# GridAPPS-D Documentation

***Release 1.0***

**The GridAPPS-D Team**

**Sep 10, 2020**

# Contents


GridAPPS-D

CHAPTER 1

# Overview

Through a series of industry centric meetings and workshops, the U.S. Department of Energy Office of Electricity Delivery and Energy Reliability (DOE-OE) gathered input from utilities throughout the United States on their experiences in implementing, or planning to implement, ADMS. The results of these meetings are documented in a February 2015 report titled Voices of Experience: Insights into Advanced Distribution Management Systems.

The report documents the potential benefits to utilities in implementing ADMS applications, and underscores the need for more affordability, a timely path for deploying ADMS, and the development and deployment of ADMS applications. The high cost and amount of time required for ADMS deployment and application development was highlighted.

In response to these needs, DOE-OE has established an ADMS program with this project specifically tasked with developing an open-source, standards based ADMS application development platform - GridAPPS-D.

## 1.1 Conceptual Design Summary

A conceptual design for GridAPPS-D was created at the beginning of the project. The conceptual design is summarized below. The full design document may be downloaded from this link - GridAPPS-D Conceptual Design

This document provides a high level, conceptual view of the platform and provides related background and contextual information. This document is intended to both educate readers about the technical work of the project and to serve as a point of reference for the project team. The document will be updated as the project progresses.

## 1.2 Architecture

A conceptual architecture for the system has five key functional elements as shown in Figure 1:

1. **Tools** help developers enhance the functionality of their applications. Examples might include off-line power flow, optimization tool boxes, state estimators, statistical processing, etc.
2. **I/O** allows convenient access to the power system model and data through standards-based queries and messages. Conversely, applications can send control signals to the simulator using standard message schemas.

3. **Development utilities** include loggers, debuggers, access control, test managers, user interface toolkits, and other application support functions.
4. **Data bus** is based on industry standards like IEC 61968 and 61970 (i.e. the Common Information Model), plus more to be identified.
5. **Distribution simulator** represents the power system operating in real time. Initially, this will be GridLAB-D, but future versions may include EPRI's OpenDSS, ns-3 for communications, and other federated co-simulators.

Figure 1 also shows the relationships between GridAPPS-D, the ADMS application developer and commercial tools. Two different classes of data flow are shown:

1. Control and configuration data are shown with dashed lines; this allows the application developer to manage the platform.
2. Data flowing as a part of an application are shown with solid lines.

For more detailed information about the architecture and design, see *UML from the Functional Specification*

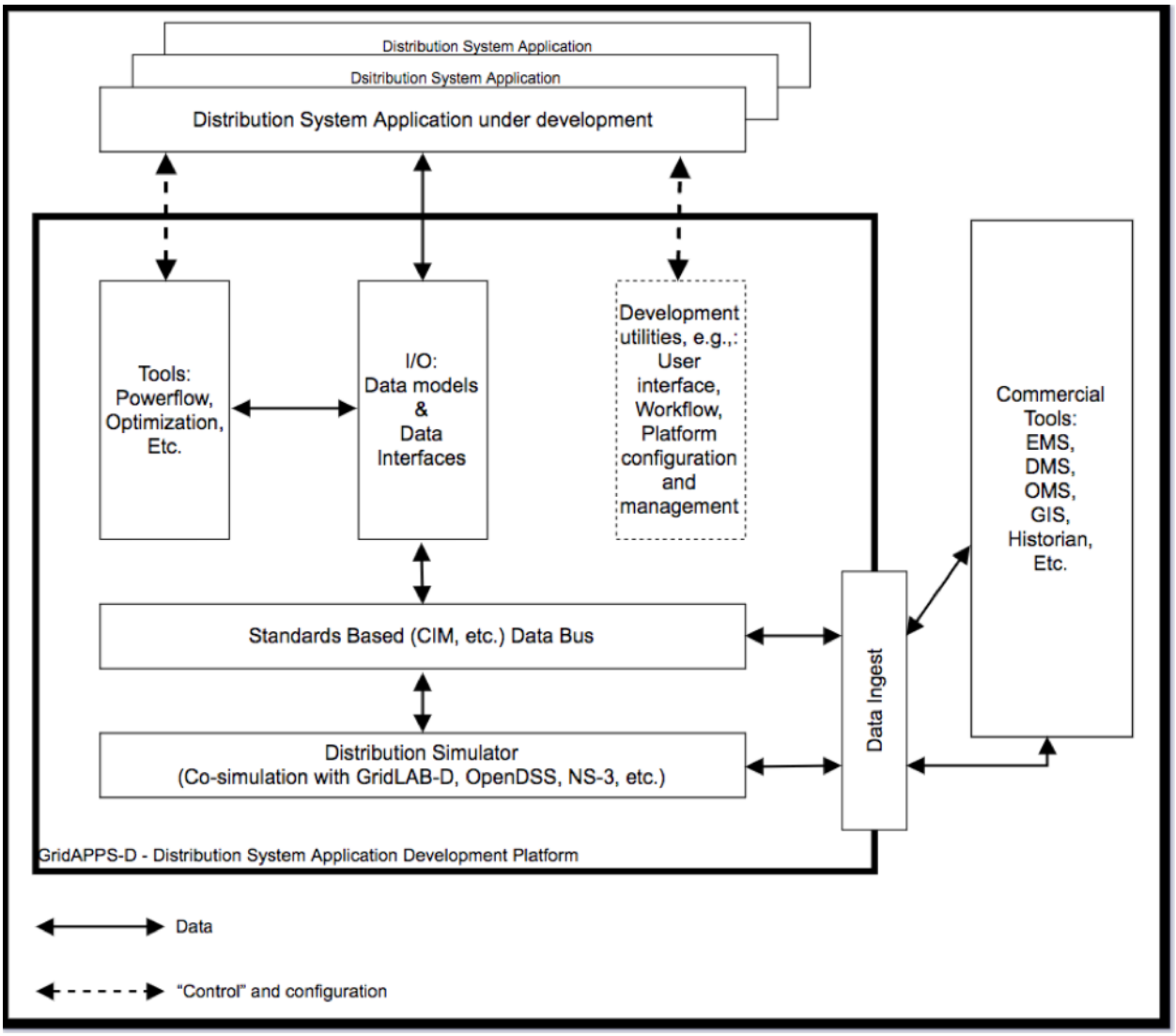



*Figure 1: GridAPPS-D provides a method for developers (top) to run their new applications on a real-time simulator with extensive modeling and tool support (heavy box). GridAPPS-D is built around standard data models like the CIM (center). It readily interfaces to existing software products (right), which may also 1) use components of GridAPPS-D*

*and 2) supplement or replace the built-in distribution simulator (bottom), facilitating the deployment of new ADMS applications to existing software products.*

## 1.3 Definition of Terms

Process Manager - Process Manager keeps track of all the processes running on the platform. These processes may include simulators, requests, applications and other managers. It is also the starting point for a request received by the platform.

Configuration Manager - It receives simulation configuration request from Process Manager and parses it to build the necessary configuration files.

Data Manager - The data manager accesses the database to build the model files used by the simulator.

Simulation Manager - The simulation manager launches the simulator and other required applications such as the FNCS bridge, FNCS, and the VoltVar application. It is in charge of managing the timing of the simulation and reporting output from the simulation out to the simulation status topic.

FNCS-GOSS Bridge - Serves as a bridge between FNCS and Simulation Manager.

FNCS - FNCS is a network co-simulator used to communicate between simulator and FNCS-GOSS bridge

Platform - Refers to GridAPPS-D platform.

RC1 - Release Cycle 1.

Simulation - A real world distribution system currently done by GridLAB-D

Simulator - In current release GridLAB-D serves as the simulator.

VoltVar Application -

Vizualization - A web-based visualization application is developed in RC1 to view power system model with real time values from simulation result.

GOSS - Grid Optics Software System is a middleware architecture designed as a prototype future data analytics and integration platform

GridLAB-D - GridLAB-D is a distribution level powerflow simulator. It acts as the real world distribution system in GridAPPS-D.

Power System Model - IEEE 8500 model is used in RC1.

Model - See Power System Model

CIM - Common Information Model is a standard for representing electrical network and exchange information.

## 1.4 References

## 1.5 Release History

### 1.5.1 Version: Release Cycle 1 (RC1)

Release Date: May 2017

Version description: This is the first version for internal release of GridAPPS-D platform. This is not ready for public use yet.

Functional requirements covered in this release:

- 102/202 Command Interface
- 301 Real-time Simulation Data
- 310 Hosted Application, but short-cutting the registration process (partial)
- 401 Distribution Co-Simulator (partial)
- 402 Process Manager (partial)
- 404 Data Manager (partial)
- 405 Simulation Manager (partial)
- 406 Power System Model Manager (partial)
- 413 Platform Manager (encapsulating 401 and 403-406)

### 1.5.2 Version: 2019.01.0

Release Date: January 2019

GridAPPS-D v2019.01.0 release contains following features/updates:

1. **Platform updates:**
    - Simulation can run as fast as possible as well as real-time (every 3 seconds)
    - Simulation can run with houses if present in the model.
    - **Following components can be controlled while the simulation is running:**
        - Open or close capacitors
        - Open or close switches
        - Change tap setting for regulators
        - Changing control modes for regulators
        - Change inverter P & Q output
        - Set control modes for regulators and capacitors
    - Simulation request creates the input weather file.
    - **gridappsd-python:**
        - (@Craig: list out updates here)
    - **cim2glm:**
        - Optional house cooling load components
    - Single-phase power electronics and fuse ratings
    - Inverter parameters changed from rotating machines to power electronics
    - Solar and storage
    - Measurements exported to the circuit metadata (JSON file); SimObject identifies the corresponding GridLAB-D object
    - Supplemental scripts to populate feeder with measurements and houses
    - Rotating machines, only parameters essential for the UAF lab microgrid
    - In GridLAB-D export of loads, each node or triplex_node will have separate submeters for houses, PV inverters, battery inverters and rotating machines, i.e., not patterned after net metering

2. **Data updates:**

   2.1 Power grid models:

   - Power grid models are stored in blazegraph database in its own docker container.
   - **Following models are pre-loaded**
     - EPRI_DPV_J1
     - IEEE123
     - IEEE13
     - R2_12_47_2
     - IEEE8500
     - IEEE123_pv
   - User can upload customized model (@Tara: attach readthedocs link)

   2.2 Weather:

   - Weather data in stored in InfluxDB using Proven.
   - InfluxDB has its own docker container with pre-loaded weather data.
   - API added to query weather data.
   - Feature added to create weather file for a simulation
   - Details of pre-loaded weather data in current release: (@Eric: meta-data details please)

   2.3 Simulation Input

   - Simulation input commands sent by applications/services are stored in InfluxDB using Proven.
   - API added to query input data.

   2.4 Simulation Output

   - Output from simulator is stored in InfluxDB using Proven.
   - API added to query output data.

   2.5 Logs

   - API added for query based on pre-defined filters or custom SQL string.
   - Changed logs to have epoch time format.

3. **Applications and Services:**

   3.1 Viz

   - User can select to run simulation at real-time or as fast as possible
   - User can select to add houses in the simulation
   - User can open or close switches and capacitors by clocking on them
   - Cleaner display of log messages while simulation is running
   - User can query simulation logs after simulation is done.
   - Toggle switches open/close
   - Querying logs through Viz (still working on this)
   - **Bug fixes**

  - fixed the stomp client in Viz,
  - added missing capacitor labels
  - redirect non-root urls to root (localhost:8080)

3.2 Sample application: (@Craig/Andy: please review/add)

- Source code at https://github.com/GRIDAPPSD/gridappsd-sample-app
- Sample app runs in its own container
- Register with gridapps-d platform when platform start.
- Re-register automatically if platform restart.
- Redundant log messages removed.
- Works with user selected model instead of hard-coded ones.

3.3 State Estimator (TODO: @Andrew)

3.4 RDRD(WSU) (TODO: @Anamika/Shiva)

3.5 DER Dispatch (@TODO: @Jeff)

3.6 VVO (@TODO: @Brandon)

5. **Source Code:**

- goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.01.0
- gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.01.0
- gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.01.0
- cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.01.0
- proven-cluster - https://github.com/pnnl/proven-cluster (@Eric: link for release branches)
- proven-docker - https://github.com/GRIDAPPSD/proven-docker
- proven-client - https://github.com/pnnl/proven-client
- proven-message - https://github.com/pnnl/proven-message
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.01.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/feature/1146
- sample-app https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.01.0

6. **Docker Container:**

GridAPPS-D creates and starts following docker containers:

- **gridappsd/gridappsd:2019.01.0 - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.01.0**
  - proven-client - https://github.com/pnnl/proven-client
  - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.01.0
  - gridappsd/gridappsd-base:master - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.01.0
  - zeromq - http://download.zeromq.org/zeromq-4.0.2.tar.gz
  - zeromq_czmq - https://archive.org/download/zeromq_czmq_3.0.2/czmq-3.0.2.tar.gz

  - activemq - http://mirror.olnevhost.net/pub/apache/activemq/activemq-cpp/3.9.4/activemq-cpp-library-3.9.4-src.tar.gz
  - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
  - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/feature/1146
- **gridappsd/influxdb:2019.01.0 - https://github.com/GRIDAPPSD/gridappsd-data/tree/releases/2019.01.0**
  - influxdb:latest - https://hub.docker.com/_/influxdb
- **gridappsd/blazegraph - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.01.0**
  - lyrasis/lbazegraph:2.1.4 - https://hub.docker.com/r/lyrasis/blazegraph
- **gridappsd/proven - https://github.com/GRIDAPPSD/proven-docker**
  - proven-cluster - https://github.com/pnnl/proven-cluster/tree/v1.3.3
  - proven-message - https://github.com/pnnl/proven-message/tree/v1.3.1
- **gridappsd/sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.01.0**
  - gridappsd/app-container-base - (TODO: @Craig can you provide the repository?)
- gridappsd/viz:2019.01.0 - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.01.0
- redis:3.2.11-alpine - https://hub.docker.com/_/redis
- mysql/mysql-server:5.7 - https://hub.docker.com/_/mysql

### 1.5.3 Version: 2019.02.0

Release Date: Feb 2019

1. Fixed Bugs:

- PROVEN - It can now store simulation input and output which can scale for IEEE8500 model.
- PROVEN - It can store data with real-time simulation run.
- PROVEN - Increased max data limit to unlimited.
- FNCS Goss Bridge - Corrected the timestamp format in simulation logs.

2. New Features:

- Viz - User can query log data from MySQL using Viz menu.
- Viz - Added menu to operate switches.
- FNCS GOSS bridge can do execute pause, resume and stop operations for simulation.
- Update PROVEN docker container for automated builds.

3. Source Code:

- goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.02.0
- gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.02.0
- gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.02.0
- cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.02.0
- proven-cluster - 1.3.4 https://github.com/pnnl/proven-cluster/releases/tag/v1.3.4
- proven-client - 1.3.4 https://github.com/pnnl/proven-client/releases/tag/v1.3.4

- proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.1
- proven-docker - https://github.com/GRIDAPPSD/proven-docker
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.02.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/feature/1146
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.02.0

4. **Docker Container:**

GridAPPS-D creates and starts following docker containers:

- **gridappsd/gridappsd:2019.01.0 - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.01.0**
  - proven-client - https://github.com/pnnl/proven-client
  - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.01.0
  - gridappsd/gridappsd-base:master - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.01.0
  - zeromq - http://download.zeromq.org/zeromq-4.0.2.tar.gz
  - zeromq_czmq - https://archive.org/download/zeromq_czmq_3.0.2/czmq-3.0.2.tar.gz
  - activemq - http://mirror.olnevhost.net/pub/apache/activemq/activemq-cpp/3.9.4/activemq-cpp-library-3.9.4-src.tar.gz
  - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
  - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/feature/1146
- **gridappsd/influxdb:2019.01.0 - https://github.com/GRIDAPPSD/gridappsd-data/tree/releases/2019.01.0**
  - influxdb:latest - https://hub.docker.com/_/influxdb
- **gridappsd/blazegraph - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.01.0**
  - lyrasis/lbazegraph:2.1.4 - https://hub.docker.com/r/lyrasis/blazegraph
- **gridappsd/proven - https://github.com/GRIDAPPSD/proven-docker**
  - proven-cluster - https://github.com/pnnl/proven-cluster/tree/v1.3.3
  - proven-message - https://github.com/pnnl/proven-message/tree/v1.3.1
- **gridappsd/sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.01.0**
  - gridappsd/app-container-base - (TODO: @Craig can you provide the repository?)
- gridappsd/viz:2019.01.0 - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.01.0
- redis:3.2.11-alpine - https://hub.docker.com/_/redis
- mysql/mysql-server:5.7 - https://hub.docker.com/_/mysql

### 1.5.4 Version 2019.03.0

1. Bugs Fixed
   - Sending a command to change set point to the PV inverter has no effect.

- Time series query return no data after simulation run.
- Viz: Switch operations not working on Firefox browser. Time on x-axis on plots is not displayed correctly.

2. New Features
   - GridAPPS-D – VOLTTRON initial interface created. https://github.com/VOLTTRON/volttron/tree/rabbitmq-volttron/examples/GridAPPS-DAgent
   - Fault injection: Simulator can receive faults. Fault schema created in Test Manager. Workflow for fault processing documented on readthedocs.
   - Viz: Created menu for capacitors, regulators.
   - Proven: Facilitates direct disclosure of JSON messages to Proven via Hazelcast or REST; eliminating need for the proven-message library. Improved throughput and scalability for Proven's data disclosure component. Disclosed data is now distributed or staged across the cluster to be used by future JET processing pipelines.
3. Documentation
   - CIM100 documented
   - Steps added for creating and testing an application
   - Updated documentation on Simulation API
4. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.03.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.03.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.03.0
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.03.0
   - proven-cluster - 1.3.4 https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.3
   - proven-client - 1.3.4 https://github.com/pnnl/proven-client/releases/tag/v1.3.4
   - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.3
   - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.03.0
   - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
   - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.03.0
   - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/feature/1146
   - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.03.0

### 1.5.5 Version 2019.06.0

1. Bugs Fixed
   - Updated configuration, power grid model and simulation API for CIM100 and app evaluation features addition.
   - All logs are being published to topic instead of queue.
   - Fixed TypError bug in gridappsd-sensor-service.
2. New Features

- Communication outages: Platform supports input and/or output outage request with simulation for all or some selected power grid components. Outages are initiated and removed at the requested start and end time.
- Fault injection: Platform can receive faults with simulation request and forwards them to co-simulator.
- Viz UI updated: Input form added for communication outage and fault parameter selection. Input form moved from single page to separate tabs.
- CIM version update: Updated CIM version to CIM100. Added support for Recloser and Breaker in model parsing.
- New methods in Python wrapper: Capability added in gridappsd-python to start, stop and run a simulation directly from python using yaml or json.
- Sample app container move to Python 3.6 as default. Updated gridappsd-sample-app to use updated container.
- Debug scripts added: Added scripts in gridappsd-docker to run platform, co-simulator and simulator in separate terminals for debugging purposes.
- Sensor service in available in gridappsd container by default. Sensor service is no longer required to be added in gridappsd container via docker-compose file.
- Default log level is changed from DEBUG to ERROR for limiting the amount of log messages on terminal.
- **Breaking API change** - Simulation input and output topics changed in gridappsd-python from FNCS_INPUT_TOPIC to SIMULATION_INPUT_TOPIC and FNCS_OUTPUT_TOPIC to SIMULATION_OUTPUT_TOPIC.
- **Breaking API change** - Simulation request return a json with simulation id and list of events with their uuids instead of just simulation id.

3. Documentation
    - Using GridAPPS-D documentation section updated for new UI input form with communication outages and faults selection.
4. Source Code
    - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.06.0
    - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.06.0
    - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.06.0
    - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.06.0
    - proven-cluster - 1.3.4 https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.3
    - proven-client - 1.3.4 https://github.com/pnnl/proven-client/releases/tag/v1.3.4
    - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.3
    - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.06.0
    - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
    - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.06.0
    - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/feature/1146
    - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.06.0

### 1.5.6 Version 2019.07.0

1. Bugs Fixed
    - Time series query filter are updated in the API as well documentation.
    - Selecting houses is now working with the simulation.
    - Following bugs resolved for Viz
        - Line name is not based on previously selected values.
        - Removing a selected app-name closes input form
        - Change Event Id to Event tag
        - Change attribute to a multi-value select box
        - Help-text 'Add input item' does not go away on CommOutage tab
        - Object mrid is not correct for multiple phases selection.
    - Pos added for load break switches
2. New Features
    - Platform now stores input and output from services and applications output/input in time series data store.
    - Simulation can run with new 9500 node model
    - Support for synchronous machines added in CIM model in blazegraph.
    - End-to-end fault injecting and processing pipeline is now working.
    - Powergrid api added to query object id, object dictionary and object measurements.
    - New keys added in glm file to support faults.
    - Viz can display plot for new 9500 model.
    - Added log api in gridappsd-python
    - Measurement for switch positions for all models
    - Explicit setting for manual mode in reg and capacitora in the RegulatingControl.mode attribute.
    - GridAPPS base constainer has folowwing changes
        - Switch to openjdk
        - New version of fncs
        - CZMQ_VERSION changeed to 4.2.0
        - ZMQ_VERSION changes to 4.3.1
        - GridLAB-D switched from feature/1146 to develop
3. Source Code
    - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.07.0
    - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.07.0
    - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.07.0
    - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.07.0
    - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.4
    - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.5

- proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
- proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.06.0
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.07.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.07.0

### 1.5.7 Version 2019.08.0

1. New Features
   - Viz added capability to select power/voltage/tap measurments for custom plotting
   - Control attributes are back for Capacitors
   - Added Voltage Violation service that publishes list of measurement ids with per unit voltages that are out of range every 15 minutes
   - Viz added display for Voltage Violation service output
   - Viz can display Lot/Long coordinated for 9500 node model.
   - Breaking Change: JSON format for timeseries query response is flattend out
   - Resolved 500 Internal server error for storing simulation input.
   - Houses are created and uploaded to Blazegraph for 123 node model
   - Additonal column process_type added for logs to distinguish process id for simulation
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.08.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.08.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.08.0
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.08.0
   - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.5
   - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.5
   - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
   - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.08.0
   - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
   - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.08.0
   - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
   - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.08.0

### 1.5.8 Version 2019.08.1

1. New Features
    - Viz: Change simulation pause button to start button when simulation completes.
    - Bug fix: Simulation id dropdown is not showing selected id in Browse-data-logs.
    - Bug fix: Timeseries queries returning same object multiple times.
    - Bug fix: Weather file containes only 10 minute data even if simulation duration is longer.
2. Source Code
    - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.08.1
    - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.08.1
    - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.08.1
    - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.08.0
    - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.5
    - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.5
    - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
    - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.08.0
    - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
    - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.08.1
    - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
    - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.08.1

### 1.5.9 Version 2019.09.0

1. New Features
    - Fault Processing: Faults are working on radial feeders.
    - Note: Faults are not working on meshed systems. If you have a meshed system then send switch open message to simulate the fault.
2. Source Code
    - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.09.0
    - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.09.0
    - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.09.0
    - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.09.0
    - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.5
    - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.5
    - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
    - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.09.0
    - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop

- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.09.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.09.0

### 1.5.10 Version 2019.09.1

1. New Features
   - BREAKING CHANGE: Measurements in simulation output message changed from array to dictionary.
   - Simulation are now working for 9500 model with houses.
   - Added missing measurement in blazegraph for houses.
   - Voltage violation service and Viz app updated to work with new simulation output format.
   - Faults are working with 9500 model.
   - Viz app: User can select services and their input parameters in simulation request form.
   - Viz app: Y-axis label corrected if plot value is same during the simulation run.
   - Simulation request API updated to take user input parameters for services.
   - Timezone corrected for pre-loaded weather data.
   - Operational limit set on the power grid models in blazegraph.
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.09.1
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.09.1
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.09.1
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.09.1
   - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7
   - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
   - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
   - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.09.1
   - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
   - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.09.1
   - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
   - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.09.1

### 1.5.11 Version 2019.10.0

1. New Features
   - Alarms service created. It publishes alarm whenver a switch or capacitor is opened or closed. It is added as a pre-requisite for sample app.
   - Load profile data pre-loaded in timeseries data store InfluxDB.

- Load profile file ieeeziploadplayer is created dynamically based on simulation start time and duration.
- API updated in platform and Proven to query load profile data.
- Timeseries API updated to accept timestamps in seconds instead of micro or nanosecond.
- Timeseries API updated to accept query filters in an array instead of single value.
- Viz app: User can search and highlight objects on network by name and mrid.
- Viz app: User can re-center network graph.
- Viz app: Displays alarms in a saperate tab when simulation is running. Notifies when a new alarm is received in alarm tab.
- Viz app: User can upload scheduled commands json file with communication outage and faults.
- Viz app: Switches are displayed as closed/opened based on simulation output value.
- Viz app: Display image for switches are changes to green/red squares and moved between nodes.
- Bug fixes in DSS configuration.
- GridLAB-D updated to latest develop version.
- OpenDSSCmd updated to 1.2.3.
- Powergrid models - Updated Generator.dss to include kVA for generators.
- Added kva base to glm file, so setting kw=0 does not make the kva base also 0.
- Internal house loads added. Schedule file is created for simulation when useHouses=true.
- Sensor service bugs fixed.
- API added to export Vnom opendss file.

2. Source Code

- goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.10.0
- gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.10.0
- gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.10.0
- cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.10.0
- proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7
- proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
- proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
- proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.10.0
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.10.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.10.0

### 1.5.12 Version 2019.12.0

1. New Features
    - Updated and documented MRID UUID generator to ensure compliance with UUID 4
    - Integrate DNP3 service with GridAPPS-D container
    - Created API to get user role based on login
    - Added a user for testmanager to distinguish between simulation commands and alarms
    - Removed hardcoded corrdinate identifcation from Viz
    - Added capability to change model state before starting a simulation.
    - Added feature on UI to upload a file with faults and comunication output
    - Created user login page on UI
    - Added light/dark toggle themeon UI
    - Wrote a SWING_PQ node for each potential island in power grid model.
    - Fixed issues for app eveluations as reported by app developers or evluation team
    - Updated ci/cd scripts for repositories to support travis.ci updates
2. Source Code
    - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2019.12.0
    - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2019.12.0
    - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2019.12.0
    - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2019.12.0
    - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7
    - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
    - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
    - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2019.12.0
    - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
    - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2019.12.0
    - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
    - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2019.12.0

### 1.5.13 Version 2020.01.0

1. New Features
    - Alarms are varified before publishing.
    - Fixed floating switches issue on Viz app.
    - Release process documeted at gridappsd-docker-build repository readme
    - Created an automated, repeatable way to upload data in blazegraph
    - Documented model state update for starting a simulation

2. Source Code
    - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.01.0
    - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.01.0
    - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.01.0
    - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.01.0
    - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7
    - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
    - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
    - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2020.01.0
    - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
    - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.01.0
    - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
    - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.01.0

### 1.5.14 Version 2020.02.0

1. New Features
    - Alarms status is published as Open/Close instead of 0/1.
    - Added resume/pause-at API for simulation.
    - Added the EnergyConsumer.p attribute as a writable property in the FNCS GOSS Bridge
    - Fixed floating switches issue on Viz app.
    - Added units on the plots.
    - Viz allow user to go to nodes by clicking on plots.
    - Labels added for overlapping line on Viz plots.
    - Operator login issue resolved.
    - First integration test added in gridappsd-testing repo.
2. Source Code
    - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.02.0
    - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.02.0
    - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.02.0
    - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.02.0
    - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7
    - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
    - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
    - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2020.02.0
    - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop

- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.02.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.02.0

### 1.5.15 Version 2020.03.0

1. New Features
   - Viz app can display lines and nodes with power outage.
   - Changes are made in Viz app to start and show data from State Estimator service.
   - Viz app can render battery nad solar panel shapes.
   - Fixes are made to support no player file in simulation config.
   - Timestamp display added for voltage violation on Viz.
   - Viz can start and subscribe to State-Estimator service.
   - Integration tests created for simulation api.
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.03.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.03.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.03.0
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.03.0
   - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7
   - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
   - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
   - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2020.03.0
   - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
   - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.03.0
   - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
   - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.03.0

### 1.5.16 Version 2020.04.0

1. New Features
   - Updated Cim2GLM library version to 18.0.3
   - Added Configuration handler for generating limits.json file
   - Increased web socket message size
   - Corrected issue where phase count is incorrect for phase s1, s2 loads
   - Corrected json parse method for TimeSeriesRequest class.

- Viz app: Updated to use simulation timestamp for voltage violation instead of current time.
- Viz app: Show "Simulation starting" message before simulation is started and hide the Pause/Stop buttons.
- Powergrid model: Added scripts and *uuid.dat files to maintain persistent mRID values
- Powergrid model: Supporting OverheadLineUnbalanced, ganged regulators and unknown spacings for 1-phase and 2-phase line.
- Integration testing infrastructure create with PyTest and Travis.

2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.04.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.04.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.04.0
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.04.0
   - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7
   - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
   - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
   - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2020.04.0
   - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
   - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.04.0
   - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
   - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.04.0

### 1.5.17 Version 2020.05.0

1. New Features
   - Updated YBus export to include model_id as parameter
   - Made changed to work with multiple load profiles measurements in InfluxDB.
   - Corrected issue of no player file if schedule name is not passed in request.
   - Fix stomp client initialization problem for Viz app on firefox where it was getting stuck in connecting state for a long time.
   - Testing summary added to integration testing.
   - Integration tests added for power grid and simulation API.
   - AWS summary web page added for integration testing report.
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.05.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.05.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.05.0
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.05.0
   - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7

- proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
- proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
- proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2020.05.0
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.05.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.05.0

### 1.5.18 Version 2020.07.0

1. New Features
   - Updated opendss to version 1.2.11
   - **Added PAUSEd to ProcessStatus list to resolve testing issue.**
     - Updated TestManager to include comparing expected results between output of 2 simulations.
     - Updated TestManager to include comparing currently running simulation to result of previously ran simulation.
   - Added a new setting to Viz UI that allows toggling logging.
   - Fixed the problem in Viz where unselecting selected services didn't remove them from the simulation configuration object
   - Powergrid model: Bumped mysql-connector-java from 5.1.40 to 8.0.16 in /CIM/cim-parser
   - More integration tests added for power grid and simuation API.
   - Integration tests added for alarms and timeseries API.
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.07.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.07.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.07.0
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.07.0
   - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.5.7
   - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.6
   - proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.5.4
   - proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2020.07.0
   - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
   - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.07.0
   - gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
   - sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.07.0

## 1.6 Contact Us

GridAPPS-D team can be reached at gridappsd@pnnl.gov

CHAPTER 2

# Installing GridAPPS-D

GridAPPS-D is available using docker containers

## 2.1 Requirements

- git
- docker version 17.12 or higher
- docker-compose version 1.16.1 or higher

## 2.2 Docker and prerequisite install on OS X

- **git**
  - OS X requires xcode

```
xcode-select --install
```

## 2.3 Clone or download the repository

```
git clone https://github.com/GRIDAPPSD/gridappsd-docker
cd gridappsd-docker
```

## 2.4 Install Docker on Ubuntu

- run the docker-ce installation script

```
./docker_install_ubuntu.sh
```

- log out of your Ubuntu session and log back in to make the docker groups change active

## 2.5 Start the docker container services

```
./run.sh
```

**The run.sh does the following**

- download the mysql dump file
- download the blazegraph data
- start the docker containers
- ingest the blazegraph data
- connect to the gridappsd container

## 2.6 Start gridappsd

Now we are inside the executing container

```
root@737c30c82df7:/gridappsd# ./run-docker.sh
```

Open your browser to http://localhost:8080/

## 2.7 Exiting the container and stopping the containers

```
Use Ctrl+C to stop gridappsd from running
exit
./stop.sh
```

## 2.8 Restarting the containers

```
./run.sh
```

Reconnecting to the running gridappsd container

```
user@foo>docker exec -it gridappsddocker_gridappsd_1 bash
```

# CHAPTER 3

# Using GridAPPS-D

## 3.1 Start GridAPPS-D platform

Connect to the running GridAPPS-D container

```
user@foo>docker exec -it gridappsddocker_gridappsd_1 bash
```

Now we are inside the executing container. Start the platform.

```
root@737c30c82df7:/gridappsd# ./run-docker.sh
```

Open your browser to http://localhost:8080/ and click the menu button.

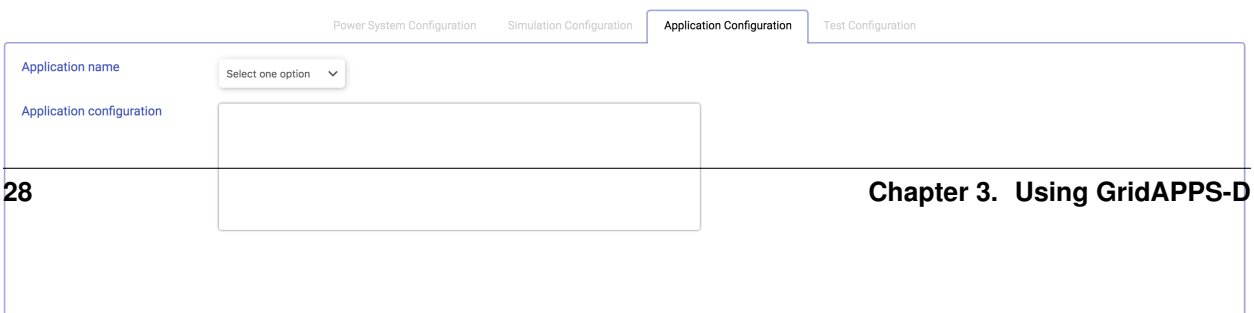

## 3.2 Start a Simulation

Choose Simulations from the menu.

To run a demo simulation keep the selected and entered values as it is. Otherwise select/enter Powergrid, Simulation and Application configuration values. Click the submit button to save the configuration.



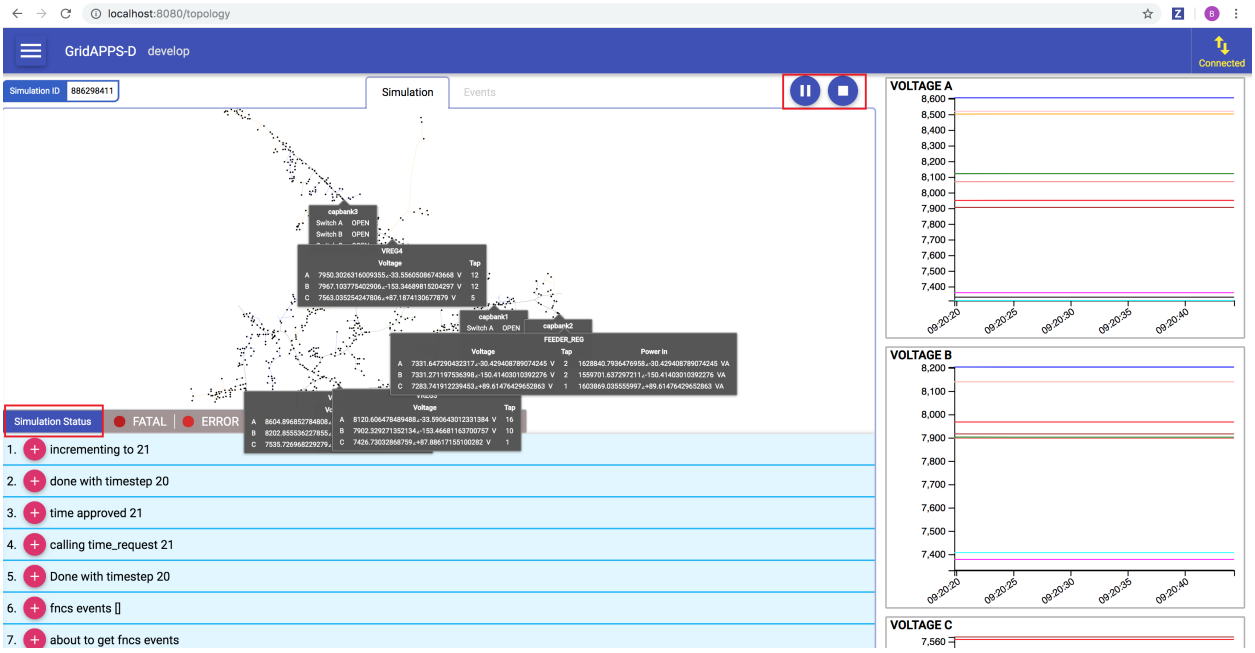
Click the tri-angle



to start the simulation.

The demo simulation runs 2 minutes of load variations with the sample-app controlling capacitor banks on the IEEE 8500-node test system [CIT2]. Most of Figure 1 is devoted to a map layout view of the test circuit, with updated labels for capacitor banks and voltage regulators. On the right-hand side, strip chart plots of the phase ABC voltages at capacitors and regulators, phase ABC substation power levels, and phase ABC regulator taps are continually updated. Capacitor bank labels on the circuit map view change between OPEN and CLOSED to show the bank status as load varies and the VVO application issues control commands. While GridAPPS-D runs the demo, GridLAB-D [CIT8] simulates power system operation and exchanges information with the sample-app using GOSS [CIT6] and FNCS [CIT7].

Following image shows the demo simulation output of the sample-app running on the IEEE 8500-node test system. Simulation Status at the bottom of the screen will display the simulation log messages. The simulation can be paused or stopped using the play and stop button.




## 3.3 Stop GridAPPS-D platform

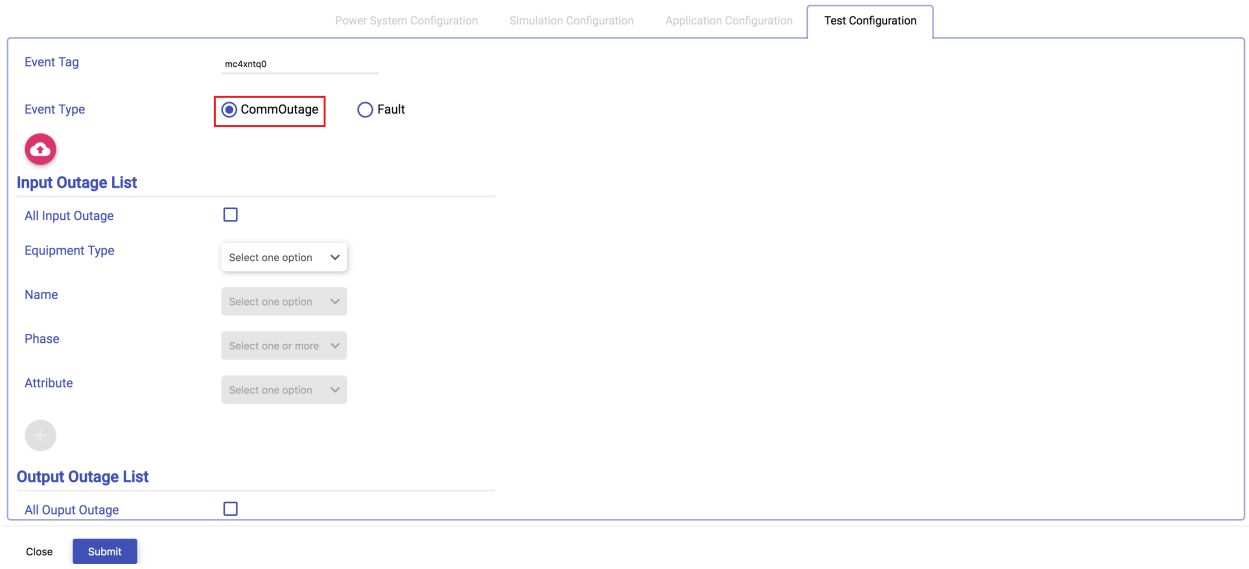
For an orderly shutdown of the platform:

```
Use Ctrl+C to stop gridappsd from running
```

## 3.4 Adding Events

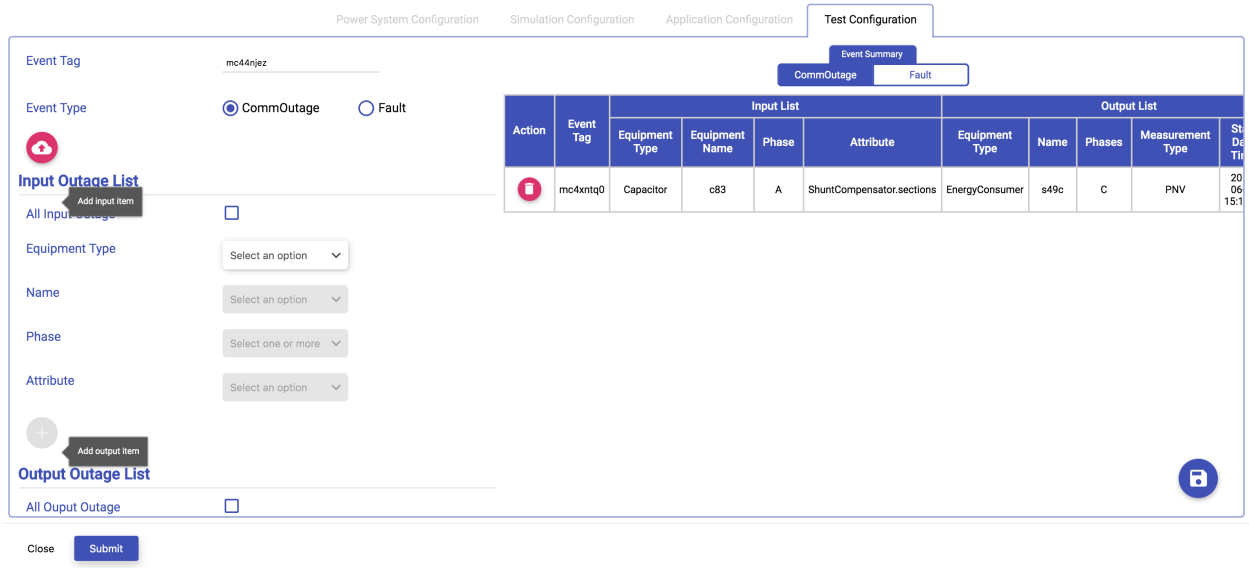
Communication outage and fault events can be added using the Test Configuration page

Select the CommOutage radio button for adding Communication Outage Events




Select the Fault radio button for adding Fault Events

Added events can be viewed in a tabular format on the right side of the page



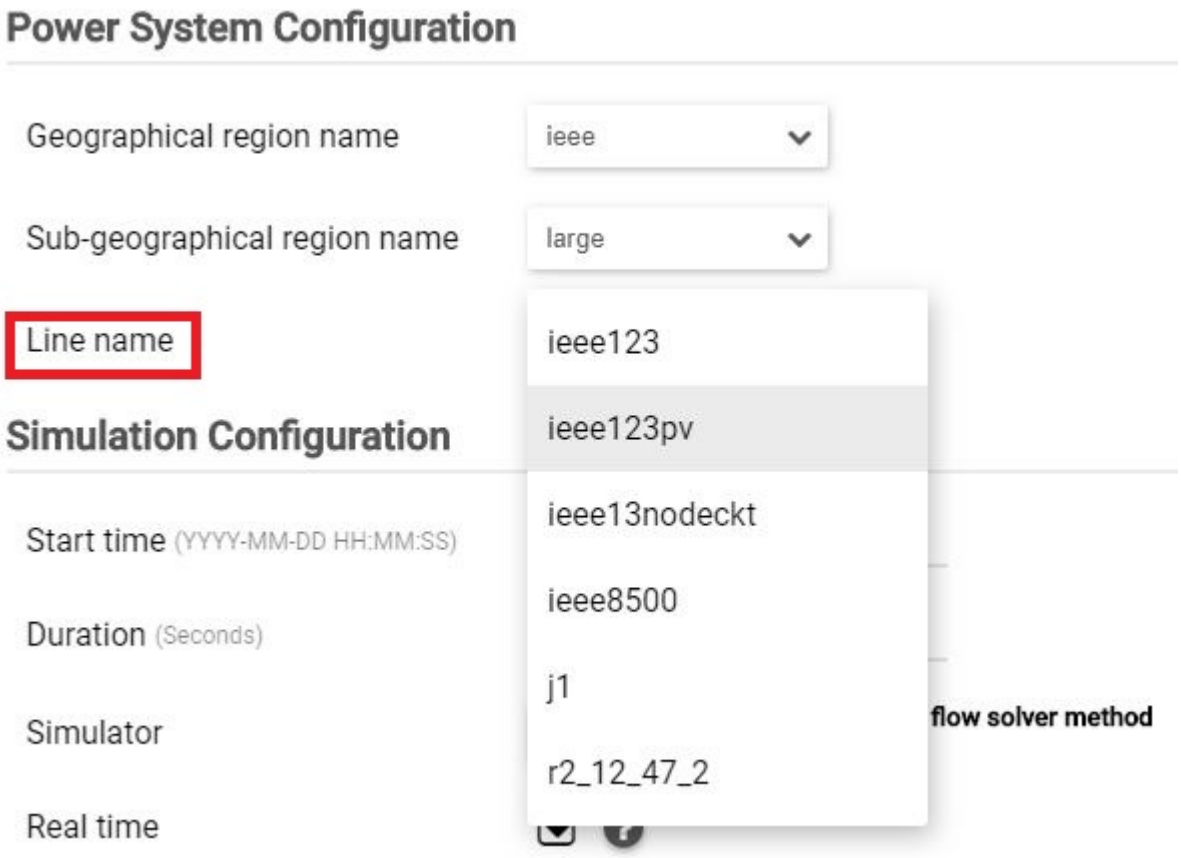

The added events for a simulation can be seen in the events view

## 3.5 Uploading Model into Blazegraph

With the platform running, use curl to load the file into Blazegraph.

```
curl -s -D- -H 'Content-Type: application/
↪xml' --upload-file 'model.xml' -X POST 'http://localhost:8889/bigdata/sparql'
```

Once the model is uploaded to Blazegraph, the new model will be shown in the Simulation Configuration Form in the visualization under the line name dropdown. If the Viz app was already open, you will need to restart the browser to see the new model(This is due to caching).




## 3.6 Inserting Measurements into Blazegraph

Clone the Powergrid-Models repository

```
git clone https://github.com/GRIDAPPSD/Powergrid-Models.git
```

Install the required python module

```
pip install SPARQLWrapper
```

Modify the Powergrid-Models/Meas/constants.py file. Change the blazegraph_url to "http://localhost:8889/bigdata/sparql".

Create a temporary directory for the measurements files

```
mkdir tmp
cd tmp
```

List the feeder and feeder id

```
python3 ../Powergrid-Models/Meas/ListFeeders.py
```

Generate the measurements file using the feeder and feeder id from the previous step

```
python3 ../Powergrid-Models/Meas/ListMeasureables.py ieee123pv _E407CBB6-8C8D-9BC9-
↪589C-AB83FBF0826D
```

Load the measurements into Blazegraph

```
for f in `ls -1 *txt`; do
  python3 ../Powergrid-Models/Meas/InsertMeasurements.py $f
done
```

## 3.7 Using Platform API

Applications and services can use either publish/subscribe mechanism or Python API to interact with GridAPPS-D platform.

Publish/Subscribe mechanism can be implemented using any of the language bindings for ActiveMQ messaging framework.

Python API wraps the publish/subscribe messaging and makes the interaction easier for Python apps/services. For more information on Python API and how to use it, look at https://github.com/GRIDAPPSD/gridappsd-python and https://github.com/GRIDAPPSD/gridappsd-sample-app.

Following sections describe the messaging APIs and the corresponding Python API function to interact with platform. Where no Python API function is mentioned, following generic functions can be used.

```
send(self, topic, message)
get_response(self, topic, message, timeout=5)
subscribe(self, topic, callback, id=None)
```

## 3.8 Authenticating

**Authentication**

Authentication in GridAPPS-D is handled using JSON Web Tokens (JWT). This allows the user to obtain a token, and continue to use it for the duration of their session.

*Retrieving and using a token*

1. **Send a token request to "/topic/pnnl.goss.token.topic" with the content a base64 encrypted version of the <username>:<pa**
   The credentials that should be used for the token request will be the username and password. These are only allowed to request a token.
2. You will receive a token back on /queue/temp.token_resp.<username> The response will be only the token, no json wrapping.
3. Use the token in place of the username for all future requests, the password can be empty. Note: if the server restarts or the token expires you will need to retrieve a new token.

*Anatomy of a token*

We use JSON web tokens, for more details view - https://jwt.io/introduction/ A JSON Web Token consists of three parts: Header, Payload and Signature. Each of these parts is concatenated with a period in the form of header.claims.signature. The header and payload are Base64 encoded and the signature is generated by algorithmically signing the encrypted header and payload. The header consists of metadata including the type of token and the hashing algorithm used to sign the token. The payload contains the claims data that the token is encoding.

The payload will look something like this

```
{"sub":"operator1","nbf":1597702372898,"iss":"GridOPTICS Software System","exp
↪":1598134372898,"iat":1597702372898,"jti":"6b615b4c-e51d-4241-9d3f-96eb29dbc8bb",
↪"roles":["testmanager","application","service","admin","operator","evaluator"]}
```

**The elements included in the token are:**

- sub contains the username of the authenticated user
- nbf, not before, or when the token is first valid. This will generally be the same as when it was issued
- iss, who issued the token
- exp, when the token expires
- iat, when the token was issued
- jti, the token identifier uuid
- roles, the authenticated roles for that user.

## 3.9 Powergrid Model API

The Powergrid Model Data Manager API allows you to query the powergrid model data store.

### 3.9.1 Query Request Queue

Query request should be sent on following queue: goss.gridappsd.process.request.data.powergridmodel

### 3.9.2 Query Model Info

Returns list of names/ids for models, substations, subregions, and regions for all available feeders.

Allowed parameter is:

- Result Format – XML/JSON/CSV, Will return results as a list in the format selected.

Example Request:

```
{
        "requestType": "QUERY_MODEL_INFO",
        "resultFormat": "JSON"
}
```

Example Response for result format JSON:

```
{
        "models": [{
                "modelName": "ieee123",
                "modelId": "_C1C3E687-6FFD-C753-582B-632A27E28507",
                "stationName": "ieee123_Substation",
                "stationId": "_FE44B314-385E-C2BF-3983-3A10C6060022",
                "subRegionName": "large",
                "subRegionId": "_1CD7D2EE-3C91-3248-5662-A43EFEFAC224",
                "regionName": "ieee",
                "regionId": "_24809814-4EC6-29D2-B509-7F8BFB646437"
}, .......
```

### 3.9.3 Query Model Names

Returns list of names for all available models.

Allowed parameter is:

- Result Format – XML/JSON/CSV, Will return results as a list in the format selected.

Example Request: goss.gridappsd.process.request.data.powergridmodel

```
{
        "requestType": "QUERY_MODEL_NAMES",
        "resultFormat": "JSON"
}
```

Example Response for result format JSON:

```
{
        "modelNames": ["_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3",
        "_5B816B93-7A5F-B64C-8460-47C17D6E4B0F",
        "_67AB291F-DCCD-31B7-B499-338206B9828F",
        "_9CE150A8-8CC5-A0F9-B67E-BBD8C79D3095",
        "_C1C3E687-6FFD-C753-582B-632A27E28507"]
}
```

Python API function:

```
query_model_names(self, model_id=None)
```

### 3.9.4 Query

Returns results from a generic SPARQL query against one or all models.

Allowed parameters are:

- modelId (optional) - when specified it searches against that model, if empty it will search against all models

- queryString - SPARQL query, for more information see https://www.w3.org/TR/rdf-sparql-query/ See below for example.
- resultFormat – XML/JSON , The format you wish the result to be returned in. Can be either JSON or XML. Will return result bindings based on the select part of the query string. See below for example.

Example Request: goss.gridappsd.process.request.data.powergridmodel

```
{
        "requestType": "QUERY",
        "resultFormat": "JSON",
        "queryString": "select ?feeder_name ?subregion_name ?region_name WHERE {?line␣
↪r:type c:Feeder.?line c:IdentifiedObject.name  ?feeder_name.?line c:Feeder.
↪NormalEnergizingSubstation ?substation.?substation r:type c:Substation.?substation␣
↪c:Substation.Region ?subregion.?subregion  c:IdentifiedObject.name  ?subregion_name␣
↪.?subregion c:SubGeographicalRegion.Region  ?region . ?region   c:IdentifiedObject.
↪name  ?region_name}"
}
```

Example Response:

```
{
"head": {
         "vars": [ "line_name" , "subregion_name" , "region_name" ]
 } ,
"results": {
        "bindings": [
         {
                "line_name": { "type": "literal" , "value": "ieee8500" } ,
                "subregion_name": { "type": "literal" , "value": "ieee8500_SubRegion"␣
↪},
                "region_name": { "type": "literal" , "value": "ieee8500_Region" }
          }
          ]
}
}
```

Python API function:

```
query_data(self, query, database_type=POWERGRID_MODEL, timeout=30)
```

### 3.9.5 Query Object

Returns details for a particular object based on the object Id.

Allowed parameters are:

- modelId (optional) - when specified it searches against that model, if empty it will search against all models
- objectID – mrid of the object you wish to return details for.
- resultFormat – XML/JSON , Will return result bindings based on the select part of the query string.

Example Request: goss.gridappsd.process.request.data.powergridmodel

```
{
        "requestType": "QUERY_OBJECT",
        "resultFormat": "JSON",
```

(continues on next page)

(continued from previous page)

```
        "objectId": "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3"
}
```

Example Response:

```
{
  "head": {
    "vars": [ "property" , "value" ]
  } ,
  "results": {
    "bindings": [
      {
        "property": { "type": "uri" , "value": "http://iec.ch/TC57/2012/CIM-schema-
↪cim17#Feeder.NormalEnergizingSubstation" } ,
        "value": { "type": "uri" , "value": "http://localhost:9999/blazegraph/
↪namespace/kb/sparql#_F1E8E479-5FA0-4BFF-8173-B375D25B0AA2" }
      } ,
      {
        "property": { "type": "uri" , "value": "http://iec.ch/TC57/2012/CIM-schema-
↪cim17#IdentifiedObject.mRID" } ,
        "value": { "type": "literal" , "value": "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3
↪" }
      } ,
      {
        "property": { "type": "uri" , "value": "http://iec.ch/TC57/2012/CIM-schema-
↪cim17#IdentifiedObject.name" } ,
        "value": { "type": "literal" , "value": "ieee8500" }
      } ,
      {
        "property": { "type": "uri" , "value": "http://iec.ch/TC57/2012/CIM-schema-
↪cim17#PowerSystemResource.Location" } ,
        "value": { "type": "uri" , "value": "http://localhost:9999/blazegraph/
↪namespace/kb/sparql#_AD650B25-8A04-EA09-95D4-4F78DD0A05E7" }
      } ,
      {
        "property": { "type": "uri" , "value": "http://www.w3.org/1999/02/22-rdf-
↪syntax-ns#type" } ,
        "value": { "type": "uri" , "value": "http://iec.ch/TC57/2012/CIM-schema-cim17
↪#Feeder" }
      }
    ]
  }
}
```

Python API function:

```
query_object(self, object_id, model_id=None):
```

### 3.9.6 Query Object Types

Returns the available object types in the model

Allowed parameters are:

- modelId (optional) - when specified it searches against that model, if empty it will search against all models
- resultFormat – XML/JSON /CSV, Will return results as a list in the format selected.

Example Request: goss.gridappsd.process.request.data.powergridmodel

```
{
        "requestType": "QUERY_OBJECT_TYPES",
        "modelId": "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3",
        "resultFormat": "JSON"
}
```

Example Response:

```
{
        "objectTypes": ["http://iec.ch/TC57/2012/CIM-schema-cim17#ConnectivityNode",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#TransformerTank",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#PowerTransformer",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#LinearShuntCompensator",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#EnergySource",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#ACLineSegment",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#LoadBreakSwitch",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#EnergyConsumer"]
}
```

Python API function:

```
query_object_types(self, model_id=None)
```

### 3.9.7 Query Model

Returns all or part of the specified model. Can be filtered by object type

Allowed parameters are:

- modelId - when specified it searches against that model, if empty it will search against all models
- objectType (optional) – type of objects you wish to return details for.
- filter – SPARQL formatted filter string
- resultFormat – XML/JSON, Will return result in the format selected.

Example Request: goss.gridappsd.process.request.data.powergridmodel

```
{
        "requestType": "QUERY_MODEL",
        "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "resultFormat": "JSON",
        "filter": "?s cim:IdentifiedObject.name '650z'",
        "objectType": "http://iec.ch/TC57/CIM100#ConnectivityNode"
}
```

Example Response:

```
[{
        "id": "_0F9BF9EE-B900-71C2-B892-0287A875A158",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#ConnectivityNode.
↪ConnectivityNodeContainer": "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#ConnectivityNode.TopologicalNode":
↪"_AE5EDB3A-9177-AEA6-78EF-3DDBA4557D94",
        "http://iec.ch/TC57/2012/CIM-schema-cim17#IdentifiedObject.mRID": "_0F9BF9EE-
↪B900-71C2-B892-0287A875A158",
```

(continues on next page)

(continued from previous page)

```
        "http://iec.ch/TC57/2012/CIM-schema-cim17#IdentifiedObject.name": "q14733",
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#type": "http://iec.ch/TC57/2012/
↪CIM-schema-cim17#ConnectivityNode"
}]
```

### 3.9.8 Query Object Ids

Returns details for a particular object based on the object Id.

Allowed parameters are:

- modelId - when specified it searches against that model, if empty it will search against all models
- objectType (optional) – type of objects you wish to return details for.
- resultFormat – XML/JSON/CSV , Will return result bindings based on the select part of the query string.

Example Request: goss.gridappsd.process.request.data.powergridmodel

```
{
        "modelId": "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3",
        "requestType": "QUERY_OBJECT_IDS",
        "resultFormat": "JSON",
        "objectType": "LoadBreakSwitch"
}
```

Example Response:

```
{
  "objectIds": [
        "_0D2157F2-CD4D-9F68-9212-F663C472AF1C",
        "_18D43D9E-36D1-3A2C-AC8F-439232FC1EE2",
        "_323C2BDB-69AA-A10C-CEC5-628C77B83268",
        "_D7AA7B55-E700-F1E8-B3EB-CB2FB07F8A37",
        .......
  ]
}
```

### 3.9.9 Query Object Dictionary

Returns details for either all objects of a particular type or a particular object based on the object Id. Either the object type or id is required, but not both.

Allowed parameters are:

- modelId - model that you wish to return objects from.
- objectType (not required if objectId is set) – type of object you wish to return details for.
- objectId (not required if objectType is set) - mrid of the object you wish to return details for, if set this will override objectType.
- resultFormat – XML/JSON , Will return result bindings based on the select part of the query string.

Example Request: goss.gridappsd.process.request.data.powergridmodel

```
{
        "modelId": "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3",
        "requestType": "QUERY_OBJECT_DICT",
        "resultFormat": "JSON",
        "objectType": "LinearShuntCompensator",
        "objectId": "_EF2FF8C1-A6A6-4771-ADDD-A371AD929D5B"
}
```

Example Response:

```
{
 [
   {
        "id": "_2199D08B-9352-2085-102F-6B207E0BEBA3",
        "ConductingEquipment.BaseVoltage": "_C0A00494-BB68-7476-57E3-9741545AE287",
        "Equipment.EquipmentContainer": "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3",
        "IdentifiedObject.mRID": "_2199D08B-9352-2085-102F-6B207E0BEBA3",
        "IdentifiedObject.name": "capbank0a",
        "PowerSystemResource.Location": "_19B9D45D-F556-01D4-8094-3AE64D5E63A0",
        "LinearShuntCompensator.b0PerSection": "100",
        "LinearShuntCompensator.bPerSection": "0.0077160494",
        "LinearShuntCompensator.g0PerSection": "0",
        "LinearShuntCompensator.gPerSection": "0",
        "ShuntCompensator.aVRDelay": "100",
        "ShuntCompensator.grounded": "true",
        "ShuntCompensator.maximumSections": "1",
        "ShuntCompensator.nomU": "7200",
        "ShuntCompensator.normalSections": "1",
        "ShuntCompensator.phaseConnection": "PhaseShuntConnectionKind.Y",
        "type": "LinearShuntCompensator"
   },....
 ]
}
```

### 3.9.10 Query Object Measurements

Returns details for measurements within a model, can be for all objects of a particular type or for those connected to a particular object based on the objectId. If neither objectType or objectId is provided it will provide all measurements belonging to the model.

Allowed parameters are:

- modelId - model that you wish to return measurements from.
- objectType (optional) – type of object you wish to return measurements for.
- objectId (optional) - mrid of the object you wish to return measurements for. If set this will override objectType.
- resultFormat – XML/JSON , Will return result bindings based on the select part of the query string.

Example Request: goss.gridappsd.process.request.data.powergridmodel

```
{
        "modelId": "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3",
        "requestType": "QUERY_OBJECT_MEASUREMENTS",
        "resultFormat": "JSON",
        "objectType": "LinearShuntCompensator",
```

(continues on next page)

(continued from previous page)

```
        "objectId": "_2199D08B-9352-2085-102F-6B207E0BEBA3"
}
```

Example Response:

```
  [
{
      "measid": "_59d526ff-32c0-4947-ab58-45f283636786",
      "type": "PNV",
      "class": "Analog",
      "name": "ACLineSegment_ln5532752-2_Voltage",
      "bus": "m1047534",
      "phases": "A",
      "eqtype": "ACLineSegment",
      "eqname": "ln5532752-2",
      "eqid": "_7A02B3B0-2746-EB24-45A5-C3FBA8ACB88E",
      "trmid": "_6B5B889C-E7E1-3444-CC63-7A589AC0DA8F"
  },....
   ]
```

### 3.9.11 Put Model

**Note:** Future Capability. Not yet available.

Inserts a new model into the model repository. This could validate model format during insertion **Keep cim/model version in mind**

Allowed parameters are:

- modelId – id to store the new model under, or update existing model
- modelContent – expects either RDF/XML or JSON formatted powergrid model
- inputFormat – XML/JSON

## 3.10 Configuration File API

### 3.10.1 Request all GridLAB-D configuration files

Generates all configuration files necessary to run a sumulation using the GridLAB-D simulator. Returns the diretory where all of the configuration files are stored.

- Required: configurationType, parameters[model_id,directory,simulationname,simulation_start_time,simulation_duration,simulati
- Optional: parameters[i_fraction, p_fraction, z_fraction, load_scaling_factor, schedule_name,solver_method]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType": "GridLAB-D All",
  "parameters": {
    "load_scaling_factor": "1.0",
    "i_fraction": "1.0",
```

(continues on next page)

(continued from previous page)

```
    "model_id": "_C1C3E687-6FFD-C753-582B-632A27E28507",
    "p_fraction": "0.0",
    "simulation_id": "12345",
    "z_fraction": "0.0",
    "simulation_broker_host": "localhost",
    "simulation_name": "ieee8500",
    "simulation_duration": "60",
    "simulation_start_time": "1518958800",
    "solver_method": "NR",
    "schedule_name": "ieeeziploadd",
    "simulation_broker_port": "61616",
    "directory": "/tmp/gridlabdsimulation/"
  }
}
```

Response: <directory where files have been stored>

### 3.10.2 Request GridLAB-D Base File

Generates the main GLM file required by the GridLAB-D simulator

- Required: configurationType, parameters[model_id]
- Optional: parameters[simulation_id, i_fraction, p_fraction, z_fraction, load_scaling_factor, schedule_name]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType": "GridLAB-D Base GLM",
  "parameters": {
    "i_fraction": "1.0",
    "z_fraction": "0.0",
    "model_id": "_C1C3E687-6FFD-C753-582B-632A27E28507",
    "load_scaling_factor": "1.0",
    "schedule_name": "ieeeziploadd",
    "p_fraction": "0.0"
  }
}
```

Response:

```
object regulator_configuration {
name "rcon_reg1a";
connect_type WYE_WYE;
      Control MANUAL; // LINE_DROP_COMP;
.......
```

### 3.10.3 Request GridLAB-D Symbols File

Generates the symbols file with XY coordinates used by the GridLAB-D simulator

- Required: configurationType, parameters[model_id]
- Optional: parameters[simulation_id]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType": "GridLAB-D Symbols",
  "parameters": {
    "model_id": "_C1C3E687-6FFD-C753-582B-632A27E28507"
  }
}
```

Response: :: {“feeders”:[

**{“name”:”ieee123”,** “mRID”:”_C1C3E687-6FFD-C753-582B-632A27E28507”, “substation”:”IEEE123”, “substationID”:”_FE44B314-385E-C2BF-3983-3A10C6060022”, “subregion”:”Medium”, “subregionID”:”_1CD7D2EE-3C91-3248-5662-A43EFEFAC224”, “region”:”IEEE”, “regionID”:”_73C512BD-7249-4F50-50DA-D93849B89C43”, “swing_nodes”:[ {“name”:”source”,”bus”:”150”,”phases”:”ABC”,”nominal_voltage”:2401.8,”x1”:100.0,”y1”:1500.0} ], “synchronousmachines”:[ ], “capacitors”:[

### 3.10.4 Request CIM Dictionary file

Generates a dictionary file which maps between the mrid identifiers used by the CIM model and the other names of model objects used by simulators.

- Required: configurationType, parameters[model_id]
- Optional: parameters[simulation_id]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType":"CIM Dictionary",
  "parameters":{"model_id":"_C1C3E687-6FFD-C753-582B-632A27E28507"}
 }
```

Response:

```
{"feeders":[
      {"name":"ieee123",
      "mRID":"_C1C3E687-6FFD-C753-582B-632A27E28507",
      "substation":"IEEE123",
      "substationID":"_FE44B314-385E-C2BF-3983-3A10C6060022",
      "subregion":"Medium",
      "subregionID":"_1CD7D2EE-3C91-3248-5662-A43EFEFAC224",
      "region":"IEEE",
      "regionID":"_73C512BD-7249-4F50-50DA-D93849B89C43",
      "synchronousmachines":[
      ],
      "capacitors":[
      {"name":"c83","mRID":"_232DD3A8-9A3C-4053-B972-8A5EB49FD980","CN1":"83","phases
↪":"ABC","kvar_A":200.0,"kvar_B":200.0,"kvar_C":200.0,"nominalVoltage":4160.0,"nomU
↪":4160.0,"phaseConnection":"Y","grounded":true,"enabled":false,"mode":null,
↪"targetValue":0.0,"targetDeadband":0.0,"aVRDelay":0.0,"monitoredName":null,
↪"monitoredClass":null,"monitoredBus":null,"monitoredPhase":null},
      {"name":"c88a","mRID":"_9A74DCDC-EA5A-476B-9B99-B4FB90DC37E3","CN1":"88","phases
↪":"A","kvar_A":50.0,"kvar_B":0.0,"kvar_C":0.0,"nominalVoltage":4160.0,"nomU":2402.0,
↪"phaseConnection":"Y","grounded":true,"enabled":false,"mode":null,"targetValue":0.0,
↪"targetDeadband":0.0,"aVRDelay":0.0,"monitoredName":null,

.......
```

(continues on next page)

(continued from previous page)

```
]
}]}
```

### 3.10.5 Request CIM Feeder Index file

Generates a list of the feeders available powergrid model data store

- Required: configurationType, parameters[model_id]
- Optional: parameters[simulation_id]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType":"CIM Feeder Index",
  "parameters":{"model_id":"_C1C3E687-6FFD-C753-582B-632A27E28507"}
 }
```

Response:

```
{"feeders":[
```

{“name”:”test9500new”,”mRID”:”_AAE94E4A-2465-6F5E-37B1-3E72183A4E44”,”substationName”:”ThreeSubs”,”substationID”:”_9B2C-1B8C-EC33-39D2F7948163”,”subregionName”:”Large”,”subregionID”:”_A1170111-942A-6ABD-D325-C64886DC4D7D”,”regionName”:”IEEE”,”regionID”:”_73C512BD-7249-4F50-50DA-D93849B89C43”}, {“name”:”ieee123”,”mRID”:”_C1C3E687-6FFD-C753-582B-632A27E28507”,”substationName”:”IEEE123”,”substationID”:”_FE44B314-385E-C2BF-3983-3A10C6060022”,”subregionName”:”Medium”,”subregionID”:”_1CD7D2EE-3C91-3248-5662-A43EFEFAC224”,”regionName”:”IEEE”,”regionID”:”_73C512BD-7249-4F50-50DA-D93849B89C43”},

]}

### 3.10.6 Request Simulation Output Configuration file

Generates file containing objects and properties with measurements avilable in the selected model

- Required: configurationType, parameters[model_id]
- Optional: parameters[simulation_id]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType":"GridLAB-D Simulation Output",
  "parameters":{"model_id":"_C1C3E687-6FFD-C753-582B-632A27E28507"}
 }
```

Response:

```
{
  "cap_capbank0a": [
    "switchA",
    "shunt_A",
    "voltage_A"
  ],
```

(continues on next page)

(continued from previous page)

```
  "cap_capbank1b": [
    "switchB",
    "voltage_B",
    "shunt_B"
  ],
  "cap_capbank2c": [
    "voltage_C",
    "switchC",
    "shunt_C"
  ],
  "cap_capbank0b": [
    "voltage_B",
    "switchB",
    "shunt_B"
  ],.......
```

### 3.10.7 Request all OpenDSS configuration files

Generates all configuration files necessary to run a sumulation using the OpenDSS simulator. Returns the diretory where all of the configuration files are stored.

- Required: configurationType, parameters[model_id,directory,simulationname,simulation_start_time,simulation_duration,simulati
- Optional: parameters[i_fraction, p_fraction, z_fraction, load_scaling_factor, schedule_name,solver_method]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType": "DSS All",
  "parameters": {
    "load_scaling_factor": "1.0",
    "i_fraction": "1.0",
    "model_id": "_C1C3E687-6FFD-C753-582B-632A27E28507",
    "p_fraction": "0.0",
    "simulation_id": "12345",
    "z_fraction": "0.0",
    "simulation_broker_host": "localhost",
    "simulation_name": "ieee8500",
    "simulation_duration": "60",
    "simulation_start_time": "1518958800",
    "solver_method": "NR",
    "schedule_name": "ieeezipload",
    "simulation_broker_port": "61616",
    "directory": "/tmp/dsssimulation/"
  }
}
```

Response: <directory where files have been stored>

### 3.10.8 Request OpenDSS Base File

Generates the main GLM file required by the OpenDSS simulator

- Required: configurationType, parameters[model_id]

- Optional: parameters[simulation_id, i_fraction, p_fraction, z_fraction, load_scaling_factor, schedule_name]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType": "DSS Base",
  "parameters": {
    "i_fraction": "1.0",
    "z_fraction": "0.0",
    "model_id": "_C1C3E687-6FFD-C753-582B-632A27E28507",
    "load_scaling_factor": "1.0",
    "schedule_name": "ieeeziploa d",
    "p_fraction": "0.0"
  }
}
```

Response:

```
clear
new Circuit.source phases=3 bus1=150 basekv=4.160 pu=1.00000 angle=0.00000 r0=0.00000␣
↪x0=0.00010 r1=0.00000 x1=0.00010
new Linecode.11 nphases=1 units=mi rmatrix=[1.32920 ] xmatrix=[1.34750 ] cmatrix=[11.
↪9873 ]
new Linecode.1 nphases=3 units=mi rmatrix=[0.457600 | 0.156000 0.466600 | 0.153500 0.
↪158000 0.461500 ] xmatrix=[1.07800 | 0.501700 1.04820 | 0.384900 0.423600 1.06510 ]␣
↪cmatrix=[15.0567 | -4.85904 15.8641 | -1.85195 -3.08879 14.3156 ]


.......
```

### 3.10.9 Request OpenDSS Coordinates File

Generates the symbols file with XY coordinates used by the OpenDSS simulator

- Required: configurationType, parameters[model_id]
- Optional: parameters[simulation_id]

Request: goss.gridappsd.process.request.config

```
{
  "configurationType": "DSS Coordinate",
  "parameters": {
    "model_id": "_C1C3E687-6FFD-C753-582B-632A27E28507"
  }
}
```

Response:

88,2950.0,1300.0 89,2775.0,1125.0 197,3525.0,2200.0 110,4275.0,3050.0 111,4275.0,3625.0 112,4275.0,2925.0 113,4800.0,2925.0 114,5125.0,2925.0 90,2775.0,900.0 61s,3175.0,1300.0 91,2550.0,1125.0 92,2550.0,825.0 93,2325.0,1125.0 94,2325.0,850.0 95,2025.0,1125.0 96,2025.0,925.0 97,3525.0,2100.0 98,3800.0,2100.0 10,1450.0,2150.0 99,4350.0,2100.0 11,950.0,2150.0

### 3.10.10 Request YBus Export Configuration file

Generates file containing ybus configuration for the given model or simulation.

Request: goss.gridappsd.process.request.config

```
{
  "configurationType":"YBus Export",
  "parameters":{"simulation_id":"12345"}
  }
```

If requested for a simulation then simulation id is mandatory. Otherwise use model_id as mentioned next.

```
{
"configurationType": "YBus Export",
"parameters": {
  "model_id": "_C1C3E687-6FFD-C753-582B-632A27E28507"
            }
    }
```

Additional paramters can be provided with model_id as mentioned in next request.

```
  {
  "configurationType": "YBus Export",
  "parameters": {
    "i_fraction": "1.0",
    "z_fraction": "0.0",
    "model_id": "_C1C3E687-6FFD-C753-582B-632A27E28507",
    "load_scaling_factor": "1.0",
    "schedule_name": "ieeeziploads",
    "p_fraction": "0.0"
  }
}
```

Response:

```
{
      "yParse": [
          "Row,Col,G,B",
          "1,1,517.6253721,-539.2591296",
          "2,1,-3.438703156,9.070554234",
          "3,1,-5.837170999,11.07061383",
          "4,1,-500,500",
          "84,1,-9.232329792,20.56428834",
          "85,1,1.801223903,-4.751238599",
          "86,1,3.057563114,-5.798887966"
          ..........
              ],
              "nodeList": [
                "\"97.1\"",
                "\"97.2\"",
              ..........
              ],
              "summary": [
                 "DateTime, .......
              ]
      }
```

## 3.11 Logging API

All applications and services should publish their log messages using using paltform;s log API.

### 3.11.1 Publishing Logs:

Log messages should be published on the following topic. Simulation id should be attached to the topic at the end.

```
goss.gridappsd.simulation.log.[simulation_id]
```

Message structure for publishing logs :

```
{
        "source": "",
        "processId": "",
        "timestamp": "long",
        "processStatus": "[STARTED|STOPPED|RUNNING|ERROR|PASSED|FAILED]",
        "logMessage": "",
        "logLevel": "[INFO|DEBUG|ERROR]",
        "storeToDb": [true|false]
}
```

where,

source is the filename publishing log message.

processId is the simulation id.

timestamp is in epoch format.

storeToDb is true if you want to store this message in log database for later.

### 3.11.2 Subscribing to Logs:

For the currently running simulation, subcribe to following topic with simulation id appended at the end to receive real time logs:

```
goss.gridappsd.simulation.log.[simulation_id]
```

### 3.11.3 Querying Logs:

Query request should be sent at following topic:

```
goss.gridappsd.process.request.data.log
```

User can query log data by sending either custom SQL query string or using query filters.

1. Custom query string:

Logs are stored in MySQL database in a table named log with following columns: source, processId,timestamp, processStatus, logMessage, logLevel. User can create custom SQL query string to get log data:

```
{"query":"select * from log"}
```

Custom query response:

```
{ "data": [
        { "process_id": "", "process_status": "RUNNING", "log_level": "INFO", "log_
↪message": "Starting gov.pnnl.goss.gridappsd.app.AppManagerImpl", "id": "1", "source
↪": "gov.pnnl.goss.gridappsd.app.AppManagerImpl", "timestamp": "2018-11-14 21:51:11.0
↪", "username": "system" },
        { "process_id": "", "process_status": "RUNNING", "log_level": "INFO", "log_
↪message": "Found 0 applications", "id": "2", "source": "gov.pnnl.goss.gridappsd.app.
↪AppManagerImpl", "timestamp": "2018-11-14 21:51:14.0", "username": "system" },
], "responseComplete": true, "id": "1792453601" }
```


2. Query filters:

An example for query filters are

```
{
        "source": "ProcessEvent",
        "processId": "12345678",
        "processStatus": "DEBUG",
        "logLevel": "DEBUG"
}
```

For more details on log message filter look at 'Publishing Logs' section.

Custom query response:


```
{ "data": [
        { "process_id": "414798372", "process_status": "RUNNING", "log_level": "DEBUG
↪", "log_message": "New rewuest received", "id": "8", "source": "ProcessEvent",
↪"timestamp": "2018-11-14 21:51:29.0", "username": "system" },
        { "process_id": "", "process_status": "RUNNING", "log_level": "DEBUG", "log_
↪message": "Running application", "id": "2", "source": "ProcessEvent", "timestamp":
↪"2018-11-14 21:51:30.0", "username": "system" },
], "responseComplete": true, "id": "1792453601" }
```


## 3.12 Simulation API

### 3.12.1 Start a Simulation

Returns simulation id.

Queue:

```
goss.gridappsd.process.request.simulation
```

Example Request:

{

*power_system_config: the CIM model to be used in the simulation*

```
"power_system_config": {
        "GeographicalRegion_name": "ieee8500nodecktassets_Region",
        "SubGeographicalRegion_name": "ieee8500nodecktassets_SubRegion",
        "Line_name": "ieee8500"
},
```

*simulation_config: the paramaters used by the simulation*

```
"simulation_config": {
        "start_time": "2009-07-21 00:00:00",
        "duration": "120",
        "simulator": "GridLAB-D",
        "timestep_frequency": "1000",
        "timestep_increment": "1000",
        "simulation_name": "ieee8500",
        "power_flow_solver_method": "NR",
```

*simulation_output: the objects and fields to be returned by the simulation*

```
"simulation_output": {
        "output_objects": [{
                "name": "rcon_FEEDER_REG",
                "properties": ["connect_type",
                "Control",
                "control_level",
                "PT_phase",
                "band_center",
                "band_width",
                "dwell_time",
                "raise_taps",
                "lower_taps",
                "regulation"]
        },
        .....]
},
```

*model creation config: the paramaters used to generate the input file for the simulation*

```
        "model_creation_config": {
                "load_scaling_factor": "1",
                "schedule_name": "ieeezipload",
                "z_fraction": "0",
                "i_fraction": "1",
                "p_fraction": "0",
                "model_state":{
                        "synchronousmachines":[
                                {"name":"diesel590","p":100.000,"q":140.000},
                                {"name":"diesel620","p":150.000,"q":500.000}
                        ],
                        "switches":[
                                {"name":"2002200004641085_sw","open":true},
                                {"name":"2002200004868472_sw","open":true},
                                {"name":"19407_48332_sw","open":true},
                                {"name":"tsw568613_sw","open":false}
                        ]
            }
        }
},
```

*application config: inputs to any other applications that should run as part of the simluation, in this case the voltvar application*

```
"application_config": {
        "applications": [{
                "name": "vvo",
```

(continues on next page)

(continued from previous page)

```
                "config_string": "{\"static_inputs\": {\"ieee8500\" : {\"control_
↪method\": \"ACTIVE\", \"capacitor_delay\": 60, \"regulator_delay\": 60, \"desired_
↪pf\": 0.99, \"d_max\": 0.9, \"d_min\": 0.1,\"substation_link\": \"xf_hvmv_sub\",\
↪"regulator_list\": [\"reg_FEEDER_REG\", \"reg_VREG2\", \"reg_VREG3\", \"reg_VREG4\
↪"],\"regulator_configuration_list\": [\"rcon_FEEDER_REG\", \"rcon_VREG2\", \"rcon_
↪VREG3\", \"rcon_VREG4\"],\"capacitor_list\": [\"cap_capbank0a\",\"cap_capbank0b\", \
↪"cap_capbank0c\", \"cap_capbank1a\", \"cap_capbank1b\", \"cap_capbank1c\", \"cap_
↪capbank2a\", \"cap_capbank2b\", \"cap_capbank2c\", \"cap_capbank3\"], \"voltage_
↪measurements\": [\"nd_l2955047,1\", \"nd_l3160107,1\", \"nd_l2673313,2\", \"nd_
↪l2876814,2\", \"nd_m1047574,3\", \"nd_l3254238,4\"],       \"maximum_voltages\":␣
↪7500, \"minimum_voltages\": 6500,\"max_vdrop\": 5200,\"high_load_deadband\": 100,\
↪"desired_voltages\": 7000,   \"low_load_deadband\": 100,\"pf_phase\": \"ABC\"}}}"
        }]
}
```

### 3.12.2 Subscribe to Simulation Output

Topic:

```
/topic/goss.gridappsd.simulation.output.[simulation_id]
```

Where simulation_id is response from start simulation API.

Example Message:

```
{
        "simulation_id" : "12ae2345",
    "message" : {
        "timestamp" : "1357048800",
        "measurements" : {
            "123a456b-789c-012d-345e-678f901a234b":{
                                "measurement_mrid" : "123a456b-789c-012d-345e-
↪678f901a234b"
                                "magnitude" : 3410.456,
                                "angle" : -123.456
        }
    }
}
```

### 3.12.3 Subscribe to Simulation Logs

Topic:

```
/topic/goss.gridappsd.simulation.log.[simulation_id]
```

Where simulation_id is response from start simulation API.

Example Message:

```
{
        "source": "",
        "processId": "",
        "timestamp": "",
        "processStatus": "[STARTING|STARTED|STOPPED|RUNNING|ERROR|CLOSED|COMPLETE]",
```

(continues on next page)

(continued from previous page)

```
        "logMessage": "",
        "logLevel": "[INFO|DEBUG|ERROR]",
        "storeToDb": [true|false]
}
```

### 3.12.4 Send Input to Simulation

Topic:

```
/topic/goss.gridappsd.simulation.input.[simulation_id]
```

Example Message:

```
{
  "command": "update",
  "input": {
      "simulation_id": "123456",
      "message": {
          "timestamp": 1357048800,
          "difference_mrid": "123a456b-789c-012d-345e-678f901a235c",
          "reverse_differences": [{

                  "object": "61A547FB-9F68-5635-BB4C-F7F537FD824E",
                  "attribute": "ShuntCompensator.sections",
                  "value": 1
              },
              {

                  "object": "E3CA4CD4-B0D4-9A83-3E2F-18AC5F1B55BA",
                  "attribute": "ShuntCompensator.sections",
                  "value": 0
              }
          ],
          "forward_differences": [{

                  "object": "61A547FB-9F68-5635-BB4C-F7F537FD824E",
                  "attribute": "ShuntCompensator.sections",
                  "value": 0
              },
              {

                  "object": "E3CA4CD4-B0D4-9A83-3E2F-18AC5F1B55BA",
                  "attribute": "ShuntCompensator.sections",
                  "value": 1
              }
              ]
              }
      }
}
```

### 3.12.5 Pause Simulation

Topic:

```
/topic/goss.gridappsd.simulation.input.[simulation_id]
```

Example Message:

```
{
    "command": "pause"
}
```

### 3.12.6 Resume Simulation

Topic:

```
/topic/goss.gridappsd.simulation.input.[simulation_id]
```

Example Message:

```
{
    "command": "resume"
}
```

### 3.12.7 Resume and Pause the Simulation after a Specified Number of Seconds

Topic:

```
/topic/goss.gridappsd.simulation.input.[simulation_id]
```

Example Message:

```
{
    "command": "resumePauseAt",
    "input": {
        "pauseIn": 10
    }
}
```

## 3.13 Timeseries API

The Timeseries Data API allows you to query the timeseries data such as weather, simulation output and input.

### 3.13.1 Query Request Queue

Query request should be sent on following queue: goss.gridappsd.process.request.data.timeseries

### 3.13.2 Query Weather data

The weather data is based on exported data collected from the Solar Radiation Research Laboratory (39.74N,105.18W,1829 meter elevation) January - December 2013. The original dataset was based in Mountain Standard Time (MST).

The original column names included engineering units, but could not be included on the import. Below is a mapping between the exported column headers and the fields in the Influx database management system.

```
Original Exported Data                          Influx Measurement Field Key       Field␣
↪Type
-------------------------------------       ----------------------------       -------
↪---
DATE (MM/DD/YYYY)                               DATE                               String
MST                                             MST                                String
Global CM22 (vent/cor) [W/ft^2]                 GlobalCM22                         Float
Direct CH1 [W/ft^2]                             DirectCH1                          Float
Diffuse CM22 (vent/cor) [W/ft^2]                Diffuse                            Float
Tower Dry Bulb Temp [deg F]                     TowerDryBulbTemp                   Float
Tower RH [%]                                    TowerRH                            Float
Avg Wind Speed @ 42ft [MPH]                     AvgWindSpeed                       Float
Avg Wind Direction @ 42ft [deg from N]          AvgWindDirection                   Float

Original Exported Data                          Influx Measurement Tag             Type
-------------------------------------       ----------------------------       -------
↪---
n/a                                             lat                                String
n/a                                             long                               String
n/a                                             place                              String
```

**Influx database details:**

Database name: “proven”, Measurement name: “weather”

Example Request:

```
{"queryMeasurement":"weather",
"queryFilter":{"startTime":"1357048800000000",
                        "endTime":"1357048860000000"},
"responseFormat":"JSON"}
```

Example Response for result format JSON:

```
{ "data": [ { "Diffuse": 2.5305959999999996,
                "AvgWindSpeed": 0,
                "TowerRH": 70.65,
                "long": "105.18 W",
                "MST": "08:00",
                "TowerDryBulbTemp": 16.124,
                "DATE": "1/1/2013",
                "DirectCH1": 0.08549150370000001,
                "GlobalCM22": 2.53962588,
                "AvgWindDirection": 0,
                "time": 1357048800,
                "place": "Solar Radiation Research Laboratory",
                "lat": "39.74 N" },
        { "Diffuse": 2.6431350599999996,
                "AvgWindSpeed": 0,
                "TowerRH": 70.41,
                "long": "105.18 W",
                "MST": "08:01",
                "TowerDryBulbTemp": 15.908,
                "DATE": "1/1/2013",
                "DirectCH1": 0.045951777299999996,
                "GlobalCM22": 2.6501118499999996,
```

(continues on next page)

(continued from previous page)

```
                "AvgWindDirection": 0,
                "time": 1357048860,
                "place": "Solar Radiation Research Laboratory",
                "lat": "39.74 N" } ],
  "responseComplete": true,
  "id": "1998314042" }
```

Allowed values for queryFilter are:

```
startTime[epoch number]
endTime[epoch number]
AvgWindDirection[number]
AvgWindSpeed[number]
Diffuse[number]
DirectCH1[number]
GlobalCM22[number]
MST[number]
TowerDryBulbTemp[number]
TowerRH[number]
lat[string]
long[string]
place[string]
```

### 3.13.3 Query Simulation Data

Returns simulation input or output data based on query filters

Example Request:

```
{"queryMeasurement": "simulation",
"queryFilter": {"simulation_id": "582881157"},
"responseFormat": "JSON"}
```

Example Response for result format JSON:

```
{
"data": { "measurements": [{ "name": "simulation",
                "points": [{ "row": { "entry": [
                        { "key": "hasMeasurementDifference", "value": "FORWARD" },
                        { "key": "hasSimulationMessageType", "value": "INPUT" },
                        { "key": "difference_mrid", "value": "c65d4ba9-8689-4838-970c-
↪2983b54ed2e6" },
                        { "key": "simulation_id", "value": "582881157" },
                        { "key": "time", "value": "1562614884" },
                        { "key": "attribute", "value": "ShuntCompensator.sections" },
                        { "key": "value", "value": "0.0" },
                        { "key": "object","value": "_5405BE1A-BC86-5452-CBF2-
↪BD1BA8984093" }]}},
                { "row": { "entry": [
                        { "key": "hasMeasurementDifference", "value": "FORWARD" },
                        { "key": "hasSimulationMessageType", "value": "INPUT" },
                        { "key": "difference_mrid", "value": "c65d4ba9-8689-4838-970c-
↪2983b54ed2e6" },
                        { "key": "simulation_id", "value": "582881157" },
                        { "key": "time", "value": "1562614884" },
```

(continues on next page)

(continued from previous page)

```
                        { "key": "attribute", "value": "ShuntCompensator.sections" },
                        { "key": "value", "value": "0.0" },
                        { "key": "object", "value": "_8D0EAC3F-AD56-C5A6-ED03-
↪863DBB4A8C5F"}]}}
"responseComplete": true,
"id": "1927836780" }
```

Allowed values for queryFilter are:

```
Both input and output message type:
starttime [number]
endtime [number]
measurement_mrid [string] or [array of string values]
simulation_id [string]
hasSimulationMessageType ["OUTPUT" | "INPUT"]

Ouput message type:
angle [number]
magnitude [number]

Input Message type:
hasMeasurementDifference  ["FORWARD"  | "REVERSE"]
attribute [string]
difference_mrid [string]
object [string]
value [number]
```

Please find some sample requests with various query filters

```
{"queryMeasurement": "simulation",
"queryFilter": {"simulation_id": "582881157", "hasSimulationMessageType": "INPUT"},
"responseFormat": "JSON"}

{"queryMeasurement": "simulation",
"queryFilter": {"simulation_id": "582881157", "angle": 23.706919634782313},
"responseFormat": "JSON"}

{"queryMeasurement":"simulation",
"queryFilter":{"simulation_id":"1743450224",
"measurement_mrid":["_01625641-d9ae-4c34-8302-69a9620ec69d","_ffd6abc7-159d-4f6d-868b-
↪7bf7b087ab85"]},
"responseFormat":"JSON"}
```

### 3.13.4 Query Sensor Service Data

Returns output of sensor sensor service.

Example Request:

```
{"queryMeasurement": "gridappsd-sensor-simulator",
"queryFilter": {"simulation_id": "582881157"},
"responseFormat": "JSON"}
```

Example Response for result format JSON:

```
{
        "data": {
        "measurements": [
                {
                "name": "gridappsd-sensor-simulator",
                "points": [
                        {
                                "row": {
                                "entry": [
                                        {
                                                "key": "instance_id",
                                                "value": "gridappsd-sensor-simulator-
↪1564186315783"
                                        },
                                        {
                                                "key": "hasSimulationMessageType",
                                                "value": "OUTPUT"
                                        },
                                        {
                                                "key": "measurement_mrid",
                                                "value": "_0009caa4-23ef-41b9-9db7-
↪624f3f47460c"
                                        },
                                        {
                                                "key": "angle",
                                                "value": "-152.44531328865978"
                                        },
                                        {
                                                "key": "magnitude",
                                                "value": "2470.4939175057075"
                                        },
                                        {
                                                "key": "simulation_id",
                                                "value": "1512566584"
                                        },
                                        {
                                                "key": "time",
                                                "value": "1564186297"
                                        }
                                        ]
                                        }
                        },.........]}]
        },
        "responseComplete": true,
        "id": "597021681"
}
```


Allowed values for queryFilter are:

```
starttime [number]
endtime [number]
measurement_mrid [string]
simulation_id [string]
instance_id
angle [number]
magnitude [number]
value [number]
```

### 3.13.5 Query Historical Sensor Service Data

A docker image is available that containes the pre-populated historical data generated from sensor service for 9500 node model. It containes data with following details:

- Start time: Saturday, July 13, 2013 8:00:02 AM (1373727602)
- End time: Friday, July 19, 2013 3:12:23 PM (1374271943).
- Simulation Id: 890162203

Follow these steps to access historical data:

1. Clone gridappsd-docker repo
2. Change the influxdb image in docker-compose.yml file to gridappsd/influxdb:historical
3. Execure ./run.sh -t <release tag>
4. Use Query Sensor Service Data API to request data.

## 3.14 Services

Sensor Simulator Service The *GridAPPSD's Sensor Simulator* simulates real devices based upon the magnitude of "prestine" simulated values. Currently the sensor service supports angle and magnitude measurements.

'_GridAPPS-D DNP3 Service <https://gridappsd-dnp3.readthedocs.io/en/develop/>'__ The *GridAPPS-D DNP3 Service* integrates GridAPPS-D with DNP3 based commercial tools to enable the CIM and DNP3 data exchange of the devices.

## 3.15 Hosting Application

### 3.15.1 Supported Application or Service Types

- Python
- EXE

### 3.15.2 Hosting Application

Developers can create application using GridAPPS-D API and use following instruction to host it with the platform. Applications run in their own Docker contianer. For example of an application working with GridAPPS-D, please see: https://github.com/GRIDAPPSD/gridappsd-sample-app.

**1. Create proper folder structure for the application**

Following is the recommended structure for applications working with GridAPPS-D using gridappsd-sample-app as an example:

```
.
└── gridappsd-sample_app
    ├── sample-app
    │   ├── [application exe or pythod code]
    ├── requirements.txt
    ├── sample_app.config
```

(continues on next page)

(continued from previous page)

```
    ├── Dockerfile
    └── setup.py
```

Where,

- **gridappsd-sample-app** is a folder and name of the application.
- **sample-app** is a folder that contains the application's source/build code.
- **requirements.txt** is required for Python based application and lists all the pre-requisite packages.
- **sample_app.config** is a file used by GridAPPS-D to launch the application from inside application container. More details are provided in Step 2.
- **Dockerfile** contains all the commands to assemble a Docker image for the application. More details are provided in Step 3.
- **setup.py** is build script file for python based applications.

**2. Create config file for application**

Config file is used by GridAPPS-D platform to register and launch the application. Here is the config file example using gridappsd-sample-app:

```
{
    "id":"sample-app",
    "description":"GridAPPS-D Sample Application app",
    "creator":"PNNL",
    "inputs":[],
    "outputs":[],
    "options": ["(simulationId)"],
    "type":"PYTHON",
    "execution_path": "sample_app/runsample.py",
    "launch_on_startup":false,
    "prereqs":["gridappsd-state-estimator"],
    "multiple_instances":false
}
```

Where,

- **id** is the name of the application and should match the name of the config file.
- **description** is a string that describes the application.
- **creator** is the organization/developer name
- **inputs** is the list of input topics application is listening to. *For future use. Leave it as in the example for now*.
- **outputs** is the list of input topics application is publishing to. *For future use. Leave it as in the example for now*.
- **options** are the run time arguments required by the application. Available options are: (i) simulationId: Unique identifier for the simulation, (ii) request: Simulation request sent by the user.
- **type** defines the type fo the application which can be PYTHON or EXE.
- **execution_path** is the path of the main file that starts the application relative to the top-most folder. In this example it would be the path relative to gridappsd-sample-app folder.
- **launch_on_startup** is true if application needs to be started as the platform starts and false if application needs to be started with a simulation.

- **prereqs** is list of GridAPPS-D services that need to be started before starting the application. Leave it as empty list [] if no such services are required. Services such as FNCS and FNCS-GOSS-BRIDGE are started by default with a simuolation so not needed to be specified here.
- **multiple_instances** is true if multiple instances of this application can be started for a single simulation otherwise false.

**3. Create Dockerfile for application**

Copy Dockerfile from https://github.com/GRIDAPPSD/gridappsd-sample-app/blob/master/Dockerfile. In the file replace gridappsd-sample with your applcation name and sample_app.config with the name of the config file of your application. You can add more commands in the file if needed for your application.

**4. Build the Docker container for application**

```
docker build --network=host -t sample-app .
```

Where,

- **sample-app** is the image name. Change it to your application name.

**5. Clone gridappsd-docker repository**

Clone this repository outside any application folder.

```
git clone https://github.com/GRIDAPPSD/gridappsd-docker.git
```

**6. Add application to platform**

In order to add your application to the GridAPPS-D platform you will need to modify the docker-compose.yml file included in the gridappsd-docker repository. Add the following to the file:

```
sample_app:
image: sample_app
environment:
  GRIDAPPSD_URI: tcp://gridappsd:61613
depends_on:
  - gridappsd
```

Use image name from step 4 instead of sample_app in line 1 and 2.

**7. Start platform and application container**

```
cd gridappsd-docker
./run.sh
```

This script starts application container along with platform. Application container has built-in code that allows application to register with GridAPPS-D platform when it starts.

**8. Verify that application container is running**

Use following command to list all Docker container which should include application container with *running* status.

```
docker ps -a
```

Optional - You can go inside the application container to check its content.

```
docker exec -it sample_app bash
```

Where,

- **sample_app** is the name of the container. Replace it with your application container name.

Execute *exit* to get out of the application container.

**9. Varify that application is hosted correctly**

- Go to http://localhost:8080
- Login with default user credetials already provided in login screen.
- Press Menu on the top-left corner
- Press *Configure New Simulation* menu item
- Go to *Application Configuration* tab
- Look for the application name in the drop down box.

If your application is available in that drop down box then application is hosted correctly with the platform.

For next step see documentation under *Using GridAPPS-D –> Start a Simulation*

### 3.15.3 Hosting Service

Developers can create a platform service using GridAPPS-D API and use following instruction to host it with the platform. For example of an service working with GridAPPS-D, please see: https://github.com/GRIDAPPSD/gridappsd-state-estimator.

**1. Create proper folder structure for the service.**

Following is the recommended structure for services working with gridappsd using gridappsd-state-estimator as an example:

```
        .
        └── gridappsd-state-estimator
            ├── state-estimator
            │   ├── [service exe or pythod code]
            ├── requirements.txt
            ├── state-estimator.config
            └── setup.py

- **gridappsd-state-estimator** is a folder and name of the service.
- **state-estimator** is a folder that contains the service's source/build code.
- **requirements.txt** is required for Python based service and lists all the pre-
↪requisite packages.
- **state-estimator.config** is a file used by GridAPPS-D to launch the service from␣
↪inside GridAPPS-D container. More details are provided in Step 2.
- **setup.py** is build script file for python based applications.
```

**2. Create config file for service**

Config file is used by GridAPPS-D platform to register and launch the service. Here is the config file example using gridappsd-state-estimator:

**::**

{ “id”:”state-estimator”, “description”:”State Estimator”, “creator”:”PNNL”, “inputs”:[“/topic/goss.gridappsd.fncs.output”,”/topic/goss.gridappsd.se.input”], “outputs”:[“/topic/goss.gridappsd.se.requests”,”/topic/goss.gridappsd.se.system_state”], “static_args”:[“(simulationId)”,”(request)”], “execution_path”:”services/gridappsd-state-estimator/state-estimator/bin/state-estimator”, “type”:”EXE”, “launch_on_startup”:false, “prereqs”:[], “multiple_instances”:true, “environmentVariables”:[], “user_input”: {

**"use-sensors-for-estimates": {** "help": "Use measurements from the sensor-simulator service, if the sensor-simulator is configured, to generate state estimates rather than using simulation measurements", "help_example": false, "default_value": true, "type": "bool"

}

}

}

Where,

- **id** is the name of the service and should match the name of the config file.
- **description** is a string that describes the service.
- **creator** is the organization/developer name
- **inputs** is the list of input topics service is listening to. *For future use. Leave it as in the example for now*.
- **outputs** is the list of input topics service is publishing to. *For future use. Leave it as in the example for now*.
- **options** are the run time arguments required by the service. Available options are: (i) simulationId: Unique identifier for the simulation, (ii) request: Simulation request sent by the user.
- **type** defines the type fo the service which can be PYTHON or EXE.
- **execution_path is the path of the main file that starts the service relative to the top-most folder. In this example it would be the path relative to gridappsd-sample-app folder.**
- **launch_on_startup** is true if service needs to be started as the platform starts and false if service needs to be started with a simulation.
- **prereqs** is list of GridAPPS-D services that need to be started before starting the service. Leave it as empty list [] if no such services are required. Services such as FNCS and FNCS-GOSS-BRIDGE are started by default with a simuolation so not needed to be specified here.
- **multiple_instances** is true if multiple instances of this service can be started for a single simulation otherwise false.

Example config for service:

```
{
        "id":"state-estimator",
        "description":"State Estimator",
        "creator":"PNNL",
        "inputs":["/topic/goss.gridappsd.fncs.output","/topic/goss.gridappsd.se.input
↪"],
        "outputs":["/topic/goss.gridappsd.se.requests","/topic/goss.gridappsd.se.
↪system_state"],
        "static_args":["(simulationId)"],
        "execution_path":"service/bin/state-estimator.out",
        "type":"EXE",
        "launch_on_startup":false,
        "prereqs":[],
        "multiple_instances":true,
        "environmentVariables":[]
}
```

2. Clone the repository https://github.com/GRIDAPPSD/gridappsd-docker (refered to as gridappsd-docker repository) next to this repository (they should both have the same parent folder)

```
.
├── gridappsd-docker
└── gridappsd-sample-app
```

3. Add service or service to platform

In order to add your service/service to the container you will need to modify the docker-compose.yml file included in the gridappsd-docker repository. Under the gridappsd service there is an example volumes leaf that is commented out. Uncomment and modify these lines to add the path for your service and config file. Adding these lines will mount the service/service on the container's filesystem when the container is started.

For service:

```
#    volumes:
#      - ~/git/gridappsd-sample-app/sample_app:/gridappsd/services/sample_app
#      - ~/git/gridappsd-sample-app/sample_app/sample_app.config:/gridappsd/services/
↪sample_app.config

    volumes:
      - ~/git/[my_app_directory]/[my_app]:/gridappsd/services/[my_app]
      - ~/git/[my_app_directory]/[my_app]/[my_app.config]:/gridappsd/services/[my_app.
↪config]
```

For service:

```
#    volumes:
#      - ~/git/gridappsd-sample-app/sample_app:/gridappsd/services/sample_app
#      - ~/git/gridappsd-sample-app/sample_app/sample_app.config:/gridappsd/services/
↪sample_app.config

    volumes:
      - ~/git/[my_service_directory]/[my_service]:/gridappsd/services/[my_service]
      - ~/git/[my_service_directory]/[my_service]/[my_service.config]:/gridappsd/
↪services/[my_service.config]
```

### 3.15.4 How to start a service

*Note: This process will be simplified in future releases so user could start a service through API and UI for a simulation with or without an service.*

Currently a service will be started by the platform only if it is a requirement for an service as described in the service config file under prereqs key. By default gridappsd-sensor-service and gridappsd-voltage-violation services are available in GridAPPS-D docker container.

In order to start a service with an service (sample app in this example) follow these steps:

1. Go into sample app container by executing

```
docker exec -it gridappsddocker_sample_app_1 bash
```

2. Inside sample app container execute following commands

```
apt-get update

apt-get install vim
```

4. Edit sample_app.config and add service id to the prereqs as shown below:

```
"prereqs":["gridappsd-sensor-simulator"]
```

*Note: Service id should match the value of "id" in service config file.*

5. Exit sample app container

6. Restart sample app docker container by executing

```
docker restart gridappsddocker_sample_app_1
```

7. Go into GridAPPS-D docker container by executing

```
docker exec -it gridappsddocker_gridappsd_1 bash
```

8. Start platform by executing :

   ./run-gridappsd.sh

Now when you start a simulation with sample app the service defined in prereqs will start as well.

CHAPTER 4

# System Configurations

**TODO** default values and description (conf under GOSS-GridAPPS-D, FNCS ports and simulator count)

CHAPTER 5

# GridAPPS-D Development Resources

This section is useful for developers for understanding or changing platform's internal workings and for those wishing to develop their own applications for GridAPPS-D. For developing application for GridAPPS-D platform see *Using GridAPPS-D* .

## 5.1 Design

Design section desribes the conceptual design of GridAPPS-D's managers. Each manager is resposible for executing specific groupd of functions.

### 5.1.1 Process Manager

Process Manager is responsible for starting, managing and stopping processes. All the requests to start a process like starting a simulation or querying a data store is received by Process Mananger. After receiving a request to start a process it forwards the request to corresponding manager for execution. Process Manager keeps track of all the processes and reports their status when requested.

This implements Internal Function Definition- 402 Process Manager

### 5.1.2 Log Manager

TBD

### 5.1.3 Simulation Manager

TBD

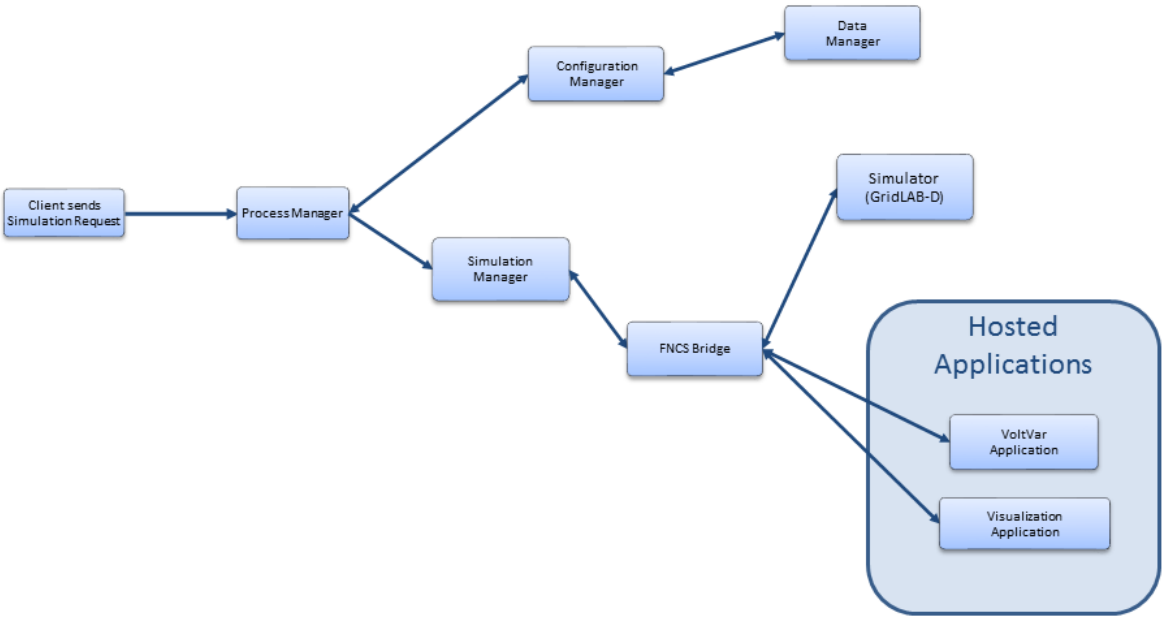
### 5.1.4 Configuration Manager

TBD

## 5.2 Eclipse IDE Setup

1. **Download or clone the repository from github**
    a. Install github desktop https://desktop.github.com/ or sourcetree https://www.atlassian.com/software/sourcetree/overview and Clone the GOSS-GridAPPS-D repository (https://github.com/GRIDAPPSD/GOSS-GridAPPS-D)
    b. Or download the source (https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/archive/master.zip)
2. Install java 1.8 SDK and set JAVA_HOME variable
3. Install Eclipse http://www.eclipse.org/downloads/packages/release/Mars/1 (Mars 4.5.1 or earlier, 4.5.2 appears to have bugs related to bundle processing) TODO what about neon?
4. Open eclipse with workspace set to GOSS-GridAPPS-D download location, eg. C:UsersusernameDocumentsGOSS-GridAPPS-D
5. Install BNDTools plugin: Help->Install New Software->Work with: http://dl.bintray.com/bndtools/bndtools/3.0.0 and Install Bndtools 3.0.0 or earlier
6. **Import projects into workspace**
    a. File->Import General->Existing Projects into workspace
    b. Select root directory, GOSS-GridAPPS-D download location
    c. Select cnf, pnnl.goss.gridappsd
7. If errors are detected, Right click on the pnnl.goss.gridappsd project and select release, then release all bundles
8. **If you would like to you a local version of GOSS-Core (Optional)**
    a. Update cnf/ext/repositories.bnd
    b. Select source view and add the following as the first line
    c. aQute.bnd.deployer.repository.LocalIndexedRepo;name=GOSS Local Release;local=/GOSS-Core2/cnf/releaserepo;pretty=true,
    d. verify by switching to bndtools and verify that there are packages under GOSS Local Relase
9. Open pnnl.goss.gridappsd/bnd.bnd, Rebuild project, you should not have errors
10. Open pnnl.goss.gridappsd/run.bnd.bndrun and click Run OSGI

## 5.3 Execution Workflow




Process Manager - The workflow begins when a simulation request is sent to the request topic monitored by the Process Manager, the process manager gathers the necessary configurations from the Configuration Manager. Then sends the configuration to the simulation manager to run the simulation.

Configuration Manager - The configuration manager parses the request and builds the necessary configuration files. It also uses the data manager to pull the model data from the CIM database.

Data Manager - The data manager accesses the CIM database to build the model files used by the simulator.

Simulation Manager - The simulation manager launches the simulator and other required applications such as the FNCS bridge, FNCS, and the VoltVar application. It is in charge of managing the timing of the simulation and reporting output from the simulation out to the simulation status topic.

FNCS Bridge - Serves as input and output from the simulator to the rest of GridAPPS-D, receives initialization, timestep, update, and finalize requests from the simulation manager and other applications, such as voltvar. It also publishes output from the simulator on a pre-defined topic for the simulation manager and other applications to subscribe to.

Simulator - In this case GridLAB-D serves as the simulator.

Hosted Application - Applications can be developed to use the data generated by the simulation and submit feedback and updates to the simulator. Two examples of this have been developed in RC1, the VoltVar application and a vizualization application

Log Manager - Process Manager recieves a log message. It retrieves the username associated with the message and forwards the message and username to Log Manager. Log Manager writes the message on a file and if store_to_db key is true in log message then log manager calls the data manager to store the log message in the database.

## 5.4 CIM Documentation

This section summarizes the use of a reduced-order CIM[1] to support feeder modeling for the North American circuits and use cases considered in GridAPPS-D. The full CIM includes over 1100 tables in SQL, each one corresponding to a UML class, enumeration or datatype. RC1 used approximately 100 such entities, mapped onto 100+ tables in SQL. Subsequent versions of GridAPPS-D use a triple-store database, which is better suited for CIM [ref].

The CIM subset described here is based on iec61970cim17v23a_iec61968cim13v11, which formed the basis of a release candidate for CIM100. This candidate CIM100 already includes changes proposed from the GridAPPS-D project.

### 5.4.1 Class Diagrams for the Profile

Figure 1 through Figure 17 present the UML class diagrams generated from Enterprise Architect[2]. These diagrams provide an essential roadmap for understanding:

1. How to ingest CIM XML from various sources into the database
2. How to generate native GridLAB-D input files from the database

For those unfamiliar with UML class diagrams:

1. Lines with an arrowhead indicate class inheritance. For example, in Figure 1, ACLineSegment inherits from Conductor, ConductingEquipment, Equipment and then PowerSystemResource. ACLineSegment inherits all attributes and associations from its ancestors (e.g., length), in addition to its own attributes and ancestors.
2. Lines with a diamond indicate composition. For example, in Figure 1, Substations make up a SubGeographical-Region, which then make up a GeographicRegion.
3. Lines without a terminating symbol are associations. For example, in Figure 1, ACLineSegment has (through inheritance) a BaseVoltage, Location and one or more Terminals.
4. Italicized names at the top of each class indicate the ancestor (aka superclass), in cases where the ancestor does not appear on the diagram. For example, in Figure 1, PowerSystemResource inherits from IdentifiedObject.

Please see *GridAPPSD_RC4.eap*[3] in the repository[4] on GitHub for the latest updates. The EnterpriseArchitect file includes a description of each class, attribute and association. It can also generate HTML documentation of the CIM, with more detail than provided here.

The diagrammed UML associations have a role and cardinality at each end, source and target. In practice, *only one end of each association* is profiled and implemented in SQL. In some cases, the figure captions indicate which end, but see the CIM profile for specific definitions, as described in the object diagram section. Usually, the end with 0..1 multiplicity is implemented instead of the end with 0..* or 1..* multiplicity.

Nearly every CIM class inherits from IdentifiedObject, from which we use two attributes:

1. mRID is the "master identifier" that must be unique and persistent among all instances. It's often used as the RDF resource identifier, and is often a GUID.

[1] See http://cimug.ucaiug.org/default.aspx and the EPRI CIM Primer at: http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000003002006001

[2] Suggest "Corporate Edition" from http://www.sparxsystems.com/ for working with CIM UML. The free CIMTool is still available at http://wiki.cimtool.org/index.h tml, but support is being phased out.

[3] OSPRREYS is an older name for GridAPPS-D

[4] https://github.com/GRIDAPPSD/Powergrid-Models/CIM

2. Name is a human-readable identifier that need not be unique.

Figure 1 shows how equipment is organized into a Feeder. Each piece of Equipment has Terminals that connect at ConnectivityNodes, which represent the "buses". Please note that in GridAPPS-D, we do not use TopologicalNode at all. In some CIM use cases, primary for transmission, the TopologicalNode coalesces ConnectivityNodes that are connected by buswork and closed switches into a single connection point. Instead, substation buswork models need to explicitly include the low-impedance branches and switches between ConnectivityNodes. We assume that modern distribution power flow solvers have no need of TopologicalNode.

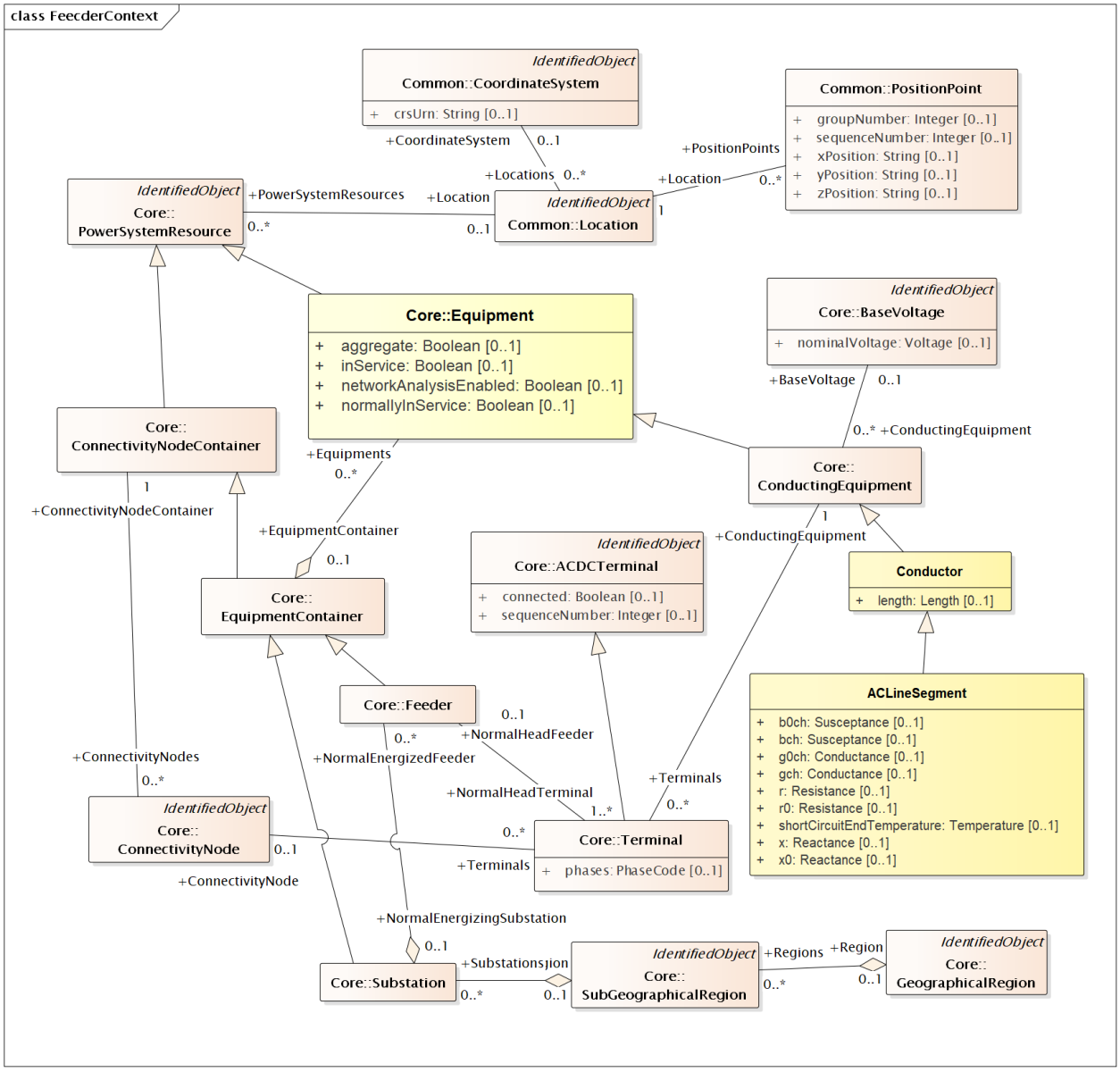



Figure 1: Placement of ACLineSegment into a Feeder. In GridAPPS-D, the Feeder is the EquipmentContainer for all power system components and the ConnectivityNodeContainer for all nodes. It's energized from a Substation, which is part of a SubGeographicalRegion and GeographicalRegion for proper context with other CIM models. For visualization, ACLineSegment can be drawn from a sequence of PositionPoints associated via Location. The Terminals are free-standing; two of them will "reverse-associate" to the ACLineSegment as ConductingEquipment, and each terminal also has one ConnectivityNode. The Terminal:phases attribute is not used; instead, phases will be defined in the ConductingEquipment instances. The associated BaseVoltage:nominalVoltage attribute is important for many of the classes that don't have their own rated voltage attributes, for example, EnergyConsumer.

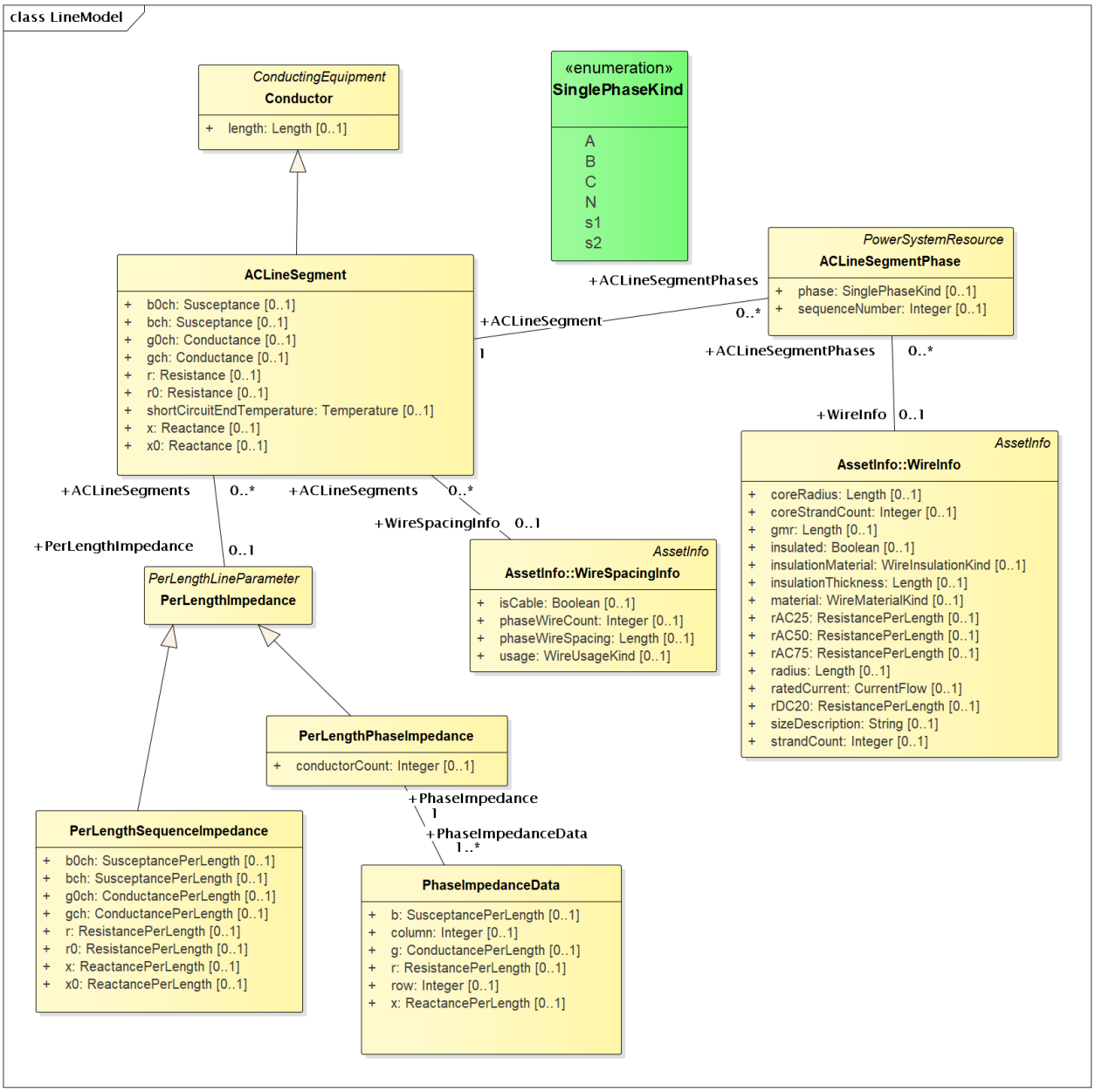



Figure 2: There are four different ways to specify ACLineSegment impedances. In all cases, Conductor:length is required. The first way is to specify the individual ACLineSegment attributes, which are sequence impedances and admittances, leaving PerLengthImpedance null. The second way is to specify the same attributes on an associated PerLengthSequenceImpedance, in which case the ACLineSegment attributes should be null. The third way is to associate a PerLengthPhaseImpedance, leaving the ACLineSegment attributes null. Only conductorCount from 1 to 3 is supported, and there will be 1, 3 or 6 reverse-associated PhaseImpedanceData instances that define the lower triangle of the Z and Y matrices per unit length. The row and column attributes must agree with ACLineSegment-Phase:sequenceNumber. The fourth way to specify impedance is by wire/cable and spacing data, by association to WireSpacingInfo and WireInfo. If there are ACLineSegmentPhase instances reverse-associated to the ACLineSegment, then per-phase modeling applies. There are several use cases for ACLineSegmentPhase: 1) single-phase or two-phase primary, 2) low-voltage secondary using phases s1 and s2, 3) associated WireInfo data where the WireSpacingInfo association exists, 4) assign specific phases to the matrix rows and columns in PerLengthPhaseImpedance. It is the application's responsibility to propagate phasing through terminals to other components, and to identify any miswiring. (Note: this profile does not use WireAssemblyInfo, nor the fromPhase and toPhase attributes of Pha-

seImpedanceData.)

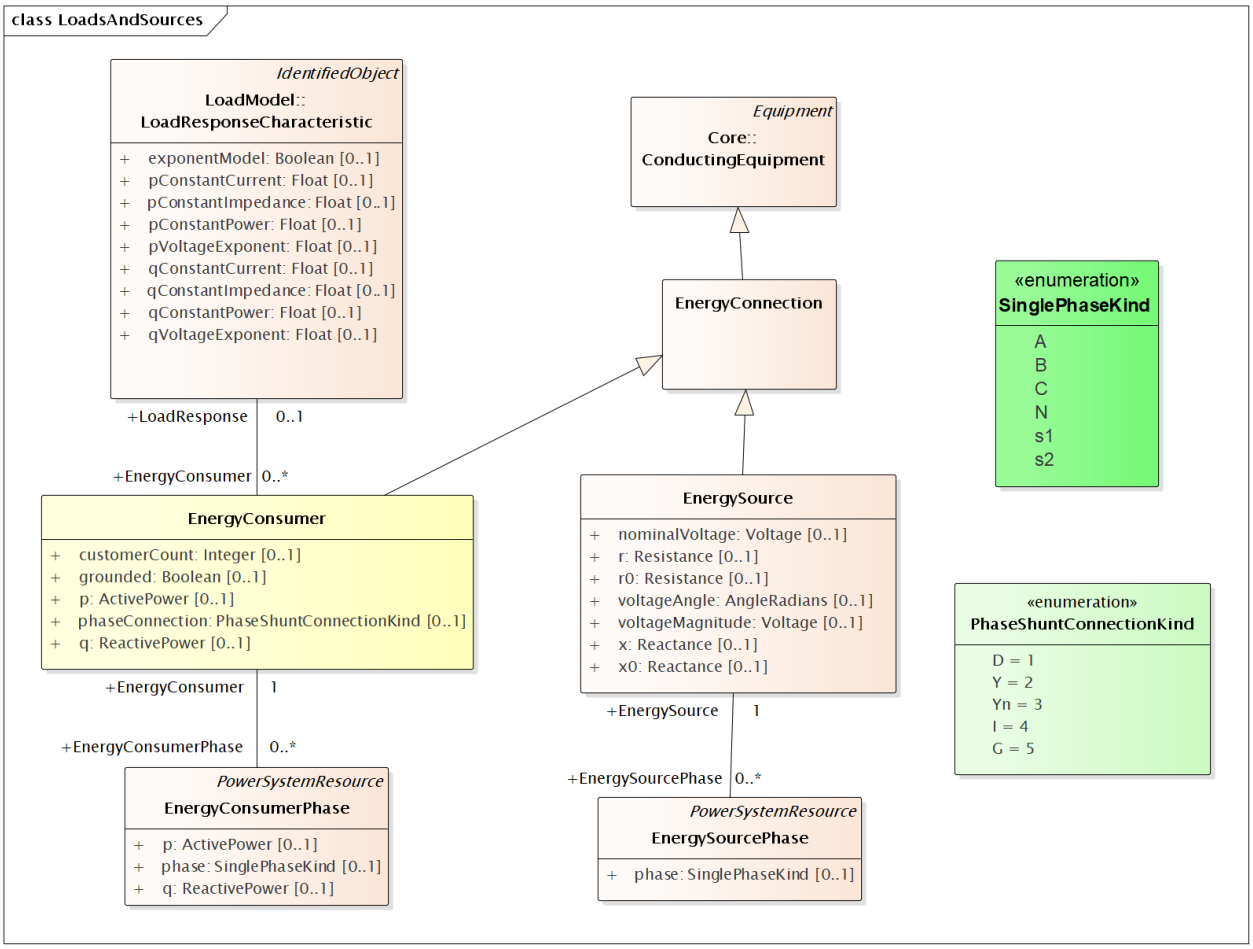



Figure 3: The EnergySource is balanced three-phase, balanced two-phase or single-phase Thevenin source. GridAPPS-D use a three-phase EnergySource to represent the transmission system. See Figures 13-14 for DER modeling. The EnergyConsumer is a ZIP load, possibly unbalanced, with an associated LoadResponse instance defining the ZIP coefficients. For three-phase delta loads, the phaseConnection is D and the three reverse-associated EnergyConsumerPhase instances will have phase=A for the AB load, phase=B for the BC load and phase=C for the AC load. A three-phase wye load may have either Y or Yn for the phaseConnection. Single-phase and two-phase loads, including secondary loads, should have phaseConnection=I (for individual).

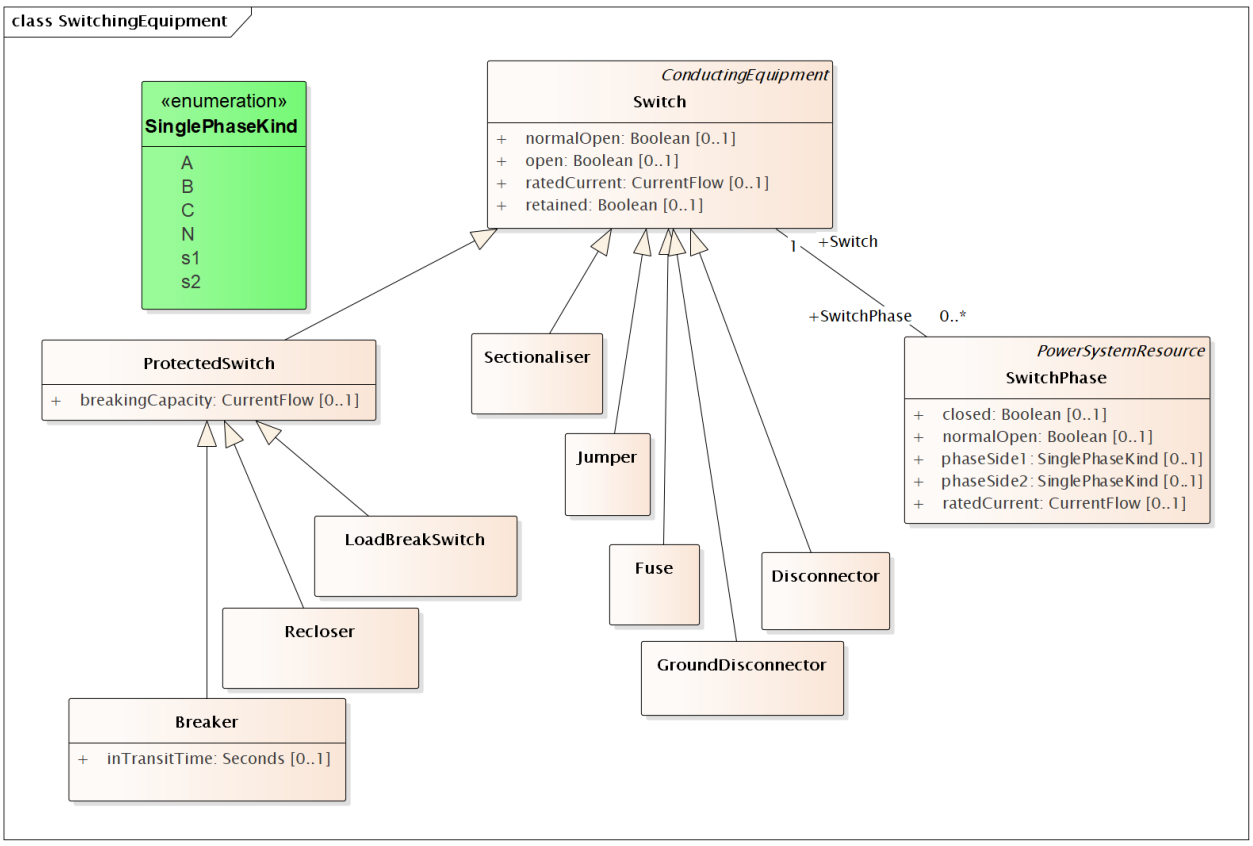



Figure 4: There are eight different kinds of Switch supported in the CIM, and all of them have zero impedance. They would all behave the same in power flow analysis, and all would require many more attributes than are defined in CIM to support protection analysis. The use cases for SwitchPhase include 1) single-phase, two-phase and secondary switches, 2) one or two conductors open in a three-phase switch or 3) transpositions, in which case phaseSide1 and phaseSide2 would be different. RatedCurrent may different among the phases, e.g., individual fuses on the same pole may have different ratings.

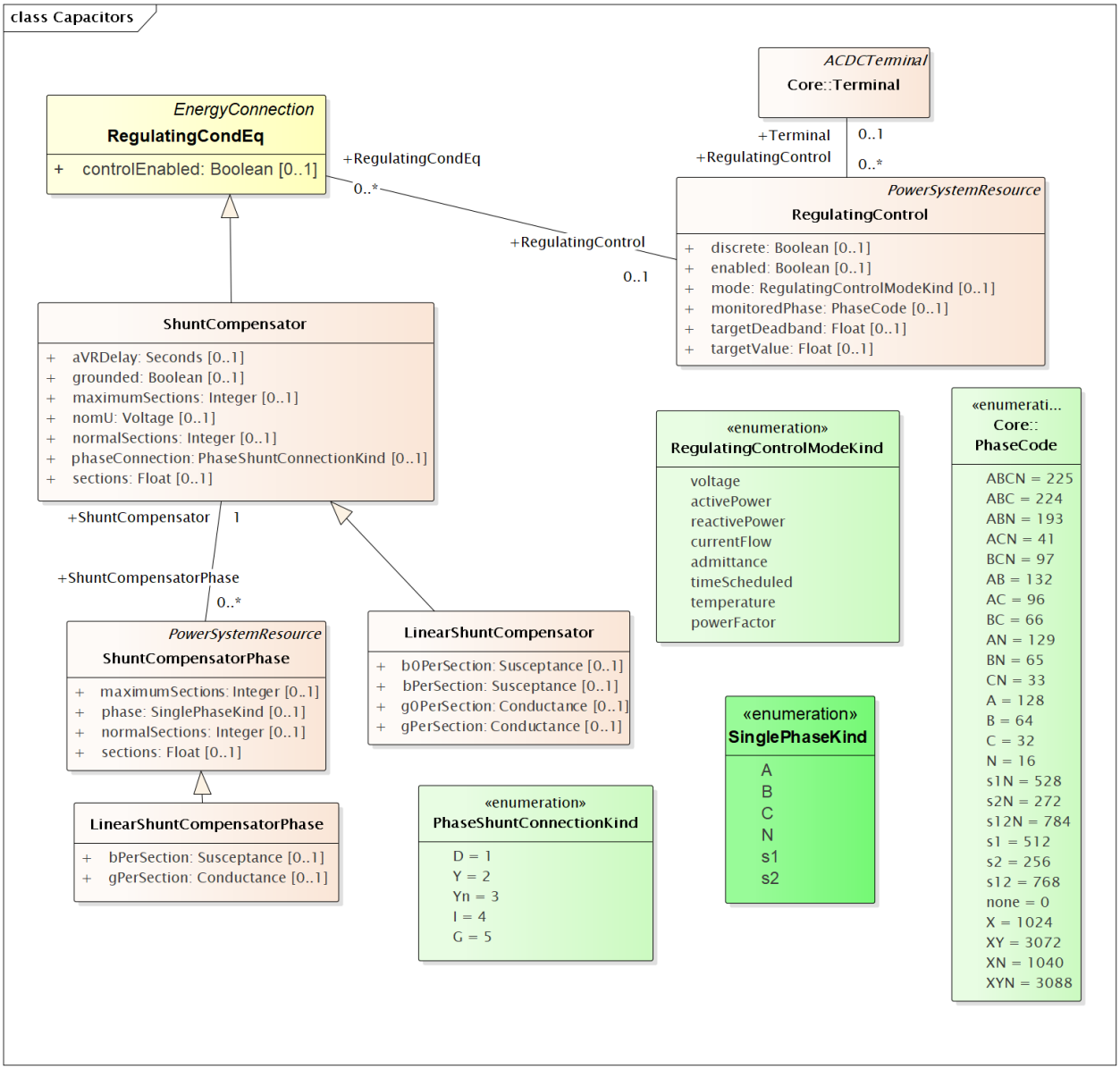



Figure 5: On the left, LinearShuntCompensator and LinearShuntCompensatorPhase define capacitor banks, in a way very similar to EnergyConsumer in Figure 3. The kVAR ratings must be converted to susceptance based on the nominal voltage, nomU. Note that aVRDelay is really a capacitor control parameter, to be used in conjunction with RegulatingControl on the right-hand side. The RegulatingControl associates to the controlled capacitor bank via RegulatingCondEq, and to the monitored location via Terminal. There is no support for a PT or CT ratio, so targetDeadband and targetValue have to be in primary volts, amps, vars, etc. Capacitor banks may _respond_ to any of the RegulatingControlModeKind choices, but it's not expected that capacitor switching will successfully regulate to the targetValue.

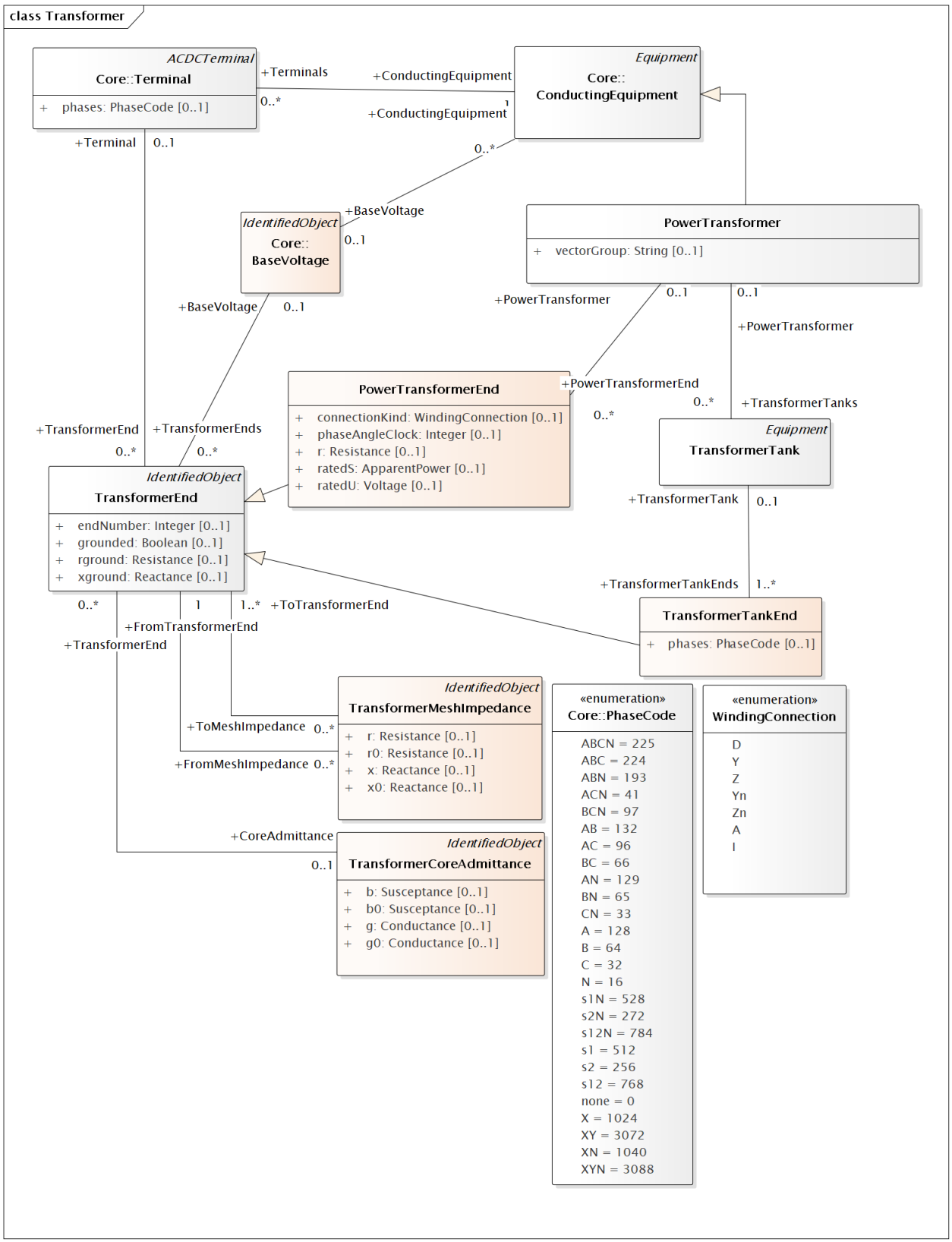



Figure 6: PowerTransformers may be modeled with or without tanks, and in both cases vectorGroup should be specified according to IEC transformer standards (e.g., Dy1 for many substation transformers). The case without tanks

is most suitable for balanced three-phase transformers that won't reference catalog data; any other case should use tank-level modeling. In the tankless case, each winding will have a PowerTransformerEnd that associates to both a Terminal and a BaseVoltage, and the parent PowerTransformer. The impedance and admittance parameters are defined by reverse-associated TransformerMeshImpedance between each pair of windings, and a reverse-associated TransformerCoreAdmittance for one winding. The units for these are ohms and siemens based on the winding voltage, rather than per-unit. WindingConnection is similar to PhaseShuntConnectionKind, adding Z and Zn for zig-zag connections and A for autotranformers. If the transformer is unbalanced in any way, then TransformerTankEnd is used instead of PowerTransformerEnd, and then one or more TransformerTanks may be used in the parent PowerTransformer. Some of the use cases are 1) center-tapped secondary, 2) open-delta and 3) EHV transformer banks. Tank-level modeling is also required if using catalog data, as described with Figure 9. (TransformerStarImpedance and several PowerTransformer attributes are not used. Star impedance attributes on PowerTransformerEnd and magnetic saturation attributes on TransformerEnd are not used.)

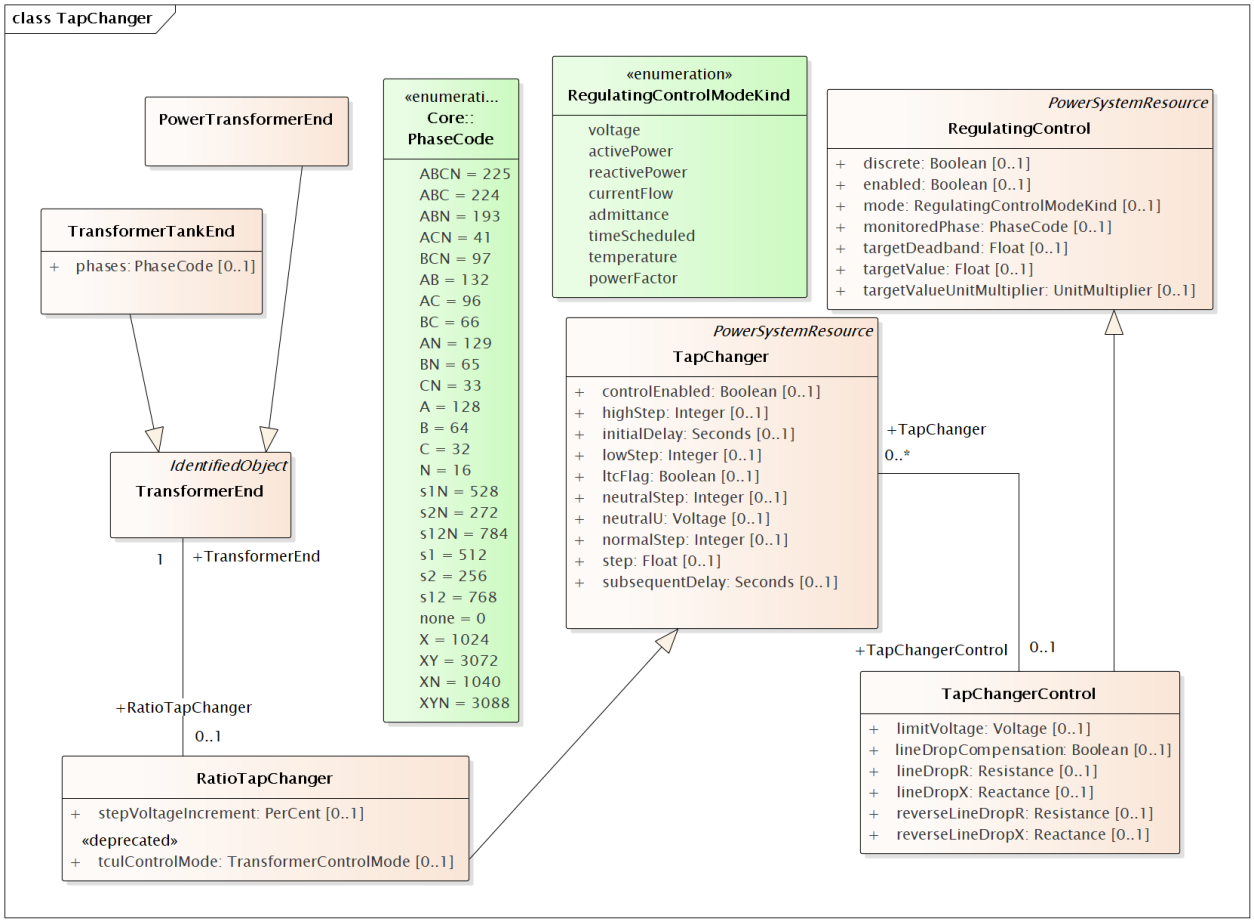



Figure 7: A RatioTapChanger can represent a transformer tap changer on the associated TransformerEnd. The RatioTapChanger has some parameters defined in a direct-associated TapChangerControl, which inherits from RegulatingControl some of the same attributes used in capacitor controls (Figure 5). Therefore, a line voltage regulator in CIM includes a PowerTransformer, a RatioTapChanger, and a TapChangerControl. The CT and PT parameters of a voltage regulator can only be described via the AssetInfo mechanism, described with Figure 8. The RegulationControl.mode must be voltage. (Note: RegulationSchedule, RatioTapChangerTable and PhaseTapChanger are not used.)

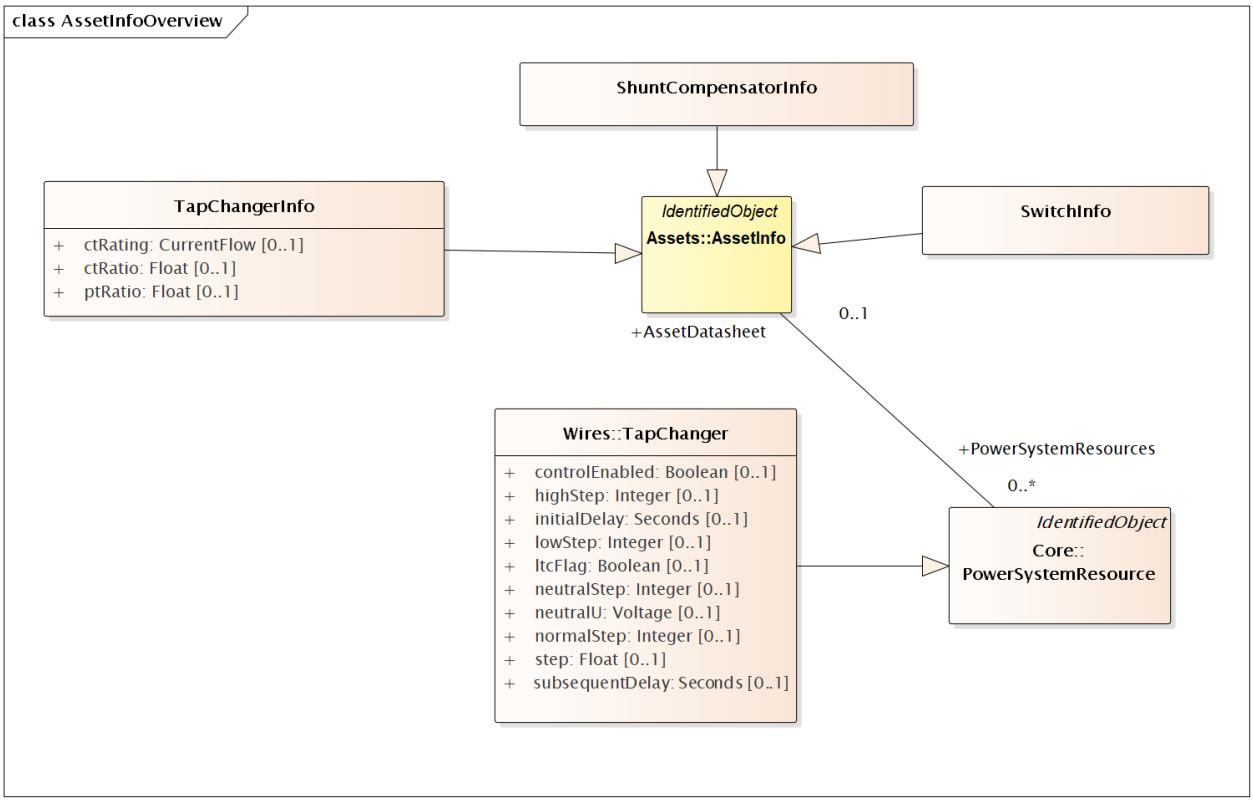



Figure 8: Many distribution software packages use the concept of catalog data, aka library data, especially for lines and transformers. We use the AssetInfo package to implement this in CIM. Here, the TapChangerInfo class includes the CT rating, CT ratio and PT ratio parameters needed for line drop compensator settings in voltage regulators. Catalog data is a one-to-many relationship. In this case, many TapChangers can share the same TapChangerInfo data, which saves space and provides consistency. Older versions of CIM had many-to-many catalog relationships, but now only one AssetDataSheet may be associated per Equipment. (Note: many datasheet attributes are not shown here and not yet used in GridAPPS-D).

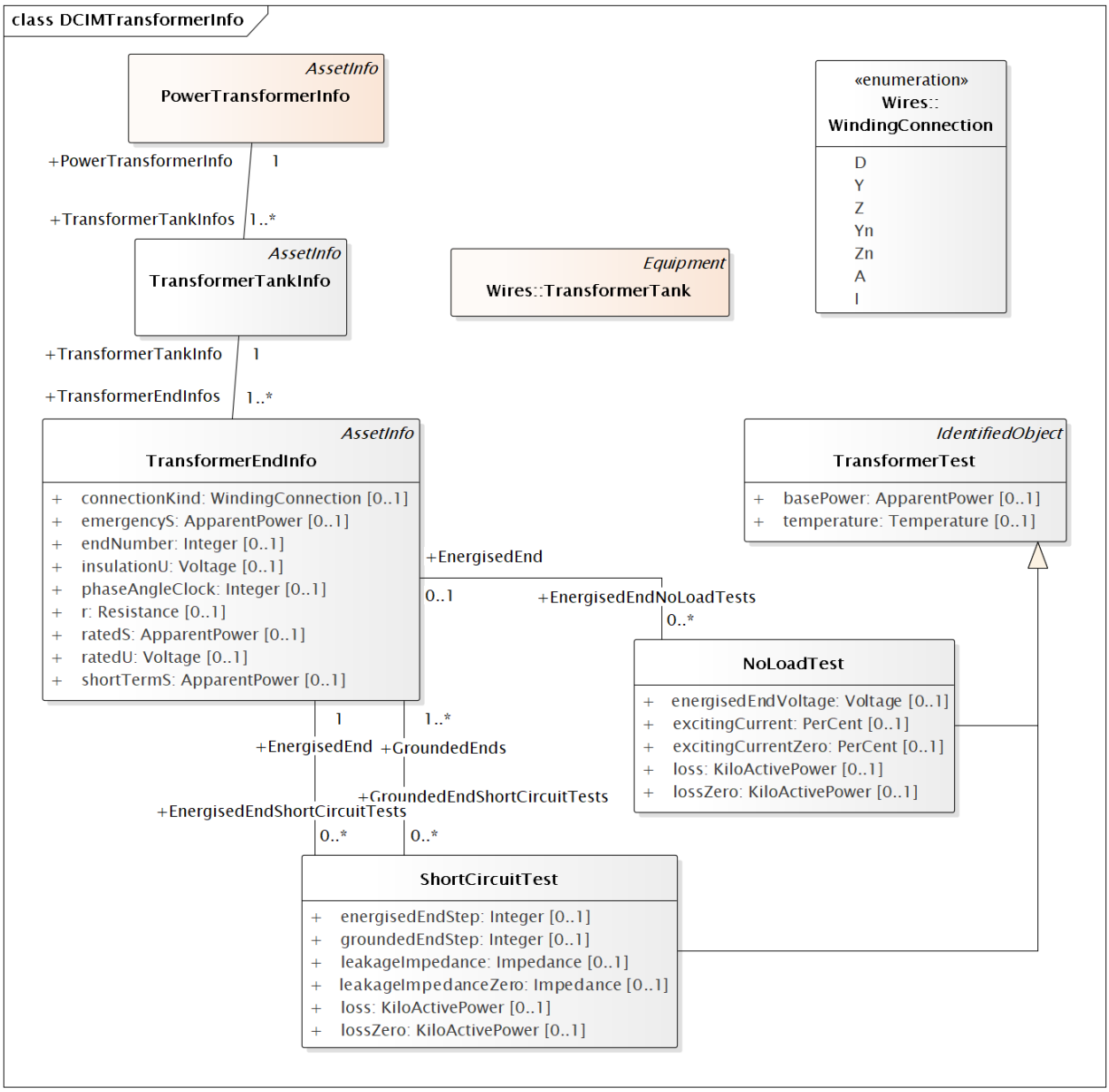



Figure 9: The catalog mechanism for transformers will associate a TransformerTank (Figure 6) with TransformerTankInfo (here), via the AssetDataSheet mechanism described in Figure 8. The PowerTransformerInfo collects TransformerTankInfo by reverse association, but it does not link with PowerTransformer. In other words, the physical tanks are cataloged because transformer testing is done on tanks. One possible use for PowerTransformerInfo is to help organize the catalog. It's important that TransformerEndInfo:endNumber (here) properly match the TransformerEnd:endNumber (Figure 6). The shunt admittances are defined by NoLoadTest on a winding / end, usually just one such test. The impedances are defined by a set of ShortCircuitTests; one winding / end will be energized, and one or more of the others will be grounded in these tests. (OpenCircuitTest is not used, nor are the current, power and voltage attributes of ShortCircuitTest).

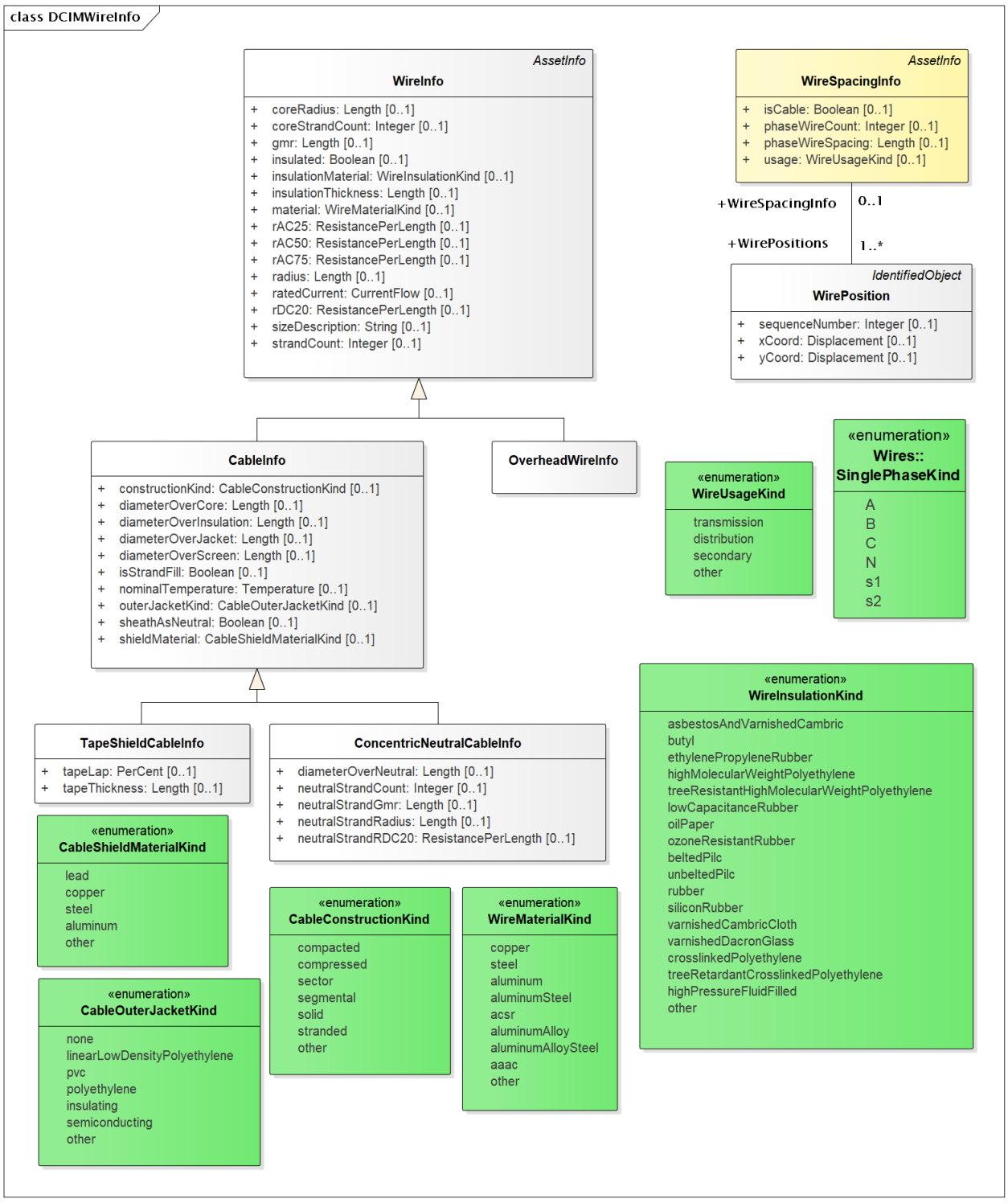



Figure 10: The catalog / library mechanism for ACLineSegment will have a WireSpacingInfo associated as in Figure 9. This will indicate whether the line is overhead or underground. phaseWireCount and phaseWireSpacing define optional bundling, so these will be 1 and 0 for distribution. The number of phase and neutral conductors is actually defined by the number of reverse-associated WirePosition instances. For example, a three-phase line with neutral would have four of them, sequenceNumber from 1 to 4. Each WirePosition's phase is determined by the ACLineSegment-Phase with matching sequenceNumber, i.e., the phases need not be numbered in any particular order. On the left-hand side, concrete classes OverheadWireInfo, TapeShieldCableInfo and ConcentricNeutralCableInfo may be associated to

ACLineSegmentPhase. It's the application's responsibility to calculate impedances from this data. In particular, soil resistivity and dielectric constants are not included in the CIM. Typical dielectric constant values might be defined for each WireInsulationKind.

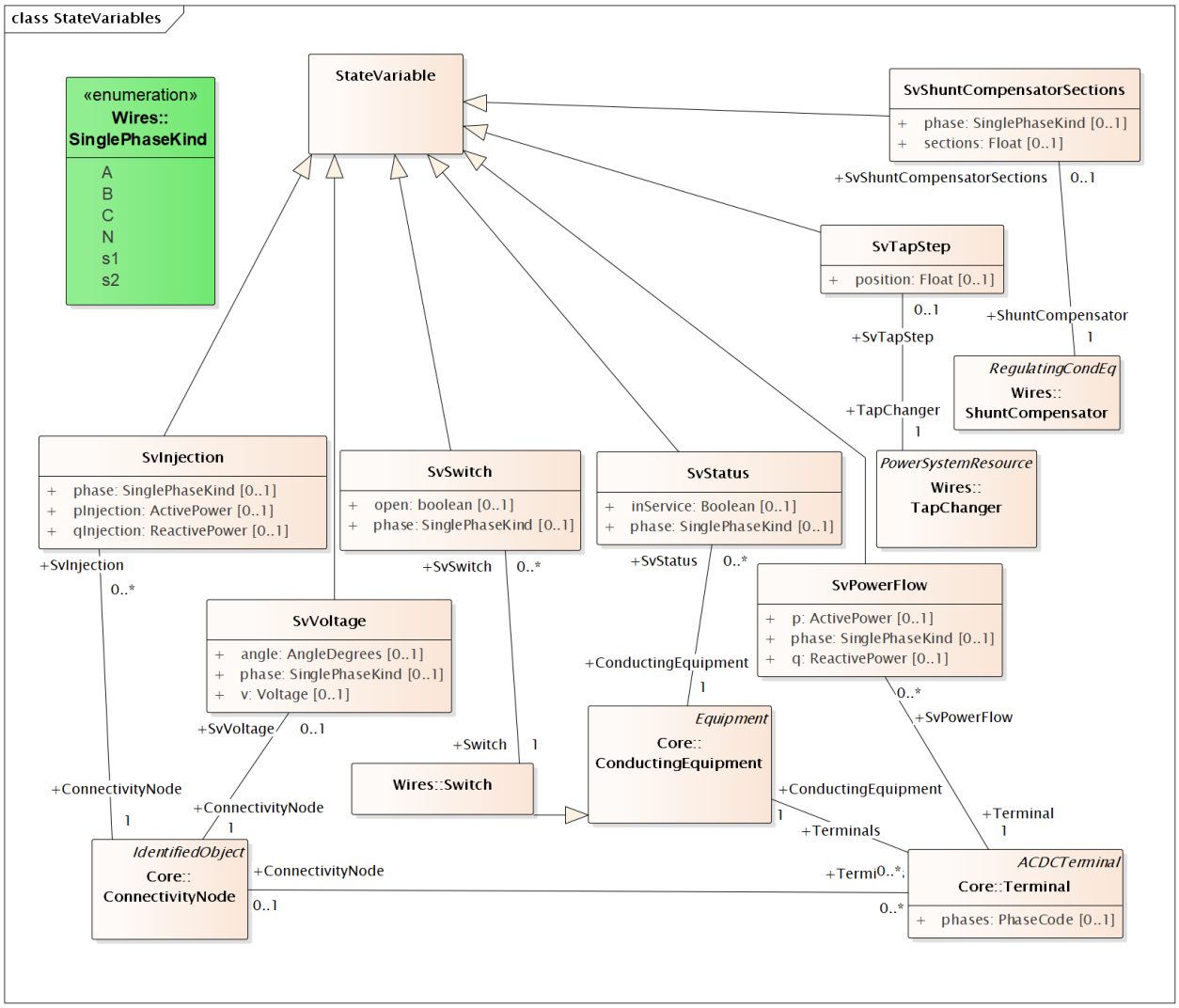



Figure 11: The CIM state variables package was designed to report power flow solution values on the distribution system. It could also report state estimator solutions as a special case of power flow solutions. Voltages are measured on ConnectivityNodes (i.e., not TopologicalNodes), power flows are measured at Terminals into the ConductingEquipment, step positions are measured on TapChangers, status is measured on ConductingEquipment, and on/off state is measured on ShuntCompensators for Switches. The "injections" have been included here, but there may not be a use case for them in distribution. On the other hand, solution values for current are very common in distribution system applications. These should be represented as SvPowerFlow values at the solved SvVoltage.

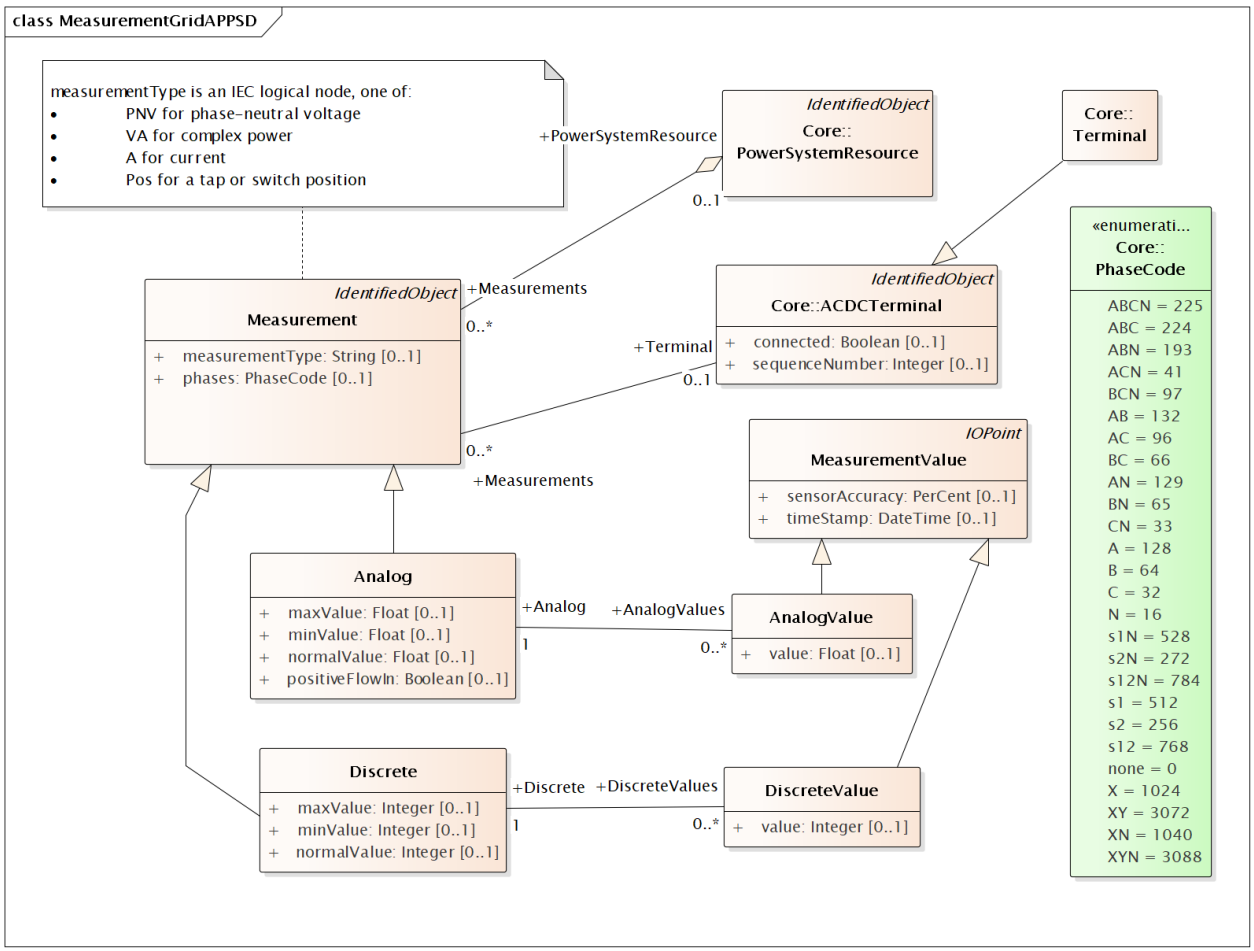



Figure 12: Measurements are defined in the Meas package. They differ from the state variables package, in that the values are measured here and not calculated or estimated. Each Measurement is associated to a PowerSystemResource, and in GridAPPS-D for now, also a Terminal that belongs to the same PowerSystemResource. (Non-electrical measurements, for example weather, would not have the Terminal). The measurementType is a string code from IEC 61850, with PNV, VA, A and POS currently supported. The Measurement has a name, mRID, and phases. In GridAPPS-D, each phase is measured individually so multi-phase codes like ABC should not be used. Pos measurements will be Discrete, for such things as tap position, switch position, or capacitor bank position. The others will be Analog, with magnitude and optional angle in degrees. Each MeasurementValue will have a timeStamp and mRID inherited from IdentifiedObject, so the values can be traced. (Note: IOPoint is a placeholder class with no attributes, inherting from IdentifiedObject. Further, it's acceptable to supply an empty or short non-unique name for each MeasurementValue.)

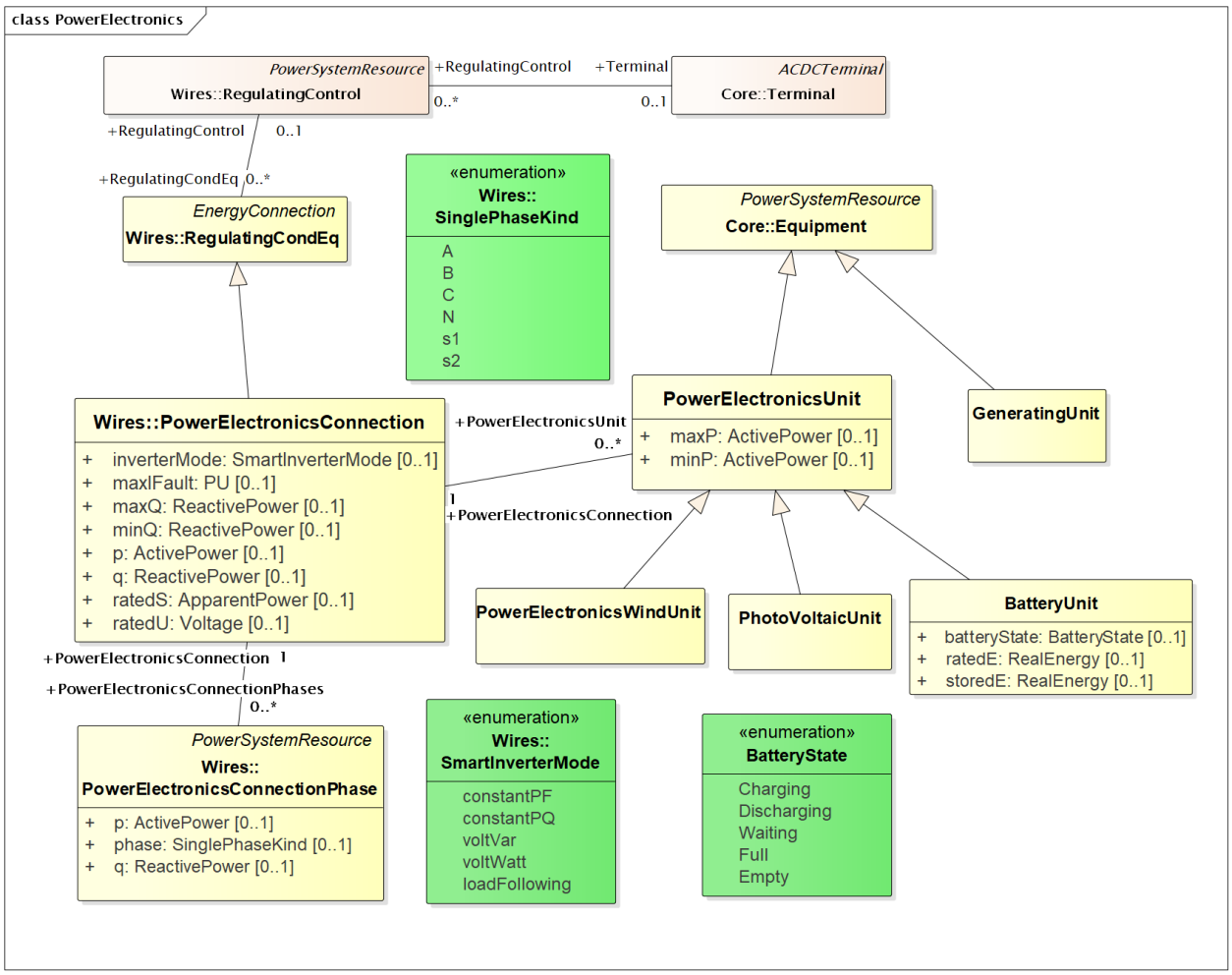



Figure 13: Power Electronics attributes are the minimum needed to support a time series power flow solution. For simple short-circuit calculations, maxIFault is provided as the inverter fault contribution in per-unit of rated current. When PowerElectronicsConnectionPhase is not present, the inverter is assumed to be balanced three-phase. The type of associated PowerElectronicsUnit determines whether the inverter is for solar or storage (wind is not currently used in GridAPPS-D). If the inverter employs a SmartInverterMode of voltVar, voltWatt or loadFollowing (storage only), then a Terminal should be associated through RegulatingControl, especially for loadFollowing. If the inverter will regulate its own Terminal, then the explicit Terminal association may not be needed. However, there are more attributes needed in CIM to define smart inverter functions. This might be done in harmonization with IEC 61850, which does define smart inverter function parameters. The existing CIM RegulatingControl attributes are probably not applicable, so they have been hidden in Figure 13.

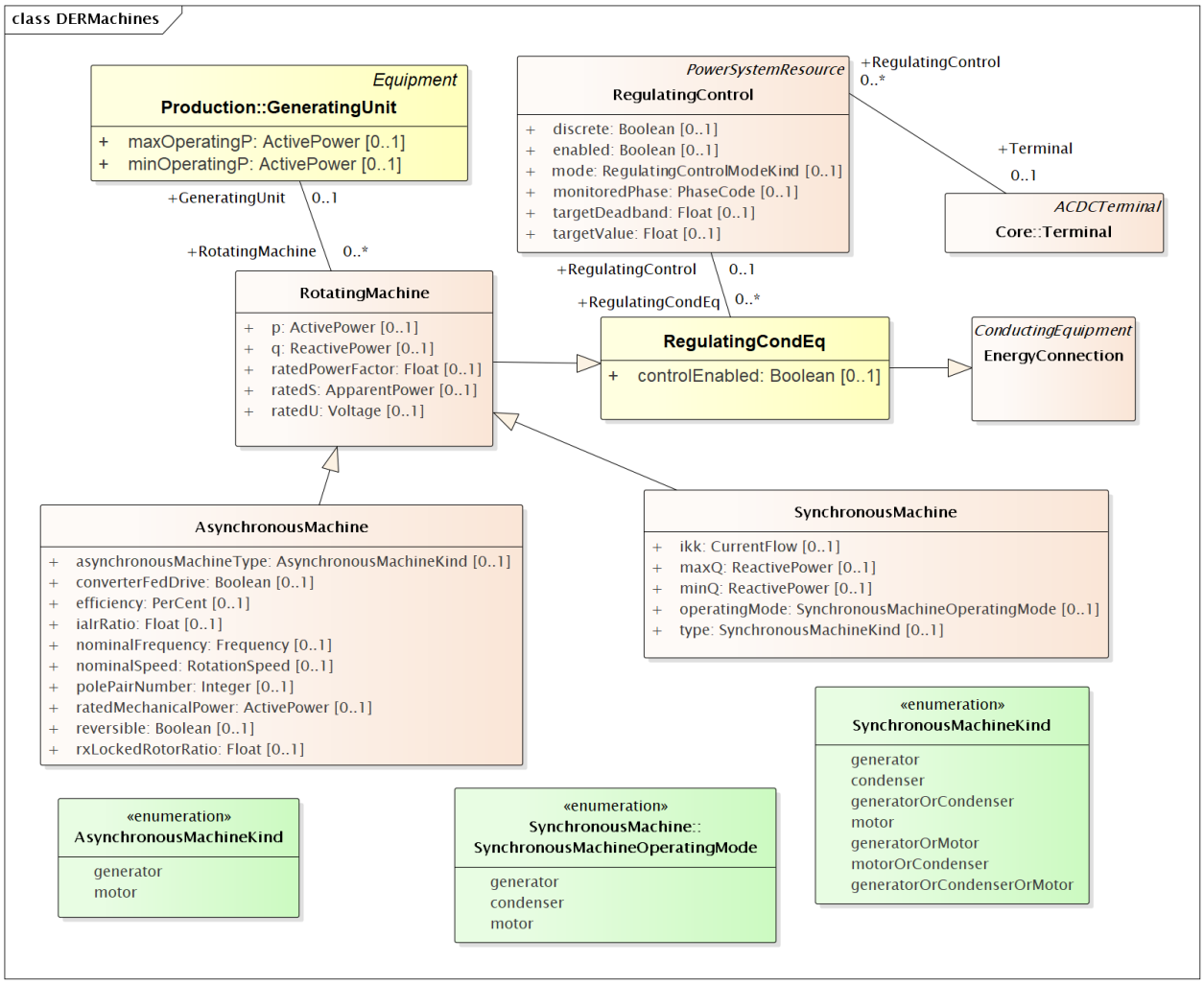



Figure 14: Rotating Machines are three-phase balanced, either synchronous or asynchronous. The SynchronousMachine ikk attribute and most of the AsynchronousMachine attributes are provided to support short-circuit calculations according to IEC 60909. The GeneratingUnit class is needed to define minimum and maximum power limits. In the full CIM, GeneratingUnit is an abstract class with descendants HydroUnit, ThermalUnit and NuclearUnit, but in GridAPPS-D we don't currently distinguish between those types. If the SynchronousMachine regulates voltage, then the RegulatingControl (with attribute values) and Terminal associations need to be provided.

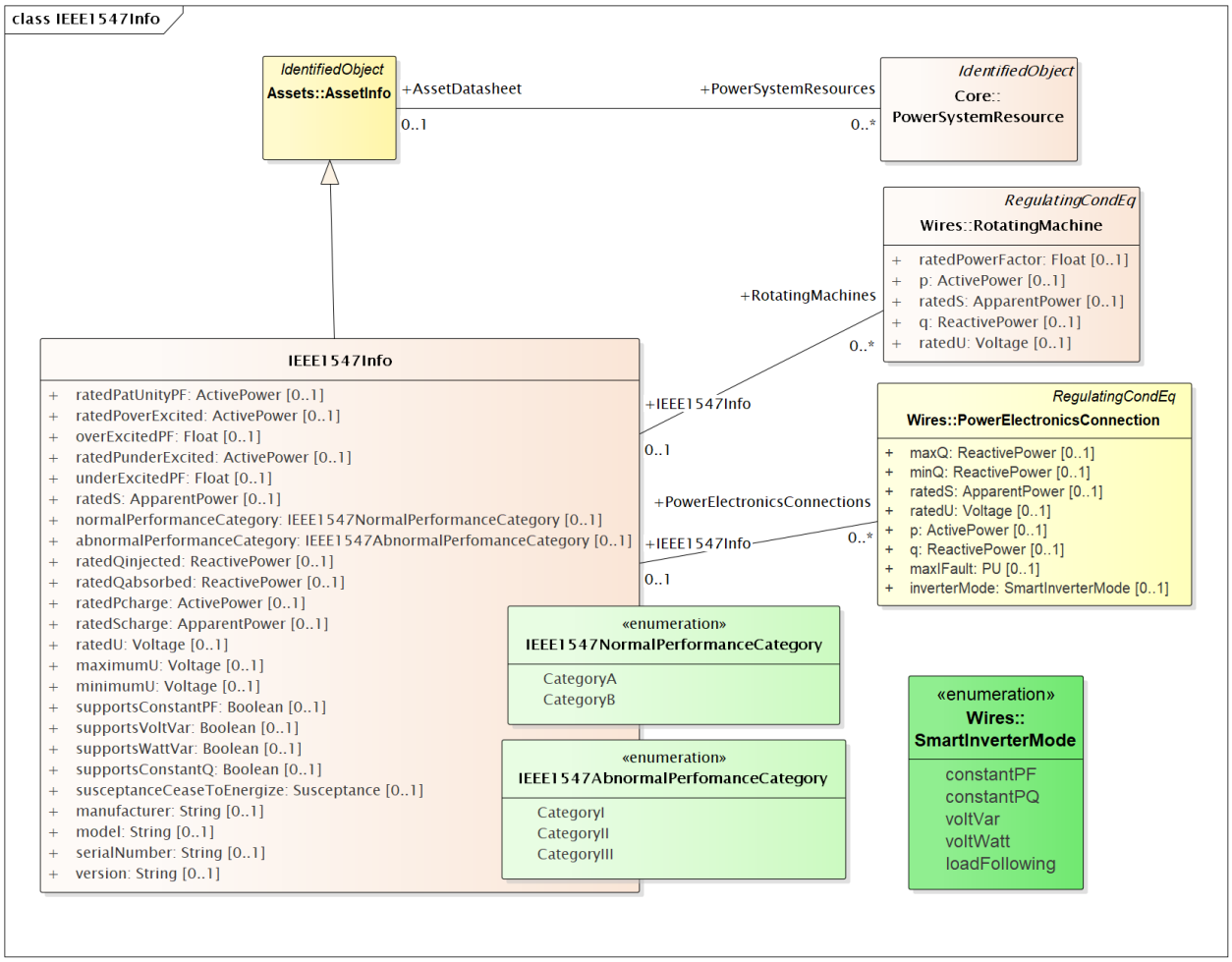



Figure 15: The IEEE1547Info class describes nameplate information, including ratings, which shall be available for DER that complies with IEEE Std. 1547-2018. Both inverters (Figure 13) and rotating machines (Figure 14) can reference this class for nameplate information in the network model. Preliminary values for these attributes would be available from an application to interconnect DER, and then updated as the project moves through commissioning to operational status.

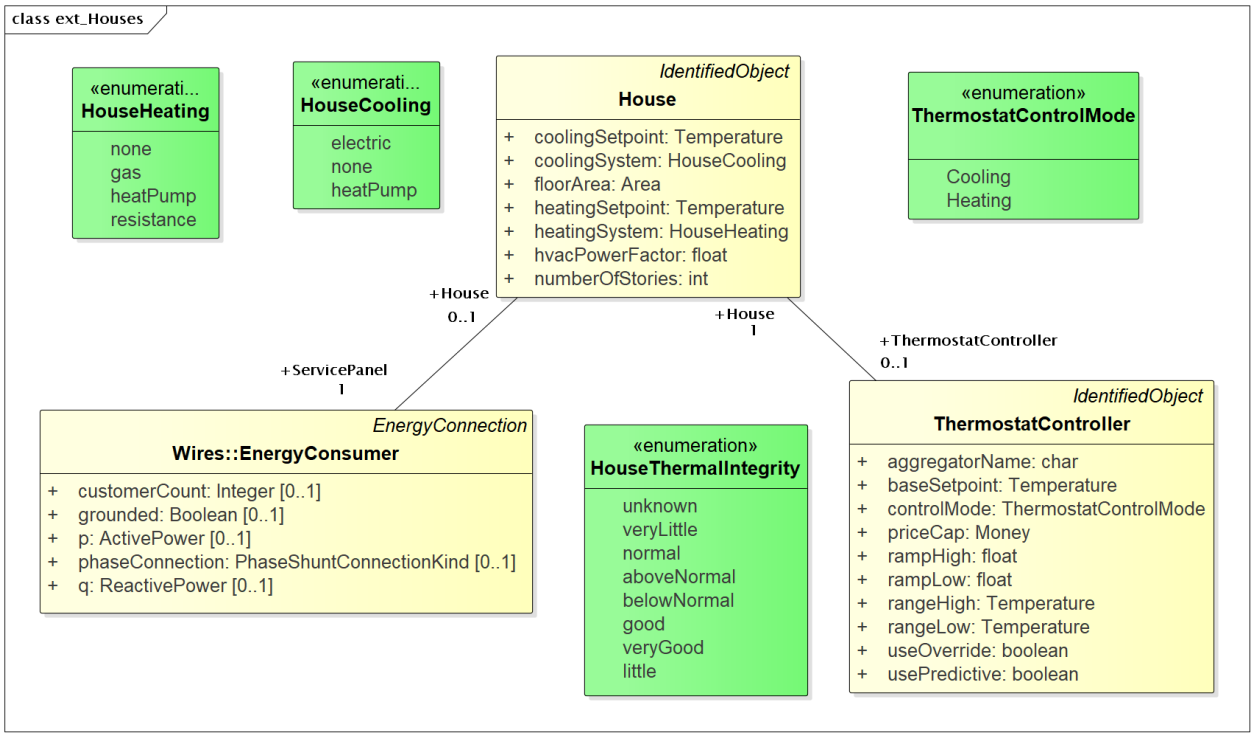



Figure 16: Houses are used to create 2nd-order thermal models of the building envelope, with internal ThermostatController and heating/cooling systems. The purpose is to introduce realistic load stochastic behaviors that are independent from and faster-moving than data typically available to an electric utility. To enable repeatable simulations, the House data structures have been defined here as a CIM extension. The House must be attached to one EnergyConsumer that incorporates other end-use loads, and connects to the distribution system. The House attributes are the minimum necessary to define a GridLAB-D house model, and during simulation, the house heating/cooling system will add to the ServicePanel loads. Therefore, the application should reduce the nominal value of EnergyConsumer.p in order to "make room" for the heating/cooling load that will switch on and off, responding to the ThermostatController and the weather. The ThermostatController contains the minimum attributes needed for PNNL's double-ramp, double-auction market mechanism. In the future, this will be harmonized with CIM market structures in the 62235 package.

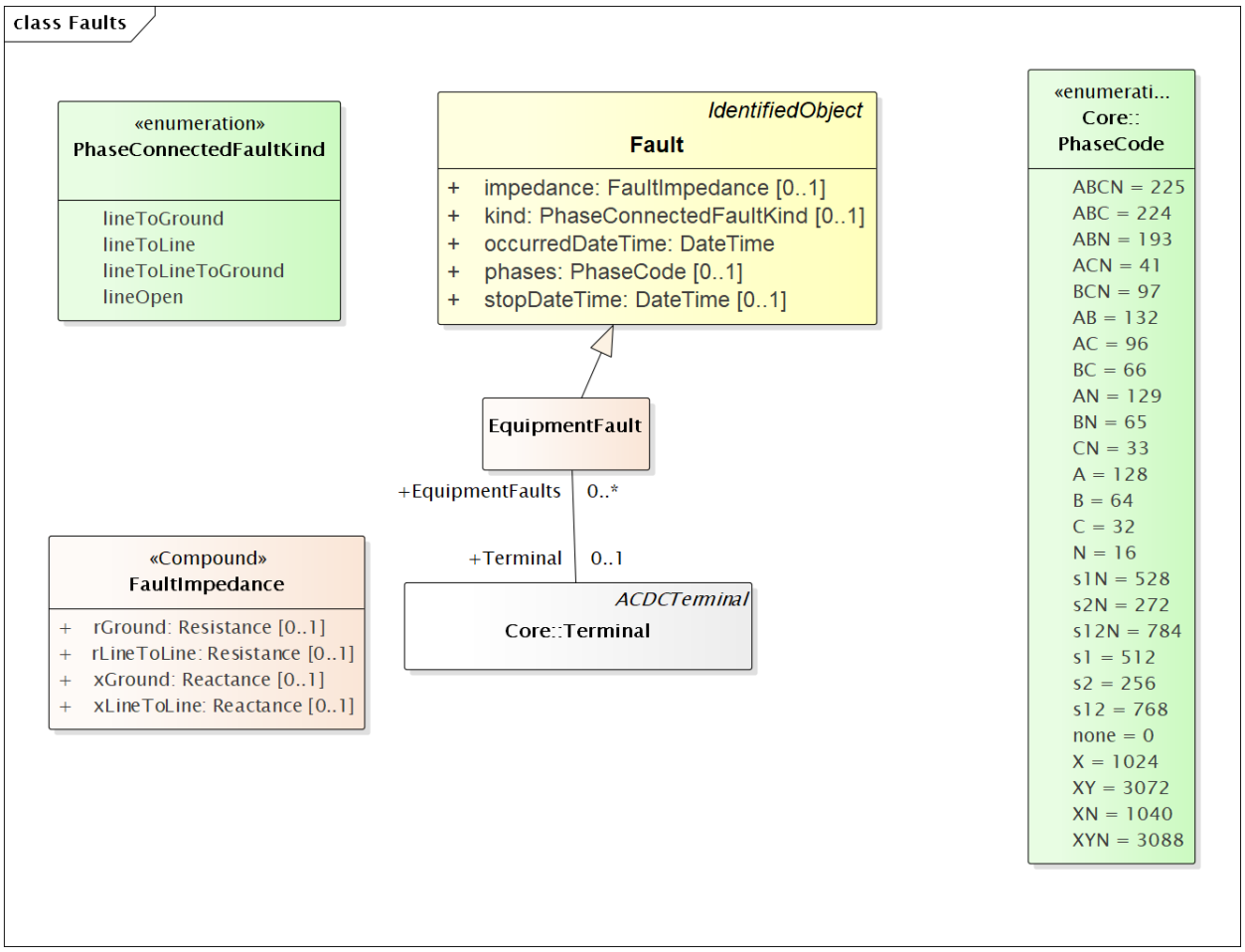



Figure 17: Faults include open conductors and short circuits (optionally including ground) on any combination of phases. In GridAPPS-D, every Fault will be an EquipmentFault associated to a Terminal (i.e., we are not using LineFault, which requires a lengthFromTerminal1 attribute). The occurredDateTime supports the scripting of fault sequences. The stopDateTime is optional. If provided, it will be the time at which a sustained fault has been repaired. If not provided, then the fault is temporary and will clear itself as soon as it's been deenergized.

### 5.4.2 Typical Queries

These queries focus on requirements of the first volt-var application.

1. Capacitors (Figure 5, Figure 18, Figure 19, Figure 20)
    a. Create a list of capacitors with bus name (Connectivity Node in Figure 1), kVAR per phase, control mode, target value and target deadband
    b. For a selected capacitor, update the control mode, target value, and target deadband
2. Regulators (Figure 7, Figure 8, Figure 18, Figure 35)
    a. List all transformers that have a tap changer attached, along with their bus names and kVA sizes
    b. Given a transformer that has a tap changer attached, list or update initialDelay, step, subsequentDelay, mode, targetDeadband, targetValue, limitVoltage, lineDropCompensation, lineDropR, lineDropX, reverseLineDropR and reverseLineDropX
3. Transformers (Figure 6, Figure 9)

a. Given a bus name or load (Figure 3), find the transformer serving it (Figure 22, Figure 25)

b. Find the substation transformer, defined as the largest transformer (by kVA size and or highest voltage rating)

c. List the transformer catalog (Figure 9, Figure 26) with name, highest ratedS, list of winding ratedU in descending order, vector group (https://en.wikipedia.org/wiki/Vector_group used with connectionKind and phaseAngleClock), and percent impedance

d. List the same information as in item c, but for transformers (Figure 6) and also retrieving their bus names. Note that a transformer can be defined in three ways

i. Without tanks, for three-phase, multi-winding, balanced transformers (Figure 22 and Figure 23).

ii. With tanks along with TransformerTankInfo (Figure 9) from a catalog of "transformer codes", which may describe balanced or unbalanced transformers. See Figure 25 and Figure 26.

iii. With tanks for unbalanced transformers, and TransformerTankInfo created on-the-fly. See Figure 25 and Figure 26.

e. Given a transformer (Figure 6), update it to use a different catalog entry (TransformerTankInfo in Figure 9)

4. Lines (Figure 2, Figure 10, Figure 18)

a. List the line and cable catalog entries that meet a minimum ratedCurrent and specific WireUsageKind. For cables, be able to specify tape shield vs. concentric neutral, the WireInsulationKind, and a minimum insulationThickness. (Figure 33)

b. Given a line segment (Figure 2) update to use a different linecode (Figure 10, Figure 32)

c. Given a bus name, list the ACLineSegments connected to the bus, along with the length, total r, total x, and phases used. There are four cases as noted in the caption of Figure 2, and see Figure 29 through Figure 32.

d. Given a bus name, list the set of ACLineSegments (or PowerTransformers and Switches) completing a path from it back to the EnergySource (Figure 3). Normally, the applications have to build a graph structure in memory to do this, so it would be very helpful if a graph/semantic database can do this.

5. Voltage and other measurements (Figure 1, Figure 11)

a. Given a bus, attach a voltage solution point (SvVoltage, Figure 36)

b. List all voltage solution points and their buses, and for each bus, list the phases actually present

c. For tap changer position (SvTapStep, Figure 37), attach and list values as in items a and b

d. For capacitor switch status (SvShuntCompensatorSections, Figure 38), attach and list values as in items a and b

6. Loads (Figure 3, Figure 34)

a. Given a bus name, list and total all of the loads connected by phase, showing the total p and q, and the composite ZIP coefficients

7. Switching (Figure 4, Figure 28)

a. Given a bus name, trace back to the EnergySource and list the switches encountered, grouped by type (i.e. the leaf class in Figure 4). Also include the ratedCurrent, breakingCapacity if applicable, and open/close status. If SwitchPhase is used, show the phasing on each side and the open/close status of each phase.

b. Given switch, toggle its open/close status.

### 5.4.3 Object Diagrams for Queries

This section contains UML object diagrams for the purpose of illustrating how to perform typical queries and updates. For those unfamiliar with UML object diagrams:

1. Each object will be an instance of a class, and more than one instance of a class can appear on the diagram. For example, Figure 18 shows two ConnectivityNode instances, one for each end of a ConductingEquipment.
2. The object name (if specified and important) appears before the colon (:) above the line, while the UML class appears after the colon. Every object in CIM will have a unique ID, and a name (not necessarily unique), even if not shown here.
3. Some objects may be shown with run-time state below the line. These are attribute value assignments, drawn from those available in the UML class or one of the class ancestors. The object may have more attribute assignments, but only those directly relevant to the figure captions are shown in the diagrams of this section.
4. Object associations are shown with solid lines, role names, and multiplicities similar to the UML class diagrams. One important difference is that only one way of navigating a particular association will be defined in the profile. For example, the lower left corner of Figure 1 shows a two-way link between Terminal and ConnectivityNode in the UML class diagram. However, Figure 18 shows that only one direction has been defined in the profile. Each Terminal has a direct reference to its corresponding ConnectivityNode. In order to navigate the reverse direction from ConnectivityNode to Terminal, some type of conditional query would be required. In other words, the object diagrams in this section indicate which associations can actually be used in GridAPPS-D.
5. In some cases, the multiplicities on the object diagrams are more restrictive than on the class diagrams, due to profiling. For example, EnergyConsumer and ShuntCompensator must have exactly one Terminal, not 1..*.

The object diagrams are intended to help you break down the CIM queries into common sub-tasks. For example, query #1 works with capacitors. It's always possible to select a capacitor (aka LinearShuntCompensator) by name. In order to find the capacitor at a bus, say "bus1" in Figure 12, one would retrieve all Terminals having a ConnectivityNode reference to "bus1". Each of those Terminals will have a ConductingEquipment reference, and you want the Terminal(s) for which that reference is actually a LinearShuntCompensator. In this CIM profile, only leaf classes (e.g. LinearShuntCompensator) will be instantiated, never base classes like ConductingEquipment. There can be more than one capacitor at a bus, more than one load, more than one line, etc.

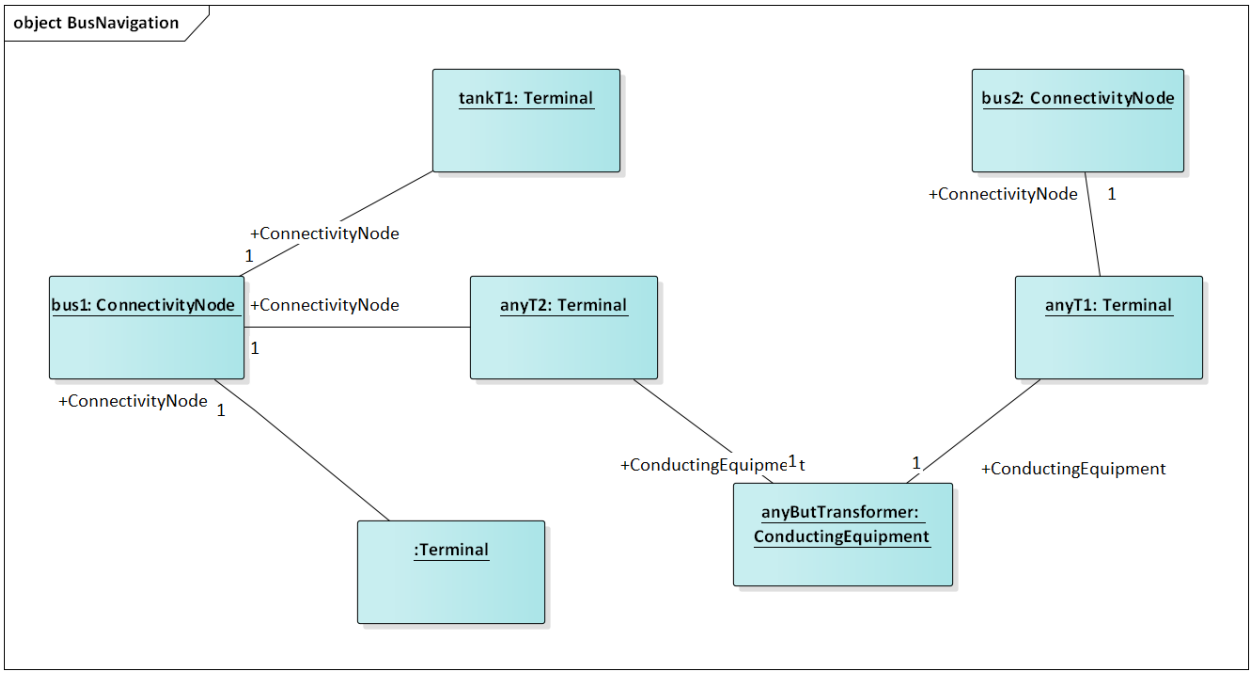



Figure 18: In order to traverse buses and components, begin with a ConnectivityNode (left). Collect all terminals

referencing that ConnectivityNode; each Terminal will have one-to-one association with ConductingEquipment, of which there are many subclasses. In this example, the ConductingEquipment has a second terminal referencing the ConnectivityNode called bus2. There are applications for both Depth-First Search (DFS) and Bread-First Search (BFS) traversals. Note 1: the Terminals have names, but these are not useful. In some cases, the Terminal sequenceNumber attribute is needed to clearly identify ends of a switch. Note 2: in earlier versions of GridAPPS-D, we had one-to-one association of TopologicalNode and ConnectivityNode, but these are no longer necessary. Note 3: transformers are subclasses of ConductingEquipment, but we traverse connectivity via transformer ends (aka windings). This is illustrated later.

In order to find capacitors (or anything else) associated with a particular "feeder", Figure 19 shows that you would query for objects having EquipmentContainer reference to the Feeder object. In GridAPPS-D, we only use Feeder for equipment container in CIM, and this would correspond to one entire GridLAB-D model. There is also a BaseVoltage reference that will have the system nominal voltage for the capacitor's location. However, in order to work with equipment ratings you should use ratedS and ratedU attributes where they exist, particularly for capacitors and transformers. These attributes are often slightly different than the "system voltage". Most of the attribute units in CIM are SI, with a few exceptions like percent and kW values on transformer test sheets (i.e., CIM represents the test sheet, not the equipment).

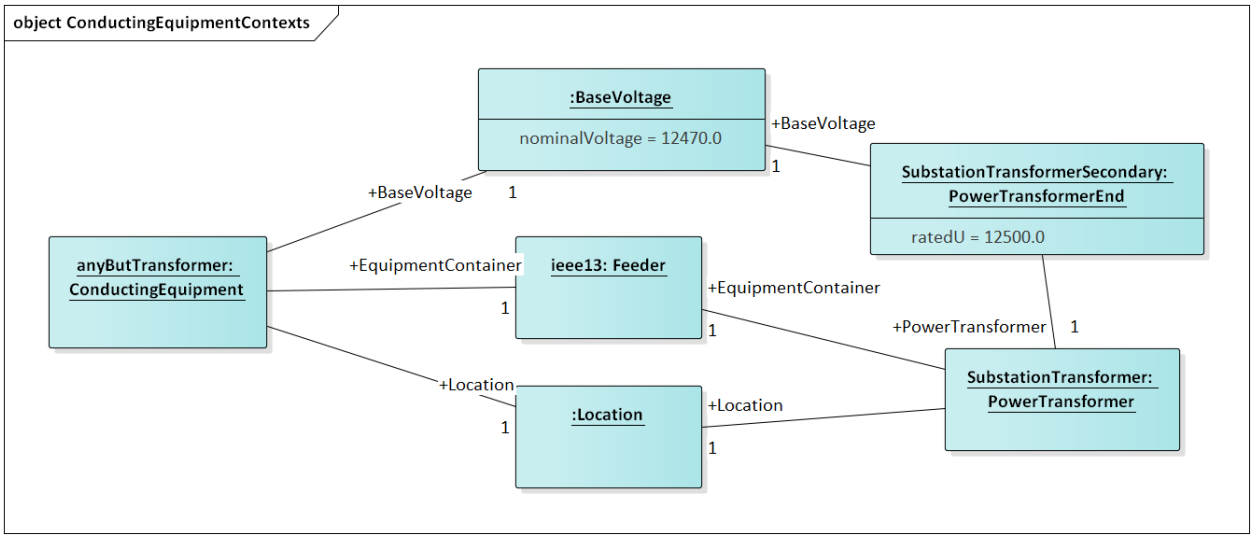



Figure 19: All conducting equipment lies within an EquipmentContainer, which in GridAPPS-D, will be a Feeder object named after the feeder. It also has reference to a BaseVoltage, which is typically one of the ANSI preferred system voltages. Power transformers are a little different, in that each winding (called "end" in CIM) has reference to a BaseVoltage. Note that equipment ratings come from the vendor, and in this case ratedU is slightly different from nominalVoltage. All conducting equipment has a Location, which contains XY coordinates (see Figure 1). The Location is useful for visualization, but is not essential for a power flow model.

Completing the discussion of capacitors, Figure 20 provides two examples for single-phase, and three-phase with local voltage control. As shunt elements, capacitors have only one Terminal instance. Loads and sources have one terminal, lines and switches have two terminals, and transformers have two or more terminals. Examples of all those are shown later. In Figure 20, the capacitor's kVAR rating will be based on its nameplate ratedU, not the system's nominalVoltage.

Often, the question will arise "what phases exist at this bus?".There is no phasing explicitly associated with a ConnectivityNode, and we don't use the Terminal phases attribute in preference to the "wires phase model" classes. For example, thephases at a line segment terminal can always be obtained from the ACLineSegmentPhase instances. To answer the question about bus phasing, we'd have to query for all ConductingEquipment instances having Terminals connected to that bus, as in Figure 18. The types of ConductingEquipment that may have individual phases include LinearShuntCompensators (Figure 20), ACLineSegments, PowerTransformers (via TransformerEnds), EnergyConsumers, EnergySources, PowerElectronicsConnections, and descendants of Switch. If the ConductingEquipment has

such individual phases, then add those phases to list of phases existing at the bus. If there are no individual phases, then ABC all exist at the bus. Note this doesn't guarantee that all wiring to the bus is correct; for example, you could still have a three-phase load served by only a two-phase line, which would be a modeling error. In Figure 20, we'd find phase C at Bus611 and phases ABC at Bus675. Elsewhere in the model, there should be ACLineSegments, PowerTransformers or Switch descendants delivering phase C to Bus611, all three phases ABC to Bus675.

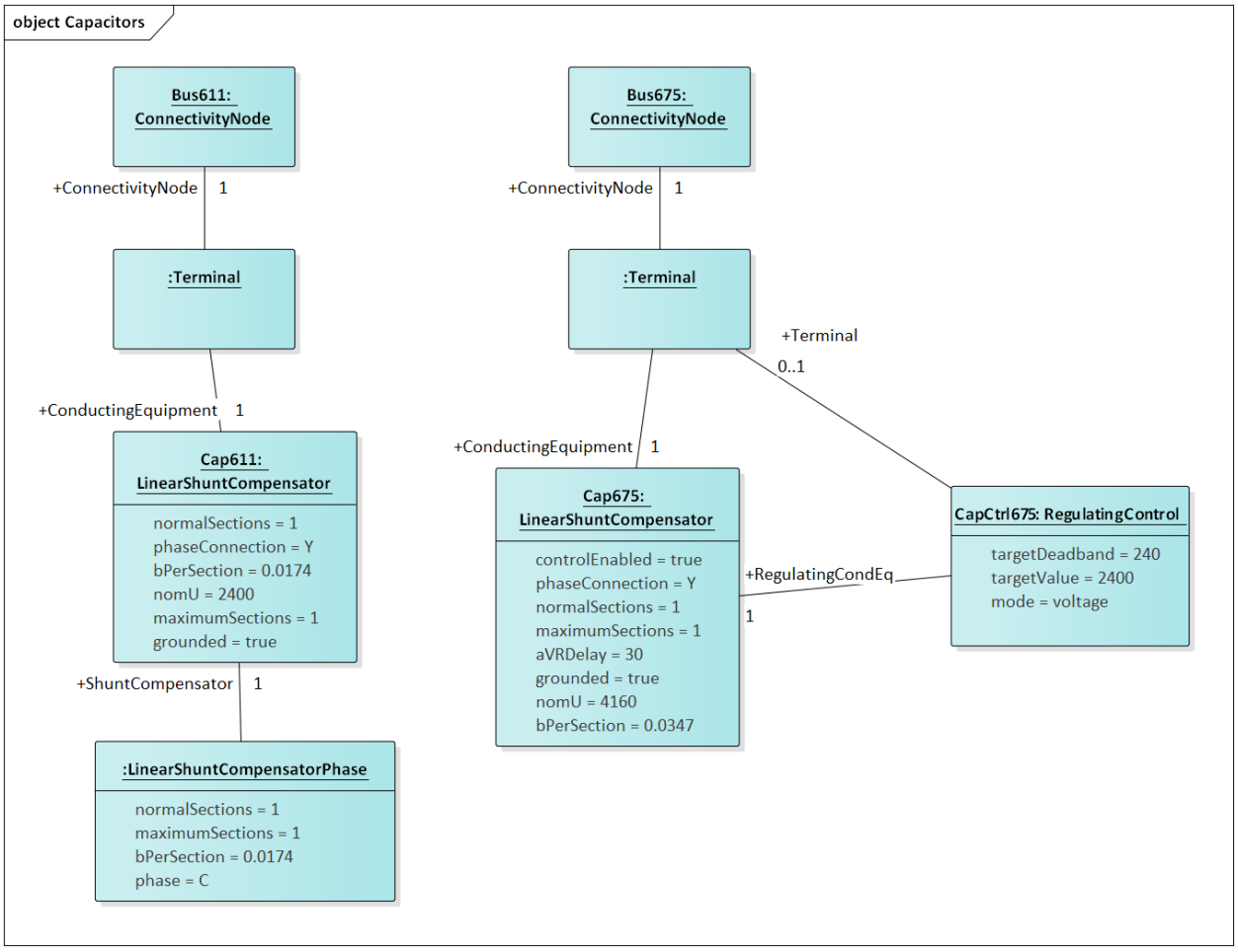



Figure 20: Capacitors are called LinearShuntCompensator in CIM. On the left, a 100 kVAR, 2400 V single-phase bank is shown on phase C at bus 611. bPerSection = 100e3 / 2400^2 [S], and the bPerSection on LinearShuntCompensatorPhase predominates; these values can differ among phases if there is more than one phase present. On the right, a balanced three-phase capacitor is shown at bus 675, rated 300 kVAR and 4160 V line-to-line. We know it's balanced three phase from the absence of associated LinearShuntCompensatorPhase objects. bPerSection = 300e4 / 4160^2 [S]. This three-phase bank has a voltage controller attached with 2400 V setpoint and 240 V deadband, meaning the capacitor switches ON if the voltage drops below 2280 V and OFF if the voltage rises above 2520 V. These voltages have to be monitored line-to-neutral in CIM, with no VT ratio. In this case, the control monitors the same Terminal that the capacitor is connected to, but a different conducting equipment's Terminal could be used. The control delay is called aVRDelay in CIM, and it's an attribute of the LinearShuntCompensator instead of the RegulatingControl. It corresponds to "dwell time" in GridLAB-D.

Figure 21 through Figure 26 illustrate the transformer query tasks, plus Figure 35 for attached voltage regulators. The autotransformer example is rated 500/345/13.8 kV and 500/500/50 MVA, for a transmission system. The short circuit test values are $Z_{HL}$=10%, $Z_{HT}$=25% and $Z_{LT}$=30%. The no-load test values are 0.05% exciting current and 0.025% no-load losses. These convert to r, x, g and b in SI units, from $Z_{LT}$= $U_{rated}$* $U_{rated}$/ $S_{rated}$, where $S_{rated}$and $U_{rated}$are based on the "from" winding (aka end). The same base quantities would be used to convert r, x, g and b back to per-unit or percent. The open wye – open delta impedances are already represented in percent or kW, from the test reports.

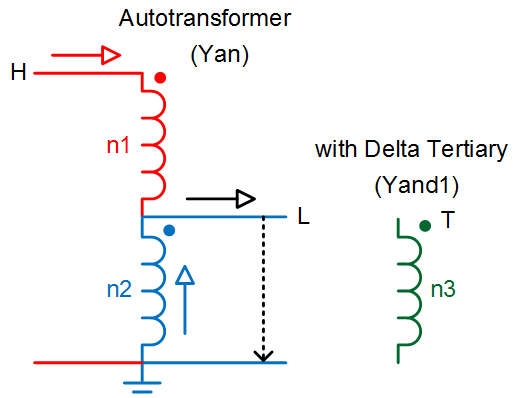



Figure 21: Autotransformer with delta tertiary winding acts like a wye-wye transformer with smaller delta tertiary. The vector group would be Yynd1 or Yyd1. For analyses other than power flow, it can be represented more accurately as the physical series (n1) – common (n2) connection, with a vector group Yand1. In either case, it's a three-winding transformer.

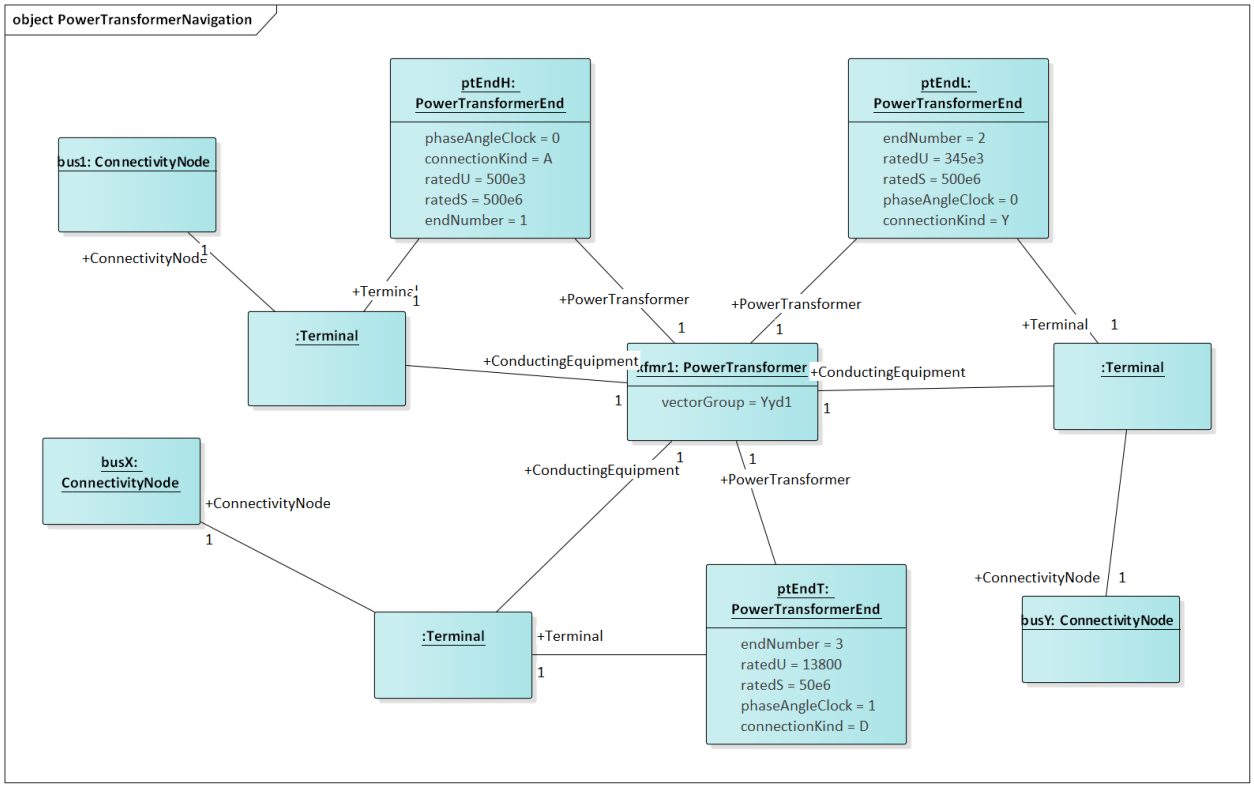



Figure 22: A three-winding autotransformer is represented in CIM as a PowerTransformer with three PowerTransformerEnds, because it's balanced and three-phase. The three Terminals have direct ConductingEquipment references to the PowerTransformer, so you can find it from bus1, busX or busY. However, each PowerTransformerEnd has a back-reference to the same Terminal, and it's own reference to BaseVoltage (Figure 13); that's how you link the matching buses and windings, which must have compatible voltages. Terminals have a sequenceNumber, but the PowerTransformerEnd's endNumber is what establishes correct linkage to catalog data as discussed later. By convention, ends with highest ratedU have the lowest endNumber, and endNumber establishes that end's place in the vectorGroup.

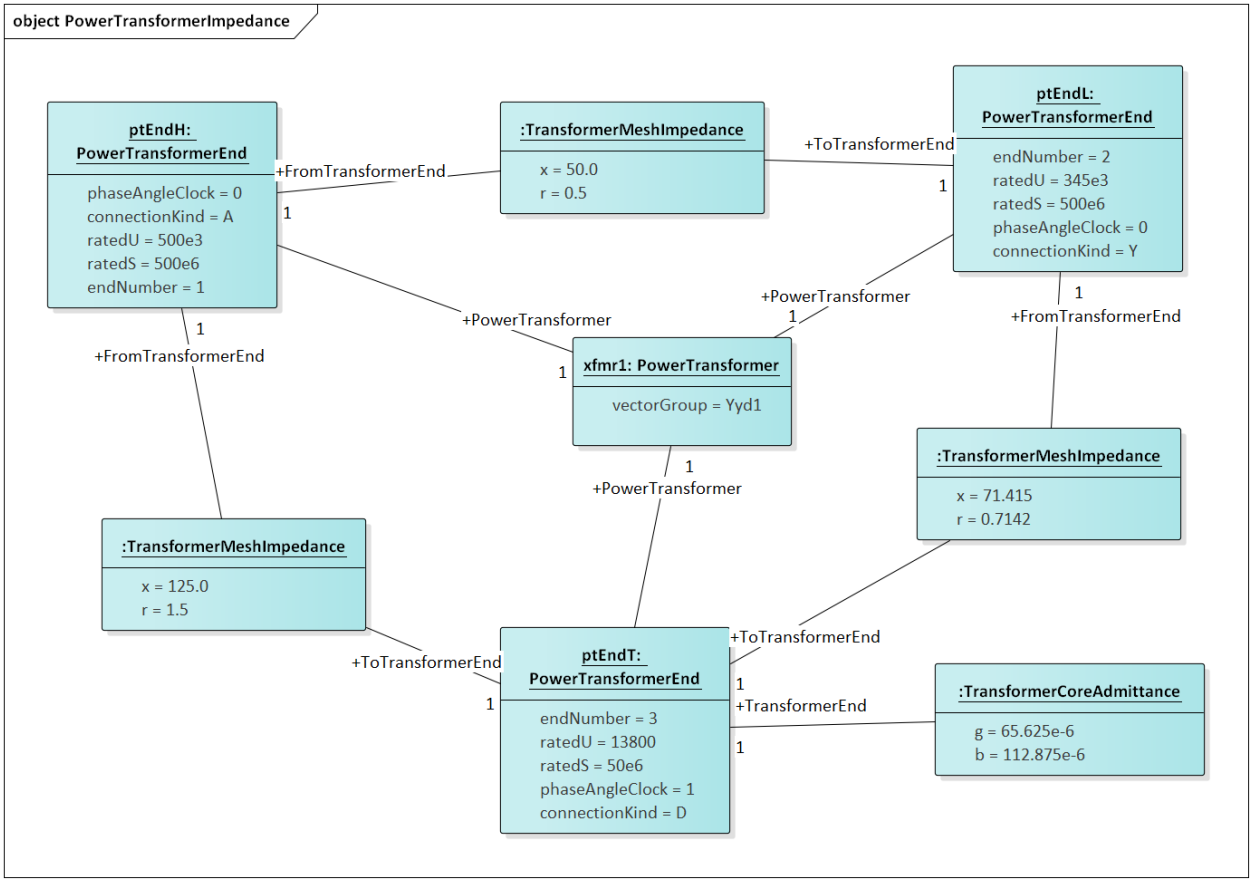



Figure 23: Power transformer impedances correspond to the three-winding autotransformer example of Figure 15 and Figure 16. There are three instances of TransformerMeshImpedance connected pair-wise between the three windings / ends. The x and r values are in Ohms referred to the end with highest ratedU in that pair. There is just one TransformerCoreAdmittance, usually attached to the end with lowest ratedU, and the attribute values are Siemens referred to that end's ratedU.

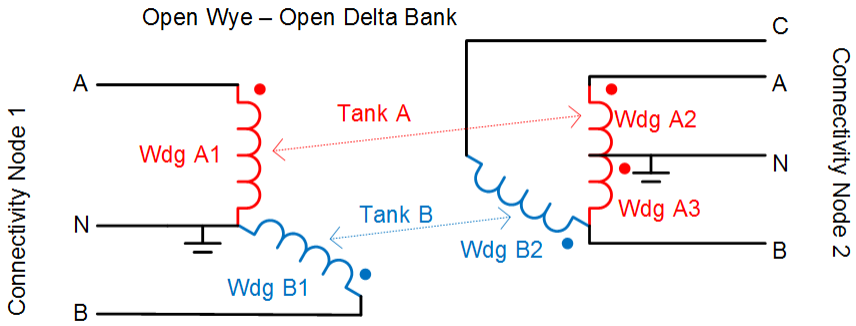



Figure 24: Open wye - open delta transformer banks are used to provide inexpensive three-phase service to loads, by using only two single-phase transformers. This is an unbalanced transformer, and as such it requires tank modeling in CIM. Physically, the two transformers would be in separate tanks. Note that Tank A is similar to the residential center-tapped secondary transformer, except the CIM phases for the secondary would include s1 and s2 instead of A and B.

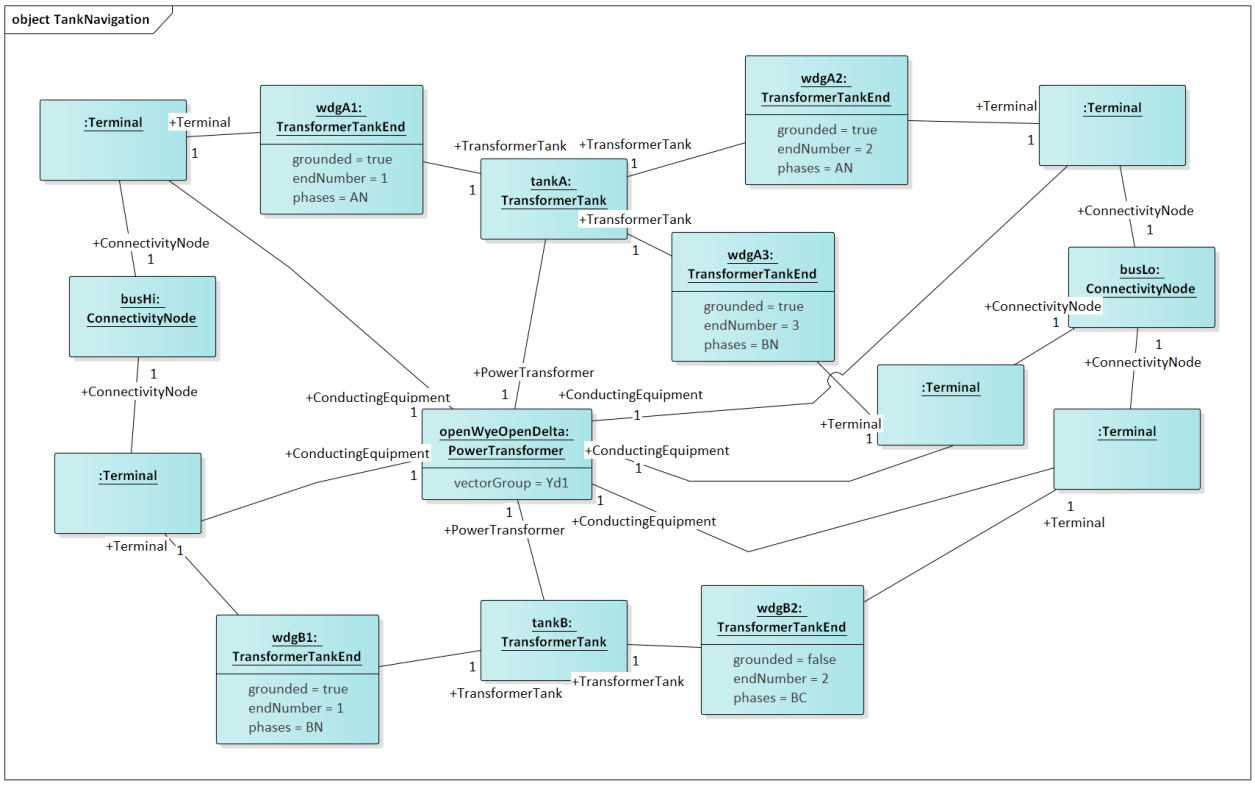



Figure 25: Unbalanced PowerTransformer instances comprise one or more TransformerTanks, which own the TransformerTankEnds. Through the ends, wdgHi collects phases ABN and busLo collects phases ABCN. Typically, phase C will also exist at wdgHi, but this transformer doesn't require it. We still assign vectorGroup Yd1 to the supervising PowerTransformer, as this is the typical case. The modeler should determine that. By comparison to Figure 24, there is a possible ambiguity in how endA3 represents the polarity dot at the neutral end of Wdg A3. An earlier CIM proposal would have assigned phaseAngleClock = 6 on wdgA3, but the attribute was removed from TransformerTankEnd. It may not be possible to infer the correct winding polarities from the vectorGroup in all cases. There is a phaseAngleClock attribute on TransformerTankEndInfo, but that represents a shelf state of the tank, not necessarily connections in the field. Therefore, it may be necessary to propose the phaseAngleClock attribute for TransformerTankEnd.

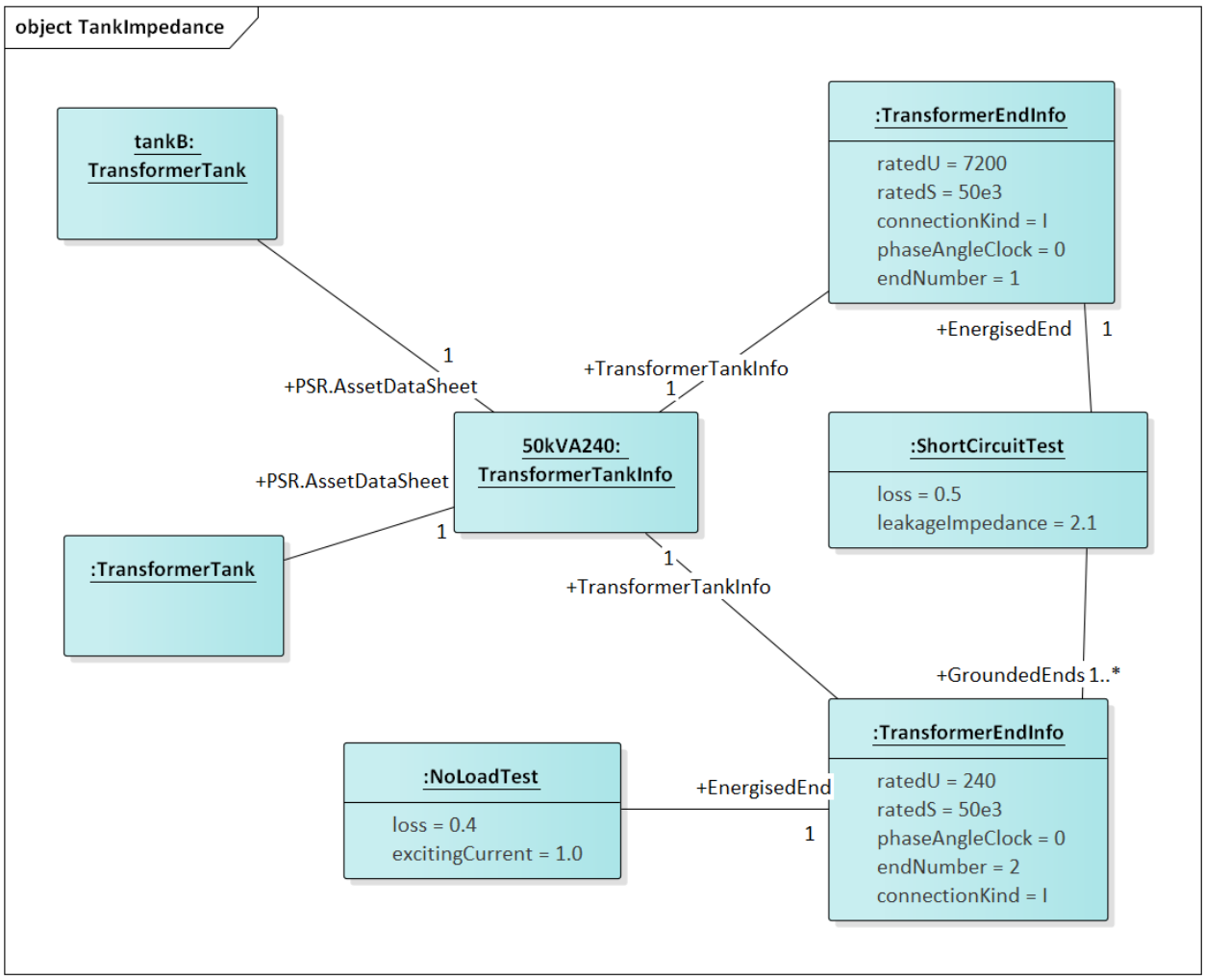



Figure 26: This Asset catalog example defines the impedances for Tank B of the open wye – open delta bank. This is a 50 kVA, 7200 / 240 V single-phase transformer. It has 1% exciting current and 0.4 kW loss in the no-load test, plus 2.1% reactance and 0.5 kW loss in the short-circuit test. A multi-winding transformer could have more than one grounded end in a short-circuit test, but this is not common. The catalog data is linked with an AssetDataSheet association shown to the left. Furthermore, endNumber on the TransformerEndInfo has to match endNumber on the TransformerTankEnd instances associated to Tank B. Instead of catalog information, we could have used mesh impedance and core admittance as in Figure 18, but we'd have to convert the test sheets to SI units and we could not share data with other TransformerTank instances, both of which are inconvenient.

Figure 27 through Figure 33 illustrate the query tasks for ACLineSegments and Switches, which will define most of the circuit's connectivity. The example sequence impedances were based on $Z_1 = 0.1 + j0.8$ Ω/mile and $Z_0 = 0.5 + j2.0$ Ω /mile. For distribution systems, use of the shared catalog data is more common, either pre-calculated matrix (Figure 31) or spacing and conductor (Figure 32 and Figure 33). In both cases, impedance calculation is outside the scope of CIM (e.g. GridLAB-D internally calculates line impedance from spacing and conductor data).

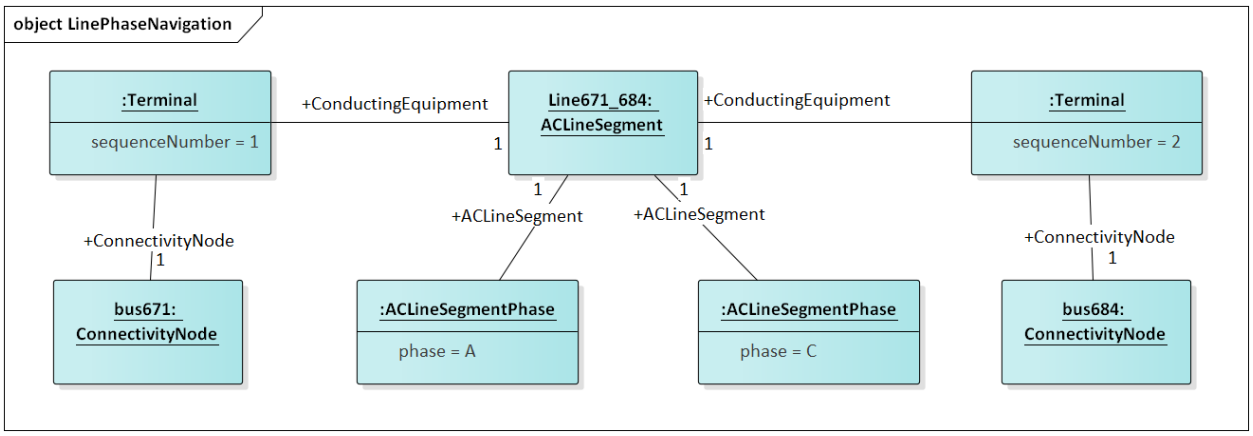



Figure 27: An ACLineSegment with two phases, A and C. If there are no ACLineSegmentPhase instances that associate to it, assume it's a three-phase ACLineSegment. This adds phases AC to bus671 and bus684.

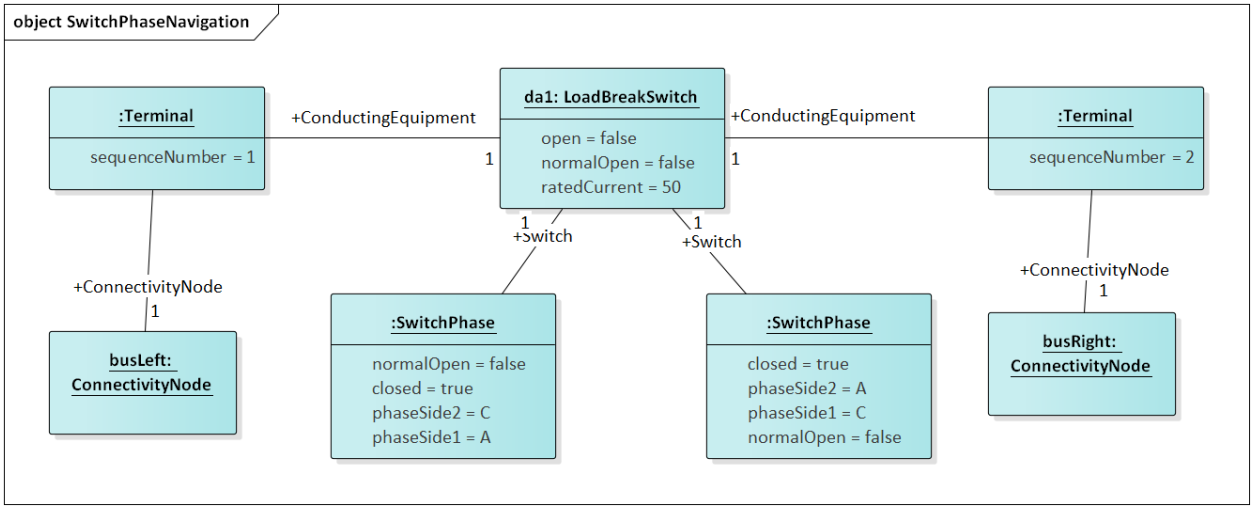



Figure 28: This 50-Amp load break switch connects phases AC between busLeft and busRight. Without associated SwitchPhase instances, it would be a three-phase switch. This switch also transposes the phases; A on side 1 connects with C on side 2, while C on side 1 connects with A on side 2. This is the only way of transposing phases in CIM. Note the Terminal.sequenceNumber is essential to differentiate phaseSide1 from phaseSide2. Also note that LoadBreakSwitch has the open attribute inherited from Switch, while SwitchPhase has the converse closed attribute. In order to open and close the switch, these attributes would be toggled appropriately. See Figure 4 for other types of switch.

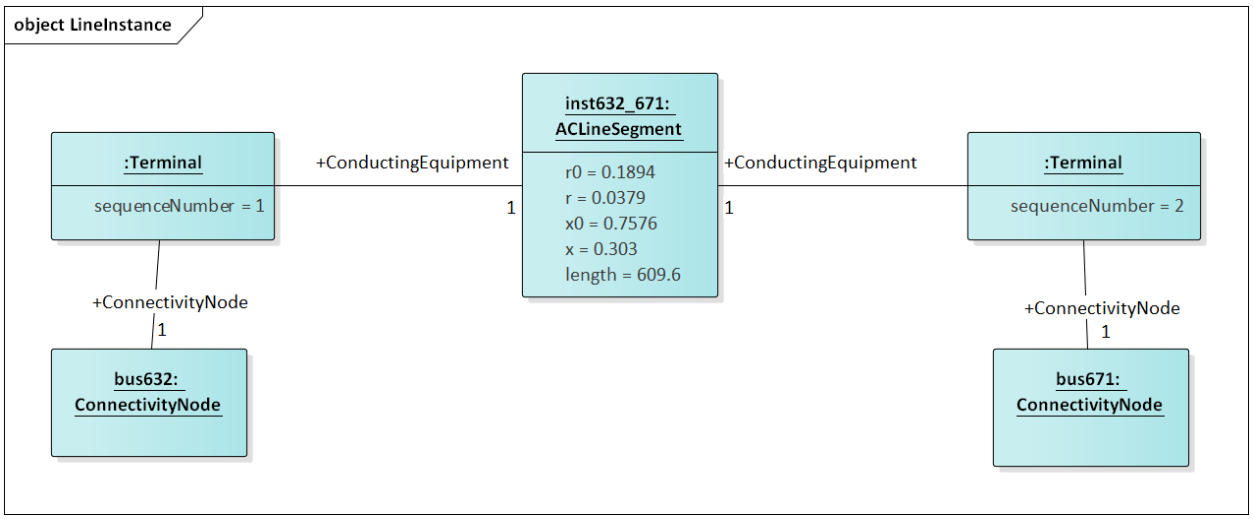



Figure 29: This is a balanced three-phase ACLineSegment between bus632 and bus671, 2000 feet or 609.6 m long. Sequence impedances are specified in ohms, as attributes on the ACLineSegment. This is a typical pattern for transmission lines, but not distribution lines.

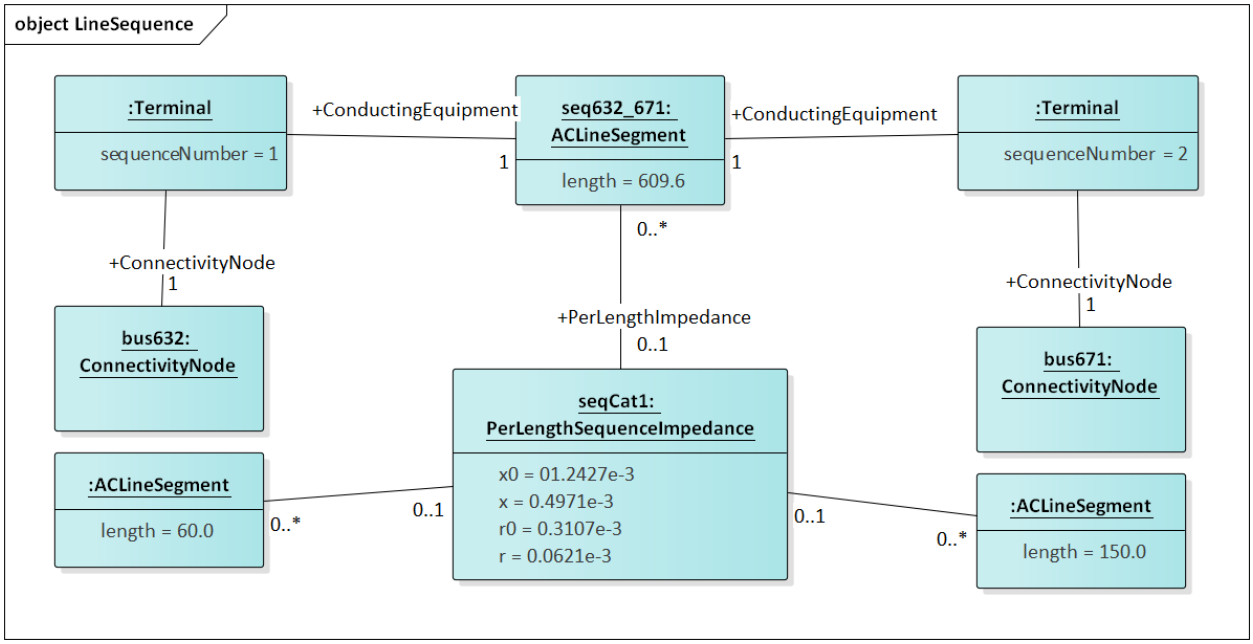



Figure 30: The impedances from Figure 24 were divided by 609.6 m, to obtain ohms per meter for seqCat1. Utilities often call this a "line code", and other ACLineSegment instances can share the same PerLengthImpedance. A model imported into the CIM could have many line codes, not all of them used in that particular model. However, those line codes should be available for updates by reassigning PerLengthImpedance.

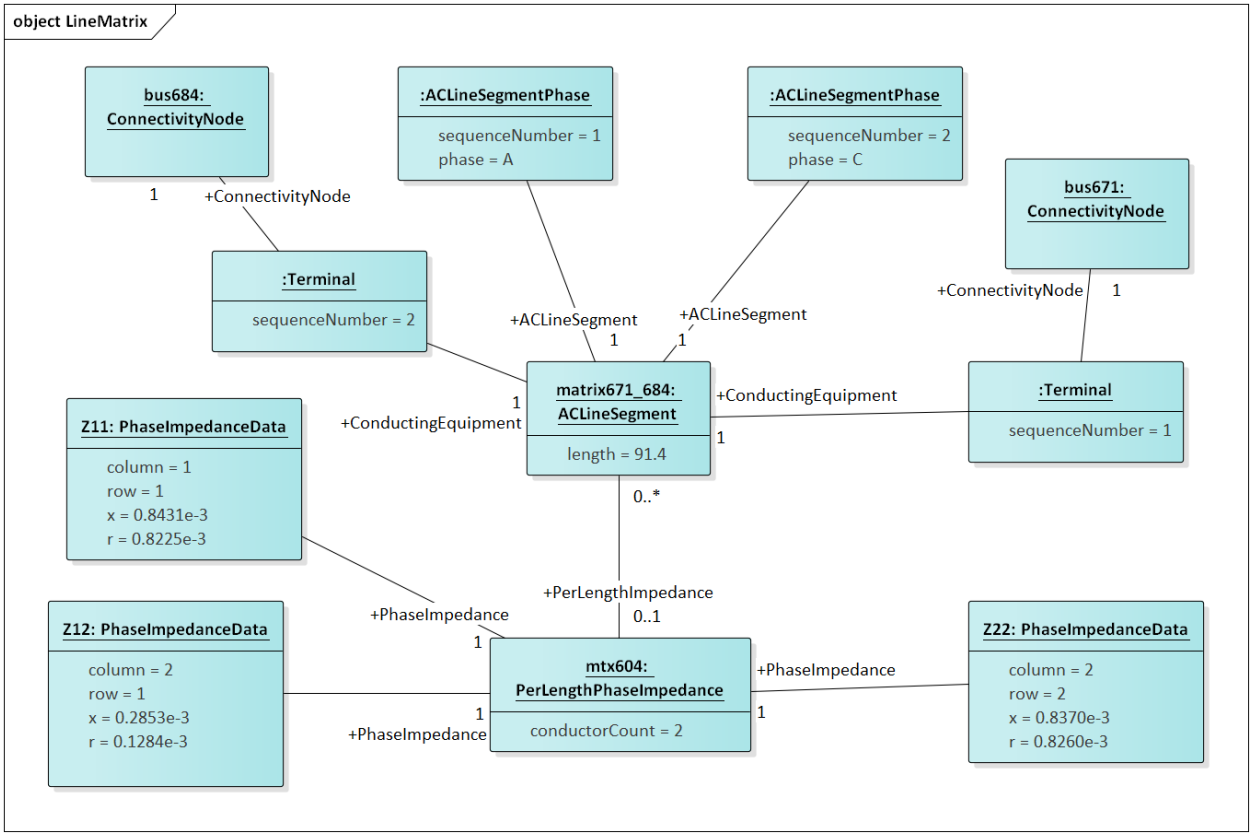



Figure 31: This is a two-phase line segment from bus671 to bus684 using a line code, which has been specified using a 2x2 symmetric matrix of phase impedances per meter, instead of sequence impedances per meter. This is more common for distribution than either Figure 29 or Figure 30. It's distinguished from Figure 30 by the fact that PerLengthImpedance references an instance of PerLengthPhaseImpedance, not PerLengthSequenceImpedance. The conductorCount attribute tells us it's a 2x2 matrix, which will have two unique diagonal elements and one distinct off-diagonal element. The elements are provided in three PhaseImpedanceData instances, which are named here for clarity as Z11, Z12 and Z22. However, only the row and column attributes are meaningful to identify the matrix element. In this example, Z11 and Z22 are slightly different. In order to swap phases A and C, we would swap the sequenceNumber values on the ACLineSegmentPhase instaces. As presented here, mtx604 can apply to phasing AB, BC or AC.

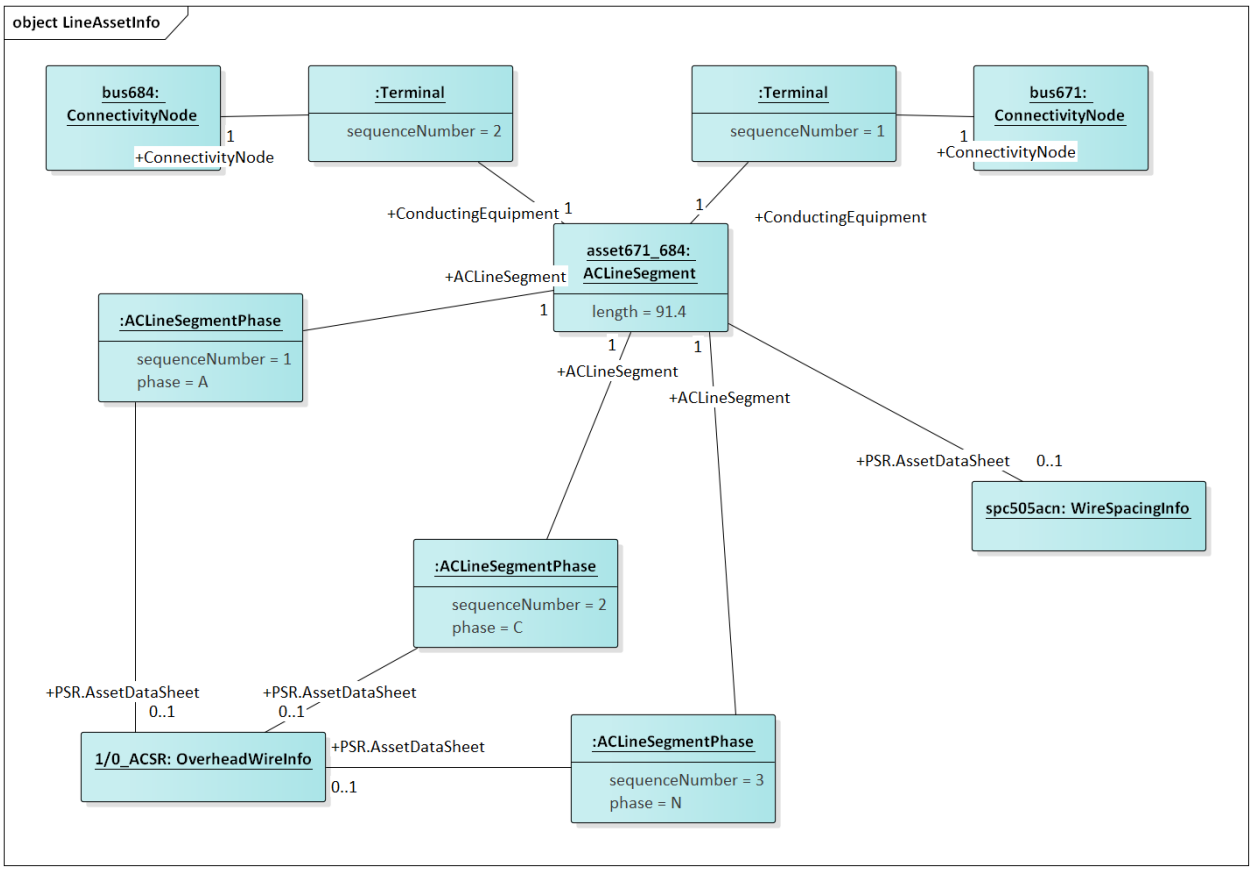



Figure 32: The two-phase ACLineSegment impedance defined by sharing wire and spacing data from a catalog. Each ACLineSegmentPhase links to an OverheadWireInfo instance via the AssetDataSheet association. If the neutral (N) is present, we have to specify its wire information for a correct impedance calculation. In this case, ACN all use the same wire type, but they can be different, especially for the neutral. Similarly, the WireSpacingInfo associates to the ACLineSegment itself via a AssetDataSheet assocation.

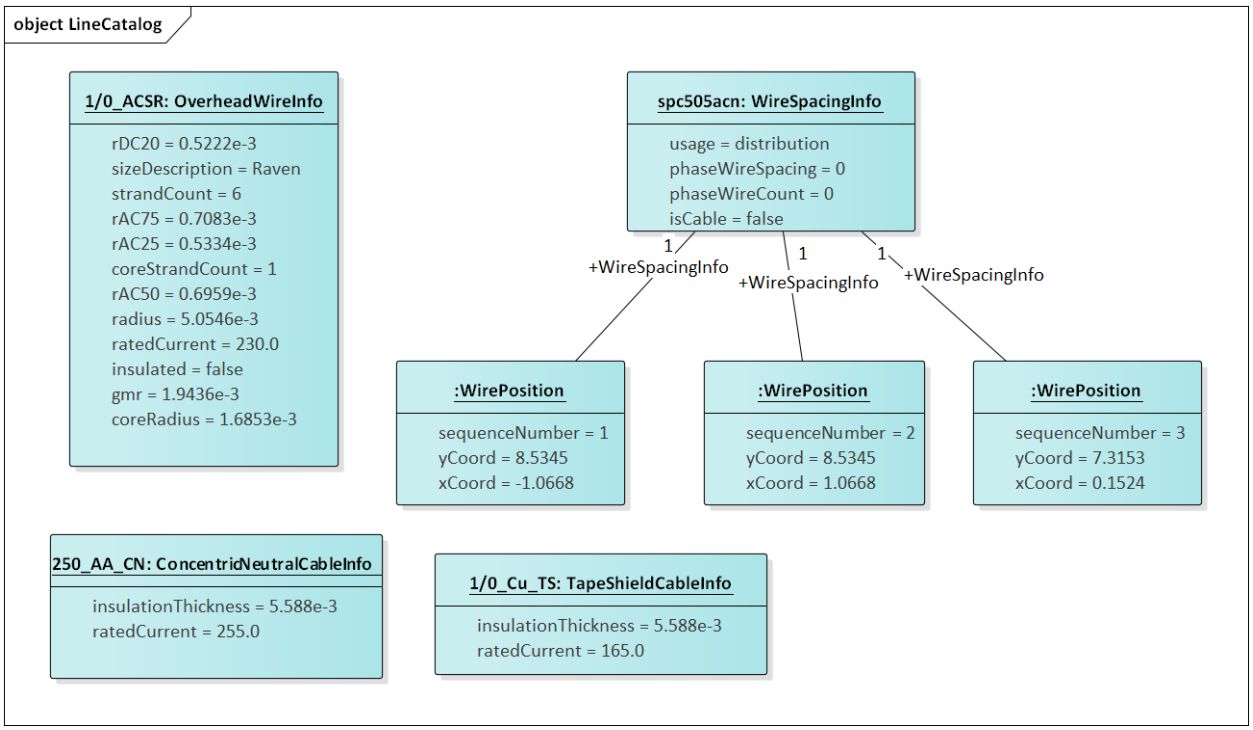



Figure 33: The upper five instances define catalog attributes for Figure 27. The WirePosition xCoord and yCoord units are meters, not feet, and they include sequenceNumber assignments to match ACLineSegmentPhase sequenceNumbers. The phaseWireSpacing and phaseWireCount attributes are for sub-conductor bundling on EHV and UHV transmission lines; bundling is not used on distribution. The number of WirePositions that reference spc505acn determine how many wires need to be assigned. Eliminating the neutral, this would produce a 2x2 phase impedance matrix. Although the pattern appears general enough to support multiple neutrals and transmission overbuild, the CIM doesn't actually have the required phasing codes. When isCable is true, the WirePosition yCoord values would be negative for underground depth. To find overhead wires of a certain size or ampacity, we can put query conditions on the ratedCurrent attribute. To find underground conductors, we query the ConcentricNeutralCableInfo or TapeShieldCableInfo instead of OverheadWireInfo. All three inherit the ratedCurrent attribute from WireInfo. Cables don't yet have a voltage rating in CIM AssetInfo, but you can use insulationThickness as a proxy for voltage rating in queries. Here, 5.588 mm corresponds to 220 mils, which is a common size for distribution.

Figure 34 illustrates the loads, which are called EnergyConsumer in CIM. The houses and appliances from GridLAB-D are not supported in CIM. Only ZIP loads can be represented. Further, any load schedules would have to be defined outside of CIM. Assume that the CIM loads are peak values.

Figure 35 illustrates the voltage regulator function. Note that GridLAB-D combines the regulator and transformer functions, while CIM separates them. Also, the CIM provides voltage and current transducer ratios for tap changer controls, but not for capacitor controls.

Figure 36 through Figure 38 illustrate how solved values can be attached to buses or other components.

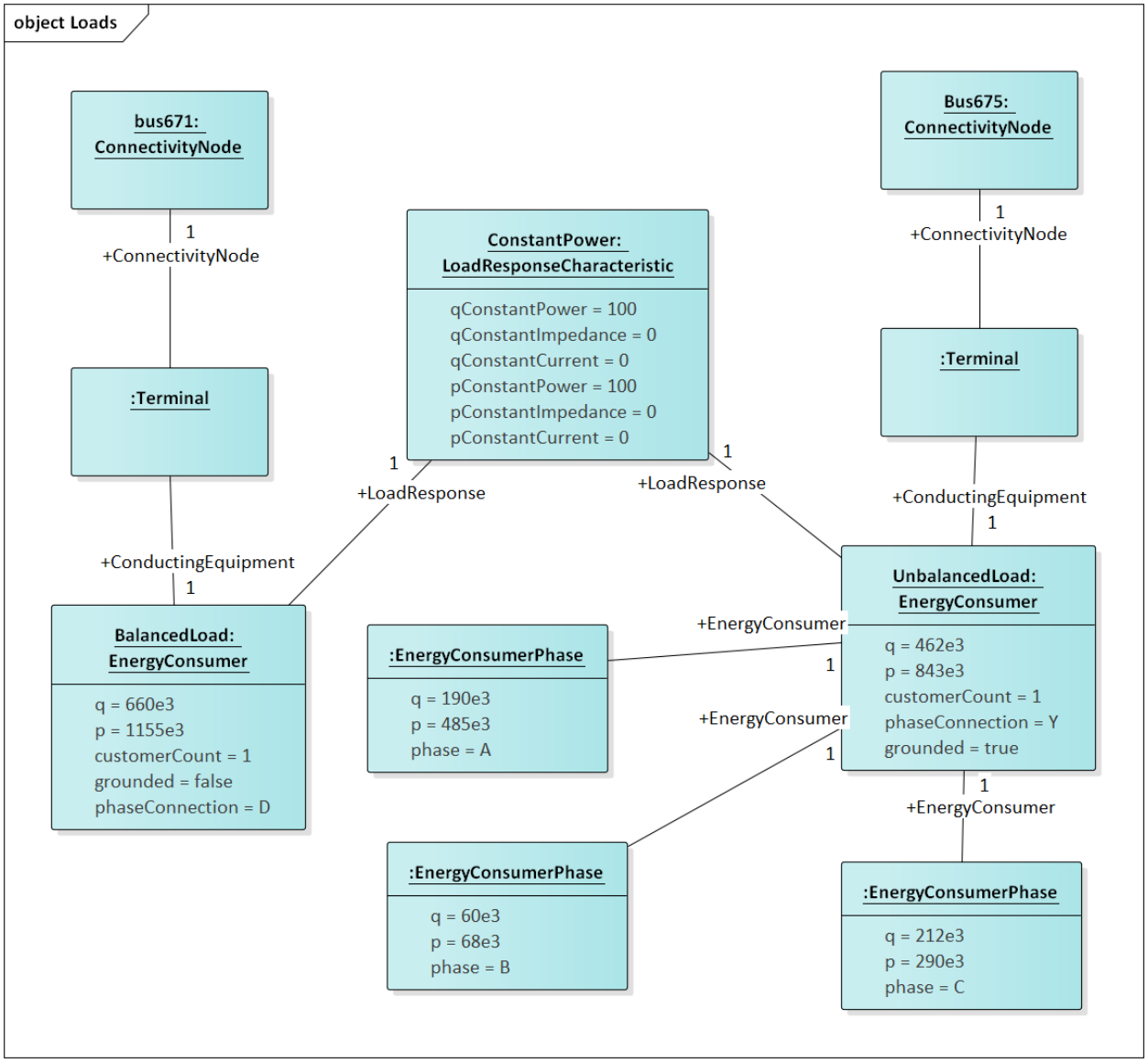



Figure 34: The three-phase load (aka EnergyConsumer) on bus671 is balanced and connected in delta. It has no ratedU attribute, so use the referenced BaseVoltage (Figure 19) if a voltage level is required. On the right, a three-phase wye-connected unbalanced load on bus675 is indicated by the presence of three EnergyConsumerPhase instances referencing UnbalancedLoad. For consistency in searches and visualization, UnbalancedLoad.p should be the sum of the three phase values, and likewise for UnbalancedLoad.q. In power flow solutions, the individual phase values would be used. Both loads share the same LoadResponse instance, which defines a constant power characteristic for both P and Q, because the percentages for constant impedance and constant current are all zero. The two other most commonly used LoadResponseCharacteristics have 100% constant current, and 100% constant impedance. Any combination can be used, and the units don't have to be percent (i.e., use a summation to determine the denominator for normalization).

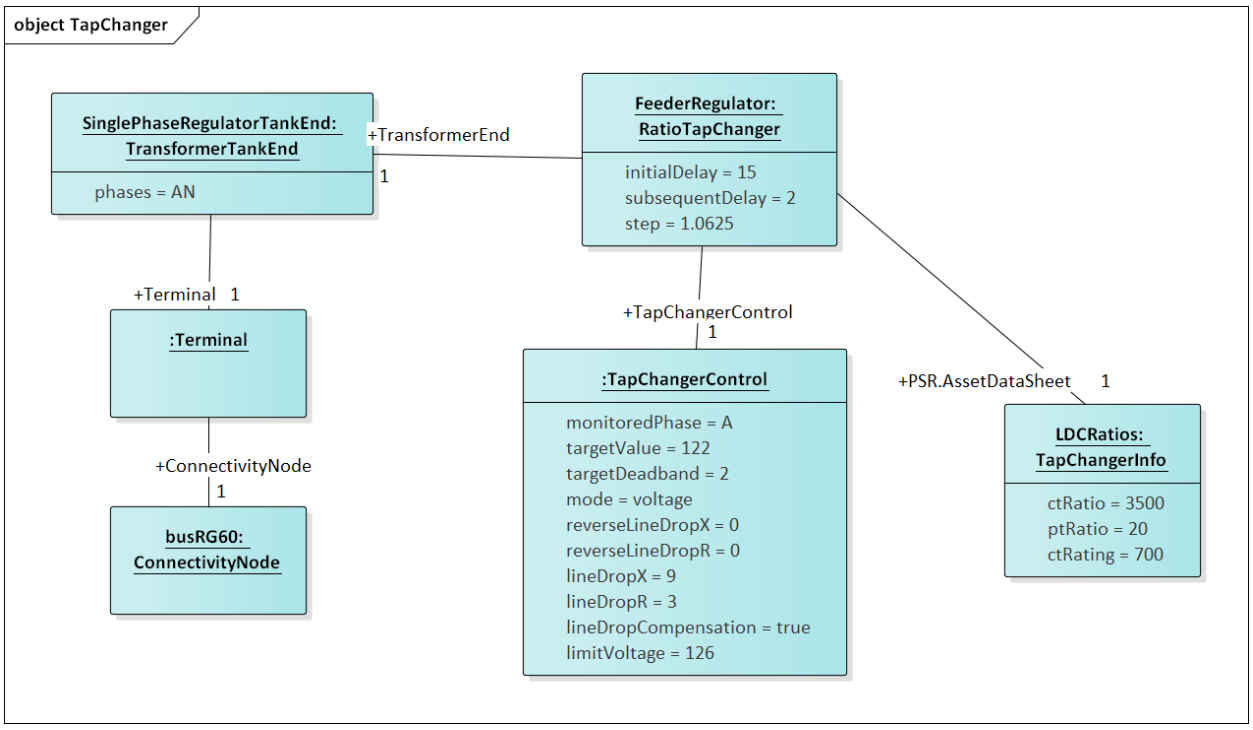



Figure 35: In CIM, the voltage regulator function is separated from the tap-changing transformer. The IEEE 13-bus system has a bank of three independent single-phase regulators at busRG60, and this example shows a RatioTapChanger attached to the regulator on phase A, represented by the TransformerTankEnd having phases=A or phases=AN. See Figure 25 for a more complete picture of TransformerTankEnds, or Figure 22 for a more complete picture of PowerTransformerEnds. Either one can be the TransformerEnd in this figure, but with a PowerTransformerEnd, all three phase taps would change in unison (i.e. they are "ganged"). Most regulator attributes of interest are found in RatioTapChanger or TapChangerControl instances. However, we need the AssetDataSheet mechanism to specify ctRatio, ptRatio and ctRating values. These are inherent to the equipment, whereas the attributes of RatioTapChanger and TapChangerControl are all settings per instance. For the IEEE 13-bus example, there would be separate RatioTapChanger and TapChangerControl instances for phases B and C.

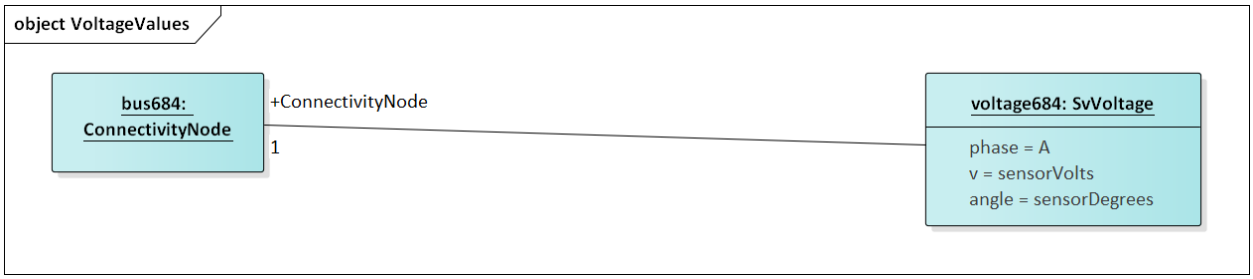



Figure 36: In this profile, a solved voltage value attaches to ConnectivityNode in GridAPPS-D. Positive sequence or phase A is implied, unless the phase attribute is specified.

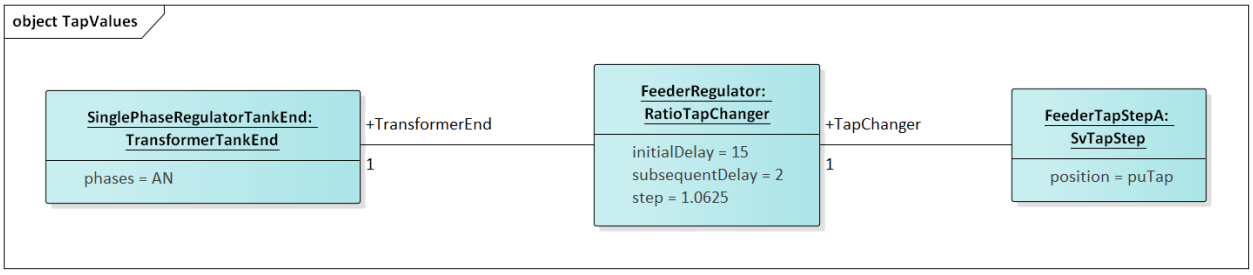


Figure 37: SvTapStep links to a TransformerEnd indirectly, through the RatioTapChanger. There is no phasing ambiguity because TransformerTankEnd has its phases attribute, while PowerTransformerEnd always includes ABC. Units for SvTapStep.position are per-unit.

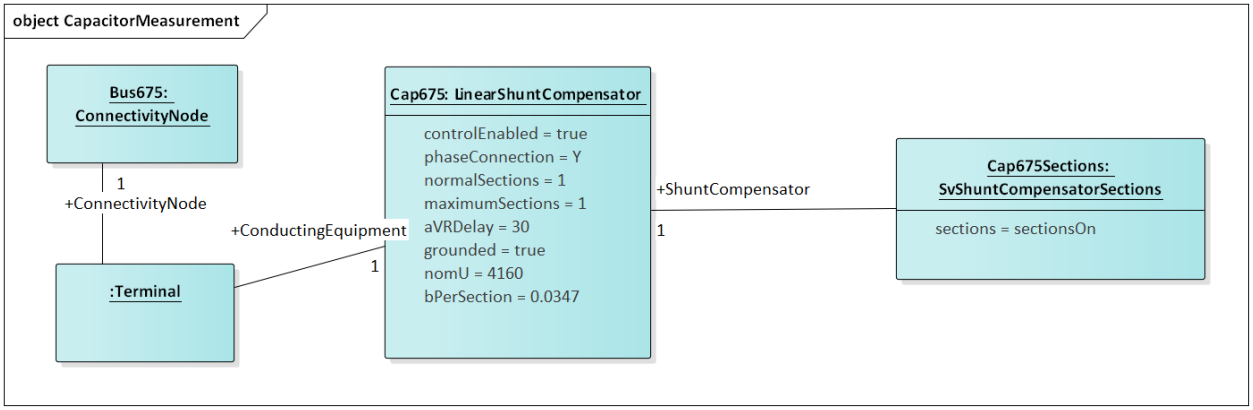



Figure 38: The on/off value for a capacitor bank attaches directly to LinearShuntCompensator. If the phase attribute is not specified, then this value applies to all phases.

### 5.4.4 Metering Relationship to Loads in the CIM

Figure 39 shows how emulated trouble calls will be connected to loads (EnergyConsumers) for test scenarios. The TroubleTicket is associated with Customer, CustomerAgreement and UsagePoint, which can then be associated to Equipment or any of its descendants. Figure 39 shows the linkage to EnergyConsumer or EnergySource, but it can also be linked to RegulatingCondEq (e.g., rotating machine and inverter-based DER). There are many attributes of Customer, CustomerAgreement and UsagePoint that are not yet used in GridAPPS-D, and not shown in Figure 40. These would be important for future metering and customer management applications. For now, the only TroubleTicket attributes to be used are dateTimeOfReport, resolvedDateTime and troubleKind. The PNNLTroubleCallKind was added because the existing troubleCode attribute is a non-standardized String. However, the comment attribute could be used for optional comments on each TroubleTicket.

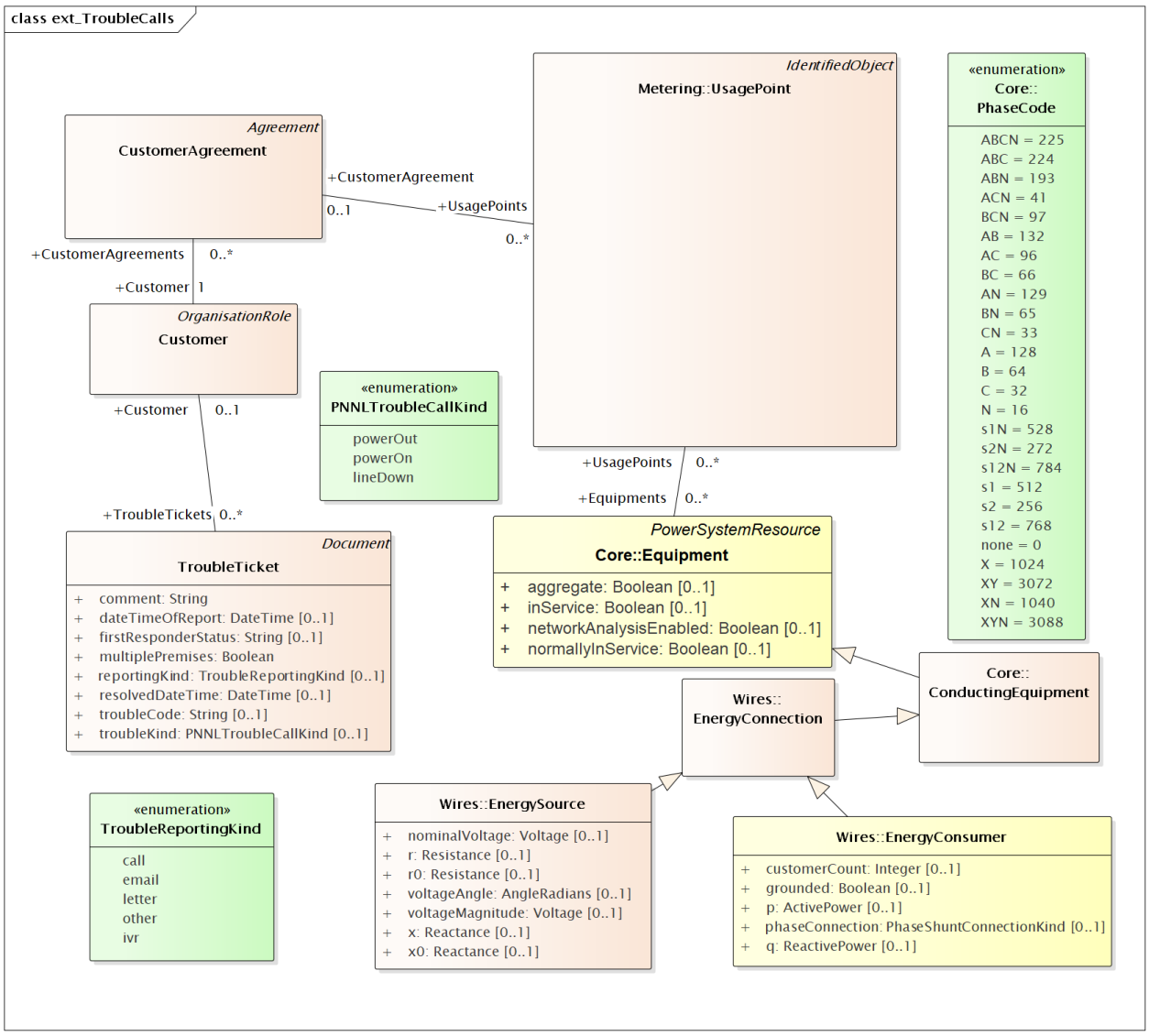



Figure 39: Trouble Calls route through Metering Usage Points to EnergyConsumers

### 5.4.5 CIM Enhancements for RC4

Possible CIM enhancements:

1. Different on and off delay parameters for RegulatingControl (Figure 5)
2. Current ratings for PerLengthImpedance (Figure 2). At present, some users rely on associated WireInfo, ignoring all attributes except currentRating.
3. Transducers for RegulatingControl (Figure 5)
4. Dielectric constant and soil resistivity (Figure 10)
5. Clock angles for TransformerTankEnd (i.e. move phaseAngleClock from PowerTransformerEnd to TransformerEnd (Figure 6)
6. Add the Fault.stopDateTime attribute
7. Single-phase asynchronous and synchronous machines.

### 5.4.6 CIM Profile in CIMTool

CIMTool was used to develop and test the profile for RC1, because it:

1. Generates SQL for the MySQL database definition
2. Validates instance files against the profile

The CIMTool developer will not be able to support the tool in future, so we may use the new Schema Composer feature in Enterprise Architect when it's ready for CIM profiling.

In order to view the profile, import the archived Eclipse project *OSPRREYS_CIMTOOL.zip* into CIMTool. Please see the CIM tutorial slides provided by Margaret Goodrich for user instructions.

Four instance files were validated against the profile in CIMTool. In order to generate them, we use a current version of OpenDSS with the *Export CDPSMcombined* command on four IEEE test feeders that come with OpenDSS:

1. **~/src/opendss/Test/IEEE13_CDPSM.dss** is the IEEE 13-bus test feeder with per-length phase impedance matrices and a delta tertiary added to the substation transformer. PV and storage were also added.
2. **~/src/opendss/Test/IEEE13_Assets.dss** is the IEEE 13-bus test feeder with catalog data for overhead lines, cables and transformers. Capacitor controls have also been added.
3. **~/src/opendss/Distrib/IEEETestCases/8500-Node/Master.dss** is the IEEE 8500-node test feeder with balanced secondary loads.
4. **~/src/opendss/Distrib/IEEETestCases/8500-Node/Master-unbal.dss** is the IEEE 8500-node test feeder with unbalanced secondary loads.

Either the 3rd or 4th feeder will be used for the volt-var application. The 1st and 2nd feeders are used to validate more parts of the CIM profile used in RC1. In all four cases, CIMTool reports only two kinds of validation error:

1. **Isolated connectivity node**: CIMTool expects two or more Terminals per ConnectivityNode, but dead ended feeder segments will have only one on the last node. This is not really an error, at least for distribution systems.
2. **Minimum cardinality**: For TapChangerControl instances, the inherited RegulatingControl.RegulatingCondEq association is not specified. This is not really an error, as the association is only needed for shunt capacitor controls. Figure 40 shows that RegulatingCondEq was not selected for TapChangerControl in the profile, so this may reflect a defect in the validation code. Efforts to circumvent it were not successful.

With these caveats, the profile and instances validate against each other, for feeder models that solve in OpenDSS.

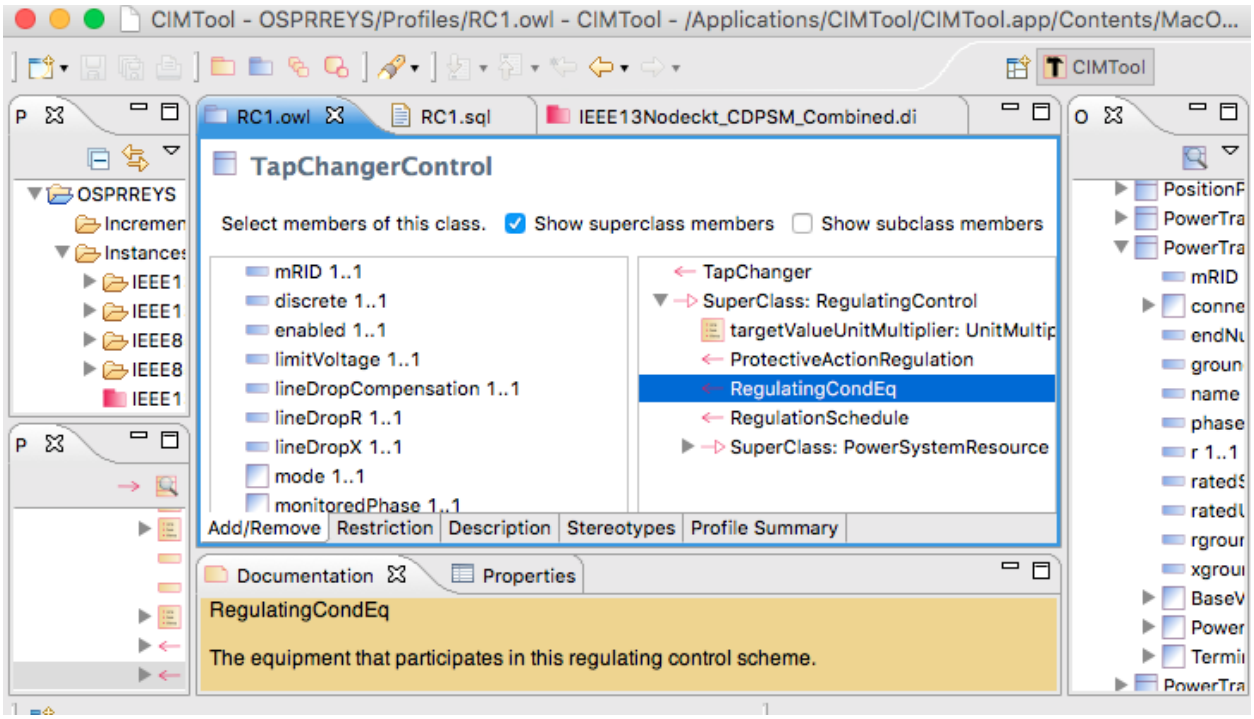



Figure 40: Editing a Profile in CIMTool

### 5.4.7 Legacy Data Definition Language (DDL) for MySQL

As shown at the top of Figure 40, CIMTool builds *RC1.sql* to create tables in a relational database, but the syntax doesn't match that required for MySQL. The following manual edits were made:

1. Globally change **CHAR VARYING(30)** to **varchar(50)** with a blank space pre-pended before the varchar
2. Globally change **“** to **‘**
3. In foreign keys to enumerations, change the referenced attribute from **mRID** to **name**
4. In foreign keys to **EquipmentContainer** or **ConnectivityNodeContainer**, change the referenced table to **Line**
5. In foreign keys to **ShuntCompensator**, change the referenced table to **LinearShuntCompensator**
6. In foreign keys to **TapChanger**, change the referenced table to **RatioTapChanger**.
7. The CIM UML incorporates several polymorphic associations, which can't be implemented directly in SQL. Base parent class tables were added for:
    a. **AssetInfo**, which can be referenced via the Parent attribute from ConcentricNeutralCableInfo, TapeShieldCableInfo, OverheadWireInfo, WireSpacingInfo, TapChangerInfo and TransformerTankInfo
    b. **TransformerEnd**, which can be referenced via the Parent attribute from PowerTransformerEnd and TransformerTankEnd
    c. **PerLengthImpedance**, which can be referenced via the Parent attribute from PerLengthSequenceImpedance and PerLengthPhaseImpedance
    d. **Switch**, which can be referenced via the SwtParent attribute from Breaker, Fuse, Sectionaliser, Recloser, Disconnector, Jumper and LoadBreakSwitch.
    e. **ConductingEquipment**, which can be referenced via the Parent attribute from ACLineSegment, EnergySource, EnergyConsumer, LinearShuntCompensator, PowerTransformer, and all of the Switch types.

8. The catalog data mechanism in Figure 8 required two new tables, one for polymorphic associations and another for many-to-many joins:

    a. **PowerSystemResource**, which can be referenced via the PSR attribute from ACLineSegment, ACLineSegmentPhase, RatioTapChanger and TransformerTank.

    b. **AssetInfoJoin**, which references AssetInfo and PowerSystemResource. This table actually supplants the Asset class in Figure 8.

9. The ShortCircuitTest in Figure 9 has a one-to-many association to TransformerEndEnfo, and we need to implement the many side by adding:

    a. **GroundedEndJoin**, which references TransformerEndInfo and ShortCircuitTest.

10. The ToTransformerEnd association in Figure 6 is one-to-many, so CIMTool did not export it to SQL. Rather than create a join table, a ToTransformerEnd attribute was added to TransformerMeshImpedance. This supports only one-to-one association, which is justified because the one-to-many case is very rare, and GridLAB-D cannot model transformers having the one-to-many association. This restriction may be removed in future versions having a semantic or graph database.

    Except for the first two items, all of these adjustments arose from the absence of inheritance or polymorphism in SQL. These adjustments will make the updates, queries and views more complicated. However, they allow referential integrity to be enforced, which is one of the most important reasons to use SQL and relational databases. Other types of data store could be a more natural fit to the CIM UML, but they may not have the performance of a relational database.

    In GitHub:

1. *RC1.sql* is the manually adjusted SQL export from CIMTool
2. *LoadRC1.sql* will **re-create the GridAPPS-D database in MySQL**, incorporate *RC1.sql*, and finally document the foreign keys. It should run without error.

## 5.5 Platform UML Diagrams

### 5.5.1 UML from the Functional Specification

This section presents a selection of GridAPPS-D domain (class) diagrams to supplement the *OSPRREYS Functional Specification* document. The purpose is to enhance understanding of the functional specification, by providing graphical walkthroughs of some important use cases. The reader should be familiar with definitions in the functional specification, and with Universal Modeling Language (UML) diagrams.

GridAPPS-D is organized as a suite of internal function managers, twelve of them composing the Platform Manager as shown in Figure 1. All GridAPPS-D functions and interactions are mediated by one (or more) of these function managers. When running, the GridAPPS-D 413 Platform Manager will be composed of one (and only one) of each internal manager numbered 401 – 412. These internal managers work together to accomplish various GridAPPS-D functions.

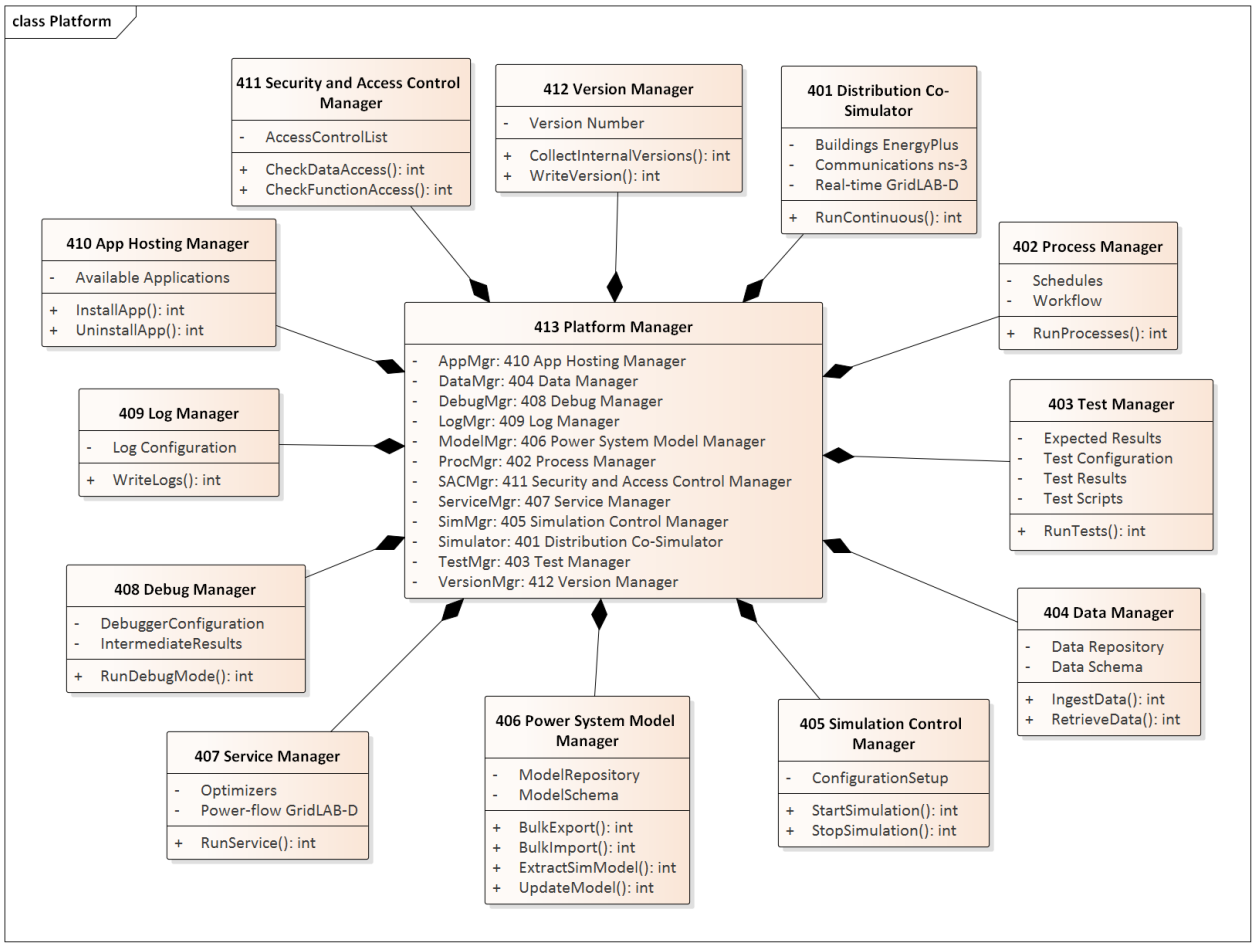



Figure 1: Composition of the GridAPPS-D Platform Manager

Within each class block, some top-level attributes are listed with (-) signs in the middle division, and some top-level methods are listed with (+) signs in the lower division. For example, we already know that 401 Distribution Co-Simulator will need component simulators (i.e. attributes) for buildings (open-source EnergyPlus), communications (open-source ns-3), and the electric power distribution grid (open-source GridLAB-D running in a real-time mode). It will also need at least one method that runs the suite of simulators in a mode emulating continuous real-time operation. Taking another example, 407 Service Manager also contains an attribute for GridLAB-D to provide power flow calculations, but run as a service to applications.

As the design evolves, classes in Figure 1 will acquire many more attributes and methods. The attributes themselves may reference complicated classes and data structures. Therefore, the UML model will expand each class into layer and sub-layer diagrams to more clearly show these evolving details. We can still use the top-level diagrams to make sure that the major components are in place for the important use cases.

Figure 2 illustrates the case of a user executing an application, in the role of EF7 from the functional specification. We initially focused on volt-var optimization (VVO), and then added a more complicated demand response (DR) application that fits the same basic pattern. As a prerequisite, some entity has provided both applications to GridAPPS-D for registration and hosting, in a process detailed later. For now, we assume the application(s) have been installed and will focus first on running VVO.

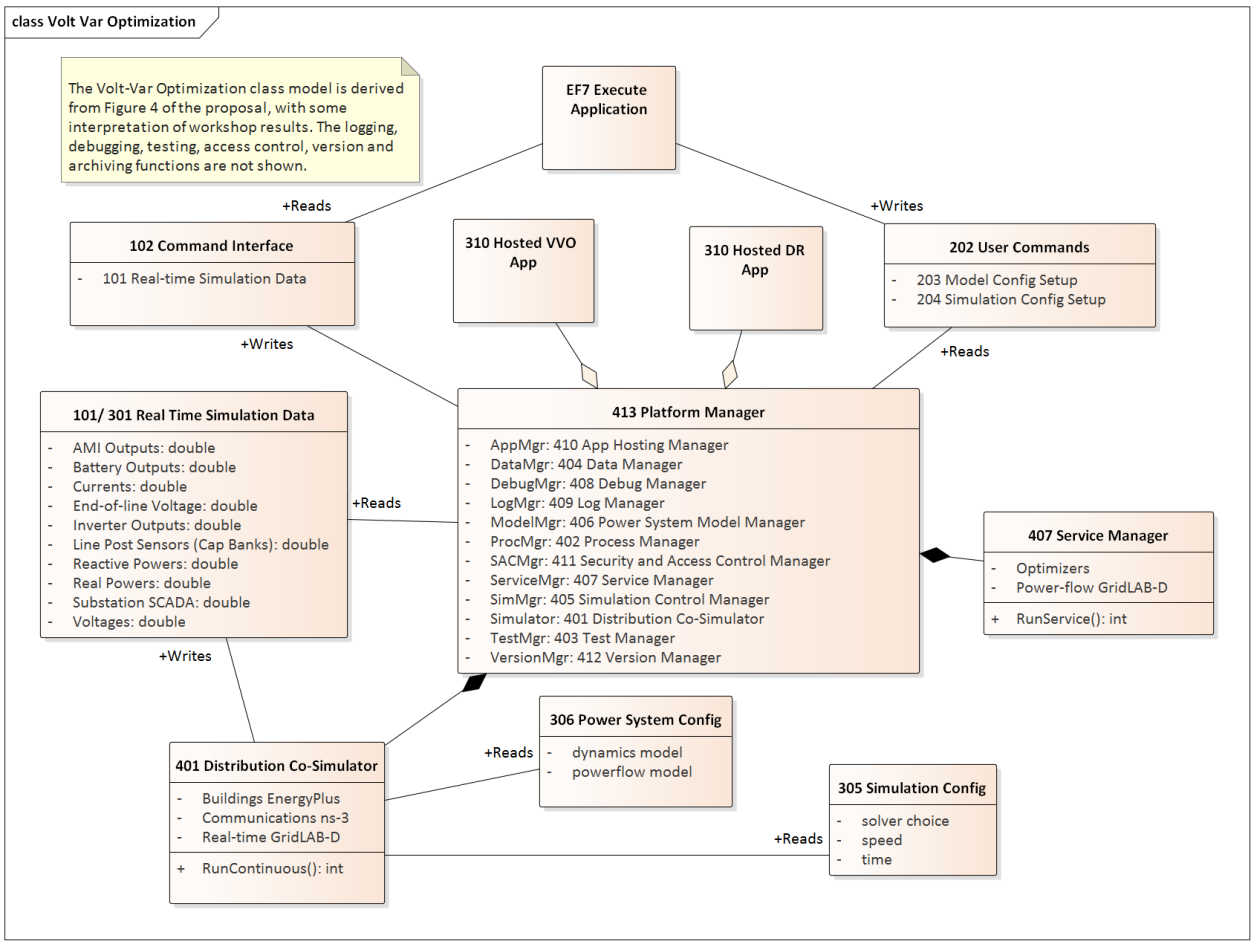



Figure 2: Executing an application

All user interaction with GridAPPS-D occurs through a command interface, numbered 202 when the user writes commands to GridAPPS-D, and numbered 102 when the user gets data from GridAPPS-D. To run VVO, the user will issue 203 Model Configuration Setup and 204 Simulation Configuration Setup to GridAPPS-D, which then delegates the commands to various internal function managers (see Figure 1). The 203 Setup will probably extract the feeder model of interest, set load and weather data, etc. The 204 Setup will probably tell 401 to run GridLAB-D for a certain time period, but not to run ns-3 or EnergyPlus. The exact composition of 203 and 204 Setups will be determined later in the design process. In a process described later, internal functions 405 (Simulation Control Manager) and 406 (Power System Model Manager) will transform 201, 203 and 204 into 305 and 306, which 401 can then read and run from directly.

When it runs, 401 will generate streams of data that mimic real-time operation of the system, and these streams pass to the other parts of GridAPPS-D as 301 Real-time Simulation Data. Some of the data streams may also output to the user as 101 Real-time Simulation Data. The 310 VVO Application can act on this data to make decisions (e.g. switch capacitor banks, change regulator taps, change solar inverter settings). In this process, 310 VVO could invoke power flow calculations in GridLAB-D via 407 Service Manager, but this is different from the way 401 Co-Simulator runs. The application may use 407 services to explore alternatives or run contingency analysis, which could change the power system model, but the 401 real-time simulations always take priority and always use the "real" model.

When we considered adding the second and more complicated application, 310 DR, the structure of Figure 2 didn't change very much. The open-headed diamond symbols indicate that GridAPPS-D can host several applications, which is UML aggregation. These applications may interact via the GridAPPS-D command interface, if the applications and their command sets have been designed for it. For example, the DR application may use VVO to check and mitigate voltage limits.

A DR application is more likely than VVO to need EnergyPlus and ns-3 in the co-simulation. In response, we added those attributes to 401, and will add supporting attributes to 201, 203 and 204 as the design evolves. It should also be recognized that more sophisticated VVO applications might incorporate communications (ns-3) if available.

Figure 3 depicts the process of managing power system models, including the schema and repository within 201 Distribution System Model. Because it's based on standards (e.g. IEC 61968) and open-source tools (e.g. MySQL), the model can be created and maintained from outside GridAPPS-D, directly by EF 21, the Model Manager. This is shown at the top of Figure 3. This process is out of GridAPPS-D scope but within project scope, and it can leverage existing tools like Cimphony, Cimdesk, EA, etc.

For use by and within GridAPPS-D, all model configuration commands will pass from EF21 through the command interface to function 406, the Power System Model Manager. This function reads the base power system model data from 201, and configures it into a three-phase load flow model for solution in 106/306. The Distribution Co-Simulator uses 306, but the user might want 106 for off-line use. Working with 404 Data Manager, the 406 Power System Model Manager may also write additional data (i.e. not used in the load flow calculation) to 104/304. In this case, the 102 Model Output function will collect that data from both 104 and 106 for reporting to the user, EF7, via the command interface. Note that the base data, in 201, is not modified through this process. Instead, the base data is treated as input to GridAPPS-D.

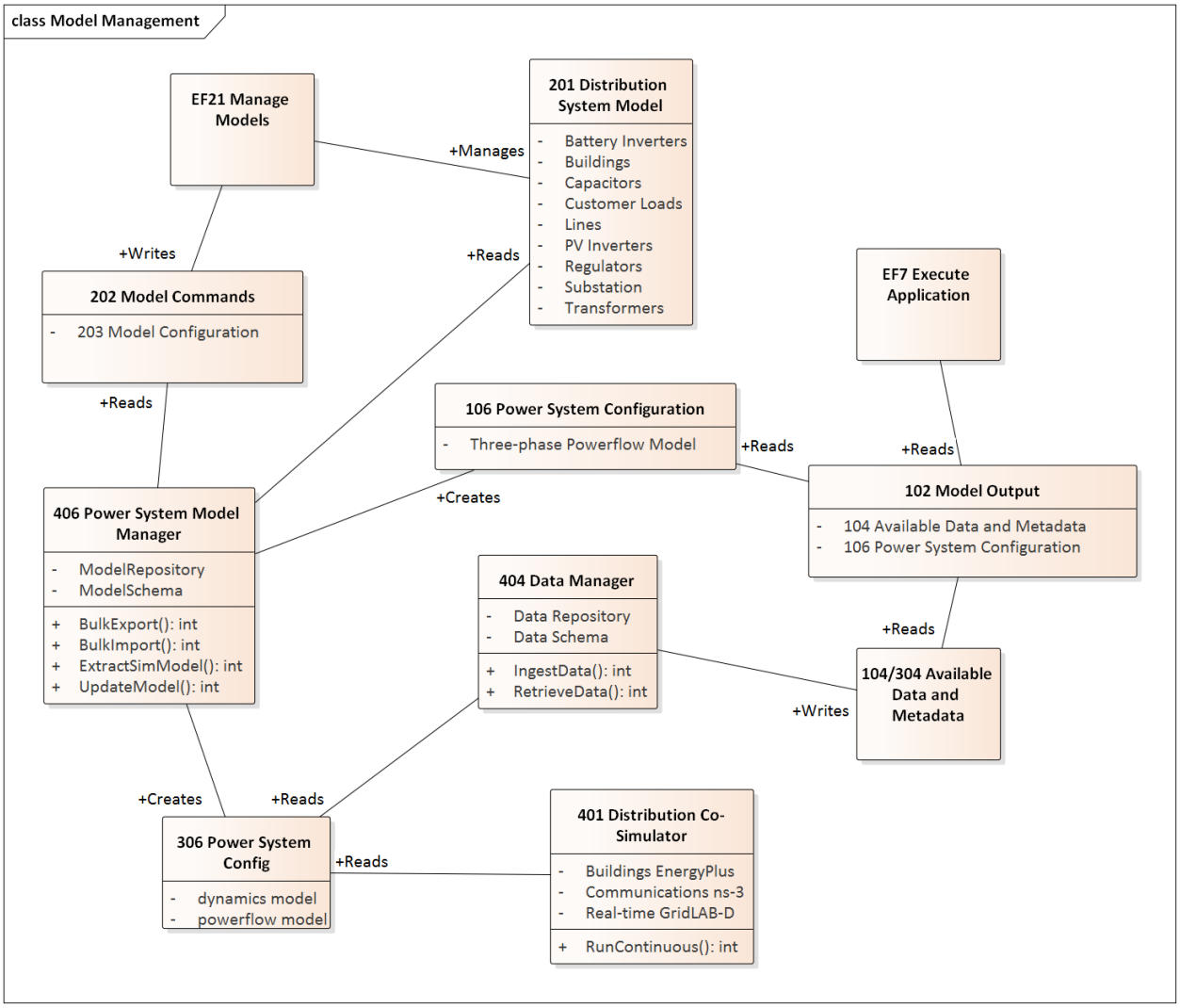



Figure 3: Internal model management

Figure 4 shows the internal Platform Manager flow when running application tests. Compared to the case of normal

usage in Figure 2, this example shows additional control and output for testing. The test commands include 203 and 204, as in Figure 2, but they also include:

- 205 Test Scripts, for the sequence of steps to perform
- 206 Test Configuration Setup, including initial conditions, etc.
- 207 Expected Results, for comparison to the actual output
- 210 Application Metadata, for information to run and instrument the application

The 403 Test Manager orchestrates the steps to run the application and collect results. As part of 103 Test Results, it will compare the real-time data (101/301) to the expected results in 207. If the testing user, EF8, requested logging, then the 409 Log Manager will create 109/309 System Logs for collection by 403 Test Manager. Logging is optional, and should have been requested as part of the 206 Test Config Setup or 204 Model Config Setup (this is not spelled out in the functional specification).

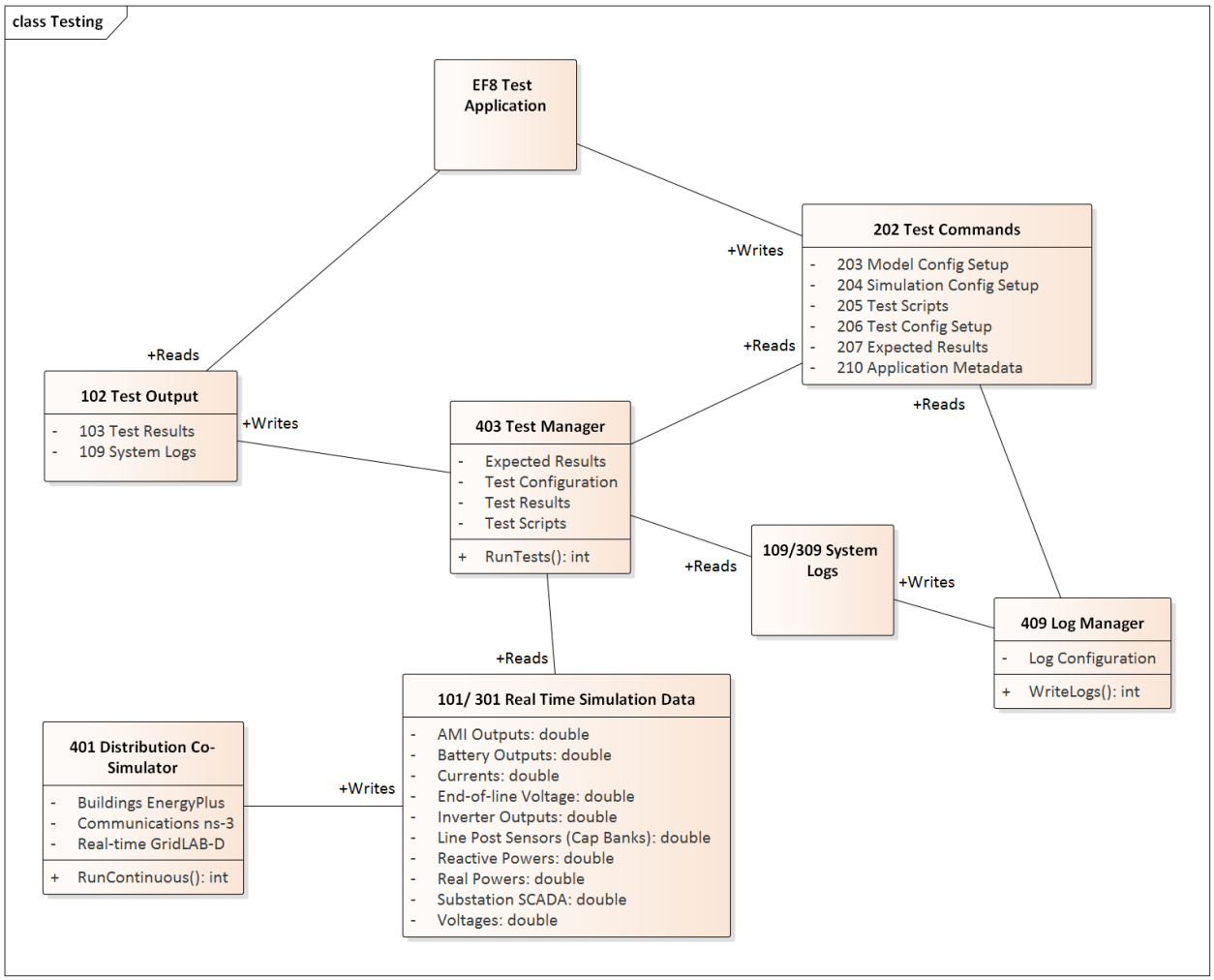



Figure 4: Testing an application or the platform

Figure 5 shows some of the internal 413 Platform Manager detail when a user, EF7, runs an application in debugging mode. Compared to Figure 2, there is much more internal output. The 212 Debug Configuration will include such things as breakpoints, watch variables, and logging requests. When run in debug mode, the 408 Debug Manager will collect the internal inputs and intermediate results from a variety of GridAPPS-D modules, including the simulator, services in use, model data, and access violations. The 404 Data Manager mediates most of this data collection (and with a change to the specification it could also mediate 101/301). The 408 Debug Manager combines this into 108

Intermediate Results, with 109 System Logs, for output to the user via the command interface. Depending on the implementation of GridAPPS-D, interactive debugging may also be supported, but is not shown in Figure 5.

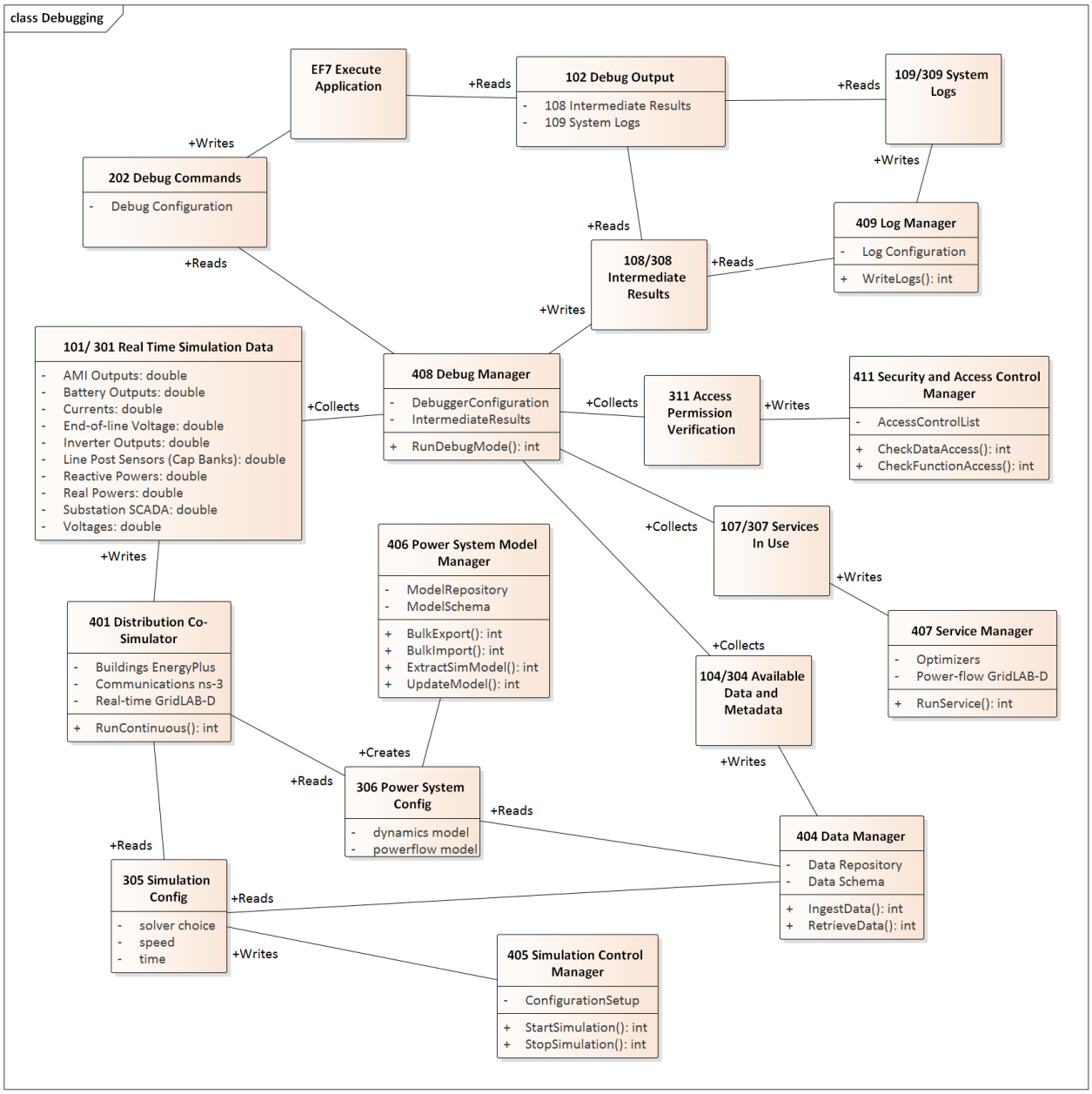



Figure 5: Debugging an application

Figure 6 shows the process of registering or updating an application to use with GridAPPS-D. The developer, in the role of EF13, must provide the application itself (211) along with the application data schema (208) and metadata (210). The data schema includes input and output parameters. The metadata includes a user-friendly name, description, calling parameters, command syntax, API functions used, etc. Using this information, 410 Application Hosting Manager will install and register the application, and its data, with 407 Service Manager and 404 Data Manager. After completing these steps, 412 Version Manager will output the current version information via the command interface; the current version includes information about which applications are installed along with the application versions.

In order to perform application management, EF13 also needs to provide user credentials to be checked against the 209 Access Control List. If these credentials are valid, the 411 SAC Manager will create 311 Access Permission

Verification for all of the internal Platform Manager components. In Figure 6, the 410 Application Hosting Manager can pass 311 to 404, 407 and 412 as needed. Although not shown earlier, SAC is actually incorporated into all GridAPPS-D processes this way.

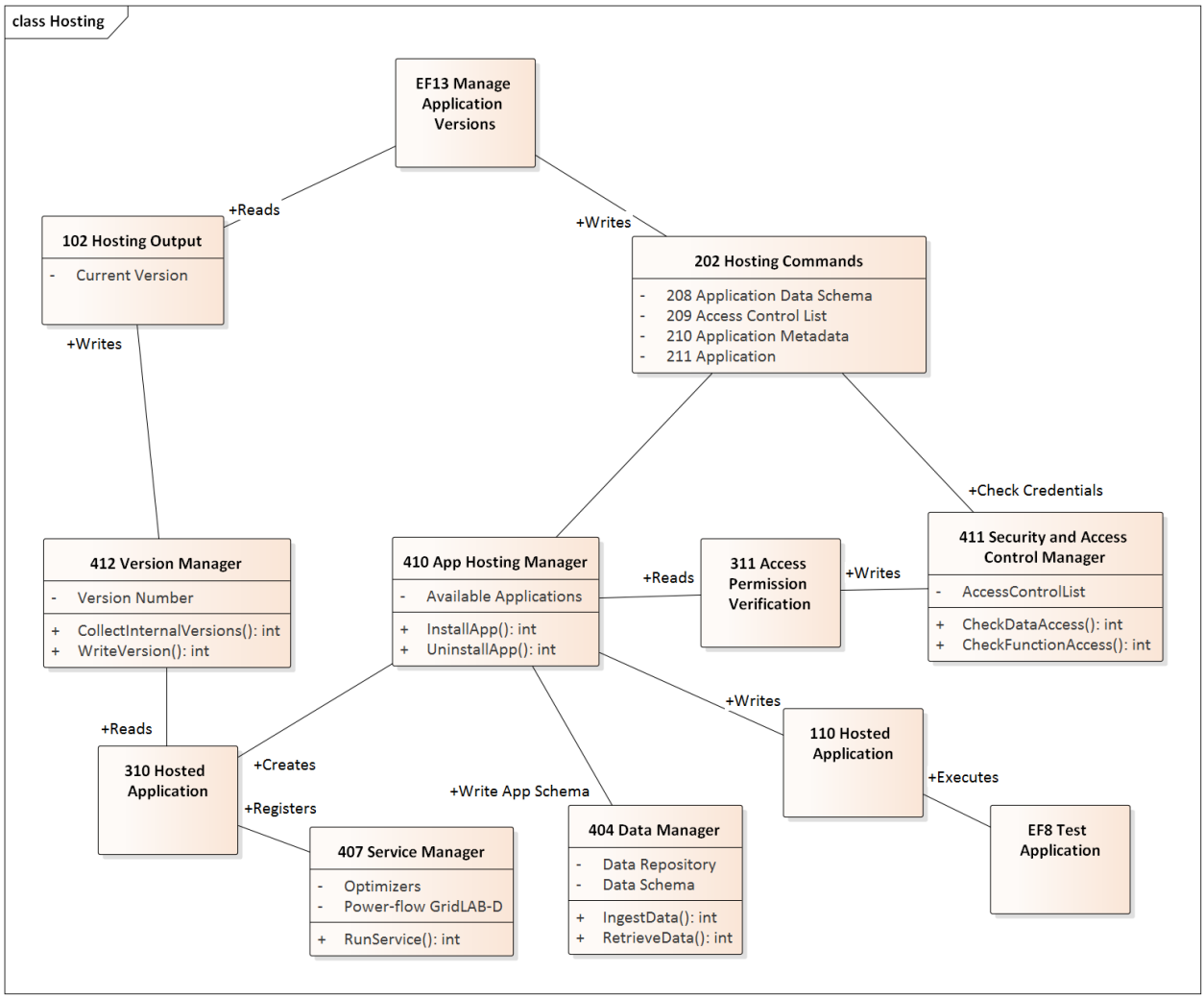



Figure 6: Hosting an application

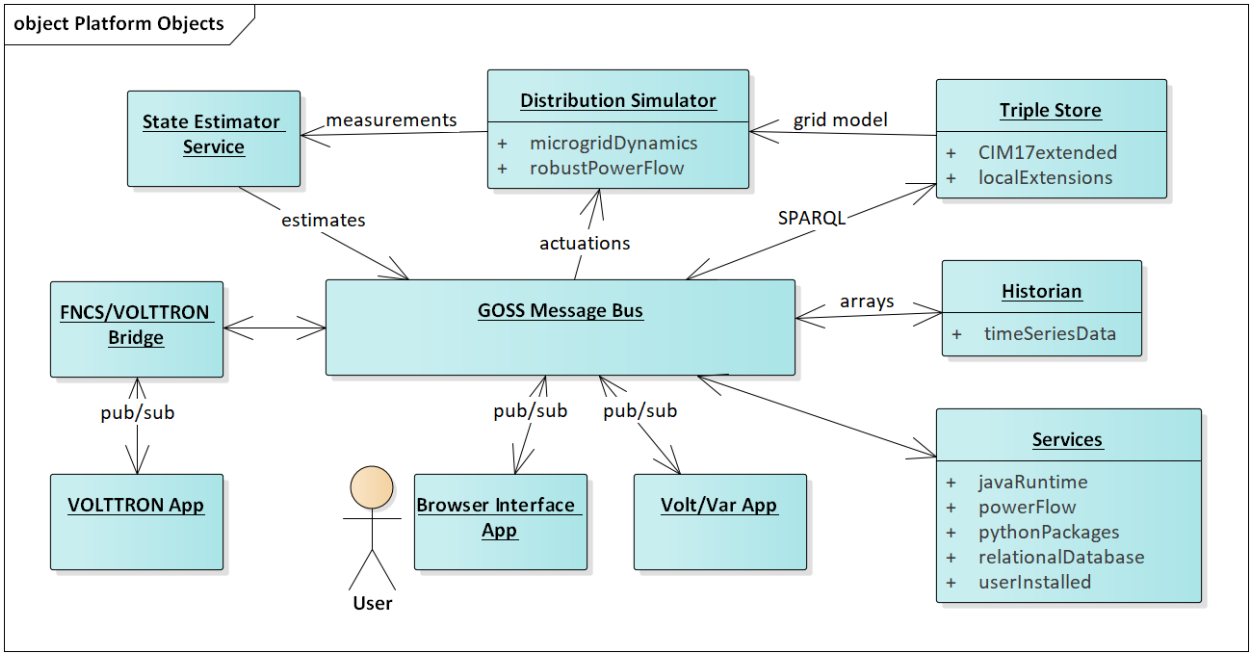



Figure 7: Platform Objects

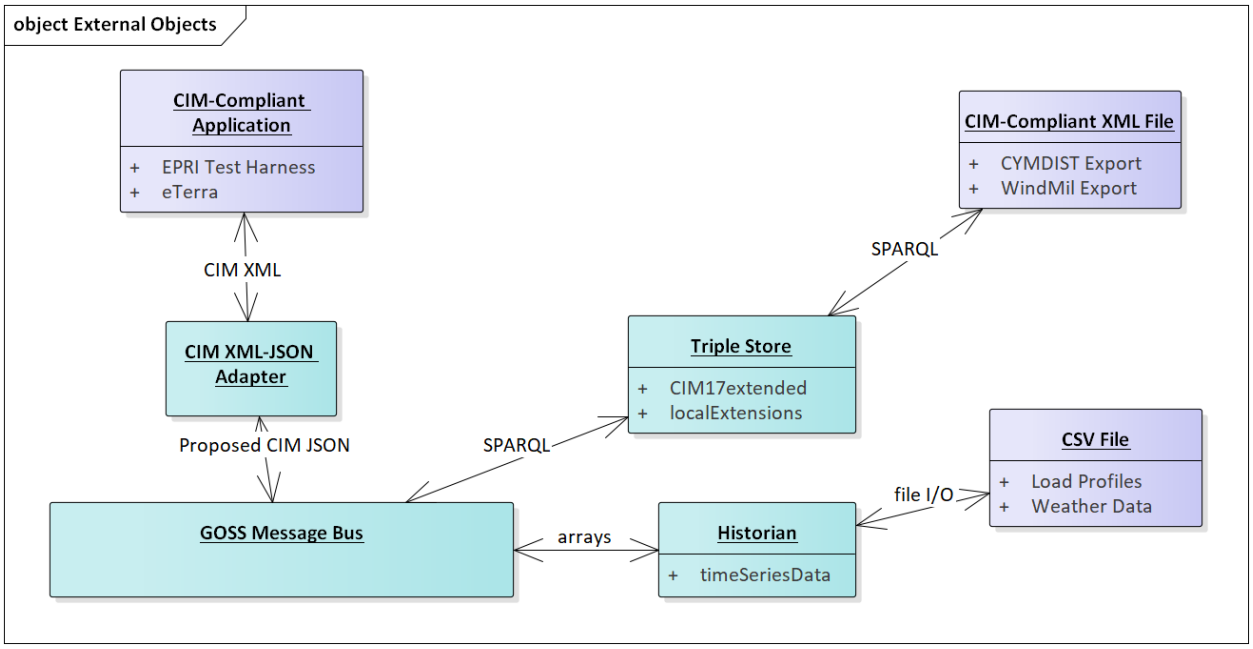



Figure 8: External Objects

### 5.5.2 UML for Release Cycle 4

Our objective is to demonstrate useful functionality, which is standards-compliant, by the end of March 2017. A simple heuristic VVO application will be running in GridAPPS-D. In terms of the Functional Requirements, we will be implementing:

- 102/202 Command Interface
- 301 Real-time Simulation Data

- 310 Hosted Application, but short-cutting the registration process
- 401 Distribution Co-Simulator (partial)
- 402 Process Manager (partial)
- 404 Data Manager (partial)
- 405 Simulation Manager (partial)
- 406 Power System Model Manager (partial)
- 413 Platform Manager (encapsulating 401 and 403-406)

This represents five out of twelve Internal Functions from the Functional Requirements, in partial form. The deadline leaves four months for detailed design and implementation, plus two months for documentation and testing. Therefore, we have chosen a minimal set of functions that can show end-to-end use of GridAPPS-D at the first milestone.

In developing the work breakdown structure (WBS), we noted that real-time simulation data is published with no time lags or errors in Release 1. However, data flow in a real DMS is affected by sensor and communication system performance, and also by the action of other subsystems. In Release 2, this might be addressed through some combination of:

- Communication and sensor models in the Distribution Co-Simulator
- Adding MDM and SCADA service attributes to the 407 Service Manager
- Filters on 301 Real-time Simulation Data

These decisions, and many others affecting Release 2 and Release 3, can be deferred until we gain experience developing Release 1.

Figure 1 shows the software components planned for Release 1. Most of these correspond to internal functions from the Functional Requirements, with some relatively minor re-factoring. The Power System Model Manager functionality has been split. The data store management and the creation of a complete GridLAB-D model appear at the bottom. Once the simulator is running, incremental changes are posted to the messaging bus.

Most of the "pink" components in Figure 1 are assigned to one task, except:

- The 310 VVO is a sub-task of the Command Interface, due to the close coupling of those efforts. The team on this task needs both power system and software skills.
- A separate task has been added for some project-level items.

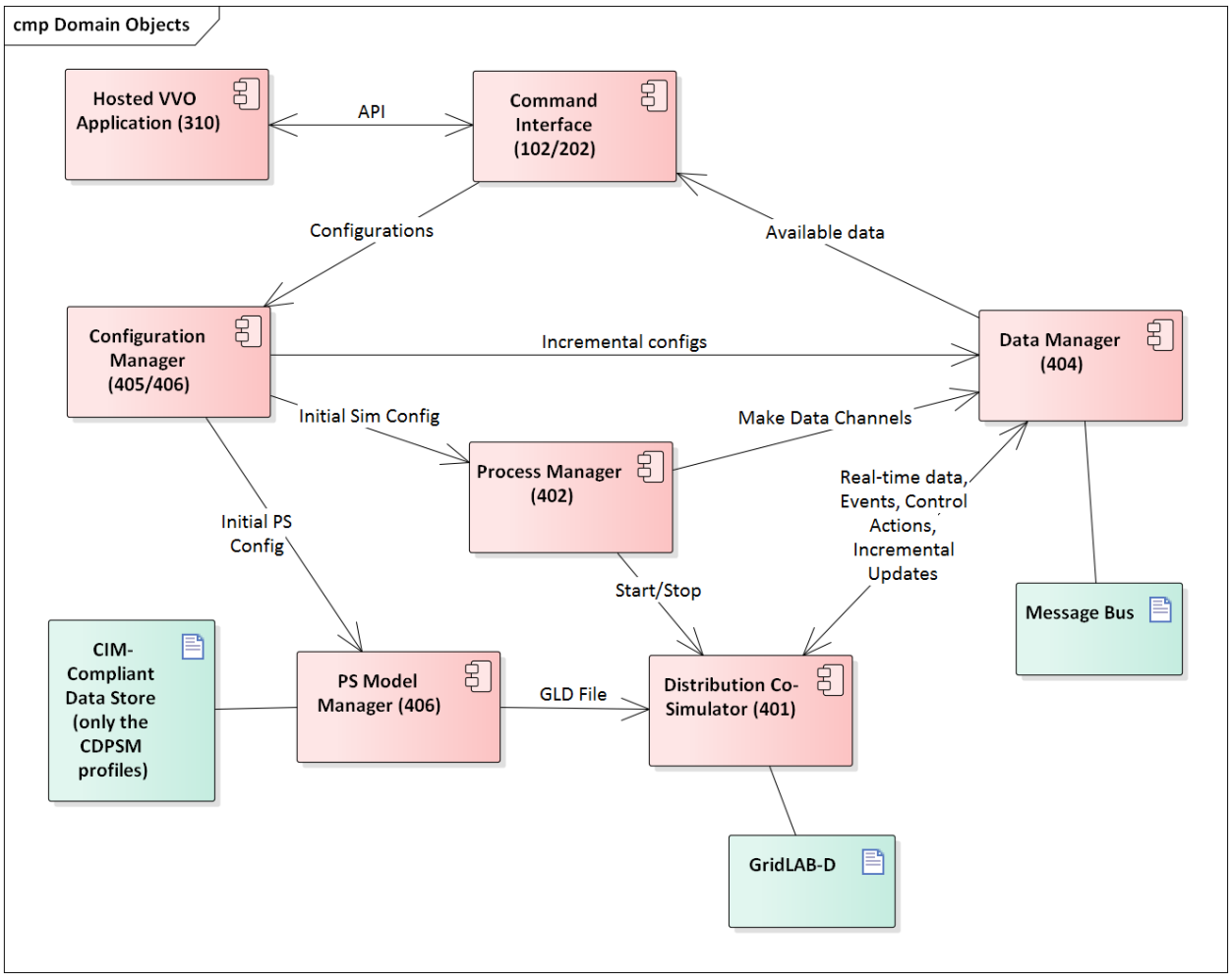



Figure 1: Component Diagram for GridAPPS-D Release 1

### 5.5.3 Initial Work Breakdown for Release Cycle 1

The Release 1 work breaks down into seven tasks, listed below. Three **critical items must be completed first**; these are highlighted in red. There are other inter-task dependencies that have not yet been called out. We plan to sequence the work over eight two-week "sprints" within the four months allocated for detailed design and development, using an agile process (Kanban).

1) Project-level Elements
    a. **Identify a power system model (note: IEEE-13 is already in CIM/CDPSM)**
    b. Design data store schema
    c. Manually ingest power system models
2) Command Interface
    a. Design APIs
        i. For all configurations in Task 4
        ii. For power system control actions (e.g. open/close switch)
    b. **Select one language binding** (e.g. Python, Java, C++, MATLAB) and implement

c. Develop a heuristic volt-var application (VVO) in the bound language

d. Integrate VVO into GridAPPS-D

3) Messaging and Data Manager

a. **Select a messaging framework (eg. ZeroMQ)**

b. Create communication APIs

c. Receives real-time data from simulator

d. Receives power system control actions

e. Handle communication between GridAPPS-D managers

f. Log messages to file

4) Configuration Manager (both Power System Config & Simulation Config)

a. Receive configurations from command interface over message bus

b. Translate configurations to native GridLAB-D

c. Translate and publish incremental update messages

d. Send configurations to Process Manager for simulation start

5) Process Manager

a. Receives configurations from the Configuration Manager

b. Send configuration to the Distribution Co-Simulator

c. Start Co-Simulation Process

d. Create simulation data channels and inform application

e. Stop simulation process

6) Distribution Co-Simulator (wraps GridLAB-D)

a. Accepts configurations from Process Manager

b. Start simulation

c. Produce and publish data in real time

d. Accept changes in real time (e.g. capacitor switching) via message bus

7) Power System Model Manager

a. Access the power system model in data store

b. Create native GridLAB-D file for initial loading into the simulator

## 5.6 CIM Validation

This section presents an overview of CIM Validation techniques that will be expanded upon in the future. The purpose of CIM validation is to assess the level of compliance GRIDAPPS-D is using in its use of CIM version 100.

### 5.6.1 Introduction

In electrical power distribution and transmission the Common Information Model (CIM) is a technology agnostic standard developed by the International Electrotechnical Commission (IEC). CIM provides the blueprints for application software like GridAPPS-D to represent data structures, message payloads, and information exchanges between applications.

To represent the model, CIM is written using the Unified Modeling Language (UML) using Sparx Enterprise Architect. The model is stored in a project file (*.eap extension file). GridAPPS-D extends the CIM to meet its application specific needs. UML (Object Management Group UML 2.5 Specification) profiles are secondary models, derived from the primary information model. Profiles represent a portion of the model to support GridAPPS-D application-specific structures and exchanges. A profile operates within the scope of the information model and is formed by extracting selected elements of the information model. These extracted elements are filtered or constrained by providing value ranges, reducing cardinality, and filtering structures.

UML profiles use stereotype notation <<>> to annotate information model elements such as classes, associations, and attributes in a target domain or technology. Stereotypes are either used to impose domain-specific criteria (for example, <<CIM:Datatype>>) or technical criteria (for example, <<table>>, <<primarykey>>) on UML elements. UML profiles are generated in a variety of ways including use of tools such as Enterprise Architect (EA) Schema Composer, CIMContextor, or CIMTool to define structural requirements. The W3C Shape Constraint Language (SHACL) will be used as extensions to further specify value constraints.

Currently GridAPPS-D is using UML diagrams as a human-readable intuitive profile description of application-specific uses of the extended CIM. Future profiles will be produced by CIMTool and SHACL to provide a more comprehensive blueprint for application development that is machine readable to compliment the diagrams.

### 5.6.2 Extending CIM

CIM is built on universally understood power grid concepts which means that the UML should be generally applicable, When the needs go beyond the general purpose solution it is possible to extend CIM for application specific purposes. When extending CIM to be compliant, the extentions comply with the rules and organization of the existing model. Otherwise an uncompliant application risks losing the advantage of using the standard, particularly for information exchanges.

Techniques for extending the CIM will not be discussed here, however the IEC TC 57 61970 part 301 document and the CIM Model Manager Guide (being released Spring 2020 by the TC57 CIM Model Managers) provides excellent guidance on best practices when extending the CIM.

### 5.6.3 Validation Techniques

### 5.6.4 Well-Formed UML Compliance

In the IEC TC 57 13, 14, and 16 Working Groups the CIM Model Managers are relied upon for any updates to the UML. Before a release occurs the JCleanCim tool (http://tanjakostic.org/jcleancim/index.html) is used to validate UML package, class, and associations against agreed upon rules for well-formed UML. The JCleanCim tool generates a log report citing any non-compliance items along with other products. It is basically like a software debug tool for CIM UML. For extensions the JCimClean tool can be used to review GridAPPS-D extensions and flag any problematic areas. In addition the JCimClean tool original log report for CIM100 can be compared against the GridAPPS-D CIMv100.

### 5.6.5 Well-Formed and Valid Profile

CIMTool can not only create Resource Description Framework Schema (RDFS) profiles from the CIM100 UML, it can also validate the generated profiles or created RDF datasets against CIM100 schema. GridAPPS-D has plans to extend the CIMTool Validation using SHACL to specify and constrain value ranges or check regular expression patterns

### 5.6.6 Final Thoughts

This section is expected to evolve in 2020 with the advancement of CIM Model Manager tools that were previously only accessible to the model managers or based on advancements of validation techniques in the profile development communities.

CHAPTER 6

# Data Model

## 6.1 IEEE 8500-Node Test Feeder

An IEEE Working Group specified a set of distribution test circuits [CIT1] and we have chosen the largest one of these as a sample circuit for GridAPPS-D [CIT2]. The 8500-Node test feeder operates at 12.47 kV and has a peak load of about 11 MW, including approximately 1100 single-phase, center-tapped transformers with triplex service drops. Loads are *balanced* between the two center-tapped windings.

The circuit includes 4 shunt capacitor banks and 4 voltage regulator banks, making it a reasonable test for solving voltage problems and for applying volt-var optimization (VVO). The circuit is also relatively lossy at peak load.

The model in GridAPPS-D came from the IEEE 8500-Node input files distributed with OpenDSS, exported to CIM from OpenDSS, and then imported to the GridAPPS-D data manager. In this automated process, four changes were implemented:

1. **Use constant-current load models, rather than constant-power load models**. This is necessary for the solution to converge at peak load. Voltages at peak load are low, and a constant-power load will draw more current under those conditions. Holding the current magnitude constant allows GridLAB-D to achieve convergence under a variety of operating conditions. This is an appropriate compromise in accuracy for real-time applications, which need to be robust through wide variations in voltage and load. In contrast, planning applications usually need more accurate load models, even at the possible expense of re-running some non-converged simulations.
2. **Disable automatic regulator and capacitor controls**. The volt-var application, described below, will supersede these settings. If a developer or user is testing the GridLAB-D model outside of GridAPPS-D, these control settings should be re-enabled in order to solve the circuit at peak load. That requires manual un-commenting edits to the GridLAB-D input file.
3. **Substitute a variable called VSOURCE for the SWING bus nominal voltage**. This needs to be set at 1.05 per-unit of nominal on the 115-kV system (i.e. 69715.065) in order to solve at peak load. Other conditions may require different source voltage values.
4. **Use a schedule for the loads** so they can vary with time during GridAPPS-D simulation. The file should be named *ziploads_schedule.player*.

CHAPTER 7

# Integrated Applications

## 7.1 Volt-var Optimization (VVO)

The sample VVO application is a Python implementation of a heuristic method that PNNL has investigated before [CIT3], [CIT4], [CIT5]. There are more advanced VVO methods that will be implemented in future work.

## 7.2 Visualization

We have created a web-based visualization of the sample VVO application. The visualization displays the topology of the IEEE 8500-Node system as an interactive graph. Capacitors and regulators are highlighted in the graph and displayed alongside tables with current values for capacitor status (OPEN or CLOSED), regulator voltage, and feeder power.

## 7.3 State Estimator Service

Given a perfect and complete set of voltage magnitude and angle measurements, along with a detailed and accurate power system model, one could calculate the real power, or any other electrical variable of interest, anywhere in the system. In practice, measurements have errors, time delays, and may even be missing. State estimation refers to the process of minimizing the errors and filling in gaps [1]. One state estimation method is called "weighted least squares", and it's analogous to drawing the best-fit line through a set of scattered points. Other methods may perform better [2]. Also, on distribution systems, it may be better to estimate branch currents instead of node voltages, but the principle is the same. In GridAPPS-D, the visualizations and applications ought to use the best available state estimator outputs, instead of raw SCADA values, for both accuracy and consistency. Therefore, the state estimator is not an application but a service in GridAPPS-D, sitting between emulated SCADA and the GOSS bus.

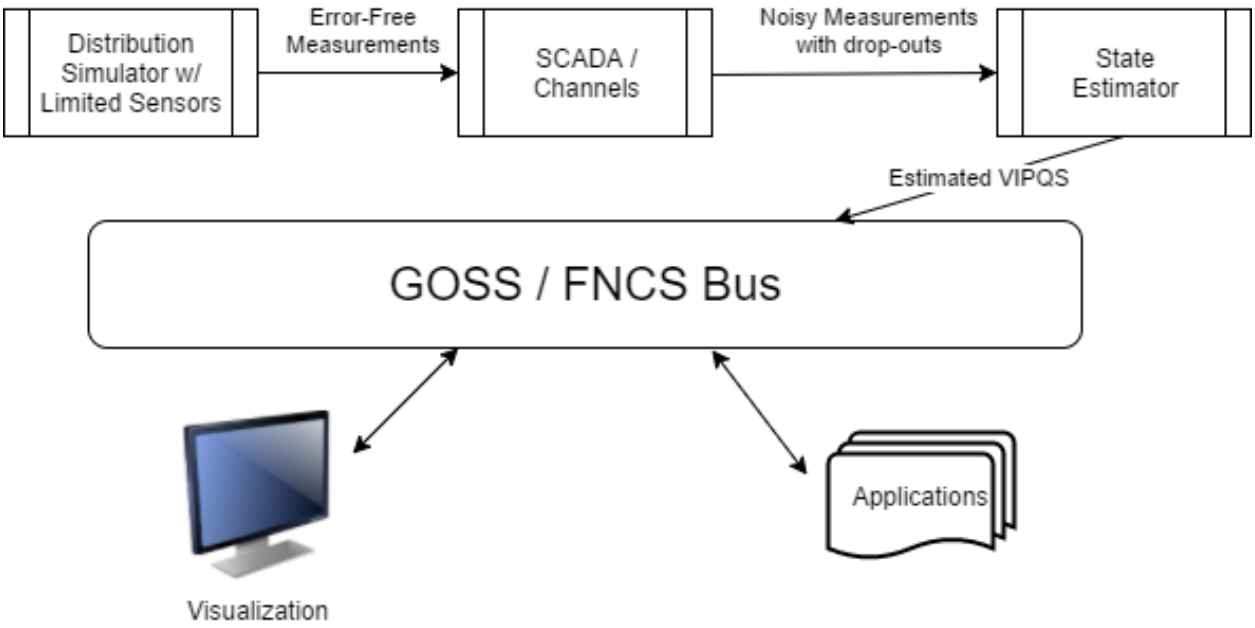



Figure 1: The state estimator processes noisy and incomplete measurements, then posting estimated voltage (V), current (I), real power (P), reactive power (Q) and switch status (S) values onto the GridAPPS-D message / data bus.

In Figure 1, the power system model (upper left) will include a limited number of sensors, corresponding to actual voltage and current transformers, line post sensors, wireless sensors, etc. In some scenarios, smart meters can also be sensors. Each such sensor will have different performance characteristics (e.g. precision, accuracy, sampling rate). Distribution systems typically do not have enough sensors to make the system observable, so there will be measurement gaps in the topology. The state estimator might fill these gaps with interpolation and graph-tracing methods on the power system model.

The supervisory control and data acquisition (SCADA) system in Figure 1 introduces more errors and failure points. Eventually, GridAPPS-D may simulate these impacts by federating ns-3 as a co-simulator. Until then, a placeholder module could be used to insert variable errors, time delays and dropouts in each measurement, whether due to sensor characteristics or the communication system. The output represents data as it would come into an operations center, and feeds the state estimator. Internally, the data flows between simulator, SCADA and state estimator might be implemented with FNCS, but this is an implementation detail. The state estimator will provide two outputs to the GOSS bus used by all GridAPPS-D applications:

1. At a time step configured by the platform, publish the best-estimate VIPQS values wherever sensors actually exist in the model, with quality attributes that still have to be established. Sensor locations delineate circuit segments, and note that all VIPQS values will be estimated at the boundaries, even if the sensor measures only V or I, for example.

2. Upon request by another application or service, publish the estimated VIPQS values for all nodes and components in the model, even at locations where no sensors exist. A variant is to publish the estimates only for selected nodes and components.

As indicated in Figure 1, other applications need to obtain estimated VIPQS values from the GOSS bus. Switch open/close states are a special case; they might be considered known values, but in practice the switch state is a measurement, which could lead to topology errors in the model. For GridAPPS-D, switch state estimates need to be a point of emphasis. Given that most distribution systems lack redundant measurements, It would be possible for an application to query these VIPQS values directly from the simulator or SCADA, bypassing the state estimator, but this is "cheating" in most situations. However, in the application development process, idealized VIPQS values could be obtained through a combination of two methods:

1. Add more sensors to the power system model

2. Set the sensor and channel errors to zero

Because the sensor outputs in GridAPPS-D come from a power flow solution that enforces Kirchhoff's Laws, the state estimator will produce ideally accurate values whenever the sensor and channel errors have been specified to be zero. The state estimator may still exhibit interpolation errors between sensor locations, but that is readily mitigated for testing purposes by adding more sensors.

With reference to RC1, the visualization and VVO applications should now subscribe to VIPQS values from the state estimator, not from the distribution simulator. They may also use or display quality metrics on the estimated values.

### 7.3.1 Design Objectives

State estimation is widely used in transmission system operations but is less common in distribution system operations due to a relatively limited value in traditional distribution systems, additional computational complexity, and a lack of sensors. Advanced distribution management platforms like GridAPPS-D provide access to model and sensor data that can be leveraged to overcome barriers to adoption and open the door to distribution system state estimators that are fast and accurate enough to be useful in utility operations.

A distribution system state estimator computes the most likely state given a set of present and/or past measurements. The full state of a distribution system consists of either the full set of complex bus voltages or the full set of complex branch currents; given the system model (admittance matrix), the remaining system parameters can be computed given the full system state.

### 7.3.2 Use Cases

- Assist power factor optimization: Utility objective is unity power-factor at the substation.
- Assist voltage optimization (planning): Utility objective is 1 p.u. voltage at last house primary.
- Real-time state estimation for advanced applications: applications can access the state estimate at a sufficient resolution to capture e.g. insolation variation caused by clouds.

### 7.3.3 Algorithms

State estimation uses system model information to produce an estimate of the state vector x given a measurement vector z. The measurement vector is related to the state vector and an error vector by the measurement function, which may be non-linear.

$$z = h(x) + e$$

Multiple formulations of the distribution system state estimation problem are possible:

1. *Node Voltage State Estimation (NVSE):* The state vector consists of node voltage magnitudes and angles for each node in the system (one reference angle can be eliminated from the state vector). This formulation of the state estimation problem is general to any topology and it is the standard for transmission system state estimation.
2. *Branch Current State Estimation (BCSE):* Radial topology and assumptions about shunt losses create a linear formulation of the state estimation problem. The state vector contains branch currents and, for a fully-constrained problem, requires one state per load, which can be less than the number of branches in the system.

Different algorithms provide different advantages for distribution system state estimation. A subset of the state estimation algorithms below will be used to achieve these goals.

1. *Weighted Least Squares Estimation (WLSE)*: a concurrent set of measurements are used to find a state vector that minimizes the weighted least squares objective function. The algorithm is memoryless with respect to previous solutions and measurements should be synchronized.

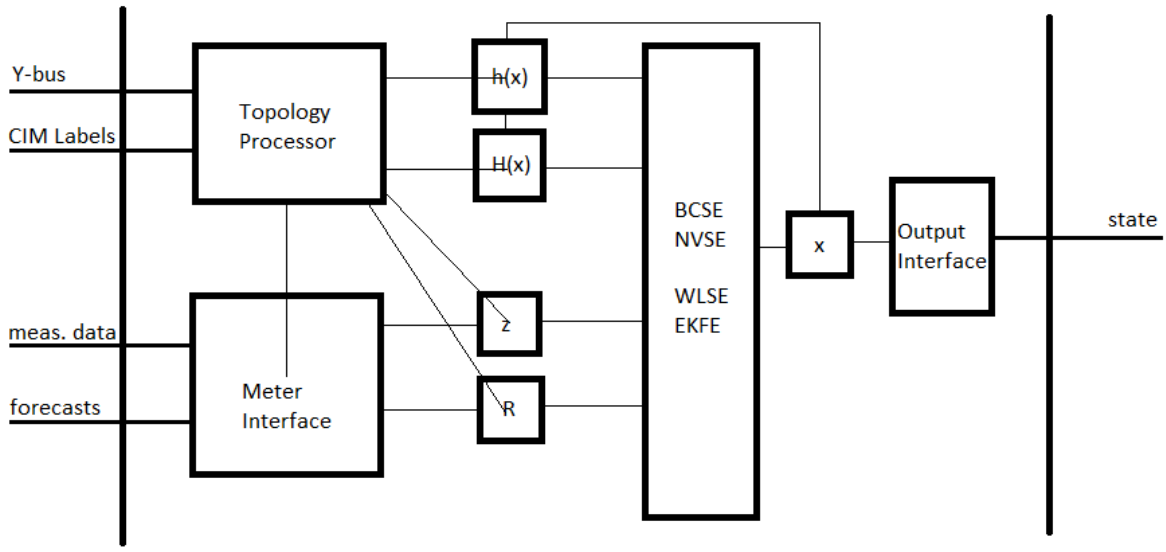
2. *Kalman Filter Estimation (KFE) and Extended Kalman Filter Estimation (EKFE)*: The Kalman filter provides a mechanism to consider past state estimates alongside present measurements. This provides additional noise rejection and allows asynchronous measurements can be considered individually. KFE is appropriate for linear BCSE and EKFE is compatible with nonlinear NVSE.
3. *Unscented Kalman Filter Estimation (UKFE)*: The unscented transform estimates the expected value and variance of the system state by observing the system outputs for inputs spanning the full dimensionality of the measurement space. Again, the Kalman filter provides a mechanism to consider past estimates.

### 7.3.4 TRL

The state estimator application will provide the capability to estimate the full system state using asynchronous measurement data. In addition a model order reduction technique will be implemented to greatly speed up the state estimation computation and to reduce the dependence on forecast-based pseudo-measurements. A paper (*Reduced-Order State Estimation for Power Distribution Systems with Sparse Sensing*) is targeted for IEEE Transactions on Power Systems.

### 7.3.5 Architecture

The state estimation service is being developed in c++. A modern c++ implementation allows the application to adapt to an evolving interface. The program architecture is shown below.




Topology Processor: initializes the measurement function and its Jacobian and determines the size of the measurement vector, the measurement covariance matrix, and the state vector.

Meter Interface: updates the measurement vector and the measurement covariance matrix as new measurement data comes available.

State Estimator: performs the state estimation operation according to the specified algorithm.

Output Interface: formats the state vector and any implicit states as an output stream.

### 7.3.6 Inputs

Upon initialization, the topology processor will receive the Y-bus from the GridLAB-D service and will query contextual information and sensor locations from the CIM database.

Periodic measurement data, including any forecasts to be used a pseudo-measurements will be required as inputs.

A "terminate" command from the platform will end the state estimation process.

### 7.3.7 Outputs

The output will include the full system state (node voltages and/or branch currents TBD).

### 7.3.8 Testing and Validation

**Evaluation metrics**

- State Error: compare state estimation output to "true" system state.
- Accuracy over baseline: compare state error of state estimator to state error of a QSTS load-flow model.
- Execution Time
- Bad Sensor Detection (binary)

**Scenarios**

- Full sensor deployment: verify that the true system state can be reproduced.
- Sparse sensor deployment: verify that the state estimator performs better than a QSTS load-flow model.
- Breaker trip: verify that switch state can be detected even when it is reported incorrectly.
- Bad sensor detection: verify that a sensor that is producing bad data can be identified.
- Dependent application support: verify that the state estimator can support e.g. the VVO application.
- Fault: for a radial system, determine the nearest common bus from multiple emulated customer calls.

### 7.3.9 Operating/Running

The state estimator will execute the topology processor at initialization and will enter a stat estimation loop. The state estimation loop will exit and the process will end upon receiving a 'terminate' command from the platform.

At initialization, a configuration file will be read for:

- State estimation mode (state vector and algorithm) selection
- Normalized residual threshold for bad measurement / sensor detection

### 7.3.10 References

[1] T. E. McDermott, "Grid Monitoring and State Estimation," in *Smart Grid Handbook*, ed: John Wiley & Sons, Ltd, 2016.

[2] A. Abur and A. Gomez Exposito, *Power system state estimation : theory and implementation*. New York, NY: Marcel Dekker, 2004.

[3] M. E. Baran and A. W. Kelley, "A branch-current-based state estimation method for distribution systems," in *IEEE Transactions on Power Systems*, vol. 10, no. 1, pp. 483-491, Feb 1995.

[4] Z. Jia, J. Chen and Y. Liao, "State estimation in distribution system considering effects of AMI data," *2013 Proceedings of IEEE Southeastcon*, Jacksonville, FL, 2013, pp. 1-6.

[5] S. C. Huang, C. N. Lu and Y. L. Lo, "Evaluation of AMI and SCADA Data Synergy for Distribution Feeder Modeling," in *IEEE Transactions on Smart Grid*, vol. 6, no. 4, pp. 1639-1647, July 2015.

[6] M. Kettner; M. Paolone, "Sequential Discrete Kalman Filter for Real-Time State Estimation in Power Distribution Systems: Theory and Implementation," in *IEEE Transactions on Instrumentation and Measurement*, vol.PP, no.99, pp. 1-13, Jun. 2017.

[7] G. Valverde and V. Terzija, "Unscented kalman filter for power system dynamic state estimation," in *IET Generation, Transmission & Distribution*, vol. 5, no. 1, pp. 29-37, Jan.

## 7.4 Model Validation Application

The state estimator basically attempts to fit measured data to a power flow model, usually assuming that the model is correct. However, a model attribute (e.g. line impedance) could also be estimated by minimizing its error residual in the state estimator's power flow solution. This process works best when applied to just one or a few suspect attributes, and/or when an archive is available to provide enough redundant measurements. The Model Validation Application will use these state estimator features off-line to help identify and correct the following types of model errors:

1. Unknown or incorrect service transformer sizes
2. Unknown or incorrect secondary circuit lengths
3. Incorrect phase identification of single-phase components
4. Phase wiring errors in line segments and switches
5. Transformer connection errors, especially reversed primary and secondary
6. Primary conductor sizes that don't decrease monotonically with distance from the source
7. Missing regulator and capacitor control settings (i.e. supply defaults from heuristic rules)
8. More than one of these on the same pole: recloser, line regulator, capacitor
9. Substation transformer impedance and turns ratio

These types of errors often appear upon the initial model import from a geographic information system (GIS), or in periodic model updates from GIS. Other error types may be added later. Many utilities do not have their secondary circuits modeled at all, but this has an important impact on AMI data. The service transformers and secondary circuits insert significant impedance between AMI meters and the primary circuit, where most of the other sensors are installed. Therefore, the first two items will require AMI data, and also enable its more effective use.

As shown in Figure 1, the Model Validator integrates with GridAPPS-D as a hosted application on the GOSS bus. Internally, it will use some of the same algorithms as the State Estimator and may share some code or binary files, but this is an implementation detail. It will need to access an archive of state-estimated VIPQS data, which may include AMI data. It will also use or incorporate an off-line power flow model, not the same one running in the GridAPPS-D distribution simulator. This may be EPRI's OpenDSS simulator [1]; compared to GridLAB-D, it's more tolerant of model errors and provides more diagnostic information about model errors.

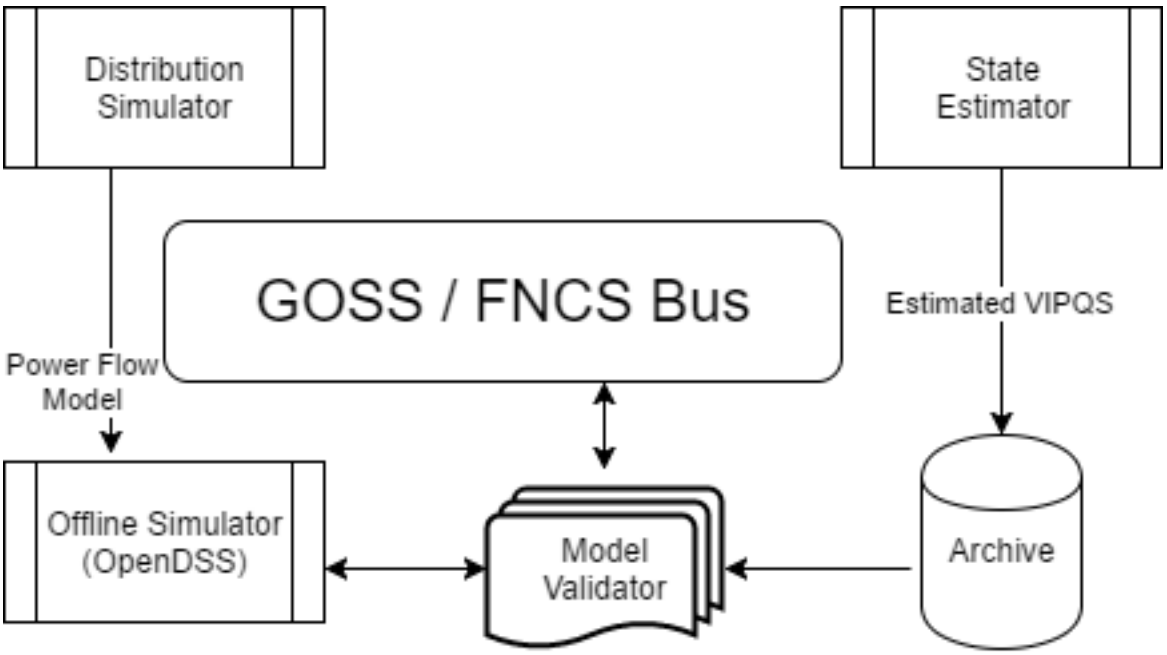



Figure 1: The Model Validator works with an archive from the state estimator, and an off-line power flow model.

### 7.4.1 Design Objectives

The model validator will detect and attempt to correct unreasonable component interconnections and network parameters. The model validation application will be implemented in Python.

### 7.4.2 Use Cases

- Valid transformer size and orientation (Utility): orientation is not captured explicitly in their GIS system.
- Discover secondary line impedance parameters (Utility) conductor type and line length are currently based on generic assumptions.
- Sanity check or estimate transformer size and impedance.
- Verify that the nominal voltage of nodes matches the base voltage of the segment: generally the winding voltage of the upstream transformer or swing bus voltage.
- Sanity check conductor sizes and line current ratings.
- Validate and fill in regulator and capacitor control settings.
- Check phase continuity (GridLAB-D may not model phase discontinuities)

### 7.4.3 Inputs

The model validator will have access to the CIM database and archived data from the state estimator.

### 7.4.4 Outputs

The model validator will one or both of the following outputs:

- Model status: log file or GUI pipe for identified issues.
- Model correction: CIM updates to correct identified issues.

### 7.4.5 Testing and Validation

**Evaluation metrics**

- Ability to detect known issues.

**Scenarios**

- Utility merger: models with different format may be interpreted differently, creating issues a CIM model.
- Data entry issue: model update does not match upgrade performed in the field

### 7.4.6 Operating/Running

The model validator script will execute once when called by the platform.

At initialization, a configuration file will be read for:

- Mode (status, quiet, verbose; see outputs section)
- Selectable validation items (use cases)

### 7.4.7 References

[1] R. C. Dugan and T. E. McDermott, "An open source platform for collaborating on smart grid research," in *Power and Energy Society General Meeting, 2011 IEEE*, 2011, pp. 1-7.

## 7.5 Transactive Systems Application

Transactive energy is a method of controlling loads and resources on the distribution system, combining both market and electrical principles [1]. One reason for including this application in DOE-funded GridAPPS-D is that PNNL has made several technical contributions and led several demonstration projects in transactive systems, also funded by DOE [2].

**Application structure**

This transactive systems application is to be implemented as a modularized 2-layer 3-level structure, as seen from Figure 3. The layer decomposition helps the control of various groups, with limited information flow between different layers. With the predefined functions in each agent type (Agent A, B, and C) in each level, the existing transactive system related work can be conveniently integrated into the application, and the new control features can be added into specific control function in each type of the agent easily.

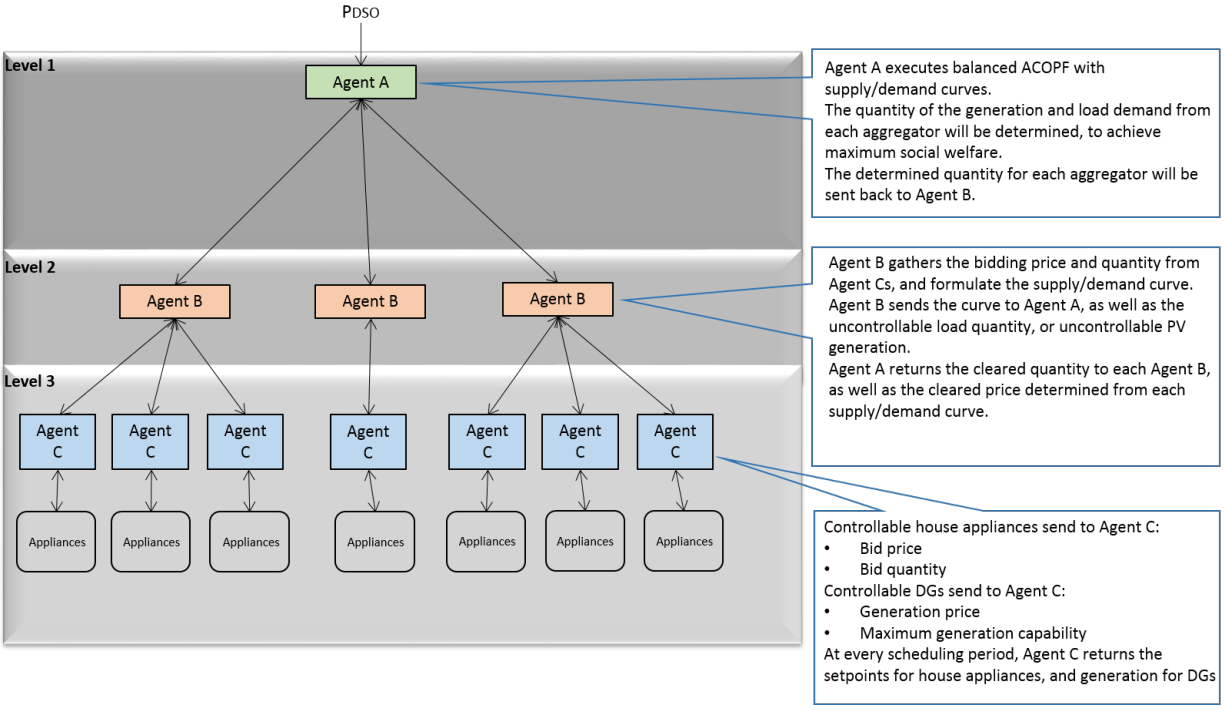



Figure 3: The structure of the modularized 2-layer 3-level transactive system application

The modularized agents opens the door for integrating different control mechanisms into the application. Users need to consider which level their control algorithm fits into, and fill in the control function of the Agent class in that level, without worrying about communications between the agents. In each level, the same type of the agent may have various control functions, which help combining benefits of different control schemes together.

Agent A, B and C will be implemented as VOLTTRON applications. VOLTTRON is an application platform for distributed sensing and control applications [3]. With the capability of hardware-in-the-loop (HIL) testing through VOLTTRON, the transactive systems application will be tested using the actual devices. A GOSS-VOLTTRON Bridge is to be implemented, for the communication between GridAPPS-D and the VOLTTRON agents in the transactive systems application.

**Application test cases**

The hierarchical control framework introduced in [4] for integrated coordination between distributed energy resources and demand response will be implemented into the application. In addition, [4] has not considered the power losses or power constrains, which will be taken into consideration in this test case. The two-layer control mechanism, including the coordination layer and device layer, fits the proposed structure of the application well. The control in each level will be implemented into corresponding function in each type of the agent. The IEEE 123-node test feeder built in GridLAB-D will be used for testing the application.

**CIM extension for the Application**

The latest versions of GridAPPS-D has used a reduced-order CIM to support feeder modeling. With transactive system application included into GridAPPS-D platform, more objects, such as house air conditioner and water heater, need to be defined in CIM. Before the definition in CIM, a simplified version of the house object and water heater object are to be implemented in GridLAB-D.

### 7.5.1 References

[1] Gridwise Architecture Council. (2017). *Transactive Energy*. Available: http://www.gridwiseac.org/about/transactive_energy.aspx

[2] Pacific Northwest National Laboratory. (2017). *Transactive Energy Simulation Platform (TESP)*. Available: http://tesp.readthedocs.io/en/latest/

[3] S. Katipamula, J. Haack, G. Hernandez, B. Akyol, and J. Hagerman, "VOLTTRON: An Open-Source Software Platform of the Future," *IEEE Electrification Magazine,* vol. 4, pp. 15-22, 2016.

[4] Di Wu, Jianming Lian, Yannan Sun, Tao Yang, Jacob Hansen, "Hierarchical control framework for integrated coordination between distributed energy resources and demand response," Electric Power Systems Research, pp. 45-54, May 2017.

## 7.6 Distribution Optimal Power Flow for Real-Time Setpoint Dispatch

### 7.6.1 Objectives

This application is designed to address the problem of optimizing the operation of aggregations of heterogeneous energy resources connected to a distribution system. We will focus on real=time optimization method and the power setting points of the distributed energy resources (DERs) will be updated on a second or subsecond timescale to maximize the operational objectives while coping with the variability of ambient conditions and noncontrollable energy assets [1]. In order to avoid massive measurements and overcome the limitation caused by model inaccuracy, this application will be implemented in a distributed manner, and only local measurements and a feedback signal from the substation aggregator are needed to determine the optimal setpoints for each controlled DER unit.

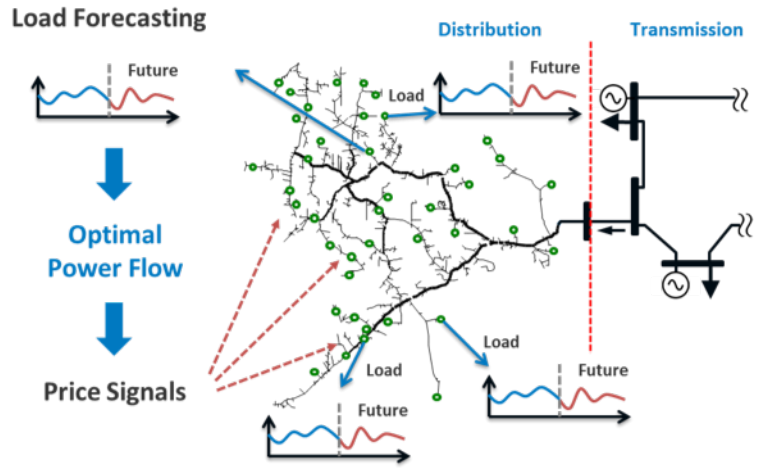



**Figure 1 The conceptual framework of distribution OPF for real=time setpoint dispatch.**

Figure 1 shows the conceptual framework of this application, and this application is targeting at TRL 3.

### 7.6.2 Design

Figure 2 describes the overall work flow of the application. Distribution OPF algorithm requires real=time measurements, distribution system model and power flow results, which will be obtained from GridAPPS=D platform through GOSS/FNCS message bus. The optimization problem formulation can be constructed using user=defined cost functions for different controllable devices. Finally the optimal setpoints for controllable devices will be solved based on the feedback information from system measurements. These setpoints will be sent back to GridLab=D grid model to update DER operations. Such a closed=loop control forms the control iteration for the studied time point, and new setpoints for the following time points will be determined in the same manner using the updated model and measurements.

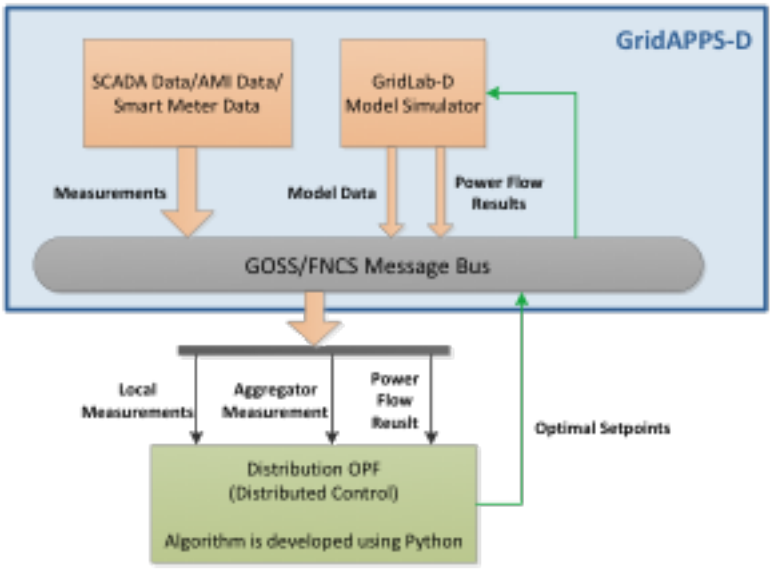



**Figure 2 The workflow of real=time setpoint dispatch application and its interaction with GridApps=D.**

### 7.6.3 Data requirements

The DER application requires p and q values from the inverters attached to PVs, loads, and capacitors. The DER application also requires setting the p and q values of inverters attached to PVs.

### 7.6.4 Testing and Validation

Evaluation metrics of this application:

- Real/reactive power at the substation
- System loss
- Voltages across the entire distribution grid: voltage magnitude, voltage fluctuation, voltage unbalance.
- Legacy control device operations: total control actions of all capacitors and regulators

Scenarios:

- Optimal Dispatch for Distributed PV Systems
- Optimal Dispatch for Distributed PV + Energy Storage
- Etc. (will be added when implementing the application)

### 7.6.5 Operating/Running

This application will be developed using Python.

### 7.6.6 References

[1] E. Dall'Anese, A. Bernstein, and A. Simonetto, "Feedback=based Projected=gradient Method for Real=time Optimization of Aggregations of Energy Resources," IEEE Global Conference on Signal and Information Processing (GlobalSIP), Montreal, Canada, Nov. 2017.

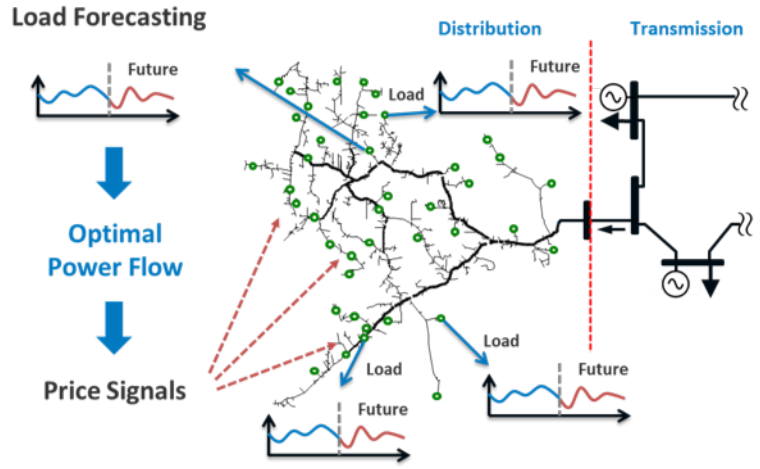
## 7.7 Short-Term Grid Forecasting

### 7.7.1 Objectives

Today's distribution systems have been experiencing a significant transformation due to an increasing amount of smart electric loads and distributed energy resources, such as electric vehicles, smart home appliances, rooftop photovoltaic systems, and energy storage. As more flexible resources integrated into distribution systems, coordination among various flexible resources plays an important role in distribution system operations to optimally manage distribution assets and flexible resources. On the other hand, with the increasing penetration of variable energy resources, it is crucial for system operators to not only monitor and estimate the current grid conditions but also forecast the future system status, which allows for proactive dispatch of controllable resources and better preparation for ever-changing grid conditions. This application develops predictive locational marginal prices (DLMPs) to proactively manage the distribution assets and flexible resources based on forecasted grid conditions. This application enables the distribution system operators to optimally incentivize individual resources to achieve system-level control objectives, such as minimizing total generation cost and optimizing the voltage profile; and it paves the way to a fully functional distribution market with granular prices that reflect the time- and location-specific values of individual participants.

### 7.7.2 Design

This application develops a high-resolution, short-term load forecasting method to accurately predict the power consumption of individual customers in distribution systems. Using historical load measurements as inputs, it trains a support vector regression model to forecast the future load. Based on the forecasted load in the short-term future, this application develops a three-phase AC optimal power flow problem to determine the predictive DLMPs in distribution systems. By accurately modeling the losses and the imbalances of distribution networks, it provides time- and location-specific pricing of individual resources.




### 7.7.3 Operating/Running

The application was built using Python 3.6. It will be run from the platform GUI.

### 7.7.4 References

[1] H. Jiang, Y. Zhang, E. Muljadi, J. J. Zhang, and D. W. Gao, "A Short-Term and High-Resolution Distribution System Load Forecasting Approach Using Support Vector Regression with Hybrid Parameters Optimization," in IEEE

Transactions on Smart Grid, vol. 9, no. 4, pp. 3341-3350, July 2018. [2] R. Yang and Y. Zhang, "Three-Phase AC Optimal Power Flow Based Distribution Locational Marginal Price," IEEE Innovative Smart Grid Technologies, Arlington, VA, Apr. 2017.

## 7.8 Solar Forecasting Application

### 7.8.1 Objectives

When observations of solar radiation are limited, persistence and smart persistence solar forecasting techniques are frequently the easiest and most effective methods to use. Often though, these techniques suffer from having no information about current cloud properties, which could improve the forecast. At NREL, a Physics-based Smart Persistence model (PSPI) [1] was created for intra-hour solar forecasting using only GHI observations and a cloud retrieval technique. This model breaks down common solar radiation components such as GHI and solar zenith angle (SZA) using a two-stream approximation [2] and methods used in [3] to forecast future GHI, cloud fraction, and cloud albedo. With this information, this technique can then be used to forecast solar power (still in development). Figure 1 below shows the conceptual framework for PSPI.

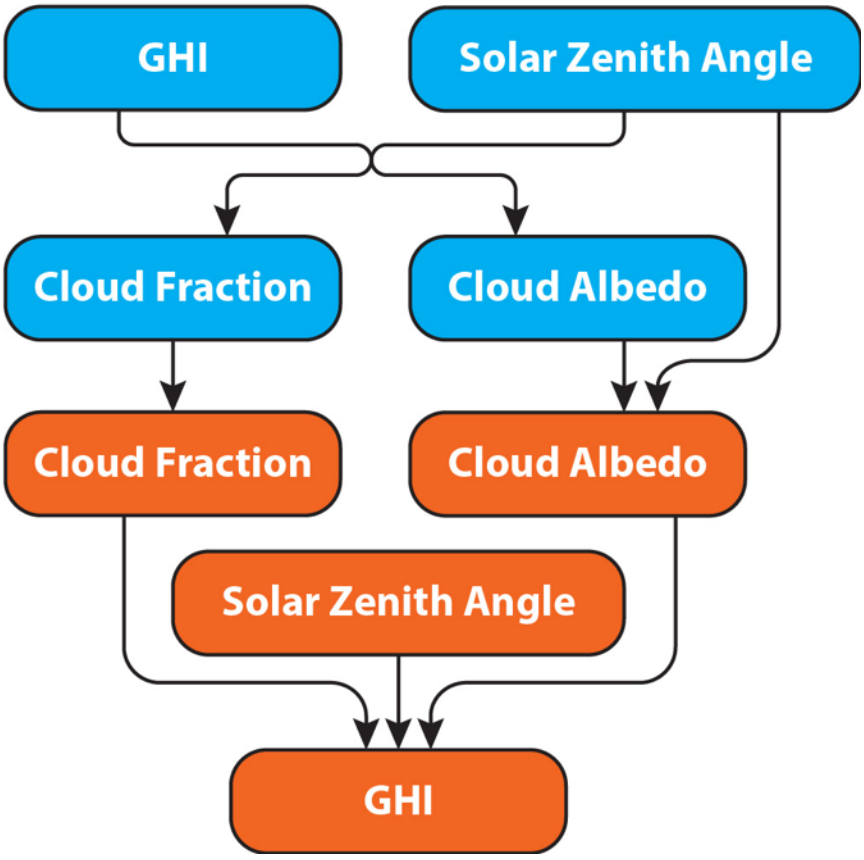



**Figure 1 Conceptual framework for PSPI. PSPI breaks up the GHI and solar zenith angle (SZA) into cloud fraction and cloud albedo components.**

### 7.8.2 Design

PSPI is designed to operate only using GHI observations. Other atmospheric parameters, such as pressure and temperature, can be ingested into the application as well if those observations exist. Currently, site-specific annual averages of these parameters are used. Other parameters, such as altitude of the site of interest, need to be adjusted prior to running

the application. Whatever atmospheric variables are available via the GridAPPS-D platform, they can be ingested into the PSPI based on the user's needs (still in development). Once all desired parameters are chosen and the application is run, intra-hour GHI forecasts can be made (5 to 60-minute forecasts) as frequently as observations arrive (usually minutely).

### 7.8.3 Testing and Validation

PSPI was tested and validated using 10 years of GHI data at the Solar Radiation Research Laboratory (SRRL) at NREL, in Golden, Colorado. More information about this process can be found in [1].

### 7.8.4 Operating/Running

The application was built using Python 3.6. It will be run from the platform GUI.

### 7.8.5 References

[1] Kumler, A., Xie, Y., & Zhang, Y. (2019). A Physics-based Smart Persistence model for Intra-hour forecasting of solar radiation (PSPI) using GHI measurements and a cloud retrieval technique. Solar Energy, 177, 494-500.

[2] Sagan, C., & Pollack, J. B. (1967). Anisotropic nonconservative scattering and the clouds of Venus. Journal of Geophysical Research, 72(2), 469-477.

[3] Xie, Y., & Liu, Y. (2013). A new approach for simultaneously retrieving cloud albedo and cloud fraction from surface-based shortwave radiation measurements. Environmental Research Letters, 8(4), 044023.

CHAPTER 8

# API Documentation

## 8.1 GridAPPS-D

## 8.2 GOSS

The GridOPTICS Software System (GOSS) manages the platform data and message bus; its overall design is described in [CIT6].

## 8.3 FNCS

The Framework for Network Co-simulation (FNCS) manages the time clock and message traffic between platform simulators; its overall design is described in [CIT7]. For API documentation see https://github.com/FNCS/fncs/wiki .

## 8.4 VVO

## 8.5 GridLAB-D

GridLAB-D is the distribution grid simulator within the platform; its overall design is described in [CIT8].

## 8.6 gov.pnnl.gridlabd.cim

This Java package converts CIM RDF to GridLAB-D format.

### 8.6.1 CDPSM_to_GLM

public class **CDPSM_to_GLM**
This class converts CIM (IEC 61968) RDF to GridLAB-D format

The general pattern is to retrieve iterators on the different types of objects (e.g. ACLineSegment) through simple SPARQL queries. Usually these iterators include just the mrID, or the mrID and name. Then Jena RDF model and resource functions are used to pull other properties from iterated objects, writing GridLAB-D input along the way. A future version will rely more heavily on SPARQL queries to do the selection and filtering, as the preferred pattern for developers working with CIM. In existing code, the EnergySource most closely follows this new preferred pattern.

Invoke as a console-mode program

**Author** Tom McDermott

**See also:** *CDPSM_to_GLM.main*, CIM User Group, CIM Profile and Queries for Feeder Modeling in GridLAB-D, GridLAB-D

#### Fields

#### baseURI

static final String **baseURI**
identifies gridlabd

#### mapNodes

static HashMap<String, *GldNode*> **mapNodes**
to look up nodes by name

#### mapSpacings

static HashMap<String, *SpacingCount*> **mapSpacings**
to look up line spacings by name

#### neg120

static final Complex **neg120**
Rotates a phasor -120 degrees by multiplication

#### nsCIM

static final String **nsCIM**
namespace for CIM; should match the CIM version used to generate the RDF

#### nsRDF

static final String **nsRDF**
namespace for RDF

### pos120

static final Complex **pos120**
Rotates a phasor +120 degrees by multiplication

### ptBaseNomV

Property **ptBaseNomV**

### ptEqBaseV

Property **ptEqBaseV**

### ptEquip

Property **ptEquip**

### ptLevBaseV

Property **ptLevBaseV**

### Methods

### AccumulateLoads

static boolean **AccumulateLoads**(*GldNode nd*, String *phs*, double *pL*, double *qL*, double *Pv*, double *Qv*, double *Pz*, double *Pi*, double *Pp*, double *Qz*, double *Qi*, double *Qp*)
Distributes a total load (pL+jqL) among the phases (phs) present on GridLAB-D node (nd)

**Parameters**

- **nd** – GridLAB-D node to receive the total load
- **phs** – phases actually present at the node
- **pL** – total real power
- **qL** – total reactive power
- **Pv** – real power voltage exponent from a CIM LoadResponseCharacteristic
- **Qv** – reactive power voltage exponent from a CIM LoadResponseCharacteristic
- **Pz** – real power constant-impedance percentage from a CIM LoadResponseCharacteristic
- **Qz** – reactive power constant-impedance percentage from a CIM LoadResponseCharacteristic
- **Pi** – real power constant-current percentage from a CIM LoadResponseCharacteristic
- **Qi** – reactive power constant-current percentage from a CIM LoadResponseCharacteristic
- **Pp** – real power constant-power percentage from a CIM LoadResponseCharacteristic
- **Qp** – reactive power constant-power percentage from a CIM LoadResponseCharacteristic

**Returns** always true

### Bus_ShuntPhases

static String **Bus_ShuntPhases** (String *phs*, String *conn*)
appends N or D for GridLAB-D loads and capacitors, based on wye or delta connection

**Parameters**

- **phs** – from CIM PhaseCode
- **conn** – contains *w* for wye connection and *d* for delta connection

**Returns** phs with N or D possibly appended

### CFormat

static String **CFormat** (Complex *c*)

**Parameters**

- **c** – complex number

**Returns** formatted string for GridLAB-D input files with 'j' at the end

### Count_Phases

static int **Count_Phases** (String *phs*)
from the phase string, determine how many are present, but ignore D, N and S

**Parameters**

- **phs** – the parsed CIM PhaseCode

**Returns** (1..3)

### FindBaseVoltage

static double **FindBaseVoltage** (Resource *res*, Property *ptEquip*, Property *ptEqBaseV*, Property *ptLevBaseV*, Property *ptBaseNomV*)
Returns the nominal voltage for conduction equipment, from either its own or container's base voltage. For example, capacitors and transformer ends have their own base voltage, but line segments don't.

**Parameters**

- **res** – an RDF resource corresponding to a ConductingEquipment instance; we need to find its base voltage
- **ptEquip** – an RDF property corresponding to the EquipmentContainer association
- **ptEqBaseV** – an RDF property corresponding to a possible BaseVoltage association on the equipment itself
- **ptLevBaseV** – an RDF property corresponding to the EquipmentContainer's BaseVoltage association
- **ptBaseNomV** – an RDF property corresponding to the nominalVoltage attribute of a CIM BaseVoltage

**Returns** the nominal voltage as found from the equipment or its container, or 1.0 if not found

### FindConductorAmps

static String **FindConductorAmps** (Model *mdl*, Resource *res*, Property *ptDataSheet*, Property *ptAmps*)
needs to return the current rating for a line segment 'res' that has associated WireInfo at 'ptDataSheet', which in turn has the current rating at ptAmps

TODO - this is not implemented; emitted syntax is for OpenDSS and the function call (below, in main) needs review

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **res** – an RDF resource corresponding to a CIM ACLineSegment
- **ptDataSheet** – an RDF property corresponding to CIM AssetDatasheet attribute
- **ptAmps** – an RDF property corresponding to CIM ratedCurrent attribute

**Returns** unusable OpenDSS input

### FirstPhase

static String **FirstPhase** (String *phs*)

**Parameters**

- **phs** – a parsed CIM PhaseCode

**Returns** the first phase found as A, B, or C

### GLDCapMode

static String **GLDCapMode** (String *s*)
translate the capacitor control mode from CIM to GridLAB-D

**Parameters**

- **s** – CIM regulating control mode enum

**Returns** MANUAL, CURRENT, VOLT, VAR

### GLD_ID

static String **GLD_ID** (String *arg*)
parse the GridLAB-D name from a CIM name, based on # position

**Parameters**

- **arg** – the CIM IdentifiedObject.name attribute, not the mrID

**Returns** the compatible name for GridLAB-D

### GLD_Name

static String **GLD_Name** (String *arg*, boolean *bus*)
convert a CIM name to GridLAB-D name, replacing unallowed characters and prefixing for a bus/node

**Parameters**

- **arg** – the root bus or component name, aka CIM name
- **bus** – to flag whether *nd_* should be prepended

**Returns** the compatible name for GridLAB-D

### GetACLineParameters

static String **GetACLineParameters** (Model *mdl*, String *name*, Resource *r*, double *len*, double *freq*, String *phs*, PrintWriter *out*)
for a standalone ACLineSegment with sequence parameters, find GridLAB-D formatted and normalized phase impedance matrix

TODO - this is always three-phase, so we don't need all 7 variations from GetSequenceLineConfigurations

-

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **name** – the root name of the line segment and its line_configuration
- **r** – an RDF resource corresponding to a CIM ACLineSegment
- **len** – the length of the ACLineSegment in feet
- **freq** – frequency in Hz for converting susceptance to capacitance
- **phs** – phasing for the written line_configuration (one of 7 variations) that needs to be referenced
- **out** – the PrintWriter instance opened from the main program, passed here so that we can share code in GetSequenceLineConfigurations

**Returns** the name of the written line_configuration

### GetBusName

static String **GetBusName** (Model *mdl*, String *eq_id*, int *seq*)
finds the bus (ConnectivityNode) name for conducting equipment

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file*
- **eq_id** – the CIM mrID of the conducting equipment
- **seq** – equals 1 to use the first terminal found, or 2 to use the second terminal found

**Returns** the GridLAB-D compatible bus name, or *x* if not found. As Terminals no longer have sequence numbers, the ordering of seq is unpredictable, so if there are two we can get bus 1 - bus 2 or bus 2 - bus 1

### GetBusPositionString

static String **GetBusPositionString** (Model *mdl*, String *id*)
for a bus (ConnectivityNode), search for X,Y geo coordinates based on connected Terminals and equipment

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **id** – name of the bus to search from

**Returns** X,Y coordinates in comma-separated value (CSV) format

### GetCableData

static String **GetCableData** (Model *mdl*, Resource *res*)
needs to return underground_line_conductor data in GridLAB-D format

TODO - this is not implemented; the emitted syntax is actually for OpenDSS

-

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **res** – an RDF resource corresponding to a CIM CableInfo (not a leaf/concrete class)

**Returns** unusable OpenDSS input

### GetCapControlData

static String **GetCapControlData** (Model *mdl*, Resource *rCap*, Resource *ctl*)

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rCap** – an RDF resource corresponding to a CIM LinearShuntCompensator (aka capacitor)
- **ctl** – an RDF resource corresponding to the CIM RegulatingControl that was found attached to the LinearShuntCompensator

**Returns** the embedded capacitor control data for a GridLAB-D capacitor object

### GetEquipmentType

static String **GetEquipmentType** (Resource *r*)
find the type of monitored equipment for controlled capacitors, usually a line or the capacitor itself

**Parameters**

- **r** – an RDF resource, will have a CIM mrID, should be a LinearShuntCompensator, ACLineSegment, EnergyConsumer or PowerTransformer

**Returns** cap, line, xf if supported in GridLAB-D; NULL or ##UNKNOWN## if unsupported

### GetGldTransformerConnection

static String **GetGldTransformerConnection** (String[] *wye*, int *nwdg*)

Map CIM connectionKind to GridLAB-D winding connections. TODO: some of the returnable types aren't actually supported in GridLAB-D

**Parameters**

- **wye** – array of CIM connectionKind attributes per winding
- **nwdg** – number of transformer windings, also the size of wye

**Returns** the GridLAB-D winding connection. This may be something not supported in GridLAB-D, which should be treated as a feature request

### GetImpedanceMatrix

static String **GetImpedanceMatrix** (Model *mdl*, String *name*, Property *ptCount*, Resource *r*, boolean *bWantSec*)

Convert CIM PerLengthPhaseImpedance to GridLAB-D line_configuration

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **name** – root name of the line_configuration(s), should be the CIM name
- **r** – an RDF resource, will have a CIM mrID, should be PerLengthPhaseImpedance
- **ptCount** – an RDF property for the PerLengthPhaseImpedance.conductorCount
- **bWantSec** – flags the inclusion of triplex, true except for debugging

**Returns** the GridLAB-D formatted impedance matrix for a line configuration. We have to write 3 of these in the case of 1-phase or 2-phase matrices. If (by name) it appears to be triplex and bWantSec is false, nothing will be returned.

### GetLineSpacing

static String **GetLineSpacing** (Model *mdl*, Resource *rLine*)

needs to return the line_spacing and wire/cncable/tscable assignments for this rLine in GridLAB-D format

TODO - this is not implemented, the emitted syntax is actually for OpenDSS

-

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rLine** – an RDF resource corresponding to a CIM ACLineSegment that should have an associated AssetInfo

**Returns** unusable OpenDSS input

### GetMatIdx

static int **GetMatIdx**(int *n*, int *row*, int *col*)
converts the [row,col] of nxn matrix into the sequence number for CIM PerLengthPhaseImpedanceData (only valid for the lower triangle) *

**Parameters**

- **n** – 2x2 matrix order
- **row** – first index of the element
- **col** – second index

**Returns** sequence number

### GetPowerTransformerData

static String **GetPowerTransformerData**(Model *mdl*, Resource *rXf*)

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rXf** – an RDF resource corresponding to CIM PowerTransformer; it should have mesh impedance data

**Returns** transformer and transformer_configuration objects in GridLAB-D format

### GetPowerTransformerTanks

static String **GetPowerTransformerTanks**(Model *mdl*, Resource *rXf*, ResIterator *itTank*, boolean *bWantSec*)
writes a PowerTransformer in GridLAB-D format, in the case where individual tranformer tanks that are connected together in a bank. GridLAB-D supports only 2-winding banks with same phasing on both sides, or single-phase, center-tapped secondary transformers.

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rXf** – an RDF resource corresponding to a CIM PowerTransformer that uses tank modeling
- **itTank** – a Jena iterator on the tanks associated with rXf, known to be non-empty before this function is called
- **bWantSec** – usually true, in order to include single-phase, center-tapped secondary transformers, which would come to this function

**Returns** transformer object in GridLAB-D format; the transformer_configuration comes from calling GetXfmrCode

### GetPropValue

static String **GetPropValue**(Model *mdl*, String *uri*, String *prop*)
unprotected lookup of uri.prop value, to be deprecated in favor of SafeProperty

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **uri** – an RDF resource, currently only an EquipmentContainer is used, and it should always exist
- **prop** – currently only IdentifiedObject.name is used, and it should always exist

**Returns** the name of the CIM object

### GetRegulatorData

static String **GetRegulatorData**(Model *mdl*, Resource *rXf*, String *name*, String *xfGroup*, String *bus1*, String *bus2*, String *phs*)

Connects a regulator in GridLAB-D format between bus1 and bus2; should be called from GetPowerTransformerTanks. In CIM, a regulator consists of a transformer plus the ratio tap changer, so if such is found, should call GetRegulatorData instead of just writing the transformer data in GetPowerTransformerTanks. Any impedance in the regulating transformer will be lost in the GridLAB-D model. Should be called from PowerTransformers that have RatioTapChangers attached, so we know that lookup will succeed

TODO: implement regulators for tank transformers

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rXf** – an RDF resource corresponding to a CIM PowerTransformer that has a RatioTapChanger associated
- **name** – the name of the PowerTransformer (already looked up before calling this function)
- **xfGroup** – the PowerTransformer's IEC vector group (already looked up before calling this function)
- **bus1** – first bus (ConnectivityNode) on the regulator (already looked up before calling this function)
- **bus2** – second bus (ConnectivityNode) on the regulator (already looked up before calling this function)
- **phs** – phases that contain A, B and/or C (already looked up before calling this function)

**Returns** regulator and regulator_configuration objects in GridLAB-D format

### GetSequenceLineConfigurations

static String **GetSequenceLineConfigurations**(String *name*, double *sqR1*, double *sqX1*, double *sqC1*, double *sqR0*, double *sqX0*, double *sqC0*)

For balanced sequence impedance, return a symmetric phase impedance matrix for GridLAB-D. We have to write 7 variations to support all combinations of 3, 2 or 1 phases used.

**Parameters**

- **name** – is the root name for these 7 variations
- **sqR1** – positive sequence resistance in ohms/mile
- **sqX1** – positive sequence reactance in ohms/mile
- **sqC1** – positive sequence capacitance in nF/mile
- **sqR0** – zero sequence resistance in ohms/mile

- **sqX0** – zero sequence reactance in ohms/mile
- **sqC0** – zero sequence capacitance in nF/mile

**Returns** text for 7 line_configuration objects

### GetWdgConnection

static String **GetWdgConnection** (Resource *r*, Property *p*, String *def*)
parse the CIM WindingConnection enumeration

**Parameters**

- **r** – an RDF resource, will have a CIM mrID, should be a transformerEnd
- **p** – an RDF property, will be a CIM attribute, should be connectionKind
- **def** – default value if property is not found, such as Y

**Returns** D, Y, Z, Yn, Zn, A or I

### GetWireData

static String **GetWireData** (Model *mdl*, Resource *res*)
needs to return overhead_line_conductor data in GridLAB-D format; res is the CIM OverheadWireInfo instance

TODO - this is not implemented; the emitted syntax is actually for OpenDSS

-

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **res** – an RDF resource corresponding to CIM OverheadWireInfo

**Returns** unusable OpenDSS input

### GetXfmrCode

static String **GetXfmrCode** (Model *mdl*, String *id*, double *smult*, double *vmult*, boolean *bWantSec*)
Translates a single TransformerTankInfo into GridLAB-D format. These transformers are described with short-circuit and open-circuit tests, which sometimes use non-SI units like percent and kW, as they appear on transformer test reports.

TODO: smult and vmult may be removed, as they should always be 1 for valid CIM XML

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **id** – CIM mRID corresponding to a CIM TransformerTankInfo
- **smult** – scaling factor for converting winding ratings to volt-amperes (should be 1)
- **vmult** – scaling factor for converting winding ratings to volts (should be 1)
- **bWantSec** – usually true to include single-phase, center-tapped secondary tranformers, which come to this function

**Returns** transformer_configuration object in GridLAB-D format

### GldPrefixedNodeName

static String **GldPrefixedNodeName** (String *arg*)
prefix all bus names with *nd_* for GridLAB-D, so they "should" be unique

**Parameters**

- **arg** – the root bus name, aka CIM name

**Returns** nd_arg

### MergePhases

static String **MergePhases** (String *phs1*, String *phs2*)
accumulate phases without duplication

### Phase_Kind_String

static String **Phase_Kind_String** (String *arg*)
parses a single phase from CIM SinglePhaseKind

**Parameters**

- **arg** – CIM SinglePhaseKind enum

**Returns** A, B, C, N, s1 or s2

### Phase_String

static String **Phase_String** (String *arg*)
parses the phase string from CIM phaseCode

**Parameters**

- **arg** – CIM PhaseCode enum

**Returns** some combination of A, B, C, N, s1, s2, s12

### SafeBoolean

static boolean **SafeBoolean** (Resource *r*, Property *p*, boolean *def*)
look up Jena boolean value

**Parameters**

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

**Returns** boolean value, or default if not found

### SafeDouble

static double **SafeDouble**(Resource *r*, Property *p*, double *def*)
look up Jena double value

**Parameters**

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

**Returns** double value, or default if not found

### SafeInt

static int **SafeInt**(Resource *r*, Property *p*, int *def*)
look up Jena integer value

**Parameters**

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

**Returns** integer value, or default if not found

### SafePhasesX

static String **SafePhasesX**(Resource *r*, Property *p*)
look up Jena phase property

**Parameters**

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute

**Returns** phases in string format, or ABCN if not found

### SafeProperty

static String **SafeProperty**(Resource *r*, Property *p*, String *def*)
look up Jena string property

**Parameters**

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

**Returns** the property (or default value) as a string

### SafeRegulatingMode

static String **SafeRegulatingMode** (Resource *r*, Property *p*, String *def*)
parse the CIM regulating control mode enum

**Parameters**

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

**Returns** voltage, timeScheduled, reactivePower, temperature, powerFactor, currentFlow, userDefined

### SafeResName

static String **SafeResName** (Resource *r*, Property *p*)
for components (not buses) returns the CIM name from r.p attribute if it exists, or the r.mrID if not, in GridLAB-D format

**Parameters**

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute

**Returns** a name compatible with GridLAB-D

### SafeResourceLookup

static String **SafeResourceLookup** (Model *mdl*, Property *ptName*, Resource *r*, Property *p*, String *def*)

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **ptName** – should be the IdentifiedObject.Name property of the resource we are looking for
- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

**Returns** the GridLAB-D formatted name of a resource referenced by r.p

### Shunt_Delta

static boolean **Shunt_Delta** (Resource *r*, Property *p*)
for loads and capacitors, returns true only if CIM PhaseShuntConnectionKind indicates delta

**Parameters**

- **r** – an RDF resource, will have a CIM mrID, should be LinearShuntCompensator or EnergyConsumer
- **p** – an RDF property, will be a CIM attribute for phaseConnection

**Returns** true if delta connection

### WirePhases

static String **WirePhases**(Model *mdl*, Resource *r*, Property *p1*, Property *p2*)

Returns GridLAB-D formatted phase string by accumulating CIM single phases, if such are found, or assuming ABC if not found. Note that in CIM, secondaries have their own phases s1 and s2. *

**Parameters**

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **r** – an RDF resource, will have a CIM mrID, should be something that can have single phases attached
- **p1** – an RDF property, will be a CIM attribute, should associate from a single phase back to r
- **p2** – an RDF property, will be a CIM attribute, should be the single phase instance's phase attribute

**Returns** concatenation of A, B, C, s1 and/or s2 based on the found individual phases

### main

public static void **main**(String[] *args*)

Reads command-line input for the converter

**Parameters**

- **args** – will be CDPSM_to_GLM [options] input.xml output_root

**Throws**

- **java.io.FileNotFoundException** – if the CIM RDF input file is not found

**Options**:

-l={0..1} load scaling factor, defaults to 1

-t={y|n} triplex; y/n to include or ignore secondaries. Defaults to yes. Use no for debugging only, as all secondary load will be ignored.

-e={u|i} encoding; UTF-8 or ISO-8859-1. No default, so this should be specified. Choose 'u' if the CIM file came frome OpenDSS.

-f={50|60} system frequency; defaults to 60

-v={1|0.001} multiplier that converts CIM voltage to V for GridLAB-D; defaults to 1

-s={1000|1|0.001} multiplier that converts CIM p,q,s to VA for GridLAB-D; defaults to 1

-q={y|n} are unique names used? If yes, they are used as unique GridLAB-D names. If no, the CIM mrID is de-mangled to create a unique GridLAB-D name, but this option is only implemented for ACLineSegments as written to some earlier GIS profiles.

-n={schedule_name} root filename for scheduled ZIPloads (defaults to none)

-z={0..1} constant Z portion (defaults to 0 for CIM-defined LoadResponseCharacteristic)

-i={0..1} constant I portion (defaults to 0 for CIM-defined LoadResponseCharacteristic)

-p={0..1} constant P portion (defaults to 0 for CIM-defined LoadResponseCharacteristic)

**Example:** java CDPSM_to_GLM -l=1 -e=u -i=1 ieee8500.xml ieee8500

Assuming Jena and Commons-Math are in Java's classpath, this will produce two output files:

1. **ieee8500_base.glm** with GridLAB-D components for a constant-current model at peak load. This file includes an adjustable source voltage and manual capacitor/tap changer states. It should be invoked from a separate GridLAB-D file that sets up the clock, solver, recorders, etc. For example, these two GridLAB-D input lines set up 1.05 per-unit source voltage on a 115-kV system:
   - #define VSOURCE=69715.065 // 66395.3 * 1.05
   - #include "ieee8500_base.glm"

   If there were capacitor/tap changer controls in the CIM input file, that data was written to ieee8500_base.glm as comments, which can be recovered through manual edits.
2. **ieee8500_busxy.glm** with bus geographic coordinates, used in GridAPPS-D but not GridLAB-D

**Cautions:** this converter does not yet implement all variations in the CIM for unbalanced power flow.

1. AssetInfo links to WireSpacing, OverheadWireInfo, ConcentricNeutralCableInfo and TapeShieldCableInfo
2. PerLengthSequenceImpedance has not been tested
3. Capacitor power factor control mode - not in GridLAB-D
4. Capacitor user-defined control mode - not in GridLAB-D
5. Capacitor controlled by load (EnergyConsumer) - need to name loads
6. Line ratings for PerLengthImpedance
7. Dielectric constant (epsR) for cables - not in CIM
8. Soil resistivity (rho) for line impedance - not in CIM
9. Multi-winding transformers other than centertap secondary-not in GridLAB-D
10. Unbalanced transformer banks - not in GridLAB-D
11. Autotransformers have not been tested
12. schedule_name implemented for secondary loads only, primary loads to be done
13. Fuse not implemented
14. Breaker not implemented
15. Jumper not implemented
16. Disconnector not implemented

**Throws**

- **java.io.UnsupportedEncodingException** – if the UTF encoding flag is wrong

**See also:** *CDPSM_to_GLM*

### 8.6.2 CDPSM_to_GLM.GldNode

static class **GldNode**

Helper class to accumulate nodes and loads.

All EnergyConsumer data will be attached to node objects, then written as load objects. This preserves the input ConnectivityNode names

TODO - another option is to leave all nodes un-loaded, and attach all loads to parent nodes, closer to what OpenDSS does

### Fields

#### bDelta

public boolean **bDelta**
will add N or D phasing, if not S

#### bSecondary

public boolean **bSecondary**
if bSecondary true, the member variables for phase A and B loads actually correspond to secondary phases 1 and 2. For GridLAB-D, these are written to phase AS, BS or CS, depending on the primary phase, which we find from the service transformer or triplex.

#### bSwing

public boolean **bSwing**
denotes the SWING bus, aka substation source bus

#### name

public final String **name**
root name of the node (or load), will have *nd_* prepended

#### nomvln

public double **nomvln**
this nominal voltage is always line-to-neutral

#### pa_i

public double **pa_i**
real power on phase A or s1, constant current portion

#### pa_p

public double **pa_p**
real power on phase A or s1, constant power portion

#### pa_z

public double **pa_z**
real power on phase A or s1, constant impedance portion

### pb_i

public double **pb_i**
real power on phase B or s2, constant current portion

### pb_p

public double **pb_p**
real power on phase B or s2, constant power portion

### pb_z

public double **pb_z**
real power on phase B or s2, constant impedance portion

### pc_i

public double **pc_i**
real power on phase C, constant current portion

### pc_p

public double **pc_p**
real power on phase C, constant power portion

### pc_z

public double **pc_z**
real power on phase C, constant impedance portion

### phases

public String **phases**
ABC allowed

### qa_i

public double **qa_i**
reactive power on phase A or s1, constant current portion

### qa_p

public double **qa_p**
reactive power on phase A or s1, constant power portion

### qa_z

public double **qa_z**
reactive power on phase A or s1, constant impedance portion

### qb_i

public double **qb_i**
reactive power on phase B or s2, constant current portion

### qb_p

public double **qb_p**
reactive power on phase B or s2, constant power portion

### qb_z

public double **qb_z**
reactive power on phase B or s2, constant impedance portion

### qc_i

public double **qc_i**
reactive power on phase C, constant current portion

### qc_p

public double **qc_p**
reactive power on phase C, constant power portion

### qc_z

public double **qc_z**
reactive power on phase C, constant impedance portion

## Constructors

### GldNode

public **GldNode** (String *name*)
constructor defaults to zero load and zero phases present

**Parameters**

- **name** – CIM name of the bus

#### Methods

#### AddPhases

public boolean **AddPhases** (String *phs*)
accumulates phases present

**Parameters**

- **phs** – phases to add, may contain ABCDSs

**Returns** always true

#### ApplyZIP

public void **ApplyZIP** (double *Z*, double *I*, double *P*)
reapportion loads according to constant power (Z/sum), constant current (I/sum) and constant power (P/sum)

**Parameters**

- **Z** – portion of constant-impedance load
- **I** – portion of constant-current load
- **P** – portion of constant-power load

#### GetPhases

public String **GetPhases** ()

**Returns** phasing string for GridLAB-D with appropriate D, S or N suffix

#### HasLoad

public boolean **HasLoad** ()

**Returns** true if a non-zero real or reactive load on any phase

#### RescaleLoad

public void **RescaleLoad** (double *scale*)
scales the load by a factor that probably came from the command line's -l option

**Parameters**

- **scale** – multiplying factor on all of the load components

### 8.6.3 CDPSM_to_GLM.SpacingCount

static class **SpacingCount**
helper class to keep track of the conductor counts for WireSpacingInfo instances

Number of Conductors is the number of phases (1..3) plus neutrals (0..1)

#### Constructors

#### SpacingCount

public **SpacingCount** (int *nconds*, int *nphases*)
construct with number of conductors and phases

**Parameters**

- **nconds** – number of phases plus neutrals (1..4)
- **nphases** – number of phase conductors (1..3)

#### Methods

#### getNumConductors

public int **getNumConductors** ()

**Returns** accessor to number of conductors

#### getNumPhases

public int **getNumPhases** ()

**Returns** accessor to number of phases

### 8.6.4 SPARQLcimTest

public class **SPARQLcimTest** extends Object
This class runs an example SQARQL query against CIM XML

Future versions of GridAPPS-D will rely more heavily on SPARQL queries to do the selection and filtering, as the preferred pattern for developers working with CIM. This example uses several triples to execute a query on LinearShuntCompensators (aka capacitors).

Invoke as a console-mode program

**Author** Tom McDermott

**See also:** *SPARQLcimTest.main*

#### Fields

#### baseURI

static final String **baseURI**
identifies gridlabd

#### nsCIM

static final String **nsCIM**
namespace for CIM; should match the CIM version used to generate the RDF

### nsRDF

static final String **nsRDF**
namespace for RDF

### Methods

### GLD_Name

static String **GLD_Name** (String *arg*, boolean *bus*)
convert a CIM name to GridLAB-D name, replacing unallowed characters

### main

public static void **main** (String[] *args*)
Reads command-line input for the converter

**Parameters**

- **args** – will be SPARQLcimTest [options] input.xml

Options: -e={uli} encoding; UTF-8 or ISO-8859-1; choose u if input.xml came from OpenDSS

CHAPTER 9

# License

1. Battelle Memorial Institute (hereinafter Battelle) hereby grants permission to any person or entity lawfully obtaining a copy of this software and associated documentation files (hereinafter the Software) to redistribute and use the Software in source and binary forms, with or without modification. Such person or entity may use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and may permit others to do so, subject to the following conditions:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Other than as used herein, neither the name Battelle Memorial Institute or Battelle may be used in any form whatsoever without the express written consent of Battelle.

1. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL BATTELLE OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

General disclaimer for use with OSS licenses

This material was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the United States Department of Energy, nor Battelle, nor any of their employees, nor any jurisdiction or organization that has cooperated in the development of these materials, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness or any information, apparatus, product, software, or process disclosed, or represents that its use would not infringe privately owned rights.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

CHAPTER 10

# Indices and tables

- genindex
- modindex
- search

# Bibliography


[CIT1] W. H. Kersting, "Radial distribution test feeders," in *2001 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.01CH37194)*, 2001, pp. 908-912 vol.2.

[CIT2] R. F. Arritt and R. C. Dugan, "The IEEE 8500-node test feeder," in *IEEE PES T&D 2010*, 2010, pp. 1-6.

[CIT3] M. E. Baran and H. Ming-Yung, "Volt/VAr control at distribution substations," in *IEEE Transactions on Power Systems,* vol. 14, pp. 312-318, 1999.

[CIT4] V. Borozan, M. E. Baran, and D. Novosel, "Integrated volt/VAr control in distribution systems," in *2001 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.01CH37194)*, 2001, pp. 1485-1490 vol.3.

[CIT5] K. P. Schneider and J. C. Fuller, "Voltage control devices on the IEEE 8500 node test feeder," in *IEEE PES T&D 2010*, 2010, pp. 1-6.

[CIT6] I. Gorton et al., "GridOPTICS(TM) A Novel Software Framework for Integrating Power Grid Data Storage, Management and Analysis," in *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, 2013, pp. 2167-2176.

[CIT7] S. Ciraci, J. Daily, J. Fuller, A. Fisher, L. Marinovici, and K. Agarwal, "FNCS: a framework for power system and communication networks co-simulation," in *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, Tampa, Florida, 2014, pp. 1-8.

[CIT8] D. P. Chassin, J. C. Fuller, and N. Djilali, "GridLAB-D: An agent-based simulation framework for smart grids," in *Journal of Applied Mathematics*, vol. 2014, no. 492320, pp. 1-12, 2014.

# Index

## H



## M



## N



## P



## Q



## R



## S



## W