

---

# **hax Documentation**

***Release 2.5.0***

**XENON1T Collaboration**

**Aug 06, 2018**



---

## Contents

---

<b>1</b>	<b>hax package</b>	<b>3</b>
1.1	Subpackages	3
1.1.1	hax.treemakers package	3
1.1.1.1	Submodules	3
1.1.1.2	hax.treemakers.common module	3
1.1.1.3	hax.treemakers.cut_booleans_examples module	6
1.1.1.4	hax.treemakers.DoubleScatter module	7
1.1.1.5	hax.treemakers.examples module	7
1.1.1.6	hax.treemakers.peak_treemakers module	7
1.1.1.7	hax.treemakers.trigger	8
1.1.1.8	Module contents	9
1.2	Submodules	9
1.3	hax.data_extractor module	9
1.4	hax.ipython module	10
1.5	hax.minitrees module	10
1.6	hax.misc module	13
1.7	hax.paxroot module	13
1.8	hax.pmt_plot module	14
1.9	hax.raw_data module	14
1.10	hax.recorrect module	15
1.11	hax.runs module	16
1.12	hax.slow_control module	18
1.13	hax.trigger_data module	19
1.14	hax.utils module	19
1.15	Module contents	20
<b>2</b>	<b>Indices and tables</b>	<b>21</b>
	<b>Python Module Index</b>	<b>23</b>



Please see the notebooks at <https://github.com/XENONIT/hax/tree/master/examples> for a tutorial and several more in-depth examples on how to use hax.

Contents:



## 1.1 Subpackages

### 1.1.1 hax.treemakers package

#### 1.1.1.1 Submodules

#### 1.1.1.2 hax.treemakers.common module

Standard variables for most analyses

**class** `hax.treemakers.common.Basics`

Bases: `hax.minitrees.TreeMaker`

Basic information needed in most (standard) analyses, mostly on the main interaction.

**Provides:**

- `s1`: The uncorrected area in pe of the main interaction's S1
- `s2`: The uncorrected area in pe of the main interaction's S2
- `x_pax`: The x-position of the main interaction from pax (by `TopPatternFit`, field-distortion corrected)
- `y_pax`: The y-position of the main interaction from pax
- `z`: The z-position of the main interaction (computed by pax using configured drift velocity)
- `drift_time`: The drift time in ns (pax units) of the main interaction
- `s1_area_fraction_top`: The fraction of uncorrected area in the main interaction's S1 seen by the top array
- `s2_area_fraction_top`: The fraction of uncorrected area in the main interaction's S2 seen by the top array
- `s1_range_50p_area`: The width of the s1 (ns), duration of region that contains 50% of the area of the peak

- `s2_range_50p_area`: The width of the s2 (ns), duration of region that contains 50% of the area of the peak
- `largest_other_s1`: The uncorrected area in pe of the largest S1 in the TPC not in the main interaction
- `largest_other_s2`: The uncorrected area in pe of the largest S2 in the TPC not in the main interaction
- `largest_veto`: The uncorrected area in pe of the largest non-lone\_hit peak in the veto
- `largest_unknown`: The largest TPC peak of type 'unknown'
- `largest_coincidence`: The largest TPC peak of type 'coincidence'. This peak type no longer exists

**Notes:**

- 'largest' refers to uncorrected area.
- 'uncorrected' refers to the area in pe without applying any position- or saturation corrections.
- 'corrected' refers to applying all available position- and/or saturation corrections (see [https://github.com/XENON1T/pax/blob/master/pax/plugins/interaction\\_processing/BuildInteractions.py#L105](https://github.com/XENON1T/pax/blob/master/pax/plugins/interaction_processing/BuildInteractions.py#L105))
- **'main interaction' is `event.interactions[0]`, which is determined by pax** (currently just the largest S1 + largest S2 after it)

**`extract_data`** (*event*)

**`class`** `hax.treemakers.common.Extended`

Bases: `hax.minitrees.TreeMaker`

Extra information, mainly motivated by cuts used for the first science run. If there are no interactions in the event, all these values will be NaN.

**Provides:**

- `s1_range_80p_area`: The width of the s1 (ns), duration of region that contains 80% of the area of the peak
- `s1_range_90p_area`: The width of the s1 (ns), duration of region that contains 90% of the area of the peak
- `s1_range_100p_area`: The width of the s1 (ns), duration of region that contains 100% of the area of the peak
- `s2_range_80p_area`: The width of the s2 (ns), duration of region that contains 80% of the area of the peak
- `s1_n_contributing_channels`: Number of PMTs contributing to the S1.
- `s2_n_contributing_channels`: Number of PMTs contributing to the S2.
- `s1_largest_hit_area`: Area of the largest hit in the S1
- `s2_largest_hit_area`: Area of the largest hit in the S2
- `s1_largest_hit_channel`: PMT channel of the largest hit in the S1
- `s2_largest_hit_channel`: PMT channel of the largest hit in the S2
- `s1_rise_time`: The time between the 10% and 50% area points of the S1
- `s2_rise_time`: The time between the 10% and 50% area points of the S2
- `s1_tight_coincidence`: Number of PMTs with a hit close (window defined in pax) to the peak's sum waveform maximum
- `s1_pattern_fit`: Poisson likelihood of main S1's hitpattern (according to MC S1(xyz) per-PMT maps)



- `s2_pattern_fit`: Poisson likelihood of main S2's hitpattern (according to MC S2(xy) per-PMT maps)
- `r_pos_correction`: r-correction added to the interaction r position to account for field distortion.
- `z_pos_correction`: z-correction added to the interaction z position to account for field distortion.
- `x_nn`: x-position of the main interaction as reconstructed by neural net. NOT Field-distortion (r,z) corrected!!!
- `y_nn`: y-position of the main interaction as reconstructed by neural net. NOT Field-distortion (r,z) corrected!!!
- `x_tpff`: x-position of the main interaction as reconstructed by Top Pattern Function Fit algorithm. no FDC
- `y_tpff`: y-position of the main interaction as reconstructed by Top Pattern Function Fit algorithm. no FDC
- `sum_s1s_before_main_s2`: Sum of all S1 areas before the main S2
- `alt_s1_interaction_drift_time`: Drift time of interaction formed with largest other S1 + main S2
- `alt_s1_interaction_z`: Z position of interaction formed with largest other S1 + main S2
- `alt_s2_interaction_x`: X position of interaction with main S1 + largest other S2 (field-distortion rz corrected)
- `alt_s2_interaction_y`: Y position of interaction with main S1 + largest other S2 (field-distortion rz corrected)
- `alt_s2_interaction_z`: Z position of interaction with main S1 + largest other S2 (field-distortion rz corrected)
- `alt_s2_interaction_s2_range_50p_area`: S2 50% area width of interaction with main S1 + largest other S2
- `alt_s2_interaction_s2_range_80p_area`: S2 80% area width of interaction with main S1 + largest other S2
- `s1_area_fraction_top_probability`: probability of s1 area fraction top given its reconstructed position
- `largest_other_s2_delay_main_s1`: The hit time mean minus main S1 hit time mean
- `largest_other_s2_delay_main_s2`: The hit time mean minus main S2 hit time mean
- `largest_other_s2_pattern_fit`: Goodness-of-fit of hit pattern to position provided by PosRecTopPatternFit

(for pax < v6.6.0, field is not stored) See also the DoubleScatter minitree for more properties of alternative interactions.

```
extra_branches = ['peaks.area_decile_from_midpoint[11]', 'peaks.tight_coincidence', 'pax.field']
extract_data (event)
```

```
class hax.treemakers.common.Fundamentals
```

Bases: `hax.minitrees.TreeMaker`

Simple minitree containing basic information about every event, regardless of its contents. This minitree is always loaded whether you like it or not :-)

**Provides:**

- `run_number`: Run number of the run/dataset this event came from (common to all treemakers)
- `event_number`: Event number within the dataset (common to all treemakers)

- `event_time`: Unix time (in ns since the unix epoch) of the start of the event window
- `event_duration`: duration (in ns) of the event

```
branch_selection = ['event_number', 'start_time', 'stop_time']
```

```
extract_data(event)
```

```
pax_version_independent = True
```

```
class hax.treemakers.common.LargestPeakProperties
```

Bases: `hax.minitrees.TreeMaker`

Largest peak properties for each type and for all peaks. If you're doing an S1-only or S2-only analysis, you'll want this info instead of Basics. In other case you may want to combine this and Basics.

```
extra_branches = ['peaks.n_hits', 'peaks.hit_time_std', 'peaks.center_time', 'peaks.n_
```

```
extract_data(event)
```

```
get_properties(peak=None, prefix="")
```

Return dictionary with peak properties, keys prefixed with prefix if peak is None, will return nans for all values

```
peak_properties_to_get = ['area', 'area_fraction_top', 'n_hits', 'hit_time_std', 'cent
```

```
peak_types = ['s1', 's2', 'lone_hit', 'unknown']
```

```
class hax.treemakers.common.TotalProperties
```

Bases: `hax.minitrees.TreeMaker`

Aggregate properties of signals in the entire event

**Provides:**

- `n_pulses`, the total number of raw pulses in the event (for pax versions >6.0.0)
- `n_peaks`, the total number of TPC peaks in the event (including lone hits)
- `n_true_peaks`, the total number of TPC peaks in the event to which at least two PMTs contribute
- `total_peak_area`, the total area (pe) in all TPC peaks in the event
- `area_before_main_s2`, same, but including only peaks that occur before the main s2 (if there is one, else 0)

```
branch_selection = ['peaks.area', 'n_pulses', 'peaks.detector', 'interactions.s2', 'pe
```

```
extract_data(event)
```

```
hax.treemakers.common.get_largest_indices(peaks, exclude_indices=())
```

Return a dic with the indices in peaks of the largest peak of each type (s1, s2, etc) excluding the inices in `exclude_peak_indices` from consideration

### 1.1.1.3 hax.treemakers.cut\_booleans\_examples module

```
class hax.treemakers.cut_booleans_examples.EnergyCut
```

Bases: `hax.minitrees.TreeMaker`

S1 and S2 size cut booleans

Require that the S1 and S2 be large enough.

**Provides:**

- `pass_s1_area_cut`: S1 bigger than 1 pe

- `pass_s2_area_cut`: S2 bigger than 150 pe

Notes:

- This only cuts signals that are too small.

**extract\_data** (*event*)

#### 1.1.1.4 `hax.treemakers.DoubleScatter` module

#### 1.1.1.5 `hax.treemakers.examples` module

**class** `hax.treemakers.examples.TimeDifferences`

Bases: `hax.minitrees.TreeMaker`

Compute time differences between S1s and S2s

The convention is that these times are how much later the second largest signal is from the main one. For example, this could be the time of the second largest S2 minus the time of the largest S2. Therefore, you'll have a positive number if the second largest S2 came after the main S2.

**Provides:**

- `dt_s1s`: Time difference between two largest S1s
- `dt_s2s`: Time difference between two largest S2s

Notes:

- Positive times means the second largest signal is later.
- The code is intentionally not general for clarity.
- This code does not use the interaction concept in any way!
- 'Largest' refers to uncorrected area

**extra\_branches** = `['peaks.hit_time_mean']`

**extract\_data** (*event*)

#### 1.1.1.6 `hax.treemakers.peak_treemakers` module

Tree makers for studying peaks on their own

Treemakers used for analyses such as the single-electron shape in time and stability.

**class** `hax.treemakers.peak_treemakers.IsolatedPeaks`

Bases: `hax.minitrees.MultipleRowExtractor`

Returns one row per peak isolated in time

Specifically returns properties of each individual peak.

**extra\_branches** = `['peaks.left', 'peaks.right', 'peaks.n_hits', 'peaks.n_contributing_c`

**extract\_data** (*event*)

**nhits\_bounds** = `(0, inf)`

**width\_bounds** = `(0, inf)`

**static yield\_peak** (*event*, *nhits\_bounds*, *width\_bounds*)

Extracts a row per peak

The peak type can be single electron and have some selection. This is a generator, and yields (peak, time\_to\_nearest).

**class** hax.treemakers.peak\_treemakers.**PeakExtractor** (\*args, \*\*kwargs)

Bases: *hax.minitrees.MultipleRowExtractor*

Base class for reading peak data in minitrees. For more information, check out example 10 in hax/examples.

**build\_cut\_string** (*cut\_list*, *obj*)

Build a string of cuts that can be applied using eval() function.

**detectors** = {'busy\_off': 4, 'busy\_on': 3, 'sum\_wv': 2, 'tpc': 0, 'veto': 1}

**event\_cut\_list** = []

**event\_cut\_string** = 'True'

**extra\_branches** = ['peaks.\*']

**extract\_data** (*event*)

**peak\_cut\_list** = []

**peak\_cut\_string** = 'True'

**peak\_fields** = ['area']

**peaktypes** = {'lone\_hit': 0, 's1': 1, 's2': 2, 'unknown': 3}

**stop\_after** = <MagicMock name='mock.inf' id='140645191671480'>

**class** hax.treemakers.peak\_treemakers.**SingleElectrons**

Bases: *hax.treemakers.peak\_treemakers.IsolatedPeaks*

**nhits\_bounds** = (15, 26.01)

**width\_bounds** = (50, 450)

#### 1.1.1.7 hax.treemakers.trigger

**class** hax.treemakers.trigger.**LargestTriggeringSignal**

Bases: *hax.minitrees.TreeMaker*

Information on the largest trigger signal with the trigger flag set in the event

**Provides:**

- trigger\_\*, where \* is any of the attributes of datastructure.TriggerSignal

**branch\_selection** = ['trigger\_signals\*', 'event\_number']

**extract\_data** (*event*)

**pax\_version\_independent** = True

**class** hax.treemakers.trigger.**Proximity**

Bases: *hax.minitrees.TreeMaker*

Information on the proximity of other events and acquisition monitor signals (e.g. busy and muon veto trigger)

**Provides:**

- previous\_x: Time (in ns) between the time center of the event and the previous x (see below for various x). This also considers any x inside the event.

- `next_x`: same, for time to next x
- **nearest\_x: Time to the nearest x. NB: If the nearest x is in the past, this time is negative!** `x` denotes the object of interest, and could be either:
- `muon_veto_trigger`.
- `busy_x`: a busy-on or busy-off signal
- `hev_x`: a high-energy veto on or -off signal
- `event`: any event (excluding itself :-)
- `1e5_pe_event`: any event with total peak area > 1e5 pe (excluding itself)
- `3e5_pe_event`: “” 3e5 pe “
- `1e6_pe_event`: “” 1e6 pe “
- `s2_area`: Area of main s2 in event
- All the information about the `muon_veto_trigger` are calculated with respect to the TPC trigger and
- not to the middle of the event.

```
aqm_labels = ['muon_veto_trigger', 'busy_on', 'hev_on', 'busy_off', 'hev_off', 'busy',
```

```
bad_mv_triggers (aqm_pulses, min_time=500)
    get me the indices of the sync signal in the MV_trigger sent to the TPC
```

```
extract_data (event)
```

```
get_data (dataset, event_list=None)
    Return data extracted from running over dataset
```

```
pax_version_independent = False
```

```
select_physical_pulses (aqm_pulses, ap_time=20000)
    get rid of afterpulses from the MV data of course!!!!
```

```
class hax.treemakers.trigger.TailCut
    Bases: hax.minitrees.TreeMaker

    get_data (dataset, event_list=None)
        Return data extracted from running over dataset

    never_store = True
```

#### 1.1.1.8 Module contents

## 1.2 Submodules

### 1.3 hax.data\_extractor module

Extract peak or hit info from processed root file

```
class hax.data_extractor.DataExtractor
    Bases: object
```

This class is meant for extracting properties that are *not* on the event level, such as peak or hit properties. For more information, check the docs of `DataExtractor.get_data()`.

```
get_data (dataset, level='peak', event_fields=['event_number'], peak_fields=['area', 'hit_time_std'],
        hit_fields=[], event_cuts=[], peak_cuts=[], stop_after=<MagicMock name='mock.inf'
        id='140645191671480'>, added_branches=[])
```

Extract peak or hit data from a dataset. Peak or hit can be toggled by specifying level = 'peak' or level = 'hit'. Example usage:

```
d = DataExtractor.get_data(dataset=run_name,level='peak',event_fields = ['event_number'],
    peak_fields=['area'],event_cuts=['event_number > 5', 'event_number < 10'],
    peak_cuts=['area > 100', 'type = "s1"] ,stop_after=10000,added_branches= ['peak.type'])
```

```
loop_body (event)
```

Function that extracts data from each event and adds array with that data to the data list.

```
hax.data_extractor.build_cut_string (cut_list, obj)
```

Build a string of cuts that can be applied using eval() function.

```
hax.data_extractor.make_branch_selection (level, event_fields, peak_fields, added_branches)
```

Make the list of branches that have to be selected.

```
hax.data_extractor.make_named_array (array, field_names)
```

Make a named array from a numpy array.

```
hax.data_extractor.root_to_numpy (base_object, field_name, attributes)
```

Convert objects stored in base\_object.field\_name to numpy array Will query attributes for each of the objects in base\_object.field\_name No, root\_numpy does not do this for you, that's for trees...

## 1.4 hax.ipython module

```
hax.ipython.code_hider ()
```

## 1.5 hax.minitrees module

Make small flat root trees with one entry per event from the pax root files.

```
class hax.minitrees.MultipleRowExtractor
```

Bases: `hax.minitrees.TreeMaker`

Base class for treemakers that return a list of dictionaries in `extract_data`. These treemakers can produce anywhere from zero or many rows for a single event.

If you're seeing this as the documentation of an actual `TreeMaker`, somebody forgot to add documentation for their treemaker.

```
process_event (event)
```

```
exception hax.minitrees.NoMinitreeAvailable
```

Bases: `Exception`

```
class hax.minitrees.TreeMaker
```

Bases: `object`

Treemaker base class.

If you're seeing this as the documentation of an actual `TreeMaker`, somebody forgot to add documentation for their treemaker.

A treemaker loops the `extract_data` function over events. This function returns a dictionary. Since dictionaries take a lot of memory, we periodically convert them into pandas dataframes (interval with which this occurs is controlled by the `cache_size` attribute). At the end of data extraction, the various dataframes are concatenated.

You must instantiate a new treemaker for every extraction.

```
branch_selection = None

cache_size = 5000

check_cache (force_empty=False)

extra_branches = ()

extra_metadata = {}

extract_data (event)

get_data (dataset, event_list=None)
    Return data extracted from running over dataset

mc_data = False

never_store = False

pax_version_independent = False

process_event (event)

uses_arrays = False
```

```
hax.minitrees.check (run_id, treemaker, force_reload=False)
    Return if the minitree exists and where it is found / where to make it.
```

#### Parameters

- **treemaker** – treemaker name or class
- **run\_id** – run name or number
- **force\_reload** – ignore available minitrees, just tell me where to write the new one.

**Returns** (treemaker, available, path). - `treemaker_class`: class of the treemaker you named. - `already_made` is True if there is an up-to-date minitree we can load, False otherwise (always if `force_reload`) - `path` is the path to the minitree to load if it is available, otherwise path where we should create the minitree.

```
hax.minitrees.extend (data, treemakers)
```

Extends the dataframe data by loading treemakers for the remaining events See <https://github.com/XENON1T/hax/pull/52> for more information.

#### Parameters

- **data** – dataframe, assumed to be event-per-row
- **treemakers** – list of treemakers to load

```
hax.minitrees.force_df_types (df_content, df_types)
```

Return dataframe with same columns and dtypes as `df_types`, with content from `df_content`.

- Extra columns are dropped.
- Missing columns are set to NaN (for floats) or INT\_NAN (for integers). Columns that are neither int or float are set to zero (e.g. “ for strings).
- Columns with different types are converted using numpy’s `astype`. When converting floats to ints, all nonfinite values are replaced with INT\_NAN

`hax.minitrees.function_over_events` (*function, dataframe, branch\_selection=None, \*\*kwargs*)

Generator which yields *function(event, \*\*kwargs)* of each processed data event in dataframe

`hax.minitrees.get_treemaker_name_and_class` (*tm*)

Return (name, class) of treemaker name or class tm

`hax.minitrees.load` (*datasets=None, treemakers='all', preselection=None, force\_reload=False, delayed=False, num\_workers=1, compute\_options=None, cache\_file=None, remake\_cache=False, event\_list=None*)

Return pandas DataFrame with minitrees of several datasets and treemakers.

#### Parameters

- **datasets** – names or numbers of datasets (without .root) to load
- **treemakers** – treemaker class (or string with name of class) or list of these to load. If value is set to 'all' then the standard science run minitrees are loaded.
- **preselection** – string or list of strings parseable by `pd.eval`. Should return bool array, to be used for pre-selecting events to load for each dataset.
- **force\_reload** – if True, will force mini-trees to be re-made whether they are outdated or not.
- **delayed** – Instead of computing a pandas DataFrame, return a `dask DataFrame` (default False)
- **num\_workers** – Number of `dask` workers to use in computation (if `delayed=False`)
- **compute\_options** – Dictionary of extra options passed to `dask.compute`
- **cache\_file** – Save/load the result to an `hdf5` file with filename specified by `cache_file`. Useful if you load in a large volume of data with many preselections.
- **remake\_cache** – If True, and cache file given, reload (don't remake) minitrees and overwrite the cache file.
- **event\_list** – List of events to process (warning: only makes sense for single dataset)

`hax.minitrees.load_cache_file` (*cache\_file*)

Load minitree dataframe + cut history from a cache file

`hax.minitrees.load_single_dataset` (*run\_id, treemakers, preselection=None, force\_reload=False, event\_list=None, bypass\_blinding=False*)

Run multiple treemakers on a single run

**Returns** (pandas DataFrame, list of dicts describing cut histories)

#### Parameters

- **run\_id** – name or number of the run to load
- **treemakers** – list of treemaker classes / names to load
- **preselection** – String or list of strings passed to `pandas.eval`. Should return bool array, to be used for pre-selecting events to load for each dataset. If string does not contain spaces, should be lax lichen name. If string contains a colon and no spaces, should be `lichen_file:lichen_name`
- **force\_reload** – always remake the minitrees, never load any from disk.
- **event\_list** – List of event numbers to visit. Disables load from / save to file.

**Bypass\_blinding** Flag to disable blinding cut. WARNING: analysts should not use this, only for production! See #211



```
hax.minitrees.load_single_minitree(run_id, treemaker, force_reload=False, re-
                                turn_metadata=False, save_file=None, event_list=None)
```

Return pandas DataFrame resulting from running treemaker on run\_id (name or number)

#### Parameters

- **run\_id** – name or number of the run to load
- **treemaker** – TreeMaker class or class name (but not TreeMaker instance!) to run
- **force\_reload** – always remake the minitree, never load it from disk.
- **return\_metadata** – instead return (metadata\_dict, dataframe)
- **save\_file** – save the results to a minitree file on disk.
- **event\_list** – List of event numbers to visit. Forces save\_file=False, force\_reload=True.

**Returns** pandas.DataFrame

```
hax.minitrees.save_cache_file(data, cache_file, **kwargs)
```

Save minitree dataframe + cut history to a cache file Any kwargs will be passed to pandas HDFStore. Defaults are:

```
complib='blosc' complevel=9
```

```
hax.minitrees.update_treemakers()
```

Update the list of treemakers hax knows. Called on hax init, you should never have to call this yourself!

## 1.6 hax.misc module

```
hax.misc.code_hider()
```

Make a button in the jupyter notebook to hide all code

```
hax.misc.dataframe_to_wiki(df, float_digits=5, title='Awesome table')
```

Convert a pandas dataframe to a dokuwiki table (which you can copy-paste onto the XENON wiki)

#### Parameters

- **df** – dataframe to convert
- **float\_digits** – Round float-ing point values to this number of digits.
- **title** – title of the table.

```
hax.misc.draw_box(x, y, **kwargs)
```

Draw rectangle, given x-y boundary tuples

## 1.7 hax.paxroot module

Utility functions for loading and looping over a pax root file

```
exception hax.paxroot.StopEventLoop
```

Bases: `Exception`

```
hax.paxroot.function_results_datasets(datasets_names, event_function=<function
                                <lambda>>, event_lists=None,
                                branch_selection=None, kwargs=None, desc="")
```

Returns a generator which yields the return values of event\_function(event) over the datasets specified in datasets\_names.

### Parameters

- **dataset\_names** – list of dataset names or numbers, or string/int of a single dataset name/number
- **event\_function** – function to run over each event
- **event\_lists** – a list of event numbers (if you’re loading in a single dataset) to visit, or a list of lists of event numbers for each of the datasets passed in `dataset_names`.
- **branch\_selection** – can be - None (all branches are read), - ‘basic’ (`hax.config[‘basic_branches’]` are read), or - a list of branches to read.
- **kwargs** – dictionary of extra arguments to pass to `event_function`. For example: `kwargs={‘x’: 2, ‘y’: 3}` -> function called like: `event_function(event, x=2, y=3)`
- **desc** – Description used in the tqdm progressbar

`hax.paxroot.get_filename(run_id)`

`hax.paxroot.get_metadata(run_id)`

Returns the metadata dictionary stored in the pax root file for `run_id`.

`hax.paxroot.loop_over_dataset(*args, **kwargs)`

Execute a function over all events in the dataset(s) Does not return anything: use `function_results_dataset` or pass a class method as `event_function` if you want results. See `function_results_datasets` for possible options.

`hax.paxroot.loop_over_datasets(*args, **kwargs)`

Execute a function over all events in the dataset(s) Does not return anything: use `function_results_dataset` or pass a class method as `event_function` if you want results. See `function_results_datasets` for possible options.

`hax.paxroot.open_pax_rootfile(run_id, load_class=True)`

Opens pax root file for `run_id`, compiling classes/dictionaries as needed. Returns TFile object. if `load_class` is False, will not load the event class. You’ll only be able to read metadata from the file.

## 1.8 hax.pmt\_plot module

`hax.pmt_plot.plot_on_pmt_arrays(color=None, size=None, geometry=‘physical’, title=None, scatter_kwargs=None, colorbar_kwargs=None)`

Plot a scatter plot of PMTs in a specified geometry, with a specified color and size of the markers. Color or size must be per-PMT array that is indexable by another array, i.e. must be `np.array` and not list. `scatter_kwargs` will be passed to `plt.scatter` `colorbar_kwargs` will be passed to `plt.colorbar` geometry can be ‘physical’, a key from `pmt_data`, or a 2-tuple of keys from `pmt_data`.

## 1.9 hax.raw\_data module

Functions for working with raw data.

**class** `hax.raw_data.HTTPSClientAuthHandler(key, cert)`

Bases: `urllib.request.HTTPSHandler`

Used for accessing GRID data and handling authentication

**getConnection** (*host, timeout*)

**https\_open** (*req*)

`hax.raw_data.cleanup_temporary_data_files()`

Removes all temporarily downloaded raw data files. Run automatically for you when your program quits

`hax.raw_data.download_from_grid(file_path_tail)`  
Downloads file\_path\_tail from grid, returns filename of temporary file

`hax.raw_data.inspect_events(run_id, event_numbers, focus='all', save_to_dir=None, config_override=None)`  
Show the pax event display for the events in run\_id,  
focus can be 'all' (default) which shows the entire event, 'largest', 'first', 'main\_s1', or 'main\_s2'

`hax.raw_data.inspect_events_from_minitree(events, *args, **kwargs)`  
Show the pax event display for events, where events is a (slice of) a dataframe loaded from a minitree Any additional arguments will be passed to inspect\_events, see its docstring for details

`hax.raw_data.inspect_peaks(run_id, event_numbers, peak_boundaries, save_to_dir=None, config_override=None)`  
Inspect the peaks starting at peak\_boundaries (in samples... sorry) in event\_numbers. Event numbers and peak\_boundaries must be list/arrays of integers of the same length.

`hax.raw_data.inspect_peaks_array(run_id, peak_array, save_to_dir=None, config_override=None)`  
Inspect peaks from a record array returned by hax.DataExtractor

`hax.raw_data.process_events(run_id, event_numbers=None, config_override=None)`  
Yields processed event(s) numbered event\_numbers from dataset run\_id (name or number) config\_override is a dictionary with extra pax options

`hax.raw_data.raw_data_processor(input_file_or_directory, config_override=None)`  
Return a raw data processor which reads events from input\_file\_or\_directory config\_override can be used to set additional pax options

`hax.raw_data.raw_events(run_id, event_numbers=None, config_override=None)`  
Yields raw event(s) numbered event\_numbers from dataset numbered dataset\_number config\_override is a dictionary with extra pax options

## 1.10 hax.recorrect module

Functions to redo late-stage pax corrections with new maps on existing minitree dataframes

These functions will be slow, since the pax interpolating map was never designed to be quick (vectorized), other processing plugins dominate the run time of pax.

`hax.recorrect.add_uncorrected_position(data)`  
Adds r, theta, u\_r, u\_x, u\_y, u\_z to data. If u\_x already exists, does nothing. Returns no value. Modifies data in place.

`hax.recorrect.recorrect_rz(data, new_map_file=None)`  
Recompute the (r,z)(r,z) field distortion correction Be sure to redo the S1(x,y,z) correction after this as well, whether or not the S1(x,y,z) map changed!

### Parameters

- **data** – input dataframe
- **new\_map\_file** – file with (r,z)(r,z) correction map to use. Defaults to map currently in pax config.

**Returns** dataframe with altered values in x, y, z (and few added columns for uncorrected position)

```
hax.recorrect.recorrect_s1xyz (data, new_map_file=<MagicMock
                               name='mock.configuration.load_configuration().__getitem__().__getitem__()'
                               id='140645191350592'>)
```

Recompute the S1(x,y,z) light yield correction. If you want to redo (r,z)(r,z), do it before doing this!

#### Parameters

- **data** – Dataframe. Only Basics minitree required.
- **new\_map\_name** – Filename of map you want to use for the correction.

**Returns** Dataframe with changed values in cs1 column

```
hax.recorrect.recorrect_s2xy (data, old_map_file='s2_xy_XENON1T_17Feb2017.json',
                               new_map_file=<MagicMock name='mock.configuration.load_configuration().__getitem__()'
                               id='140645191350592'>)
```

Recompute the (x,y) correction for a different map :param data: dataframe (Basics and Extended minitrees required) :param old\_map\_file: Map filename that was used to process the dataframe. Defaults to the map used for 6.4.2 :param new\_map\_file: Map filename that you want to use for the correction. Defaults to the pax config default. :return: dataframe with altered value in cS2 (and few added columns for uncorrected position)

TODO: This could be rewritten to use the extended minitrees, so the old map no longer needs to be specified.

## 1.11 hax.runs module

Runs database utilities

```
hax.runs.count_tags (ds)
```

Return how often each tag occurs in the datasets DataFrame ds

```
hax.runs.datasets_query (query)
```

Return names of datasets matching query

```
hax.runs.get_dataset_info (run_id, projection_query=None)
```

Returns a dictionary with the runs database info for a given run\_id. For XENON1T, this queries the runs db to get the complete run doc.

#### Parameters

- **run\_id** – name or number, or list of such, of runs to query. If giving a list, it must be sorted!
- **projection\_query** – can be - None (default): the entire run doc will be returned - string: runs db field name (with dots indicating subfields), we'll query and return only that field. - anything else: passed as projection to pymongo.collection.find

For example 'processor.DEFAULT.electron\_lifetime\_liquid' returns the electron lifetime.

```
hax.runs.get_run_info (run_id, projection_query=None)
```

Returns a dictionary with the runs database info for a given run\_id. For XENON1T, this queries the runs db to get the complete run doc.

#### Parameters

- **run\_id** – name or number, or list of such, of runs to query. If giving a list, it must be sorted!
- **projection\_query** – can be - None (default): the entire run doc will be returned - string: runs db field name (with dots indicating subfields), we'll query and return only that field. - anything else: passed as projection to pymongo.collection.find

For example 'processor.DEFAULT.electron\_lifetime\_liquid' returns the electron lifetime.

`hax.runs.get_run_name(run_id)`  
Return run name matching `run_id`. Returns `run_id` if `run_id` is string (presumably already run name)

`hax.runs.get_run_number(run_id)`  
Return run number matching `run_id`. Returns `run_id` if `run_id` is int (presumably already run int)

`hax.runs.get_run_start(run_id)`  
Return the start time of the run as a datetime

`hax.runs.get_rundb_collection()`  
Return the pymongo handle to the runs db collection. You can use this to do queries like `.find` etc.

`hax.runs.get_rundb_database()`  
Return the pymongo handle to the runs db database. You can use this to access other collections.

`hax.runs.get_rundb_password()`  
Return the password to the runs db, if we know it

`hax.runs.is_mc(run_id)`

`hax.runs.load_corrections()`  
Load all corrections that are stored on MongoDB as defined by the `corrections` field in the hax config. Corrections must be named the same as their collection name in the database 'run'.

`hax.runs.tags_selection(dsets=None, include=None, exclude=None, pattern_type='fnmatch', ignore_underscore=True)`  
Return runs by tag selection criteria.

#### Parameters

- **dsets** – pandas DataFrame, subset of datasets from `hax.runs.datasets`. If not provided, uses `hax.runs.datasets` itself (all datasets).
- **include** – String or list of strings of patterns of tags to include
- **exclude** – String or list of strings of patterns of tags to exclude. Exclusion criteria have higher priority than inclusion criteria.
- **pattern\_type** – Type of pattern matching to use. Defaults to 'fnmatch', which means you can use unix shell-style wildcards (`?`, `*`). Alternative is 're', which means you can use full python regular expressions.
- **ignore\_underscore** – Ignore the underscore at the start of some tags (indicating some degree of officialness or automation) when matching.

#### Examples:

- `tags_selection(include='blinded')` select all datasets with a `blinded` or `_blinded` tag.
- `tags_selection(include='*blinded')` ... with `blinded` or `_blinded`, `unblinded`, `blablinded`, etc.
- `tags_selection(include=['blinded', 'unblinded'])` ... with `blinded` OR `unblinded`, but not `blablinded`.
- `tags_selection(include='blinded', exclude=['bad', 'messy'])` select **blinded datasets** that aren't bad or messy

`hax.runs.update_datasets(query=None)`  
Update `hax.runs.datasets` to contain latest datasets. Currently just loads XENON100 run 10 runs from a csv file.  
query: custom query, in case you only want to update partially??

`hax.runs.version_is_consistent_with_policy(version)`  
Returns if the pax version is consistent with the pax version policy. If policy is 6.2.1, only '6.2.1' (or 'v6.2.1') gives True. If policy is 6.2, any of 6.2.0, 6.2.1 etc. gives True

`hax.runs.version_tuple(v)`

Convert a version indication string (e.g. “6.2.1”) into a tuple of integers

## 1.12 hax.slow\_control module

**exception** `hax.slow_control.AmbiguousSlowControlMonikerException`

Bases: `Exception`

**exception** `hax.slow_control.UnknownSlowControlMonikerException`

Bases: `Exception`

`hax.slow_control.get(names, run=None, start=None, end=None, url=None)`

Retrieve the data from the historian database (hax.slow\_control.get is just a synonym of this function)

### Parameters

- **names** – name or list of names of slow control variables; see `get_historian_name`.
- **run** – run number/name to return data for. If passed, start/end is ignored.
- **start** – String indicating start of time range, in arbitrary format (thanks to `parsedatetime`)
- **end** – String indicating end of time range, in arbitrary format

**Returns** pandas Series of the values, with index the time in UTC. If you requested multiple names, pandas DataFrame

`hax.slow_control.get_pmt_data_last_measured(run)`

Retrieve PMT information for a run from the historian database

**Parameters** **run** – run number/name to return data for.

**Returns** pandas DataFrame of the values, with index the time in UTC.

`hax.slow_control.get_sc_api_key()`

Return the slow control API key, if we know it

`hax.slow_control.get_sc_data(names, run=None, start=None, end=None, url=None)`

Retrieve the data from the historian database (hax.slow\_control.get is just a synonym of this function)

### Parameters

- **names** – name or list of names of slow control variables; see `get_historian_name`.
- **run** – run number/name to return data for. If passed, start/end is ignored.
- **start** – String indicating start of time range, in arbitrary format (thanks to `parsedatetime`)
- **end** – String indicating end of time range, in arbitrary format

**Returns** pandas Series of the values, with index the time in UTC. If you requested multiple names, pandas DataFrame

`hax.slow_control.get_sc_name(name, column='Historian_name')`

Return slow control historian name of name. You can pass a historian name, sc name, pid identifier, or description. For a full table, see hax.

`hax.slow_control.init_sc_interface()`

Initialize the slow control interface access and list of variables

## 1.13 hax.trigger\_data module

`hax.trigger_data.get_aqm_pulses(run_id)`

Return a dictionary of acquisition monitor pulse times in the run `run_id`. keys are channel labels (e.g. `muon_veto_trigger`). Under the keys ‘busy’ and ‘hev’, you’ll get the sorted combination of all busy/hev \_on and \_off signals.

`hax.trigger_data.get_special_file_filename(basename, run_id, special_path_key=None)`

`hax.trigger_data.get_trigger_data(run_id, select_data_types='all', format_version=2)`

Return dictionary with the trigger data from `run_id` `select_data_types` can be ‘all’, a trigger data type name, or a list of trigger data type names. If you want to find out which data types exists, use ‘all’ and look at the keys of the dictionary.

## 1.14 hax.utils module

Utilities for use INSIDE hax (and perhaps random weird use outside hax) If you have a nice function that doesn’t fit anywhere, `misc.py` is where you want to go

`hax.utils.combine_pax_configs(config, overrides)`

Combines configuration dictionaries `config` and `overrides`. `overrides` has higher priority. each config must be a dictionary containing only string->dict pairs (like the `pax/ConfigParser` configs)

`hax.utils.find_file_in_folders(filename, folders)`

Searches for `filename` in `folders`, then return full path or raise `FileNotFoundError` Does not recurse into subdirectories

`hax.utils.flatten_dict(d, separator=':', _parent_key='')`

Flatten nested dictionaries into a single dictionary, indicating levels by separator Don’t set `_parent_key` argument, this is used for recursive calls. Stolen from <http://stackoverflow.com/questions/6027558>

`hax.utils.get_user_id()`

**Returns** string identifying the currently active system user as `name@node`

**Note** user can be set with the ‘USER’ environment variable, usually set on windows

**Note** on unix based systems you can use the password database to get the login name of the effective process user

`hax.utils.get_xenon100_dataset_number(dsetname)`

Converts a XENON100 dataset name to a number

`hax.utils.human_to_utc_datetime(x)`

Return a python UTC-localized datetime object corresponding to the human-readable date/time `x` :param `x`: string with a human-readable date/time indication (e.g. “now”). If you specify something absolute, it will

be taken as UTC.

`hax.utils.load_pickles(filename, load_first=None)`

Returns list of pickles stored in `filename`. :param `load_first`: number of pickles to read. Otherwise reads until file is exhausted

`hax.utils.save_pickles(filename, *args)`

Compresses and pickles `*args` to `filename`. The pickles are stacked: load them with `load_pickles`

`hax.utils.utc_timestamp(d)`

Convert a UTC datetime object `d` to (float) seconds in the UTC since the UTC unix epoch. If you pass a timezone-naive datetime object, it will be treated as UTC.

## 1.15 Module contents

`hax.init` (*filename=None, \*\*kwargs*)

Loads hax configuration from hax.ini file filename. You should always call this before starting up hax. You can call it again to reload the hax config. Any keyword arguments passed will override settings from the configuration.



## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### h

- [hax](#), 20
- [hax.data\\_extractor](#), 9
- [hax.ipython](#), 10
- [hax.minitrees](#), 10
- [hax.misc](#), 13
- [hax.paxroot](#), 13
- [hax.pmt\\_plot](#), 14
- [hax.raw\\_data](#), 14
- [hax.recorrect](#), 15
- [hax.runs](#), 16
- [hax.slow\\_control](#), 18
- [hax.treemakers](#), 9
  - [hax.treemakers.common](#), 3
  - [hax.treemakers.cut\\_booleans\\_examples](#), 6
  - [hax.treemakers.examples](#), 7
  - [hax.treemakers.peak\\_treemakers](#), 7
  - [hax.treemakers.trigger](#), 8
  - [hax.trigger\\_data](#), 19
- [hax.utils](#), 19



## A

add\_uncorrected\_position() (in module `hax.recorrect`), 15  
 AmbiguousSlowControlMonikerException, 18  
 aqm\_labels (`hax.treemakers.trigger.Proximity` attribute), 9

## B

bad\_mv\_triggers() (`hax.treemakers.trigger.Proximity` method), 9  
 Basics (class in `hax.treemakers.common`), 3  
 branch\_selection (`hax.minitrees.TreeMaker` attribute), 11  
 branch\_selection (`hax.treemakers.common.Fundamentals` attribute), 6  
 branch\_selection (`hax.treemakers.common.TotalProperties` attribute), 6  
 branch\_selection (`hax.treemakers.trigger.LargestTriggeringSignal` attribute), 8  
 build\_cut\_string() (`hax.treemakers.peak_treemakers.PeakExtractor` method), 8  
 build\_cut\_string() (in module `hax.data_extractor`), 10

## C

cache\_size (`hax.minitrees.TreeMaker` attribute), 11  
 check() (in module `hax.minitrees`), 11  
 check\_cache() (`hax.minitrees.TreeMaker` method), 11  
 cleanup\_temporary\_data\_files() (in module `hax.raw_data`), 14  
 code\_hider() (in module `hax.ipython`), 10  
 code\_hider() (in module `hax.misc`), 13  
 combine\_pax\_configs() (in module `hax.utils`), 19  
 count\_tags() (in module `hax.runs`), 16

## D

DataExtractor (class in `hax.data_extractor`), 9  
 dataframe\_to\_wiki() (in module `hax.misc`), 13  
 datasets\_query() (in module `hax.runs`), 16  
 detectors (`hax.treemakers.peak_treemakers.PeakExtractor` attribute), 8  
 download\_from\_grid() (in module `hax.raw_data`), 14

draw\_box() (in module `hax.misc`), 13

## E

EnergyCut (class in `hax.treemakers.cut_booleans_examples`), 6  
 event\_cut\_list (`hax.treemakers.peak_treemakers.PeakExtractor` attribute), 8  
 event\_cut\_string (`hax.treemakers.peak_treemakers.PeakExtractor` attribute), 8  
 extend() (in module `hax.minitrees`), 11  
 Extended (class in `hax.treemakers.common`), 4  
 extra\_branches (`hax.minitrees.TreeMaker` attribute), 11  
 extra\_branches (`hax.treemakers.common.Extended` attribute), 5  
 extra\_branches (`hax.treemakers.common.LargestPeakProperties` attribute), 6  
 extra\_branches (`hax.treemakers.examples.TimeDifferences` attribute), 7  
 extra\_branches (`hax.treemakers.peak_treemakers.IsolatedPeaks` attribute), 7  
 extra\_branches (`hax.treemakers.peak_treemakers.PeakExtractor` attribute), 8  
 extra\_metadata (`hax.minitrees.TreeMaker` attribute), 11  
 extract\_data() (`hax.minitrees.TreeMaker` method), 11  
 extract\_data() (`hax.treemakers.common.Basics` method), 4  
 extract\_data() (`hax.treemakers.common.Extended` method), 5  
 extract\_data() (`hax.treemakers.common.Fundamentals` method), 6  
 extract\_data() (`hax.treemakers.common.LargestPeakProperties` method), 6  
 extract\_data() (`hax.treemakers.common.TotalProperties` method), 6  
 extract\_data() (`hax.treemakers.cut_booleans_examples.EnergyCut` method), 7  
 extract\_data() (`hax.treemakers.examples.TimeDifferences` method), 7  
 extract\_data() (`hax.treemakers.peak_treemakers.IsolatedPeaks` method), 7

`extract_data()` (hax.treemakers.peak\_treemakers.PeakExtractor method), 8  
`extract_data()` (hax.treemakers.trigger.LargestTriggeringSignal method), 8  
`extract_data()` (hax.treemakers.trigger.Proximity method), 9

## F

`find_file_in_folders()` (in module hax.utils), 19  
`flatten_dict()` (in module hax.utils), 19  
`force_df_types()` (in module hax.minitrees), 11  
`function_over_events()` (in module hax.minitrees), 11  
`function_results_datasets()` (in module hax.paxroot), 13  
Fundamentals (class in hax.treemakers.common), 5

## G

`get()` (in module hax.slow\_control), 18  
`get_aqm_pulses()` (in module hax.trigger\_data), 19  
`get_data()` (hax.data\_extractor.DataExtractor method), 9  
`get_data()` (hax.minitrees.TreeMaker method), 11  
`get_data()` (hax.treemakers.trigger.Proximity method), 9  
`get_data()` (hax.treemakers.trigger.TailCut method), 9  
`get_dataset_info()` (in module hax.runs), 16  
`get_filename()` (in module hax.paxroot), 14  
`get_largest_indices()` (in module hax.treemakers.common), 6  
`get_metadata()` (in module hax.paxroot), 14  
`get_pmt_data_last_measured()` (in module hax.slow\_control), 18  
`get_properties()` (hax.treemakers.common.LargestPeakProperties method), 6  
`get_run_info()` (in module hax.runs), 16  
`get_run_name()` (in module hax.runs), 16  
`get_run_number()` (in module hax.runs), 17  
`get_run_start()` (in module hax.runs), 17  
`get_rundb_collection()` (in module hax.runs), 17  
`get_rundb_database()` (in module hax.runs), 17  
`get_rundb_password()` (in module hax.runs), 17  
`get_sc_api_key()` (in module hax.slow\_control), 18  
`get_sc_data()` (in module hax.slow\_control), 18  
`get_sc_name()` (in module hax.slow\_control), 18  
`get_special_file_filename()` (in module hax.trigger\_data), 19  
`get_treemaker_name_and_class()` (in module hax.minitrees), 12  
`get_trigger_data()` (in module hax.trigger\_data), 19  
`get_user_id()` (in module hax.utils), 19  
`get_xenon100_dataset_number()` (in module hax.utils), 19  
`getConnection()` (hax.raw\_data.HTTPSClientAuthHandler method), 14

## H

hax (module), 20

hax.data\_extractor (module), 9  
hax.ipython (module), 10  
hax.minitrees (module), 10  
hax.misc (module), 13  
hax.paxroot (module), 13  
hax.pmt\_plot (module), 14  
hax.raw\_data (module), 14  
hax.recorrect (module), 15  
hax.runs (module), 16  
hax.slow\_control (module), 18  
hax.treemakers (module), 9  
hax.treemakers.common (module), 3  
hax.treemakers.cut\_booleans\_examples (module), 6  
hax.treemakers.examples (module), 7  
hax.treemakers.peak\_treemakers (module), 7  
hax.treemakers.trigger (module), 8  
hax.trigger\_data (module), 19  
hax.utils (module), 19  
`https_open()` (hax.raw\_data.HTTPSClientAuthHandler method), 14  
HTTPSClientAuthHandler (class in hax.raw\_data), 14  
`human_to_utc_datetime()` (in module hax.utils), 19

## I

`init()` (in module hax), 20  
`init_sc_interface()` (in module hax.slow\_control), 18  
`inspect_events()` (in module hax.raw\_data), 15  
`inspect_events_from_minitree()` (in module hax.raw\_data), 15  
`inspect_peaks()` (in module hax.raw\_data), 15  
`inspect_peaks_array()` (in module hax.raw\_data), 15  
`is_mc()` (in module hax.runs), 17  
IsolatedPeaks (class in hax.treemakers.peak\_treemakers), 7

## L

LargestPeakProperties (class in hax.treemakers.common), 6  
LargestTriggeringSignal (class in hax.treemakers.trigger), 8  
`load()` (in module hax.minitrees), 12  
`load_cache_file()` (in module hax.minitrees), 12  
`load_corrections()` (in module hax.runs), 17  
`load_pickles()` (in module hax.utils), 19  
`load_single_dataset()` (in module hax.minitrees), 12  
`load_single_minitree()` (in module hax.minitrees), 12  
`loop_body()` (hax.data\_extractor.DataExtractor method), 10  
`loop_over_dataset()` (in module hax.paxroot), 14  
`loop_over_datasets()` (in module hax.paxroot), 14

## M

`make_branch_selection()` (in module hax.data\_extractor), 10

make\_named\_array() (in module hax.data\_extractor), 10  
 mc\_data (hax.minitrees.TreeMaker attribute), 11  
 MultipleRowExtractor (class in hax.minitrees), 10

## N

never\_store (hax.minitrees.TreeMaker attribute), 11  
 never\_store (hax.treemakers.trigger.TailCut attribute), 9  
 nhits\_bounds (hax.treemakers.peak\_treemakers.IsolatedPeaks attribute), 7  
 nhits\_bounds (hax.treemakers.peak\_treemakers.SingleElectrons attribute), 8  
 NoMinitreeAvailable, 10

## O

open\_pax\_rootfile() (in module hax.paxroot), 14

## P

pax\_version\_independent (hax.minitrees.TreeMaker attribute), 11  
 pax\_version\_independent (hax.treemakers.common.Fundamentals attribute), 6  
 pax\_version\_independent (hax.treemakers.trigger.LargestTriggeringSignal attribute), 8  
 pax\_version\_independent (hax.treemakers.trigger.Proximity attribute), 9  
 peak\_cut\_list (hax.treemakers.peak\_treemakers.PeakExtractor attribute), 8  
 peak\_cut\_string (hax.treemakers.peak\_treemakers.PeakExtractor attribute), 8  
 peak\_fields (hax.treemakers.peak\_treemakers.PeakExtractor attribute), 8  
 peak\_properties\_to\_get (hax.treemakers.common.LargestPeakProperties attribute), 6  
 peak\_types (hax.treemakers.common.LargestPeakProperties attribute), 6  
 PeakExtractor (class in hax.treemakers.peak\_treemakers), 8  
 peaktypes (hax.treemakers.peak\_treemakers.PeakExtractor attribute), 8  
 plot\_on\_pmt\_arrays() (in module hax.pmt\_plot), 14  
 process\_event() (hax.minitrees.MultipleRowExtractor method), 10  
 process\_event() (hax.minitrees.TreeMaker method), 11  
 process\_events() (in module hax.raw\_data), 15  
 Proximity (class in hax.treemakers.trigger), 8

## R

raw\_data\_processor() (in module hax.raw\_data), 15  
 raw\_events() (in module hax.raw\_data), 15  
 recorrect\_rz() (in module hax.recorrect), 15  
 recorrect\_slxyz() (in module hax.recorrect), 15

recorrect\_s2xy() (in module hax.recorrect), 16  
 root\_to\_numpy() (in module hax.data\_extractor), 10

## S

save\_cache\_file() (in module hax.minitrees), 13  
 save\_pickles() (in module hax.utils), 19  
 select\_physical\_pulses() (hax.treemakers.trigger.Proximity method), 9  
 SingleElectrons (class in hax.treemakers.peak\_treemakers), 8  
 stop\_after (hax.treemakers.peak\_treemakers.PeakExtractor attribute), 8  
 StopEventLoop, 13

## T

tags\_selection() (in module hax.runs), 17  
 TailCut (class in hax.treemakers.trigger), 9  
 TimeDifferences (class in hax.treemakers.examples), 7  
 TotalProperties (class in hax.treemakers.common), 6  
 TreeMaker (class in hax.minitrees), 10

## U

UnknownSlowControlMonikerException, 18  
 update\_datasets() (in module hax.runs), 17  
 update\_treemakers() (in module hax.minitrees), 13  
 uses\_arrays (hax.minitrees.TreeMaker attribute), 11  
 utc\_timestamp() (in module hax.utils), 19

## V

version\_is\_consistent\_with\_policy() (in module hax.runs), 17  
 version\_tuple() (in module hax.runs), 17

## W

width\_bounds (hax.treemakers.peak\_treemakers.IsolatedPeaks attribute), 7  
 width\_bounds (hax.treemakers.peak\_treemakers.SingleElectrons attribute), 8

## Y

yield\_peak() (hax.treemakers.peak\_treemakers.IsolatedPeaks static method), 7