```
Help on class ElecMeter in nilmtk.elecmeter:

nilmtk.elecmeter.ElecMeter = class
ElecMeter(nilmtk.hashable.Hashable,
nilmtk.electric.Electric)
 |  Represents a physical electricity meter.
 |
 |  Attributes
 |  ----------
 |  appliances : list of Appliance objects
connected immediately downstream
 |    of this meter.  Will be [] if no
appliances are connected directly
 |    to this meter.
 |
 |  store : nilmtk.DataStore
 |
 |  key : string
 |      key into nilmtk.DataStore to access
data.
 |
 |  metadata : dict.
 |      See http://nilm-
metadata.readthedocs.org/en/latest/
dataset_metadata.html#elecmeter
 |
 |  STATIC ATTRIBUTES
 |  -----------------
 |
 |  meter_devices : dict, static class attribute
 |      See http://nilm-
metadata.readthedocs.org/en/latest/
dataset_metadata.html#meterdevice
 |
```

```
 |  Method resolution order:
 |      ElecMeter
 |      nilmtk.hashable.Hashable
 |      nilmtk.electric.Electric
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, store=None, metadata=None,
meter_id=None)
 |      Initialize self.  See help(type(self))
for accurate signature.
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  available_ac_types(self, physical_quantity)
 |      Finds available alternating current
types for a specific physical quantity.
 |
 |      Parameters
 |      ----------
 |      physical_quantity : str or list of
strings
 |
 |      Returns
 |      -------
 |      list of strings e.g. ['apparent',
'active']
 |
 |  available_columns(self)
 |      Returns
 |      -------
 |      list of 2-tuples of strings e.g.
```

```
[('power', 'active')]
 |
 |  available_physical_quantities(self)
 |      Returns
 |      -------
 |      list of strings e.g. ['power', 'energy']
 |
 |  building(self)
 |
 |  clear_cache(self, verbose=False)
 |      See Also
 |      --------
 |      _compute_stat
 |      _get_stat_from_cache_or_compute
 |      key_for_cached_stat
 |      get_cached_stat
 |
 |  dataset(self)
 |
 |  dominant_appliance(self)
 |      Tries to find the most dominant
appliance on this meter,
 |      and then returns that appliance object.
Will return None
 |      if there are no appliances on this
meter.
 |
 |  dropout_rate(self, ignore_gaps=True,
**loader_kwargs)
 |      Parameters
 |      ----------
 |      ignore_gaps : bool, default=True
 |          If True then will only calculate
dropout rate for good sections.
```

```
 |      full_results : bool, default=False
 |      **loader_kwargs : key word arguments for
DataStore.load()
 |
 |      Returns
 |      -------
 |      DropoutRateResults object if
`full_results` is True,
 |      else float
 |
 |  dry_run_metadata(self)
 |
 |  get_cached_stat(self, key_for_stat)
 |      Parameters
 |      ----------
 |      key_for_stat : str
 |
 |      Returns
 |      -------
 |      pd.DataFrame
 |
 |      See Also
 |      --------
 |      _compute_stat
 |      _get_stat_from_cache_or_compute
 |      key_for_cached_stat
 |      clear_cache
 |
 |  get_metadata(self)
 |
 |  get_source_node(self, **loader_kwargs)
 |
 |  get_timeframe(self)
 |
```

```
 |  good_sections(self, **loader_kwargs)
 |      Parameters
 |      ----------
 |      full_results : bool, default=False
 |      **loader_kwargs : key word arguments for
DataStore.load()
 |
 |      Returns
 |      -------
 |      if `full_results` is True then return
nilmtk.stats.GoodSectionsResults
 |      object otherwise return list of
TimeFrame objects.
 |
 |  instance(self)
 |
 |  is_site_meter(self)
 |
 |  key_for_cached_stat(self, stat_name)
 |      Parameters
 |      ----------
 |      stat_name : str
 |
 |      Returns
 |      -------
 |      key : str
 |
 |      See Also
 |      --------
 |      clear_cache
 |      _compute_stat
 |      _get_stat_from_cache_or_compute
 |      get_cached_stat
 |
```

```
 |  label(self, pretty=True)
 |      Returns a string describing this meter.
 |
 |      Parameters
 |      ----------
 |      pretty : boolean
 |          If True then just return the type
name of the dominant appliance
 |          (without the instance number) or
metadata['name'], with the
 |          first letter capitalised.
 |
 |      Returns
 |      -------
 |      string : A label listing all the
appliance types.
 |
 |  load(self, **kwargs)
 |      Returns a generator of DataFrames loaded
from the DataStore.
 |
 |      By default, `load` will load all
available columns from the DataStore.
 |      Specific columns can be selected in one
or two mutually exclusive ways:
 |
 |      1. specify a list of column names using
the `columns` parameter.
 |      2. specify a `physical_quantity` and/or
an `ac_type` parameter to ask
 |          `load` to automatically select
columns.
 |
 |      If 'resample' is set to 'True' then the
```

```
default behaviour is for
 |      gaps shorter than max_sample_period will
be forward filled.
 |
 |      Parameters
 |      ----------
 |      physical_quantity : string or list of
strings
 |          e.g. 'power' or 'voltage' or
'energy' or ['power', 'energy'].
 |          If a single string then load columns
only for that physical quantity.
 |          If a list of strings then load
columns for all those physical
 |          quantities.
 |
 |      ac_type : string or list of strings,
defaults to None
 |          Where 'ac_type' is short for
'alternating current type'.  e.g.
 |          'reactive' or 'active' or
'apparent'.
 |          If set to None then will load all AC
types per physical quantity.
 |          If set to 'best' then load the
single best AC type per
 |          physical quantity.
 |          If set to a single AC type then load
just that single AC type per
 |          physical quantity, else raise an
Exception.
 |          If set to a list of AC type strings
then will load all those
 |          AC types and will raise an Exception
```

```
if any cannot be found.
 |
 |      columns : list of tuples, using NILMTK's
vocabulary for measurements.
 |          e.g. [('power', 'active'),
('voltage', ''), ('energy', 'reactive')]
 |          `columns` can't be used if `ac_type`
and/or `physical_quantity` are set.
 |
 |      sample_period : int, defaults to None
 |          Number of seconds to use as the new
sample period for resampling.
 |          If None then will use
self.sample_period()
 |
 |      resample : boolean, defaults to False
 |          If True then will resample data
using `sample_period`.
 |          Defaults to True if `sample_period`
is not None.
 |
 |      resample_kwargs : dict of key word
arguments (other than 'rule') to
 |          `pass to pd.DataFrame.resample()`.
Defaults to set 'limit' to
 |          `sample_period / max_sample_period`
and sets 'fill_method' to ffill.
 |
 |      preprocessing : list of Node subclass
instances
 |          e.g. [Clip()].
 |
 |      **kwargs : any other key word arguments
to pass to `self.store.load()`
```

```
 |
 |      Returns
 |      -------
 |      Always return a generator of DataFrames
(even if it only has a single
 |      column).
 |
 |      Raises
 |      ------
 |      nilmtk.exceptions.MeasurementError if a
measurement is specified
 |      which is not available.
 |
 |  matches(self, key)
 |      Parameters
 |      ----------
 |      key : dict
 |
 |      Returns
 |      -------
 |      Bool
 |
 |  sample_period(self)
 |
 |  save(self, destination, key)
 |      Convert all relevant attributes to a
dict to be
 |      saved as metadata in destination at
location specified
 |      by key
 |
 |  total_energy(self, **loader_kwargs)
 |      Parameters
 |      ----------
```

```
 |      full_results : bool, default=False
 |      **loader_kwargs : key word arguments for
DataStore.load()
 |
 |      Returns
 |      -------
 |      if `full_results` is True then return
TotalEnergyResults object
 |      else returns a pd.Series with a row for
each AC type.
 |
 |  upstream_meter(self, raise_warning=True)
 |      Returns
 |      -------
 |      ElecMeterID of upstream meter or None if
is site meter.
 |
 |
----------------------------------------------
----------------------
 |  Class methods defined here:
 |
 |  load_meter_devices(store) from builtins.type
 |
 |
----------------------------------------------
----------------------
 |  Data descriptors defined here:
 |
 |  device
 |      Returns
 |      -------
 |      dict describing the MeterDevice for this
meter (sample period etc).
```

```
 |
 |  key
 |
 |  name
 |
 |
----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  meter_devices = {'CurrentCostTx': {'data_logger': {'creators': ['Jack ...
 |
 |
----------------------------------------------------------------------
 |  Methods inherited from nilmtk.hashable.Hashable:
 |
 |  __eq__(self, other)
 |      Return self==value.
 |
 |  __hash__(self)
 |      Return hash(self).
 |
 |  __ne__(self, other)
 |      Return self!=value.
 |
 |
----------------------------------------------------------------------
 |  Data descriptors inherited from nilmtk.hashable.Hashable:
 |
```

```
 |  __dict__
 |      dictionary for instance variables (if
defined)
 |
 |  __weakref__
 |      list of weak references to the object
(if defined)
 |
 |
-----------------------------------------------
----------------------
 |  Methods inherited from
nilmtk.electric.Electric:
 |
 |  activation_series(self, *args, **kwargs)
 |      Returns runs of an appliance.
 |
 |      Most appliances spend a lot of their
time off.  This function finds
 |      periods when the appliance is on.
 |
 |      Parameters
 |      ----------
 |      min_off_duration : int
 |          If min_off_duration > 0 then ignore
'off' periods less than
 |          min_off_duration seconds of sub-
threshold power consumption
 |          (e.g. a washing machine might draw
no power for a short
 |          period while the clothes soak.)
Defaults value from metadata or,
 |          if metadata absent, defaults to 0.
 |      min_on_duration : int
```

```
 |          Any activation lasting less seconds
than min_on_duration will be
 |          ignored.  Defaults value from
metadata or, if metadata absent,
 |          defaults to 0.
 |      border : int
 |          Number of rows to include before and
after the detected activation
 |      on_power_threshold : int or float
 |          Defaults to
self.on_power_threshold()
 |      **kwargs : kwargs for
self.power_series()
 |
 |      Returns
 |      -------
 |      list of pd.Series.  Each series contains
one activation.
 |
 |      .. note:: Deprecated
 |        `activation_series` will be removed in
NILMTK v0.3.
 |        Please use `get_activations` instead.
 |
 |  activity_histogram(self, period='D',
bin_duration='H', **kwargs)
 |      Return a histogram vector showing when
activity occurs.
 |
 |      e.g. to see when, over the course of an
average day, activity occurs
 |      then use `bin_duration='H'` and
`period='D'`.
 |
```

```
 |      Parameters
 |      ----------
 |      period : str. Pandas period alias.
 |      bin_duration : str. Pandas period alias
e.g. 'H' = hourly; 'D' = daily.
 |          Width of each bin of the histogram.
`bin_duration` must exactly
 |          divide the chosen `period`.
 |      Returns
 |      -------
 |      hist : np.ndarray
 |          length will be `period /
bin_duration`
 |
 |  available_power_ac_types(self)
 |      Finds available alternating current
types from power measurements.
 |
 |      Returns
 |      -------
 |      list of strings e.g. ['apparent',
'active']
 |
 |      .. note:: Deprecated in NILMTK v0.3
 |                `available_power_ac_types`
should not be used.  Instead please
 |                use
`available_ac_types('power').`
 |
 |  average_energy_per_period(self,
offset_alias='D', use_uptime=True,
**load_kwargs)
 |      Calculate the average energy per period.
e.g. the average
```

```
 |      energy per day.
 |
 |      Parameters
 |      ----------
 |      offset_alias : str
 |          A Pandas `offset alias`.  See:
 |          pandas.pydata.org/pandas-docs/
stable/timeseries.html#offset-aliases
 |      use_uptime : bool
 |
 |      Returns
 |      -------
 |      pd.Series
 |          Keys are AC types.
 |          Values are energy in kWh per period.
 |
 |  correlation(self, other, **load_kwargs)
 |      Finds the correlation between the two
ElecMeters. Both the ElecMeters
 |      should be perfectly aligned
 |      Adapted from:
 |      http://www.johndcook.com/blog/
2008/11/05/how-to-calculate-pearson-correlation-
accurately/
 |
 |      Parameters
 |      ----------
 |      other : an ElecMeter or MeterGroup
object
 |
 |      Returns
 |      -------
 |      float : [-1, 1]
 |
```

```
 |  entropy(self, k=3, base=2)
 |      This implementation is provided courtesy
NPEET toolbox,
 |      the authors kindly allowed us to
directly use their code.
 |      As a courtesy procedure, you may wish to
cite their paper,
 |      in case you use this function.
 |      This fails if there is a large number of
records. Need to
 |      ask the authors what to do about the
same!
 |      The classic K-L k-nearest neighbor
continuous entropy estimator
 |      x should be a list of vectors, e.g. x =
[[1.3],[3.7],[5.1],[2.4]]
 |      if x is a one-dimensional scalar and we
have four samples
 |
 |  get_activations(self, min_off_duration=None,
min_on_duration=None, border=1,
on_power_threshold=None, **kwargs)
 |      Returns runs of an appliance.
 |
 |      Most appliances spend a lot of their
time off.  This function finds
 |      periods when the appliance is on.
 |
 |      Parameters
 |      ----------
 |      min_off_duration : int
 |          If min_off_duration > 0 then ignore
'off' periods less than
 |          min_off_duration seconds of sub-
```

```
threshold power consumption
 |          (e.g. a washing machine might draw
no power for a short
 |          period while the clothes soak.)
Defaults value from metadata or,
 |          if metadata absent, defaults to 0.
 |      min_on_duration : int
 |          Any activation lasting less seconds
than min_on_duration will be
 |          ignored.  Defaults value from
metadata or, if metadata absent,
 |          defaults to 0.
 |      border : int
 |          Number of rows to include before and
after the detected activation
 |      on_power_threshold : int or float
 |          Defaults to
self.on_power_threshold()
 |      **kwargs : kwargs for
self.power_series()
 |
 |      Returns
 |      -------
 |      list of pd.Series.  Each series contains
one activation.
 |
 |  load_series(self, **kwargs)
 |      Parameters
 |      ----------
 |      ac_type : str
 |      physical_quantity : str
 |          We sum across ac_types of this
physical quantity.
 |      **kwargs : passed through to load().
```

```
 |
 |      Returns
 |      -------
 |      generator of pd.Series.  If a single
ac_type is found for the
 |      physical_quantity then the series.name
will be a normal tuple.
 |      If more than 1 ac_type is found then the
ac_type will be a string
 |      of the ac_types with '+' in between.
e.g. 'active+apparent'.
 |
 |  matches_appliances(self, key)
 |      Parameters
 |      ----------
 |      key : dict
 |
 |      Returns
 |      -------
 |      True if all key:value pairs in `key`
match any appliance
 |      in `self.appliances`.
 |
 |  min_off_duration(self)
 |
 |  min_on_duration(self)
 |
 |  mutual_information(self, other, k=3, base=2)
 |      Mutual information of two ElecMeters
 |      x,y should be a list of vectors, e.g. x
= [[1.3],[3.7],[5.1],[2.4]]
 |      if x is a one-dimensional scalar and we
have four samples
 |
```

```
 |      Parameters
 |      ----------
 |      other : ElecMeter or MeterGroup
 |
 |  on_power_threshold(self)
 |      Returns the minimum `on_power_threshold`
across all appliances
 |      immediately downstream of this meter.
If any appliance
 |      does not have an `on_power_threshold`
then default to 10 watts.
 |
 |  plot(self, ax=None, timeframe=None,
plot_legend=True, unit='W', plot_kwargs=None,
**kwargs)
 |      Parameters
 |      ----------
 |      width : int, optional
 |          Number of points on the x axis
required
 |      ax : matplotlib.axes, optional
 |      plot_legend : boolean, optional
 |          Defaults to True.  Set to False to
not plot legend.
 |      unit : {'W', 'kW'}
 |      **kwargs
 |
 |  plot_activity_histogram(self, ax=None,
period='D', bin_duration='H', plot_kwargs=None,
**kwargs)
 |
 |  plot_autocorrelation(self, ax=None)
 |      Plots autocorrelation of power data
 |      Reference:
```

```
 |      http://www.itl.nist.gov/div898/handbook/
eda/section3/autocopl.htm
 |
 |      Returns
 |      -------
 |      matplotlib.axis
 |
 |  plot_lag(self, lag=1, ax=None)
 |      Plots a lag plot of power data
 |      http://www.itl.nist.gov/div898/handbook/
eda/section3/lagplot.htm
 |
 |      Returns
 |      -------
 |      matplotlib.axis
 |
 |  plot_power_histogram(self, ax=None,
load_kwargs=None, plot_kwargs=None, range=None,
**hist_kwargs)
 |      Parameters
 |      ----------
 |      ax : axes
 |      load_kwargs : dict
 |      plot_kwargs : dict
 |      range : None or tuple
 |          if range=(None, x) then
on_power_threshold will be used as minimum.
 |      **hist_kwargs
 |
 |      Returns
 |      -------
 |      ax
 |
 |  plot_spectrum(self, ax=None)
```

```
 |      Plots spectral plot of power data
 |      http://www.itl.nist.gov/div898/handbook/
eda/section3/spectrum.htm
 |
 |      Code borrowed from:
 |      http://glowingpython.blogspot.com/
2011/08/how-to-plot-frequency-spectrum-with.html
 |
 |      Returns
 |      -------
 |      matplotlib.axis
 |
 |  power_series(self, **kwargs)
 |      Get power Series.
 |
 |      Parameters
 |      ----------
 |      ac_type : str, defaults to 'best'
 |      **kwargs :
 |          Any other key word arguments are
passed to self.load()
 |
 |      Returns
 |      -------
 |      generator of pd.Series of power
measurements.
 |
 |  power_series_all_data(self, **kwargs)
 |
 |  proportion_of_energy(self, other,
**loader_kwargs)
 |      Compute the proportion of energy of self
compared to `other`.
 |
```

```
 |      By default, only uses
other.good_sections().  You may want to set
 |
`sections=self.good_sections().intersection(othe
r.good_sections())`
 |
 |      Parameters
 |      ----------
 |      other : nilmtk.MeteGroup or ElecMeter
 |          Typically this will be mains.
 |
 |      Returns
 |      -------
 |      float [0,1] or NaN if other.total_energy
== 0
 |
 |  proportion_of_upstream(self, **load_kwargs)
 |      Returns a value in the range [0,1]
specifying the proportion of
 |      the upstream meter's total energy used
by this meter.
 |
 |  switch_times(self, threshold=40)
 |      Returns an array of pd.DateTime when a
switch occurs as defined by threshold
 |
 |      Parameters
 |      ----------
 |      threshold: int, threshold in Watts
between succcessive readings
 |      to amount for an appliance state change
 |
 |  uptime(self, **load_kwargs)
 |      Returns
```

```
 |      -------
 |      timedelta: total duration of all good
sections.
 |
 |  vampire_power(self, **load_kwargs)
 |
 |  when_on(self, on_power_threshold=None,
**load_kwargs)
 |      Are the connected appliances appliance
is on (True) or off (False)?
 |
 |      Uses `self.on_power_threshold()` if
`on_power_threshold` not provided.
 |
 |      Parameters
 |      ----------
 |      on_power_threshold : number, optional
 |          Defaults to
self.on_power_threshold()
 |      **load_kwargs : key word arguments
 |          Passed to self.power_series()
 |
 |      Returns
 |      -------
 |      generator of pd.Series
 |          index is the same as for chunk
returned by `self.power_series()`
 |          values are booleans
```