

---

# **FEAST**

***Release 3.1***

**Chandler Kemp and Clay Bell**

**Nov 10, 2020**



**CONTENTS:**

<b>1</b>	<b>FEAST modules</b>	<b>1</b>
1.1	field_simulation . . . . .	1
1.2	input_data_classes . . . . .	2
1.3	Detection Modules . . . . .	3
1.4	EmissionSimModules . . . . .	11
	<b>Python Module Index</b>	<b>17</b>



## FEAST MODULES

### 1.1 field\_simulation

A high level module for running simulations of multiple natural gas production sites and associated infrastructure.

### 1.2 input\_data\_classes

This module defines all classes used to store input data.

**class DataFile** (*notes='No notes provided', raw\_file\_name=None, data\_prep\_file=None*)

DataFile is an abstract super class that all data file types inherit from FEAST.

#### Parameters

- **notes** – A string containing notes on the object created
- **raw\_file\_name** – path (or list of paths) to a raw input file(s)
- **data\_prep\_file** – path to the file used to process input data and create the DataFile object

**class LeakData** (*notes='No notes provided', raw\_file\_name=None, data\_prep\_file=None, leak\_sizes=None*)

LeakData is designed to store all leak size data from a reference. It accommodates multiple detection methods within a single instance.

Creates a LeakData object

#### Parameters

- **notes** – a string containing notes on the object created
- **raw\_file\_name** – path to a raw input file
- **data\_prep\_file** – path to a script used to build the object from the raw data file
- **leak\_sizes** – list of leak sizes. If leaks were detected using multiple methods, leak\_sizes must be a dict with one key for each detection method

**define\_data** (*leak\_data=None, well\_counts=None, comp\_counts=None, detect\_methods=None*)

Check data formatting and set keys... there is exactly one detection method, leak\_data may be a list.

#### Parameters

- **leak\_data** – leak\_data must be a dict of emission rates if there are multiple detection methods. If there is exactly one detection method, leak\_data may be a list.
- **well\_counts** – lists the number of wells inspected by each detection method in keys

- **comp\_counts** – lists the number of components inspected by each detection method in keys
- **detect\_methods** – each detection method should have a unique key associated with it

**Returns** None

**class ProductionData** (*site\_prod=None, \*\*kwargs*)

Stores an array of production rates that may be associated with gas production sites

**Parameters**

- **prod\_dat** – an array of production rates
- **kwargs** – pass through

**class RepairData** (*notes='No notes provided', raw\_file\_name=None*)

RepairData is designed to store the costs of repairing leaks from a particular reference and associated notes.

**Parameters**

- **notes** – A string containing notes on the object created
- **raw\_file\_name** – path to a raw input file

**define\_data** (*repair\_costs=None*)

**Parameters** **repair\_costs** – list of costs to repair leaks

## 1.3 Detection Modules

### 1.3.1 Idar\_program

This module defines the LDARProgram class.

**class LDARProgram** (*time, gas\_field, tech\_dict*)

An LDAR program contains one or more detection methods and one or more repair methods. Each LDAR program records the find and repair costs associated with all detection and repair methods in the program. The LDAR program deploys runs the action methods of each detection and repair method contained in the program. The detection and repair methods determine their own behavior at each time step.

**Parameters**

- **time** – a Time object
- **gas\_field** – a GasField object
- **tech\_dict** – a dict containing all of the detection methods to be employed by the LDAR program. The dict must have the form {"name": DetectionMethod}. All of the relationships between detection methods and between detection methods and repair methods must be defined by the dispatch\_objects specified for each method.

**action** (*time, gas\_field*)

Runs the detect method for every tech in tech\_dict and runs the repair method :param time: the simulation time object :param gas\_field: the simulation gas\_field object :return:

### 1.3.2 abstract\_detection\_method

**class DetectionMethod** (*time, detection\_variables=None, op\_envelope=None, ophrs=None*)

DetectionMethod is an abstract super class that defines the form required for all detection methods

**Parameters** **time** – a Time object

**static check\_min\_max\_condition** (*condition, params*)

Checks a min-max condition defined by params. Supports float, integer, list and array based min max conditions. If the min-max condition is specified as a min float/integer and max float/integer, the numbers are placed in min and max lists each with length 1. The function returns True if the condition is between the min and max values, False otherwise. If the min and max values are array-like, the function returns true if the condition is between any pair of min-max values.

#### Parameters

- **condition** – condition to check (must be a number)
- **params** – a dict with ‘min’ and ‘max’ keys. The min and max values can be numbers or array-like.

#### Returns

**check\_op\_envelope** (*gas\_field, time, site\_index=None*)

Returns the status of the operating envelope. The method supports 8 types of operating envelope conditions:

1. A meteorological condition based on min-max values that apply to the whole field (eg. temperature)
2. A meteorological condition based on min-max values that are site-specific (eg. wind direction)
3. A meteorological condition based on a fail list that applies to the whole field (eg. precipitation type)
4. A meteorological condition based on a fail list that is site specific (possible but not expected)
5. A site condition based on min-max values that apply to the whole field (eg site production)
6. A site condition based on min-max values that are site-specific (possible but not expected)
7. A site condition based on a fail list that applies to the whole field (eg. site type)
8. A site condition based on a fail list that is site specific (possible but not expected).

#### Parameters

- **gas\_field** – A feast GasField object
- **time** – A feast Time object
- **site\_index** – Index to a specific site

**Return status** A string specifying the result of the operating envelope check. Can be one of 4 strings:

1. ‘field pass’
2. ‘field fail’
3. ‘site pass’
4. ‘site fail’

**check\_time** (*time*)

Determines whether or not the detection method is active during the present time step

**Parameters** **time** – A Time object

**Returns** True if check\_time passes, False otherwise

**choose\_sites** (*gas\_field, time, n\_sites, clear\_sites=True*)

Identifies sites to survey at this time step

**Parameters**

- **gas\_field** – A GasField object
- **time** – A Time object
- **n\_sites** – Max number of sites to survey at this time step
- **clear\_sites** – If true, clear sites selected from the queue. If False, leave sites in the queue. Leaving sites in the queue is useful for SiteMonitor type detection methods.

**Returns** None

**static empirical\_interpolator** (*test\_conditions, test\_results, sim\_conditions*)

Calculates the probability of detection by interpolating the value of test\_results between test\_conditions.

**Parameters**

- **test\_conditions** – conditions to be interpolated from
- **test\_results** – results associated with each condition listed in test\_conditions
- **sim\_conditions** – Nxk array of current conditions, where N is the number of emissions to consider, and k is the number of conditions

**Returns** an array of the probabilities of detection (dimension N)

**extend\_site\_queue** (*site\_inds*)

Add new sites to the site\_queue if they are not already in the queue

**Parameters** **site\_inds** – List of indexes to add to the queue

**Returns** None

**static find\_comp\_name** (*gas\_field, sitename, comp\_index*)

Determines the key for a component based on its index and site

**Parameters**

- **gas\_field** – a GasField object
- **sitename** – name of the site containing the component
- **comp\_index** – index of the component to consider

**Returns** The key for the component identified by comp\_index, or -1 if the component is not found.

**static find\_site\_name** (*gas\_field, site\_index*)

Determines the key for a site based on its index :param gas\_field: a GasField object :param site\_index: an integer indicating the index of the site to be considered :return: the key for the site identified by site\_index, or -1 if the key cannot be found.

**get\_current\_conditions** (*time, gas\_field, emissions, em\_id*)

Extracts conditions specified in self.detection\_variables

**Parameters**

- **time** – a Time object
- **gas\_field** – a GasField object



- **emissions** – a DataFrame of current emissions
- **em\_id** – emission indexes to consider

**Return conditions** an array (n\_emissions, n\_variables) of conditions for use in the PoD calculation

### 1.3.3 comp\_survey

This module defines the component level survey based detection class, CompSurvey.

**class CompSurvey** (*time, dispatch\_object, survey\_interval, survey\_speed, labor, site\_queue, detection\_probability\_points, detection\_probabilities, ophrs, comp\_survey\_index=0, site\_survey\_index=0, op\_env\_wait\_time=7, \*\*kwargs*)

Bases: *feast.DetectionModules.abstract\_detection\_method.DetectionMethod*

This class specifies a component level, survey based detection method. A component level method identifies the specific component that is the source of the emission at the time of detection. Examples of components include connectors, valves, open ended lines, etc. A survey based method inspects emissions at specific moments in time (as opposed to a monitor method that continuously monitors for emissions). The class has three essential attributes:

1. An operating envelope function to determine if conditions satisfy requirements for the method to be deployed
2. A probability of detection surface function to determine which emissions are detected
3. The ability to call a follow up action

#### Parameters

- **time** – a Time object
- **dispatch\_object** – an object to dispatch for follow-up actions (typically a Repair method)
- **survey\_interval** – Time between surveys (float–days)
- **survey\_speed** – Speed of surveys (float–components/hr)
- **labor** – Cost of surveys (float–\$/hr)
- **site\_queue** – Sites to survey (list of ints)
- **detection\_probability\_points** – Set of conditions at which the probability was measured (array)
- **detection\_probabilities** – Set of probabilities that were measured (array)
- **comp\_survey\_index** – Index of the component to be surveyed next (int)
- **site\_survey\_index** – Index of the site to be surveyed next (or currently under survey) (int)
- **op\_env\_wait\_time** – Time to wait if operating envelope conditions fail part way through a site before moving to the next site (float–days)
- **ophrs** – range of times of day when the method performs survey (dict–hours). eg: { ‘begin’: 8, ‘end’: 17 }

**action** (*site\_inds=None, emit\_inds=None*)

Adds sites to the queue for future inspections. This method is expected to be called by another detection method or by an LDAR program.

**Parameters**

- **site\_inds** – List of sites to add to the queue
- **emit\_inds** – Not used.

**Returns** None**detect** (*time, gas\_field, emissions*)

The detect method checks that the current time is within operating hours, selects emitters to inspect, determines which emissions are detected and dispatches the follow up action for detected emissions.

**Parameters**

- **time** – a Time object
- **gas\_field** – a GasField object
- **emissions** – a DataFrame of current emissions

**detect\_prob\_curve** (*time, gas\_field, em\_surveyed, emissions*)

This function determines which leaks are found given an array of indexes defined by “cond.” The method uses attributes of the DetectionMethod and interpolation.

**Parameters**

- **time** – a Time object
- **gas\_field** – a GasField object
- **em\_surveyed** – Array of emission\_id to consider
- **emissions** – a DataFrame of current emissions

**Return detect** the indexes of detected leaks (array of ints)**emitters\_surveyed** (*time, gas\_field, emissions*)

Determines which emitters are surveyed during the current time step. Accounts for the number of components surveyed per timestep, the number of components at each site, and the component and site at which the survey left off in the previous time step

**Parameters**

- **time** – a Time object
- **gas\_field** – a GasField object
- **emissions** – a DataFrame of current emissions

**Return emitter\_inds** emission\_id of emissions to evaluate at this timestep (list of ints)

### 1.3.4 site\_survey

The site\_survey module defines the site level level survey based detection class, SiteSurvey.

**class SiteSurvey** (*time, dispatch\_object, sites\_per\_day, site\_cost, detection\_probability\_points, detection\_probabilities, op\_envelope=None, ophrs=None, site\_queue=None, survey\_interval=None, \*\*kwargs*)

Bases: *feast.DetectionModules.abstract\_detection\_method.DetectionMethod*

SiteSurvey specifies a site level, survey based detection method. A site level detection method is sensitive to the total emissions from a site. If emissions are detected, the site is identified as the source of emissions rather than a component on the site. Survey based detection methods search for emissions at a specific moment in time (as opposed to monitor detection methods that continuously scan sites for new emissions). The class has three essential attributes:

1. An operating envelope function to determine if the detection method can be applied
2. A probability of detection surface function to determine which emissions are detected
3. The ability to dispatch a follow up action

#### Parameters

- **time** – a Time object
- **dispatch\_object** – the object that SiteSurvey will pass flagged site indexes to (DetectionMethod or Repair)
- **sites\_per\_day** – the number of sites that the method can survey in one day (int)
- **site\_cost** – the cost per site of the detection method (\$/site–float)
- **detection\_probability\_points** – The conditions at which the detection probability was measured. (NxM array, where N is the number of distinct conditions and M is the number of variables (up to two)).
- **detection\_probabilities** – The list of probabilities of detection associated with every point in detection\_probability\_points (array of shape N, where N is the number of conditions with an associated probability of detection).
- **op\_envelope** – The set of conditions underwhich the SiteSurvey may operate. The op\_envelope must be passed as a dict with the following form–  
 {‘parameter name’: {‘class’: int, ‘min’: list of minimum conditions, ‘max’: list of maximum conditions}}
- Unique minima can be defined for every site in a list if the op\_envelope ‘class’ is site specific. Multiple minima can be defined in a list for a single site if multiple ranges should be considered.
- **ophrs** – The times of day when the SiteSurvey can be deployed. Should be a dict:  
 {‘begin’: hour integer, ‘end’: hour integer}
- **site\_queue** – an ordered list of sites to be surveyed. An LDAR program may update this list.
- **survey\_interval** – The time between surveys (int–days)

**action** (*site\_inds=None, emit\_inds=None*)

Action to add sites to queue. Expected to be called by another detection method or by an LDAR program

#### Parameters

- **site\_inds** – List of sites to add to the queue
- **emit\_inds** – Not used.

**Returns** None

**detect** (*time, gas\_field, emissions*)

The detection method implements a survey-based detection method model

#### Parameters

- **time** – an object of type Time (defined in feast\_classes)
- **gas\_field** – an object of type GasField (defined in feast\_classes)
- **emissions** – an Emissions object

**Returns** None

**detect\_prob\_curve** (*time, gas\_field, site\_inds, emissions*)

This function determines which sites are passed to the dispatch\_object by SiteSurvey. The function sums all emissions at a site, determines the probability of detection given the total site emissions and present conditions, then determines whether or not the site is flagged according to the probability.

**Parameters**

- **time** – Simulation time object
- **gas\_field** – Simulation gas\_field object
- **site\_inds** – The set of sites to be considered
- **emissions** – an object storing all emissions in the simulation

**Return detect** the indexes of detected leaks

**sites\_surveyed** (*gas\_field, time*)

Determines which sites are surveyed during the current time step. Accounts for the number of sites surveyed per timestep

**Parameters**

- **gas\_field** –
- **time** –

**Return site\_inds** the indexes of sites to be surveyed during this timestep.

### 1.3.5 site\_monitor

The site\_monitor module defines the site monitor detection class, SiteMonitor.

**class SiteMonitor** (*time, dispatch\_object, time\_to\_detect\_points, time\_to\_detect\_days, ophrs=None, capital=0, site\_queue=None, \*\*kwargs*)

Bases: *feast.DetectionModules.abstract\_detection\_method.DetectionMethod*

This class specifies a site level continuous monitoring method. A site monitor continuously observes emissions from an entire site and determines when an action should be dispatched at the site. The method has three essential characteristics:

1. A list of the sites where the method applies
2. A time-to-detect surface specified as a list of conditions and associated mean detection times
3. The ability to dispatch a follow up action

**Parameters**

- **time** – a Time object
- **dispatch\_object** – the object to dispatch for follow up actions
- **time\_to\_detect\_points** – The conditions at which the time to detection was measured. (NxM array, where N is the number of distinct conditions and M is the number of variables (up to two)).
- **time\_to\_detect\_days** – The list of probabilities of detection associated with every point in detection\_probability\_points (array of shape N, where N is the number of conditions with an associated probability of detection).

- **ophrs** – The times of day when the SiteMonitor is operational. Should be a dict:  
{‘begin’: hour integer, ‘end’: hour integer}
- **capital** – The total cost of installing the site monitor system in the simulation (float–\$)
- **site\_queue** – A list of sites where the site monitor system is installed

**action** (*site\_inds=None, emit\_inds=None*)

Action to add sites to queue. Expected to be called by another detection method or by an LDAR program

**Parameters**

- **site\_inds** – List of sites to add to the queue
- **emit\_inds** – Not used.

**Returns** None

**detect** (*time, gas\_field, emissions*)

The detection method implements a continuous monitor detection method model

**Parameters**

- **time** – a Time object
- **gas\_field** – a GasField object
- **emissions** – a DataFrame containing emission data to evaluate

**Returns** None

**detect\_prob\_curve** (*time, gas\_field, site\_inds, emissions*)

Determines which sites are passed to the dispatch method. In this case, the sites to pass are determined by calculating a probability of detection based on the simulation time resolution (*time.delta\_t*) and the mean time to detection

**Parameters**

- **time** – simulation Time object
- **gas\_field** – simulation GasField object
- **site\_inds** – the set of sites to be considered
- **emissions** – an object storing all emissions in the simulation

**Return detect** the indexes of detected leaks

**static prob\_detection** (*time, ttd*)

Calculates the probability of detection during a timestep of length *time.delta\_t* and a mean time to detection *ttd*

**Parameters**

- **time** – Simulation time object
- **ttd** – mean time to detection (float–days)

**Returns** the probability of detection during this timestep

### 1.3.6 repair

This module defines the Repair class. Repair may be called by detection objects as follow up actions.

**class Repair** (*repair\_delay=0*)

Bases: object

Defines a repair process. A repair process determines when emissions are ended by an LDAR program and the associated costs.

**Parameters** **repair\_delay** – The time between when an emission is passed to Repair and when it is removed from the simulation (float–days)

**action** (*site\_inds=None, emit\_inds=None*)

adds emissions to the to\_repair queue.

**Parameters**

- **site\_inds** – not used
- **emit\_inds** – A list of emission indexes to repair

**Returns** None

**repair** (*time, emissions*)

Adjusts the emission end time based on the current time and the repair delay time. If the null emission end time comes before the repair time, the end time is not changed.

**Parameters**

- **time** – a Time object
- **emissions** – an Emission object

**Returns** None

## 1.4 EmissionSimModules

### 1.4.1 emission

A class for storing emission properties and functions for modifying emission properties throughout a simulation are defined in this module.

**class Emission** (*flux=(), reparable=True, site\_index=(), comp\_index=(), start\_time=0, end\_time=inf, repair\_cost=(), emission\_id=None*)

Bases: object

Stores all properties of all emissions that exist at a particular instant in a simulation.

**Parameters**

- **flux** – An array of emission rates (array of floats–gram/second)
- **reparable** – An array of True/False values to indicate whether or not an emission is reparable
- **site\_index** – An array indicating the index of the site that contains every emission
- **comp\_index** – An array indicating the index of the component that is the source of each emission
- **start\_time** – An array specifying the time when every emission begins

- **end\_time** – An array specifying the time when every emission will end (days)
- **emission\_id** –
- **repair\_cost** – An array storing the cost of repairing every emission (\$)

**em\_rate\_in\_range** (*t0, t1, reparable=None*)

Returns the sum of emissions that existed between t0 and t1 integrated over the time period :param t0: beginning of interval (days) :param t1: end of interval (days) :param reparable: boolean condition. If set, only returns emissions with a matching reparable property :return: Average emission rate between t1 and t0 (g/s)

**extend** (*\*args*)

Extends the existing emissions data frame with all of the entries in args :param args: a list of Emission objects :return:

**get\_current\_emissions** (*time*)

Returns all emissions that exist at time.current\_time :param time: a Time object :return: a DataFrame of current emissions

**get\_emissions\_in\_range** (*t0, t1, reparable=None*)

Returns all emissions that existed between t0 and t1 :param t0: beginning of interval (days) :param t1: end of interval (days) :param reparable: boolean condition. If set, only returns emissions with a matching reparable property :return: a DataFrame of all emissions that existed at any time in the interval t0:t1

**bootstrap\_emission\_maker** (*n\_em\_in, comp\_name, site, time, start\_time=None, reparable=True*)

Create leaks using a bootstrap method.

#### Parameters

- **n\_em\_in** – number of leaks to generate
- **comp\_name** – key to a Component object in site.comp\_dict
- **site** – a Site object
- **time** – a Time object
- **start\_time** – the times at which each emission begins
- **reparable** – Specifies whether emissions should be reparable or not (boolean)

**comp\_indexes\_fcn** (*site, comp\_name, n\_inds*)

Returns an array of indexes to associate with new emissions

#### Parameters

- **site** – a EmissionSimModules.simulation\_classes.Site object
- **comp\_name** – name of a component contained in Site.comp\_dict
- **n\_inds** – Integer of indexes to generate

**Returns** An array of indexes in the range specified for the relevant component

**emission\_objects\_generator** (*dist\_type, emission\_data\_path, custom\_emission\_maker=None*)

emission\_objects\_generator is a parent function that will be called to initialize gas fields

#### Parameters

- **dist\_type** – Type of leak distribution to be used
- **leak\_data\_path** – Path to a leak data file

**permitted\_emission** (*n\_emit, sizes, duration, time, site, comp\_name, start\_time*)

Creates an emission object specifying new permitted emissions

**Parameters**

- **n\_emit** – number of emissions to create
- **sizes** – a list of leak sizes from which to specify the emission rate
- **duration** – a float defining the duration of the emission
- **time** – a Time object
- **site** – a Site object
- **comp\_name** – Name of the component to be considered from within site.comp\_dict
- **start\_times** – array of times at which emissions start

**Returns** an Emission object

## 1.4.2 infrastructure\_classes

This module stores component, gasfield and site classes to represent infrastructure in a simulation

```
class Component (repair_cost_path=None,      emission_data_path=None,      base_reparable=None,
                  custom_emission_maker=None,      emission_production_rate=0,      emis-
                  sion_per_comp=None, episodic_emission_sizes=[0], episodic_emission_per_day=0,
                  episodic_emission_duration=0,      vent_sizes=[0],      vent_period=inf,
                  vent_starts=array([],      dtype=float64),      vent_duration=0,      name='default',
                  null_repair_rate=None, dist_type='bootstrap')
```

Bases: object

A class to store parameters defining a component (for example, name, leak production rate, leak size distribution, etc)

**Parameters**

- **repair\_cost\_path** – path to a repair cost data file
- **emission\_data\_path** – path to an emission data file
- **base\_reparable** – Defines whether emissions generated are reparable with a boolean true/false
- **custom\_emission\_maker** – Optional custom defined function for creating new emissions
- **emission\_production\_rate** – The rate at which new emissions are created (emissions per day per component)
- **emission\_per\_comp** – The number of emissions expected per component (must be less than 1) If emission\_per\_comp is left as None, then emission\_per\_comp is set equal to the emissions per component recorded in the file at emission\_data\_path.
- **episodic\_emission\_sizes** – A list of emission sizes to draw from for episodic emissions (g/s)
- **episodic\_emission\_per\_day** – The average frequency at which episodic emissions occur (1/days)
- **episodic\_emission\_duration** – The duration of episodic emissions (days)
- **vent\_sizes** – A list of emission sizes for periodic emissions (g/s)
- **vent\_period** – The time between emissions (days)
- **vent\_duration** – the time that a periodic vent persists (days)



- **vent\_starts** – the time at which the first periodic vent occurs at each component in the simulation
- **name** – A name for the instance of Component
- **null\_repair\_rate** – the rate at which fugitive emissions are repaired. If None, a steady state assumption is enforced based on `emission_production_rate` and `emission_per_comp`.
- **dist\_type** – The type of distribution to be used in determining emission rates for new emissions

**class GasField** (*time=None, sites=None, emissions=None, met\_data\_path=None*)

Bases: `object`

GasField accommodates all data that defines a gas field at the beginning of a simulation.

#### Parameters

- **time** – A FEAST time object
- **sites** – a dict of sites like this: {'name': {'number': n\_sites, 'parameters': site\_object}}
- **emissions** – A FEAST emission object to be used during the simulations
- **met\_data\_path** – A path to a met data file

**emerging\_emissions** (*time*)

Defines emissions that emerge during a simulation :param time: :return:

**static emission\_maker** (*n\_leaks, new\_leaks, comp\_name, n\_comp, time, site, n\_episodic=None*)

Updates an Emission object with new values returned by `emission_size_maker` and assigns unique indexes to them

#### Parameters

- **n\_leaks** – number of new leaks to create
- **new\_leaks** – a leak object to extend
- **comp\_name** – name of a component object included in `site.comp_dict`
- **n\_comp** – the number of components to model
- **time** – a time object
- **site** – a site object
- **n\_episodic** – number of episodic emissions to create
- **start\_time** – time at which the new emissions begin emitting

**Returns** None

**emission\_size\_maker** (*time*)

Creates a new set of leaks based on attributes of the gas field :param time: a time object (the parameter `delta_t` is used) :return new\_leaks: the new leak object

**get\_met** (*time, parameter\_names, interp\_modes='mean', ophrs=None*)

Return the relevant meteorological condition, accounting for discrepancies between simulation time resolution and data time resolution

#### Parameters

- **time** – time object
- **parameter\_names** – specify a list of meteorological conditions to return
- **interp\_modes** – can be a list of strings: mean, median, max or min

- **ophrs** – Hours to consider when interpolating met data should be of form {'begin': 5, 'end':17}

**Return met\_conds** dict of meteorological conditions

**initialize\_emissions** (*time*)

Create emissions that exist at the beginning of the simulation

**Parameters time** –

**Return initial\_emissions**

**met\_data\_maker** (*start\_hr=0*)

Creates a dict to store met data derived from a Typical Meteorological Year file. The data may be rotated so that the simulation begins at any hour in the TMY file. :param start\_hr: The hour at which the simulation should begin. :return: None

**set\_indexes** ()

Counts components for each site and assigns appropriate indexes

**class Site** (*name='default', comp\_dict=None, prod\_dat=None*)

Bases: object

A class to store the number and type of components associated with a site.

**Parameters**

- **name** – The name of the site object (a string)
- **comp\_dict** – A dict of components at the site, for example: {'name': {'number': 650, 'parameters': Component()}}
- **prod\_dat** –

### 1.4.3 simulation\_classes

simulation\_classes stores the classes used to represent time, results and financial settings in simulations.

**class Scenario** (*time, gas\_field, ldar\_program\_dict*)

Bases: object

A class to store all data specifying a scenario and the methods to run and save a realization

**Parameters**

- **time** – Time object
- **gas\_field** – GasField object
- **ldar\_program\_dict** – dict of detection methods and associated data

**check\_timestep** ()

Prints a warning if time.delta\_t is greater than the duration of some permitted emissions

**Parameters**

- **gas\_field** – a GasField object
- **time** – a Time object

**Returns** None

**run** (*dir\_out='Results', display\_status=True*)

run generates a single realization of a scenario.

**Parameters**

- **dir\_out** – path to a directory in which to save results (string)
- **display\_status** – if True, display a status update whenever 10% of the time steps are completed

**Returns** None

**save** (*dir\_out*)

Save results to a file

**Parameters** **dir\_out** – Name of directory in which to save output file.

**class Time** (*delta\_t=1, end\_time=365, current\_time=0*)

Bases: `object`

Instances of the time class store all time related information during a simulation

**Parameters**

- **delta\_t** – length of one timestep (days)
- **end\_time** – length of the simulation (days)
- **current\_time** – current time in a simulation (days)



## PYTHON MODULE INDEX

### f

`feast.DetectionModules.abstract_detection_method`,  
3  
`feast.DetectionModules.comp_survey`, 5  
`feast.DetectionModules.ldar_program`, 3  
`feast.DetectionModules.repair`, 10  
`feast.DetectionModules.site_monitor`, 9  
`feast.DetectionModules.site_survey`, 7  
`feast.EmissionSimModules.emission_class_functions`,  
11  
`feast.EmissionSimModules.infrastructure_classes`,  
13  
`feast.EmissionSimModules.simulation_classes`,  
15  
`feast.field_simulation`, 1  
`feast.input_data_classes`, 2