Fugitive Emissions Abatement Simulation Toolkit User Guide

FEAST v3.1

Guide & technical documentation

Table of Contents

Introduction

Since its publication in 2016, the Fugitive Emissions Abatement Simulation Toolkit (FEAST) has become a widely known simulation tool used by regulators and technologists to evaluate and compare emissions from natural gas infrastructure under disparate Leak Detection and Repair (LDAR) programs. An LDAR program consists of the technology used for leak detection, the implementation of the detection technology, and the leak repair process. For example, a traditional LDAR program might consist of a Flame Ionization Detector (FID), an operator who periodically surveys natural gas equipment for leaks using the FID and a crew that repairs leaks once they are identified. Every step in identifying and repairing leaks is included in the definition of an LDAR program.

FEAST models emissions from a natural gas field as a dynamic process. FEAST generates an initial set of leaks distributed throughout a natural gas field and then adds new leaks to the field through time. Every detection method in an LDAR program is assigned a probability of detection (PoD) function that determines the probability that an emission will be detected during a survey or (in the case of continuous monitor technologies) during a particular time interval. Detected emissions are classified as either vents or leaks. Only leaks are passed to the repair process in the LDAR program. The repair process removes the leaks from the simulation at the appropriate time based on parameters of the repair process, resulting in emissions mitigation relative to the null scenario. The total emissions integrated throughout the simulation provides a quantitative description of the performance of the simulated LDAR program that can be compared with other LDAR scenarios. The economic value of the LDAR program is finally estimated based on the emissions mitigated and the operating costs of the LDAR program. The dynamic model naturally accounts for the frequency of LDAR surveys, and illustrates the value of reducing the rate at which leaks are created as well as increasing the rate at which leaks are found.

Modifications between FEAST 1.0 and FEAST 3.1

For those familiar with FEAST 1.0, this section highlights the most significant changes implemented in more recent revisions. A comparison of FEAST revisions is shown in

Table 1, with a more detailed discussion following. The description here is conceptual; details of the FEAST 3.1 implementation are provided in the Model Structure section.

Table 1: Comparison of FEAST revisions

FEAST REVISION	1.0	2.0	3.0	3.1
Release Date	2016	2017	Mar. 2020	Dec. 2020
Software Language	MATLAB	Python	Python	Python
Detection Methodology	Simulated plume dispersion & concentration field	Simulated plume dispersion & concentration field	Analytical sigmoidal probability of detection curve (2D)	Empirical probability of detection surface (3D)
Survey Time	Instantaneous	Instantaneous	Finite	Finite
Tiered Detection Programs	Not supported – Leaks instantly repaired	Not supported – Leaks instantly repaired	Supported – Special class	Supported – Modular class structure
Emission Categories (Vents vs Fugitive)	Not supported	Not supported	Supported	Supported
Operating Envelope	Not supported	Not supported	Not supported	Supported
Meteorological Data	Only for dispersion model	Only for dispersion model	Not supported	Supported
Dispatch Rules	Not supported	Not supported	Not supported	Supported
Continuous monitors	Supported	Supported	Not supported	Supported

FEAST 2.0: Simulation ported from Matlab to Python

FEAST 2.0 was released as a Python package in 2017. FEAST 2.0 improved accessibility to the model since Python is free and open source. FEAST 2.0 also included minor changes to the model structure to facilitate modeling multiple instances of similar technologies and improve simulation speed.

FEAST 3.0: Probability of Detection curves, vents, finite time surveys and tiered detection

FEAST 3.0 was presented at the American Geophysical Union Fall Meeting in 2019 and released online in January 2020. Extensive documentation was not developed in anticipation of further revisions for FEAST 3.1. FEAST 3.0 included four major revisions to FEAST 2.0, as described below.

Probability of Detection (PoD) curves

FEAST 2.0 simulated plume dispersion using a Gaussian model and then invoked a process-model of leak detection technologies based on the simulated methane concentration field. FEAST 3.0 used PoD curves to determine which emissions were detected. FEAST 3.0 required the PoD curve to be defined as a specific sigmoid shaped function of emission rate characterized by the emission size that would be detected 50% of the time, and a width parameter. Since the PoD curve did not depend on the dispersion model used in FEAST 2.0, the dispersion model was eliminated in addition to the process-based models of detection.

The PoD curve approach provides a method for simulating technologies by using empirical detection rates directly rather than relying on a process-based model that must be carefully calibrated to field conditions. The PoD curve can be measured in field trials by exposing the detection method to many known emissions and recording the fraction of time that emissions are detected relative to their emission rate. The empirical result naturally accounts for complex plume dispersion patterns that might occur on well pads but are difficult to simulate. Furthermore, the PoD curve can be used to represent any technology that is sensitive to emission rate.

The dispersion model in FEAST 2.0 required windspeed data to determine downwind concentrations, but the PoD method in FEAST 3.0 was independent of windspeed. Therefore, support for wind parameters was removed from FEAST 3.0

The PoD approach in FEAST 3.0 complicated simulation of continuous monitor technologies. In FEAST 2.0, detection of an emission with a specific rate was deterministic given a particular wind speed and direction. The time between when an emission occurred and when it was detected depended on the wind conditions. In FEAST 3.0, the time to detection became probabilistic based on the PoD curve. In order to simulate the performance of a continuous monitor, the PoD had to be set based not only on the properties of the monitor, but also the frequency that the PoD curve was called during the simulation. FEAST 3.0 allowed continuous monitors to be simulated but did not provide a robust or intuitive method for their representation.

Vents

FEAST 2.0 simulated fugitive emissions only, while FEAST 3.0 simulated vents as well as fugitive emissions. Including vents provides the ability to simulate the costs incurred by false positives from detection methods that cannot distinguish between vents and fugitive emissions.

Finite time surveys

FEAST 2.0 simulated all surveys as though they occurred instantaneously, while FEAST 3.0 required a survey speed parameter to be set that determined the time between when a survey began and a detection method reached a particular emission. This added functionality allowed for an accurate representation of the maximum frequency of surveys that can be maintained in simulation with many sites and one survey team

Tiered detection LDAR programs

FEAST 2.0 simulated the cost of programs assuming that every leak that was detected could be repaired immediately. FEAST 3.0 supports LDAR programs that identify sites with anomalously high emissions which are then passed to a secondary detection method to determine the specific components that are emitting. In FEAST 3.0, the site-level survey and follow up survey were both encoded in the same LDAR program object.

FEAST 3.1: Operating envelopes, meteorological data, PoD surfaces, chained detection programs, dispatch rules, continuous monitors

FEAST 3.1 expanded on FEAST 3.0 to make the model more representative of emerging LDAR programs, provide additional flexibility in the types of LDAR programs simulated and provide support for programs incorporating continuous emission monitors. These revisions are described in further detail below.

Operating envelope

Most detection methods are limited to operating in a range of meteorological conditions. For example, a detection method may only be certified for use in winds below a particular speed or require a minimum ceiling level for overhead clouds. If operating conditions are not suitable for a detection method the deployment may be delayed until conditions improve, resulting in an increased level of emissions relative to an idealized method without these environmental limitations imposed.

FEAST 3.1 allows an operating envelope to be assigned to every detection method in an LDAR program. The operating envelope can be defined for any number of conditions from a list of supported parameters. The supported parameters include both meteorological parameters and site characteristics. While FEAST 3.0 simulated detection methods that can always be deployed, FEAST 3.1 simulates detection methods that can be only be deployed when environmental conditions are satisfied.

Meteorological data

The operating envelope described above requires a simulation of meteorological conditions in order to be useful. FEAST 1.0 and 2.0 allowed the user to provide hourly wind speed and direction data that were then used in simulations. In addition, FEAST 1.0 and 2.0 would randomly choose the Pasquill Stability class at each timestep (within limits determined by the wind speed). FEAST 3.0 provided no support for meteorological data. FEAST 3.1 supports meteorological data that specifies any of the following hourly parameters: wind speed, wind direction, temperature, relative humidity, precipitation, albedo, ceiling height, cloud cover and solar intensity. The input data structure is designed for use with typical meteorological year (TMY) data released by the National Renewable Energy Laboratory.

PoD Surfaces

While FEAST 3.0 required a PoD function with emission rate as the only independent variable, FEAST 3.1 supports a PoD surface with up to two independent variables. The independent variables can be any meteorological parameter and/or emission rate.

Supporting PoD functions with two independent variables required a method that did not assume an arbitrary functional form in order to represent technologies across multiple dimensions. The

PoD surface in FEAST 3.1 is defined using linear interpolation from empirical data. The new linear interpolation method supports PoD functions with arbitrary shape, including but not limited to the sigmoid shape assumed in FEAST 3.0. The interpolation method also eliminates the need for users to understand the functional form in order to fit their method's data to it; users simply enter the PoD data they have for a detection method and FEAST 3.1 will interpolate from that data.

Chained Detection Programs

FEAST 3.1 created a new data structure to separate detection methods, repair methods and LDAR programs into multiple objects. The new data structure facilitates simulation of LDAR programs that include multiple detection and repair methods. In FEAST 3.0, the only LDAR programs with multiple detection methods that were supported consisted of a preliminary site level survey coupled with a component level follow up survey. In FEAST 3.1, an arbitrary number of detection methods can be passed to an LDAR program, and the relationship between detection methods can be customized by the user. In addition, an LDAR program may include multiple repair methods with different time delays and costs.

Dispatch rules

FEAST 3.1 supports LDAR programs which dispatch different detection methods depending on the site or emission detection by other methods. While FEAST 3.0 only supported dispatching detection methods based on survey intervals or a site level survey in a tiered detection program, FEAST 3.1 supports dispatching detection methods based on any number of other detection methods, survey intervals or site type.

Continuous Monitors

FEAST 3.1 provides a pre-built model of continuous monitors. The continuous monitor representation allows users to specify a Mean Time to Detection (MTD) surface rather than a PoD surface. FEAST uses the MTD surface to calculate the PoD for emissions at each timestep given the time resolution used in the simulation. The MTD surface can also have up to two independent variables, such as wind speed and emission rate. The MTD surface specifies the expected time for detection if the conditions were to remain constant. While FEAST 3.0 required users to specify a PoD curve that depended on the simulation's time resolution, FEAST 3.1 requires users to specify an MTD surface that is independent of simulation settings and computes the appropriate PoD surface dynamically.

Model Structure

The Scenario Class

A FEAST *Scenario*, requires the user to define:

- A *GasField*
- One or more *LDARProgram(s)*
- A simulation *Time* object

The *GasField* defines the emissions model. The simulation will evaluate the detection process and resulting emissions mitigation from each *LDARProgram* independently. The *Time* object defines the duration and resolution of the simulation.

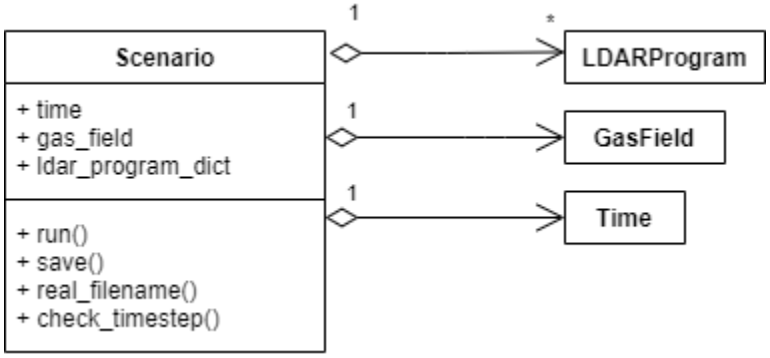

Figure 1: FEAST Scenario class and associations

The GasField Class

The data used to specify the emissions model are stored in an instance of the FEAST class *GasField*. The data structure is hierarchical (see Figure 2): it contains instances of the FEAST class *Site* which in turn contains instances of the class *Component*. The *Component* class stores all of the parameters necessary to specify a potential emission source in FEAST. The *Site* class specifies the number of each type of component that exist at each site. The *GasField* class stores the number of sites of each type, as well as meteorological data. A `GasField` also contains an instance of the `Emission` class. The `Emission` class specifies all of the emissions that will exist in the simulation without a simulated LDAR program.

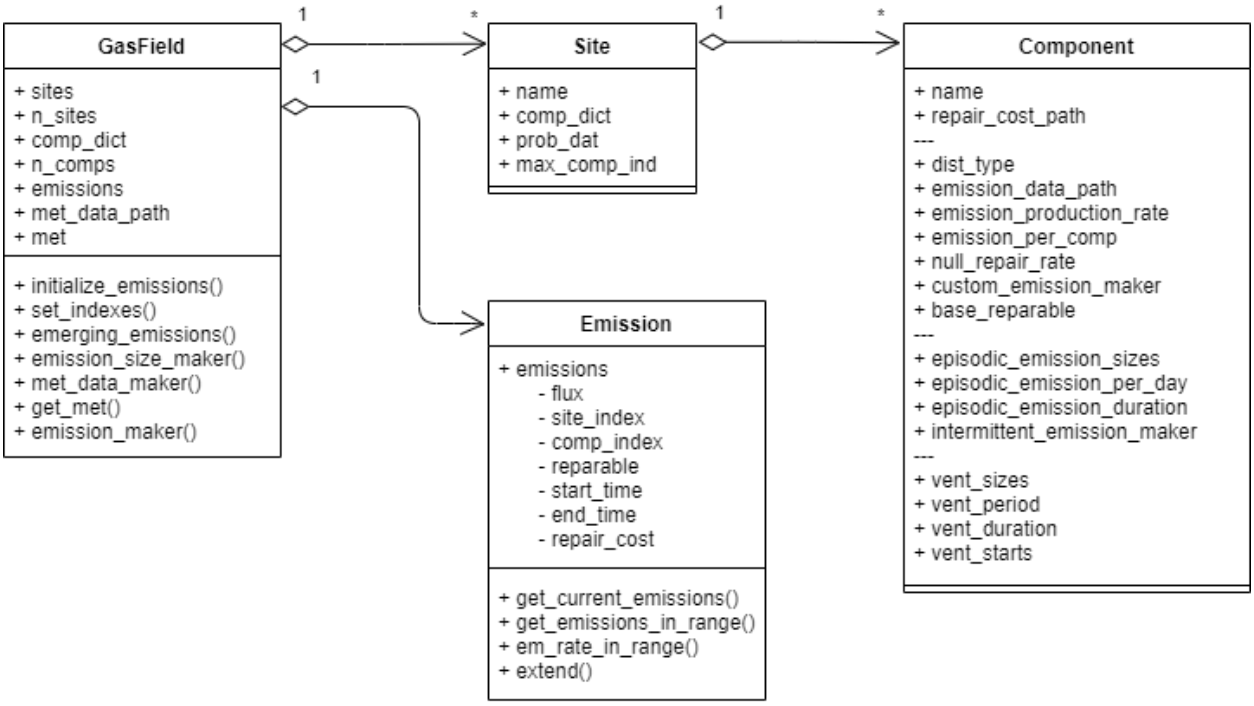

Figure 2: FEAST GasField class and associations

The LDARProgram Class

LDAR program specifications are stored an instance of the class `LDARProgram`. The `LDARProgram` class is also hierarchical (see **Error! Reference source not found.**): it contains

instances of the *DetectionMethod* class and the *Repair* class and specifies when detection methods should be deployed. Instances of the *DetectionMethod* store the PoD surface, speed, cost and any other parameters specific to the detection method. Instances of the *Repair* class specify the time delay between detection and repair. The *DetectionMethod* and *Repair* classes track specific results associated with the detection and repair processes using the *ResultAggregate* class. Each *LDARProgram* contains a copy of the *Emission* object from the *GasField* which is modified during the simulation to account for mitigated emissions. LDAR programs can be compared by their *emission_timeseries* attributes or their associated *Emission* objects after the simulation completes.

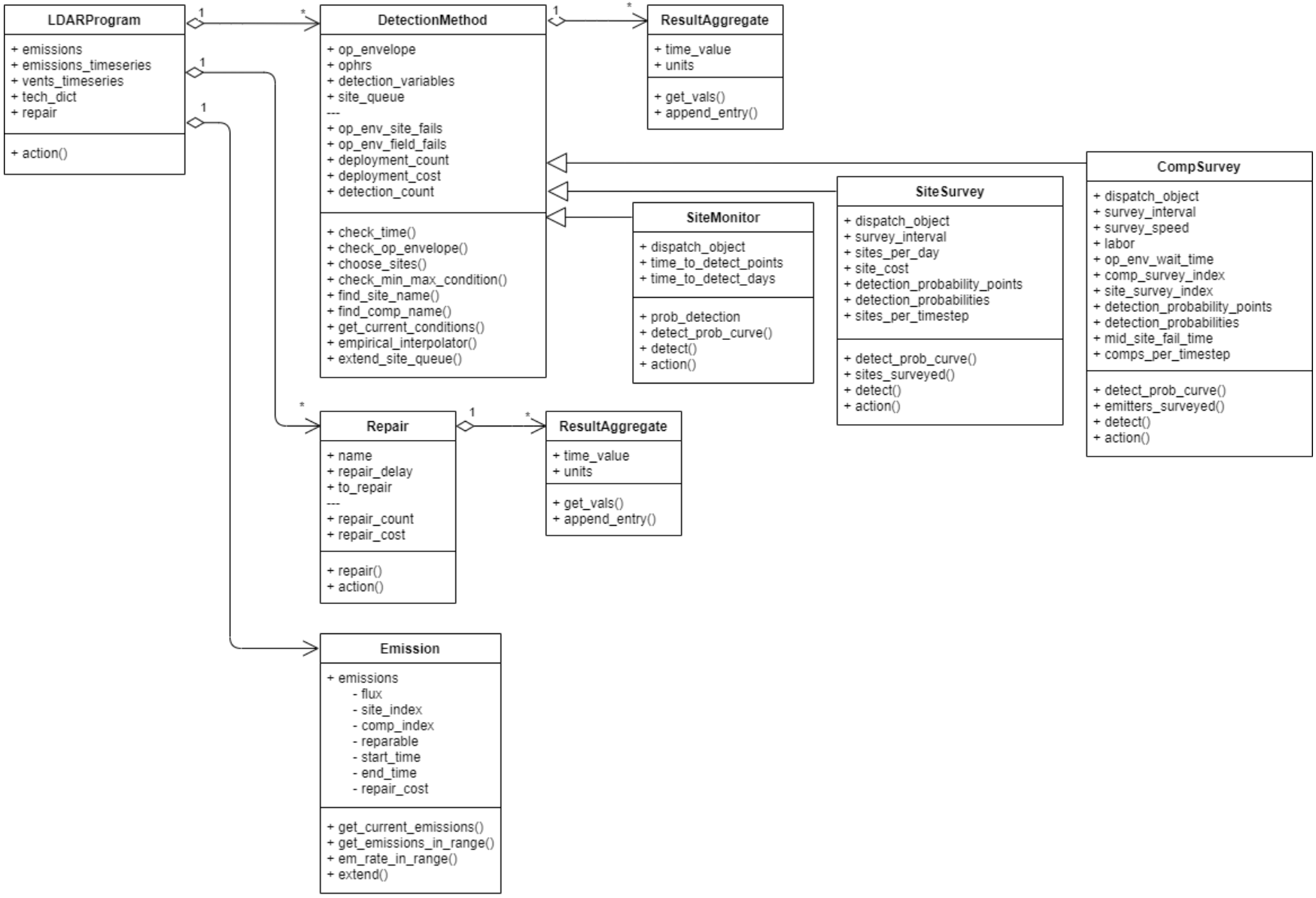

Figure 3: FEAST LDARProgram class and associations

The ResultAggregate Class

FEAST stores simulation results using the *ResultAggregate* class during runtime. The class contains an array with two columns: one column stores the time when an event occurs, and the second column stores a value. Two subclasses inherit from *ResultAggregate*.

ResultDiscrete stores quantities that occur at a specific time. The subclass provides methods to calculate the sum of quantities between two times. A second method calculates the cumulative sum versus time of quantities.

ResultContinuous stores quantities that occur continuously for a finite duration, such as an emission rate. The subclass provides a method to compute the integrated quantity during a time interval.

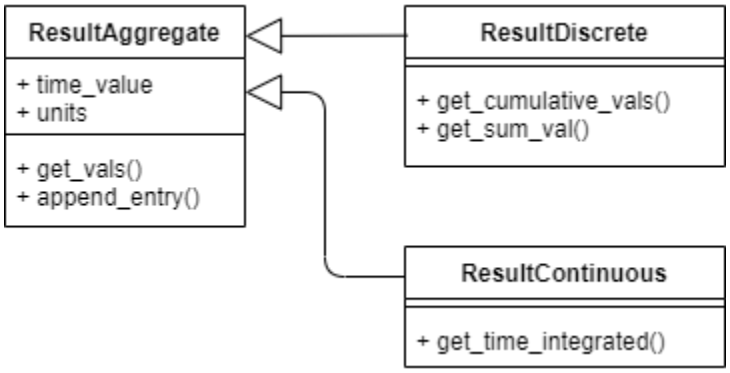

Figure 4: FEAST ResultAggregate class and associations

How to launch a simulation

Launching a simulation requires building the emission model and LDAR programs from their most basic pieces, as illustrated in Figure 5. First emission, repair cost and meteorological distributions must be loaded. Then the infrastructure *Components*, *Time* settings, and *Repair* methods can be defined. *Sites* are built from the previously defined *Components*, and *DetectionMethods* are connected to the *Repair* methods (or other *DetectionMethods*) that they dispatch. The *Sites* are combined into a *GasField* and the *DetectionMethods* are combined in one or more *LDARPrograms*. Finally, the *GasField* and *LDARPrograms* are passed to a *Scenario* to run the simulation. FEAST includes an example that demonstrates how each of these steps can be completed within python. The script is in the highest level directory of FEAST and is named ExampleRunScript.py.

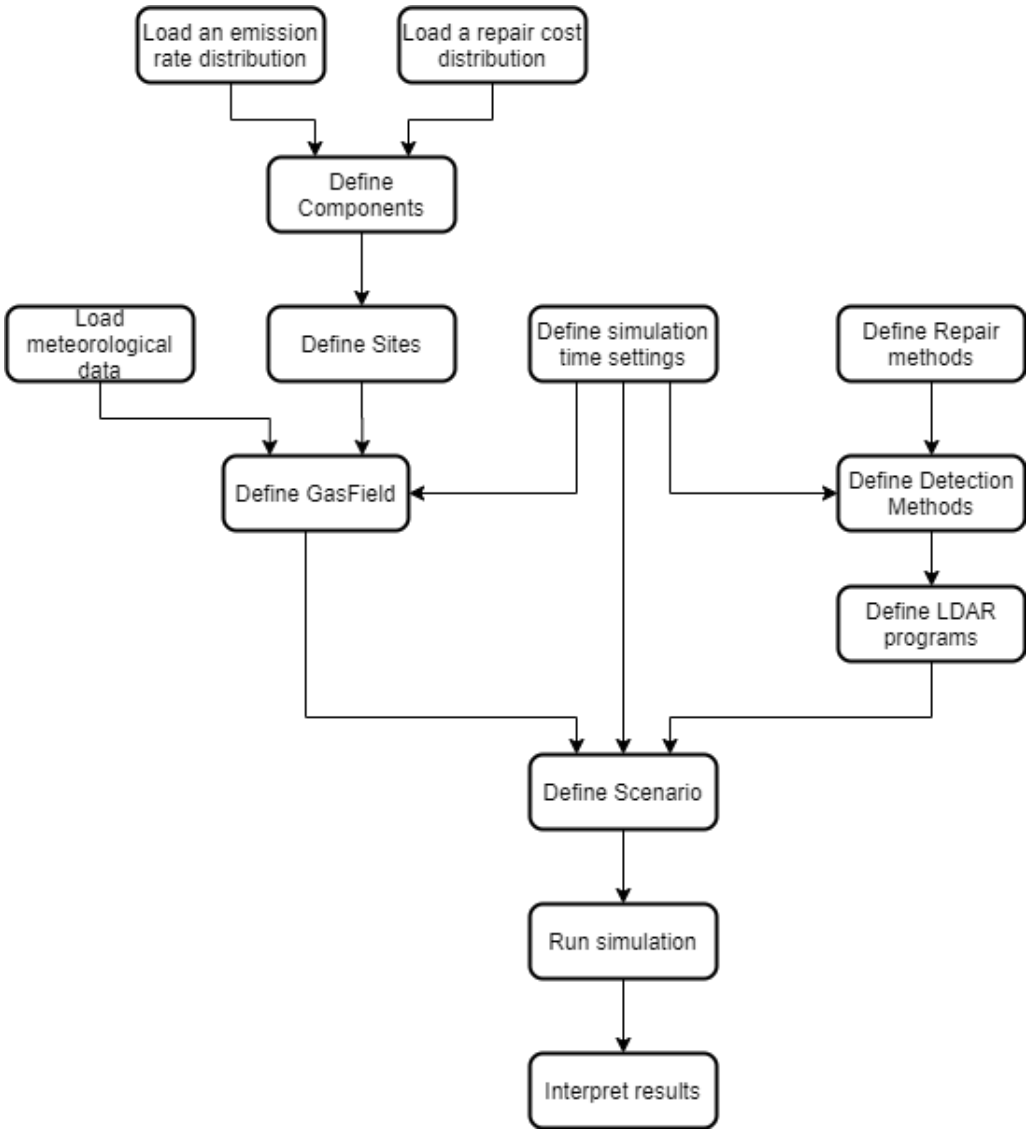

Figure 5: Simulation flow chart

How to read results

FEAST supports two save methods that can be specified at run time. The first method is 'pickle'. If the 'pickle' option is chosen, the full `Scenario` object will be saved to disk using the python package 'pickle.' When it is saved, the `Scenario` object will contain all of the settings for the simulation, as well as the characteristics of every emission that existed in the simulation under each `LDARProgram`. The 'pickle' package will again need to be used to load the results file after it is saved. The loaded python object will be an instance of the Scenario class, including the `LDARProgram` dict and `GasField` attributes.

The 'pickle' save method stores all of the details of the simulation and stores them in the familiar format of a `Scenario`. The resulting file is convenient for recalling details of a simulation, but results in a large file size and requires a consistent version of FEAST be available in order to open the file in the future.

The second save method is ‘json.’ In that case, select results from the simulation are saved in a JSON file, omitting the simulation settings. When this method is used, each *ResultAggregate* object in an *LDARProgram*, *DetectionMethod* or *Repair* object will be collected into a single dict which is then dumped to the ‘json’ file. The specific results stored may change as FEAST is revised or new *DetectionMethod* classes are added to the software.

User references

In addition to this user guide, FEAST provides extensive documentation in doc strings, an example run script, and the FEAST 3.1 documentation PDF file. The FEAST 3.1 documentation PDF file and example run script are available in the FEAST parent directory. The documentation PDF compiles all of the doc strings from the code into a human-readable format including a table of contents to illustrate more details of the model structure than described here. The example run script can be run locally on any machine to verify the FEAST has been installed correctly and demonstrate the steps to building a simulation.