes you requested).

This takes a while (+/- 30 minutes) and writes a file called `/dali/lgrandi/xenonnt/data_management_reprocessing/to_do_runs.csv` (depending on your ini file). This file has a list of runs that you can process given the options as above.

Step 2 - starting the jobs to process the data

After producing `/dali/lgrandi/xenonnt/data_management_reprocessing/to_do_runs.csv`, we need to submit the jobs to process the data. Most of the arguments are the same as above, we now also specify some self-explanatory arguments for the jobs to be submitted.

```
reprox-start-jobs \
    --package cutax \
    --context xenonnt_v6 \
    --target event_info event_pattern_fit cuts_basic \
    --ram 12000 \
    --cpu 2
```

Step 3 - move to the production folder

Now, hopefully most of the data has been processed successfully, we can now move it to the production folder. This includes a check to see if the data was processed successfully so even if a few jobs failed (or are still running), you can safely run this command below.

```
reprox-move-folders
```

Run entire workflow (steps 1-3 in a single command)

You can also do all the above in a single command, using the same arguments (see above for explanation of each.).

```
reprox-reprocess \
    --package cutax \
    --context xenonnt_v6 \
    --target event_info event_pattern_fit cuts_basic \
    --cmt-version global_v6 \
    --ram 12000 \
    --cpu 2 \
    --move-after-workflow # To move the data into the production folder
```

1.3 Advanced usage

Below are several more advanced use cases.

1.3.1 Changing the defaults of processing

You might want to play with the config file that says how many resources one uses by default. The reprocessing.ini file. You can either change the source code of this file, or you can overwrite it as follows:

```
git clone git@github.com:XENONnT/reprox.git
cp reprox/reprox/reprocessing.ini my_reprocessing_config.ini

# # Edit my_reprocessing_config.ini. For example using vim:
# vi my_reprocessing_config.ini
```

(continues on next page)

(continued from previous page)

```
# overwrite the file used using an environment variable
export REPROX_CONFIG=$(pwd)/my_reprocessing_config.ini
```

You will see that your defaults have been changed (e.g. do `reprox-reprocess --help`) reflecting the changes you made in the `.ini` file.

1.3.2 Use custom config

You might want to process some data with slightly different settings, this can be done using the`--context_kwargs` argument as follows (please don't move it into the production folder unless you know what you are doing):

```
reprox-reprocess \
    --package cutax \
    --context xenonnt_v6 \
    --target event_info event_pattern_fit cuts_basic \
    --cmt-version global_v6 \
    --ram 12000 \
    --cpu 2
    --context-kwargs '{"s1_min_coincidence": 2, "s2_min_pmts": 10}'
```

1.3.3 Using `reprox` from your jupyter notebook

You can also run the commands from above in a notebook or python script.

```
from reprox import find_data, submit_jobs, validate_run

targets = 'event_info event_pattern_fit cuts_basic'.split()

# First determine which data to process
find_data.find_data(
    targets=targets,
    exclude_from_invalid_cmt_version='global_v6'
)
# Now start running the jobs
submit_jobs.submit_jobs(targets=targets)

# Finally move the jobs to the production folder
validate_run.move_all()
```

1.3.4 Processing NV data

By default, the package assumes that only linked-mode or TPC runs are processed, if you want to instead process NV data you need to tell the scripts to also take into account the NV detector:

```
reprox-reprocess \
    --package cutax \
    --context xenonnt_v6 \
    --target events_nv \
```

(continues on next page)

(continued from previous page)

```
    --detectors neutron_veto muon_veto
    --ram 12000 \
    --cpu 2
    --cmt-version False
```

1.3.5 Using tagged versions

One might want to run with a different tag as so

```
MY_TAG=2021.12.2
source /cvmfs/xenon.opensciencegrid.org/releases/nT/$MY_TAG/setup.sh
reprox-reprocess \
    --package cutax  \
    --context xenonnt_v5  \
    --targets event_info \
    --cmt-version global_v5 \
    --ram 24000  \
    --cpu 2  \
    --move-after-workflow \
    --tag $MY_TAG
```

1.4 reprox package

1.4.1 Submodules

1.4.2 reprox.core module

Shared common methods for reprocessing, not useful in itself

reprox.core.**check_user_is_admin**(*admin_group='xenon1t-admins'*)
Check that the user is an xenon1t-admin

reprox.core.**format_context_kwargs**(*minimum_run_number*, *maximum_run_number*)

reprox.core.**get_context**(*package='cutax'*, *context='xenonnt_v7'*, *output_folder='/home/docs/checkouts/readthedocs.org/user_builds/reprox/checkouts/latest/test_folder/strax*
config_kwargs: Union[None, dict] = None, *minimum_run_number=17900*,
maximum_run_number=None)

reprox.core.**log_versions**()
Log versions (nested import makes the arg parsing quick)

reprox.core.**parse_args**(*description='nton reprocessing on dali'*, *include_find_args=False*,
include_processing_args=False, *include_workflow_args=False*)
Parse arguments to return to the user

1.4.3 reprox.find_runs module

reprox.find_runs.**determine_data_to_reprocess**(*st: strax.context.Context*, *targets: Union[str, tuple, list] = ()*, *special_modes: Union[List[str], Tuple[str]] = ('LED', 'noise', 'pmtap', 'pmtgain', 'exttrig')*, *keep_detectors: Union[str, tuple, list] = ('tpc',)*, *exclude_from_invalid_cmt: Optional[str] = 'global_v7'*, *_max_workers: int = 50*, *ignore_runs=()*) → pandas.core.frame.DataFrame

Find data that we can process. This data needs to:

1. (optional) be within the validity of a specified CMT version. Disable with exclude_from_invalid_cmt=False
2. Don't be some calibration mode (led/noise etc. data)
3. Not be available already (why would you want to reprocess that?)
4. Have the data which we need in order to compute this target.

Parameters

- **st** – Context to run with
- **targets** – Data types to produce
- **special_modes** – list of modes to exclude to determine here (usually you can do this trivially, so no need to use this function)
- **exclude_from_invalid_cmt** – A CMT version whereof we will check that the CMT version extends to those ranges where we would like to reprocess.
- **_max_workers** – Max workers for finding the stored data

Returns

reprox.find_runs.**find_data**(*targets: Union[str, list, tuple]*, *exclude_from_invalid_cmt_version: Union[bool, str] = 'global_v7'*, *context_kwargs: Union[None, dict] = None*, *keep_detectors: Union[str, tuple, list] = ['tpc']*, *ignore_runs=()*) → None

Determine which data to process, see determine_data_to_reprocess :param targets: List of targets to process :param exclude_from_invalid_cmt_version: A CMT version (optional) to

exclude runs that lie outside it's validity from

Parameters **context_kwargs** – Any context kwargs

Returns

1.4.4 reprox.process_job module

class reprox.process_job.**ProcessingJob**(*run_id*, *targets*, *submit_kwargs*)

Bases: object

Class for starting jobs and keeping an eye on their status

get_run_job_state(*read_last=10*, *ignore_patterns=['tensorflow', 'UserWarning', 'module compiled against']*) → str

Get the state of the current job

submit(***extra_kwargs*)
Submit the job to be run

submit_message = None

1.4.5 reprox.submit_jobs module

reprox.submit_jobs.**can_submit_more_jobs**(*nmax='100'*)

reprox.submit_jobs.**cycle_queue**(*queues=('xenon1t', 'dali', 'broadwl')*)

reprox.submit_jobs.**get_rundoc**(*run_id*)

reprox.submit_jobs.**n_jobs_running**()

reprox.submit_jobs.**submit_jobs**(*submit_kwargs: Union[None, dict] = None*, *targets: Union[str, List[str], Tuple[str]] = ('event_info', 'event_pattern_fit')*, *break_if_n_jobs_left_running: Union[None, int] = None*, *clear_logs: bool = False*, *sleep_s_when_queue_full: int = 60*, *submit_only: Union[None, int] = None*, *known_partitions: Union[tuple, list] = ['dali', 'xenon1t']*) → List[*reprox.process_job.ProcessingJob*]
Submit jobs to the queue for the given options

Parameters

- **submit_kwargs** – dict of options that are passed on to the job submission
- **targets** – List of datatypes to produce
- **break_if_n_jobs_left_running** – threshold when to stop reporting the status
- **clear_logs** – If true, clear the logs from previous jobs
- **sleep_s_when_queue_full** – sleep this many seconds if the
- **submit_only** – maximum number of jobs to submit
- **known_partitions** – list of partitions this user can submit to

Returns a list of all the jobs that were submitted

1.4.6 reprox.validate_run module

Validate that the data can be loaded successfully and move the data to the production folder

class reprox.validate_run.**RunValidation**(*path: str*, *context: Optional[strax.context.Context] = None*, *mode: Union[int,* reprox.validate_run.ValidationLevel*] = ValidationLevel.SHALLOW*)
Bases: object

Check that a directory (corresponding to a single datatype is

find_error() → str
Run several checks on a path to see if the processing was done correctly

class reprox.validate_run.**ValidationLevel**(*value*)
Bases: enum.IntEnum

An enumeration.

DEEP = 1

SHALLOW = 0

reprox.validate_run.**change_ownership**(*path*, *group*)

reprox.validate_run.**move_all**(*source_folder: str = '/home/docs/checkouts/readthedocs.org/user_builds/reprox/checkouts/latest/test_folder/strax_data*, *destination_folder: str = '/home/docs/checkouts/readthedocs.org/user_builds/reprox/checkouts/latest/test_folder'*, ***move_kwargs*)

Move data from all folders in <source_folder> into the destination folder and change the ownership of the folder

Parameters

- **source_folder** – The main folder where to look for folders to move
- **destination_folder** – The folder where the <path> folder should be moved to
- **move_kwargs** – Takes the following kwargs: :group: Name of the group that the permissions should be set to :validation_level: the level at which to validate the data:
  - ValidationLevel.SHALLOW: <0> for basic validation
  - **ValidationLevel.DEEP: <1> where we actually try loading the** data with (requires a context)

    context for when the validation_level is set to ValidationLevel.DEEP

Returns None

reprox.validate_run.**move_folder**(*path: str*, *destination_folder: str = '/home/docs/checkouts/readthedocs.org/user_builds/reprox/checkouts/latest/test_folder'*, *group='xenon1t-admins'*, *validation_level: int = ValidationLevel.SHALLOW*, *context: Optional[strax.context.Context] = None*) → Optional[str]

Move data from path into the destination folder and change the ownership of the folder

Parameters

- **path** – The folder to move
- **destination_folder** – The folder where the <path> folder should be moved to
- **group** – Name of the group that the permissions should be set to
- **validation_level** – the level at which to validate the data: - ValidationLevel.SHALLOW: <0> for basic validation - ValidationLevel.DEEP: <1> where we actually try loading the

  data with (requires a context)
- **context** – for when the validation_level is set to ValidationLevel.DEEP

Returns error string if an error occurred

Raises **FileExistsError** – when there is already a folder at the destination path

1.4.7 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

reprox, 11
reprox.core, 7
reprox.find_runs, 8
reprox.process_job, 8
reprox.submit_jobs, 9
reprox.validate_run, 9

INDEX

C

can_submit_more_jobs() (*in module reprox.submit_jobs*), 9
change_ownership() (*in module reprox.validate_run*), 10
check_user_is_admin() (*in module reprox.core*), 7
cycle_queue() (*in module reprox.submit_jobs*), 9

D

DEEP (*reprox.validate_run.ValidationLevel attribute*), 9
determine_data_to_reprocess() (*in module reprox.find_runs*), 8

F

find_data() (*in module reprox.find_runs*), 8
find_error() (*reprox.validate_run.RunValidation method*), 9
format_context_kwargs() (*in module reprox.core*), 7

G

get_context() (*in module reprox.core*), 7
get_run_job_state() (*reprox.process_job.ProcessingJob method*), 8
get_rundoc() (*in module reprox.submit_jobs*), 9

L

log_versions() (*in module reprox.core*), 7

M

module
    reprox, 11
    reprox.core, 7
    reprox.find_runs, 8
    reprox.process_job, 8
    reprox.submit_jobs, 9
    reprox.validate_run, 9
move_all() (*in module reprox.validate_run*), 10
move_folder() (*in module reprox.validate_run*), 10

N

n_jobs_running() (*in module reprox.submit_jobs*), 9

P

parse_args() (*in module reprox.core*), 7
ProcessingJob (*class in reprox.process_job*), 8

R

reprox
    module, 11
reprox.core
    module, 7
reprox.find_runs
    module, 8
reprox.process_job
    module, 8
reprox.submit_jobs
    module, 9
reprox.validate_run
    module, 9
RunValidation (*class in reprox.validate_run*), 9

S

SHALLOW (*reprox.validate_run.ValidationLevel attribute*), 9
submit() (*reprox.process_job.ProcessingJob method*), 8
submit_jobs() (*in module reprox.submit_jobs*), 9
submit_message (*reprox.process_job.ProcessingJob attribute*), 9

V

ValidationLevel (*class in reprox.validate_run*), 9