GridAPPS-D

Release 2021.04.0

The GridAPPS-D Team and Community

Sep 16, 2021

INSTALLATION RUNTIME

GridAPPS-D

GridAPPS-D™ is an open-source platform that accelerates development and deployment of portable applications for advanced distribution management and operations. It is built in a linux environment using Docker, which allows large software packages to be distributed as containers.

Its purpose is to reduce the time and cost to integrate advanced functionality into distribution operations, to create a more reliable and resilient grid.

GridAPPS-D enables standardization of data models, programming interfaces, and the data exchange interfaces for:

- devices in the field
- distributed apps in the systems
- applications in the control room

The platform provides

- robust testing tools for applications
- distribution system simulation capabilities
- standardized research capability
- reference architecture for the industry
- application development kit

The GridAPPS-D source code is publically available from GitHub. The GridAPPS-D™ project is sponsored by the U.S. Department of Energy and receives ongoing updates from a team of core developers at PNNL.

The GridAPPS-D team encourages and appreciates community involvement, including issues and pull requests on GitHub, participation in monthly app developers meetings, and posts on the discussion board.

Questions and support requests should be filed in the GridAPPS-D Forum Discussion Board

General issues and bugs can be reported in the GridAPPS-D Forum Issues Page

Bugs in the GridAPPS-D platform can be reported in full detail using the GOSS-GridAPPS-D Issues Page

CHAPTER

ONE

WINDOWS 10 INSTALLATION

This section contains detailed installation instructions and runtime environment tips for running GridAPPS-D and its dependencies on a Windows 10 machine.

1.1 Virtual Machine & Docker Setup

A typical Windows 10 installation does not include several of the tools needed to run the GridAPPS-D Platform Several software packages need to be installed prior to installing GridAPPS-D in the next step

Installation Steps:

- *1. Verify System Requirements*
- *2. Verify OS Build*
- *3. Install Windows Subsystem for Linux*
  - *3.1. Enable WSL*
  - *3.2. Upgrade to WSL2*
  - *3.3. Install Linux Ubuntu OS*
  - *3.4. Set up Ubuntu in WSL*
- *4. Install Docker for Windows*

It is also possible to use Virtualbox to create a virtual Ubuntu Linux machine. This approach may be used if it is desired to use Eclipse, etc. for Java development. However, performance of Virtualbox is significantly worse than WSL2 for running GridAPPS-D simulations and python application development.

1.1.1 System Requirements

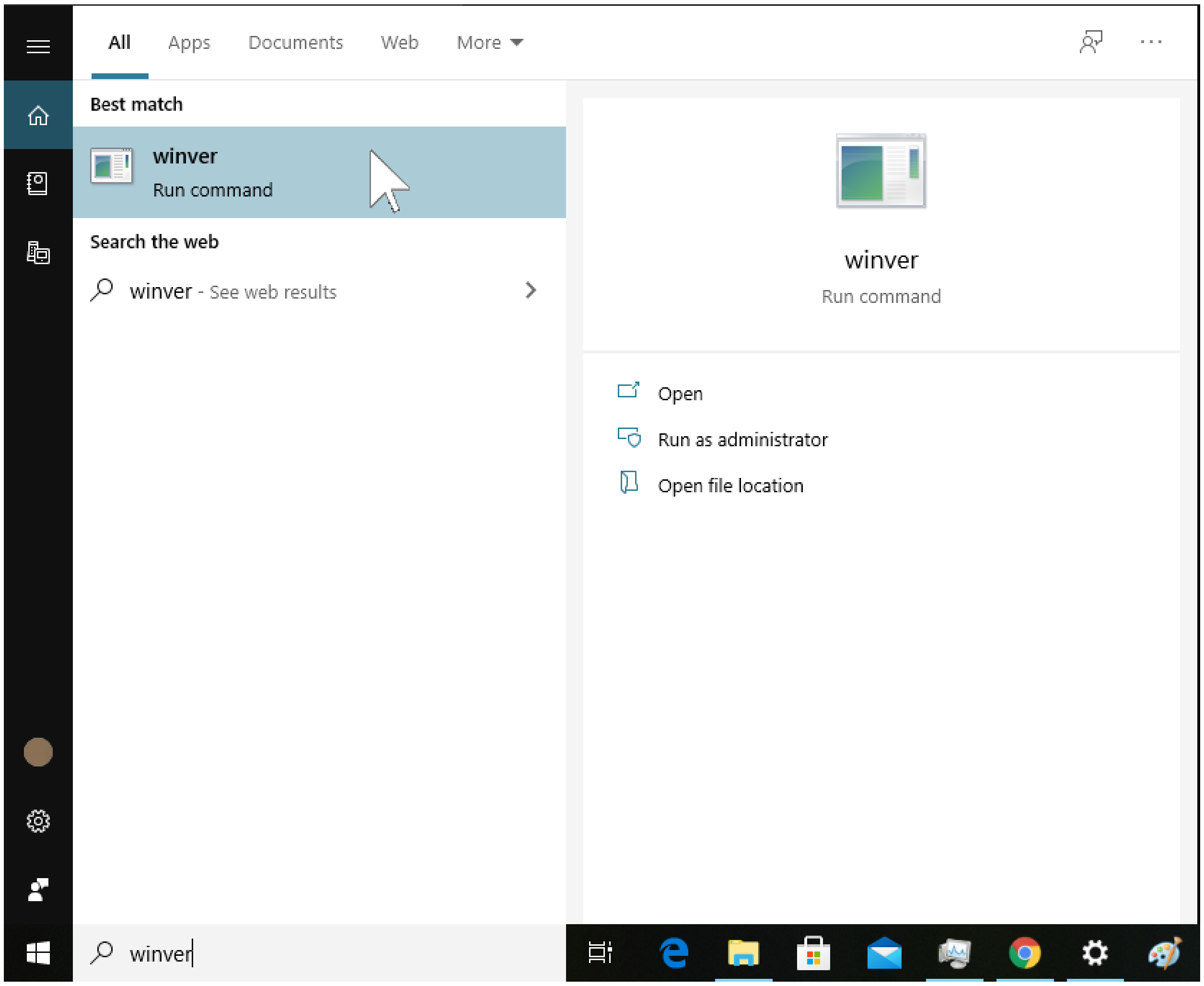
- **OS:**
  - Windows 10, Version 2004 or higher, with Build 19041 or higher
- **RAM:**
  - 8GB (*absolute minimum* for 13 and 123 node models, *may encounter memory overload during installation*);
  - 16GB (preferred for small models, minimum for 8500/9500 node models);
  - 32GB (recommended for application development)
- **Disk Space:**

– 15GB required for installation

Note: The download size is quite large, so it is recommended to use a fiber or ethernet interent connection, rathered than a metered hotspot to avoid excessive data usage charges.

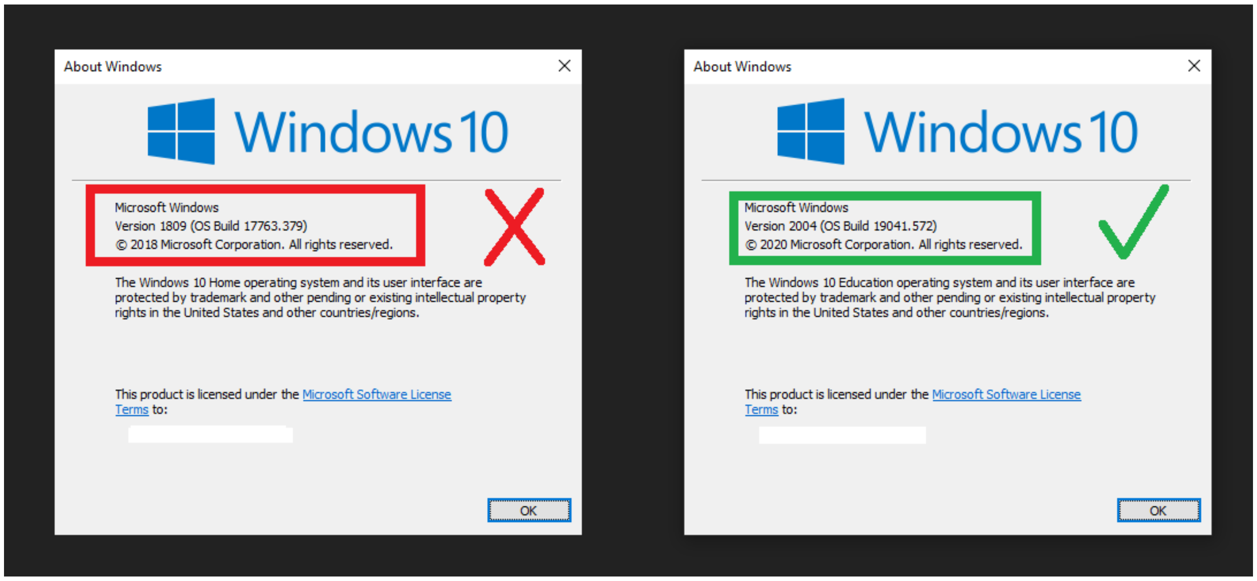
1.1.2 Windows 10 OS Build Requirments

To check your OS build, type `winver` in the Cortana seach bar:



Check to see if your OS is

- For x64 systems: **Version 1903 or higher, with Build 18362 or higher.**
- For ARM64 systems: **Version 2004 or higher, with Build 19041 or higher.**


About Windows

Microsoft Windows
Version 1809 (OS Build 17763.379)
© 2018 Microsoft Corporation. All rights reserved.

The Windows 10 Home operating system and its user interface are protected by trademark and other pending or existing intellectual property rights in the United States and other countries/regions.

This product is licensed under the Microsoft Software License Terms to:

About Windows

Microsoft Windows
Version 2004 (OS Build 19041.572)
© 2020 Microsoft Corporation. All rights reserved.

The Windows 10 Education operating system and its user interface are protected by trademark and other pending or existing intellectual property rights in the United States and other countries/regions.

This product is licensed under the Microsoft Software License Terms to:

If not, run **Windows Update** to get the latest verion of Windows 10 available for your machine. It may take some time for the new OS to download. Multiple restarts are typical while upgrading the windows version.

1.1.3 Disconnect from Corporate VPN

There is a known issue with WSL2 and some corporate / laboratory VPNs. During WSL2 and docker installation, the Domain Name Server is set to that of your corporate intranet if your machine is connected to a VPN. This will cause issues when accessing github, etc. from within WSL2.

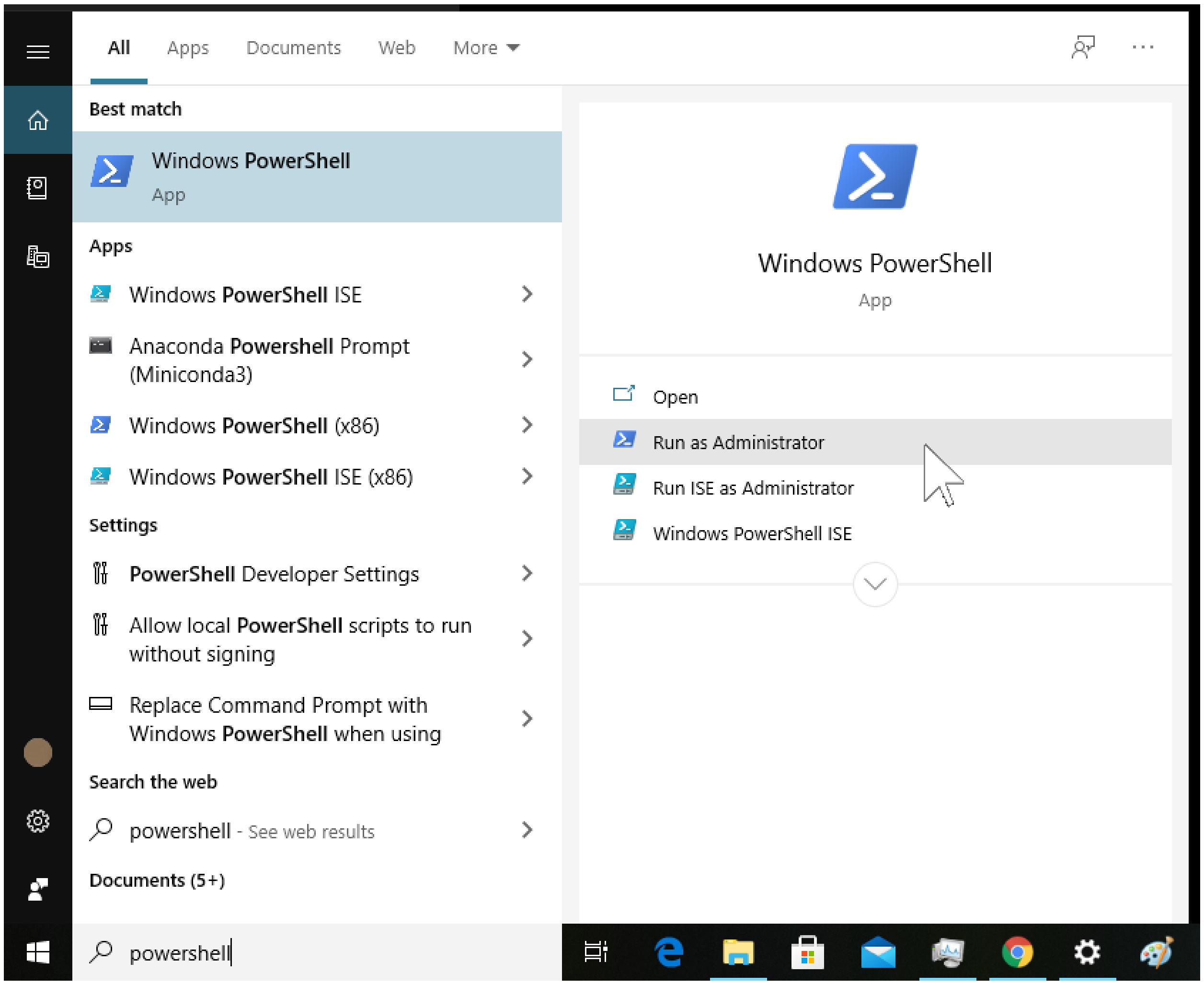
This issue has also been reported by users using Virtualbox VMs.

1.1.4 Install Windows Subsystem for Linux

GridAPPS-D and the associated docker containers will run using the Windows Subsystem for Linux (WSL), which is a new feature to Windows 10 that enables linux code to run natively in Windows without a separate virtual machine. The steps in this section are also available on the Microsoft website

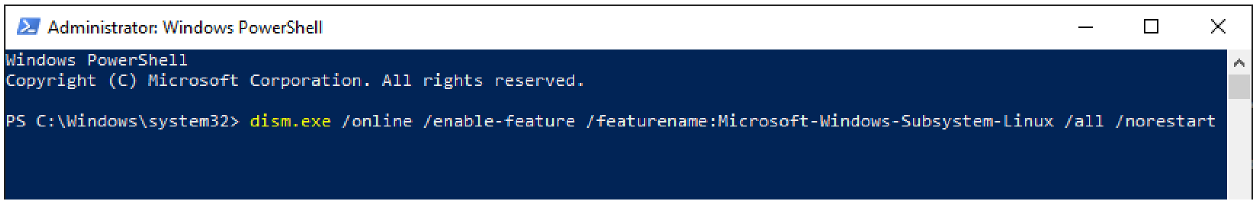
Enable WSL

Open Windows PowerShell as an administrator:



Enable WSL by entering

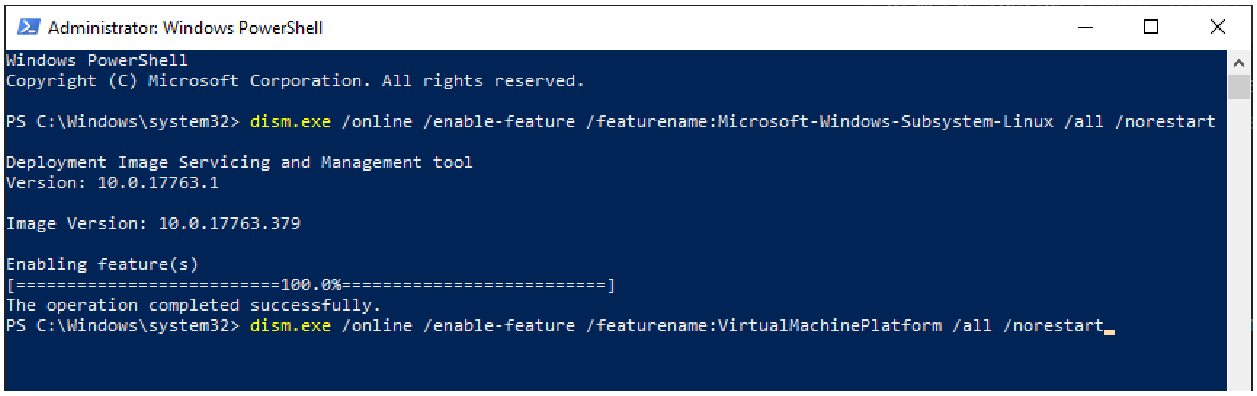
```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux
/all /norestart
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

Then, without restarting, enable the virtual machine platform by entering

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /
norestart
```



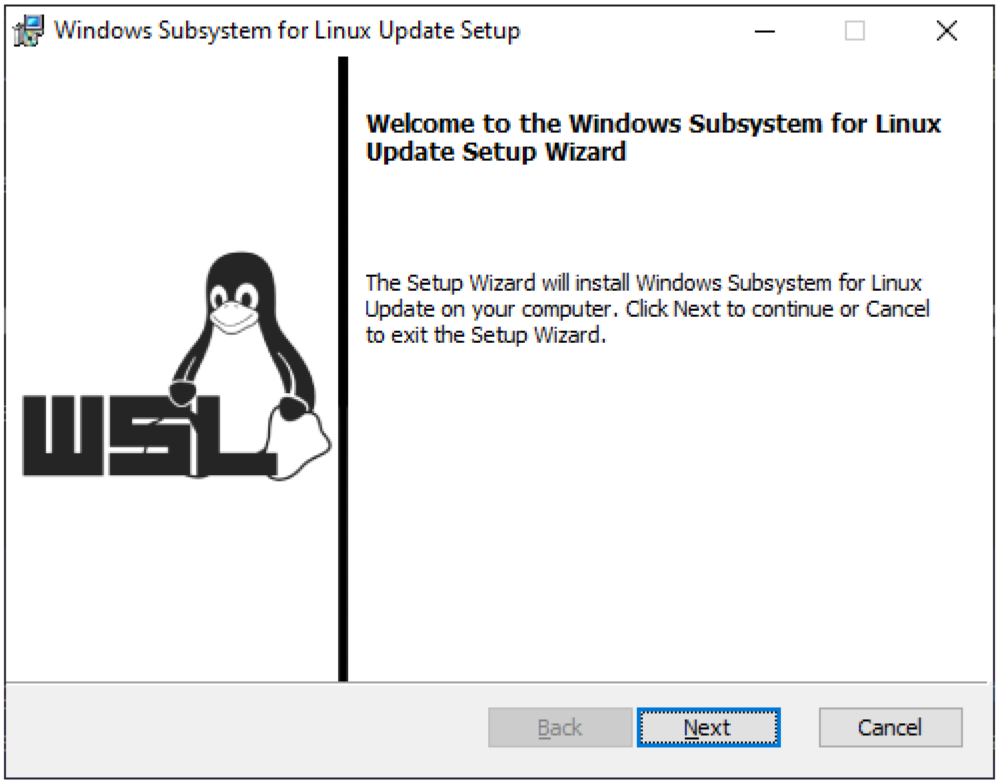
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

Deployment Image Servicing and Management tool
Version: 10.0.17763.1

Image Version: 10.0.17763.379

Enabling feature(s)
[==========================100.0%==========================]
The operation completed successfully.
PS C:\Windows\system32> dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

When completed, restart your machine. It may take a few minutes for the new settings to be applied while restarting.

Upgrade to WSL2

Download the latest WSL2 package .msi installer from the Microsoft repository

Run the update package to install WSL2 using the wizard:



Windows Subsystem for Linux Update Setup

Welcome to the Windows Subsystem for Linux Update Setup Wizard

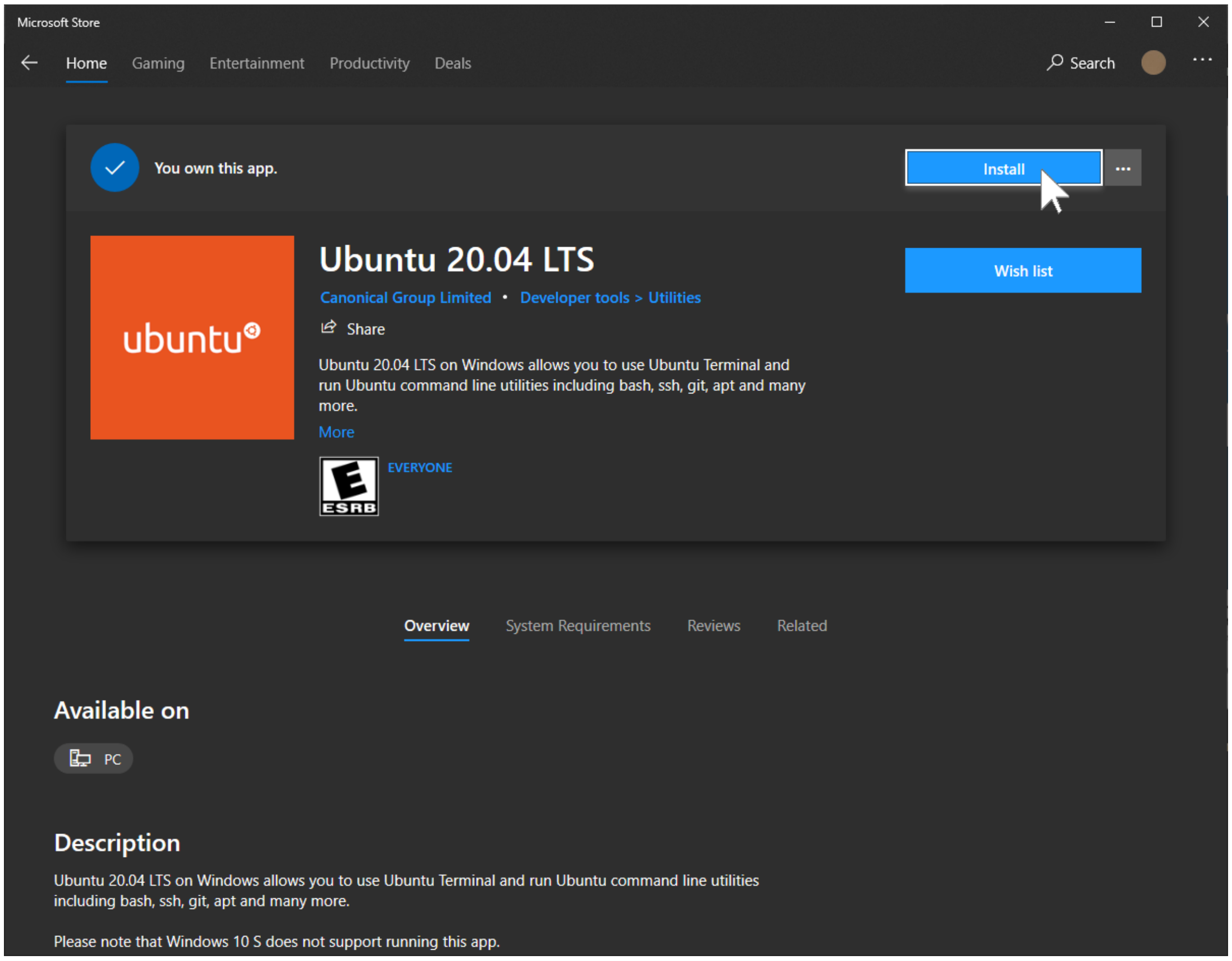
The Setup Wizard will install Windows Subsystem for Linux Update on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Open Windows PowerShell again and update the settings to use WSL2 by entering

```
wsl --set-default-version 2
```

Install Linux Ubuntu OS

Open the Microsoft Store app, and search for `Ubuntu` and install the desired version (available versions are 16.04, 18.04, and 20.04)



Ubuntu 20.04 LTS

Canonical Group Limited • Developer tools > Utilities

Share

Ubuntu 20.04 LTS on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

More

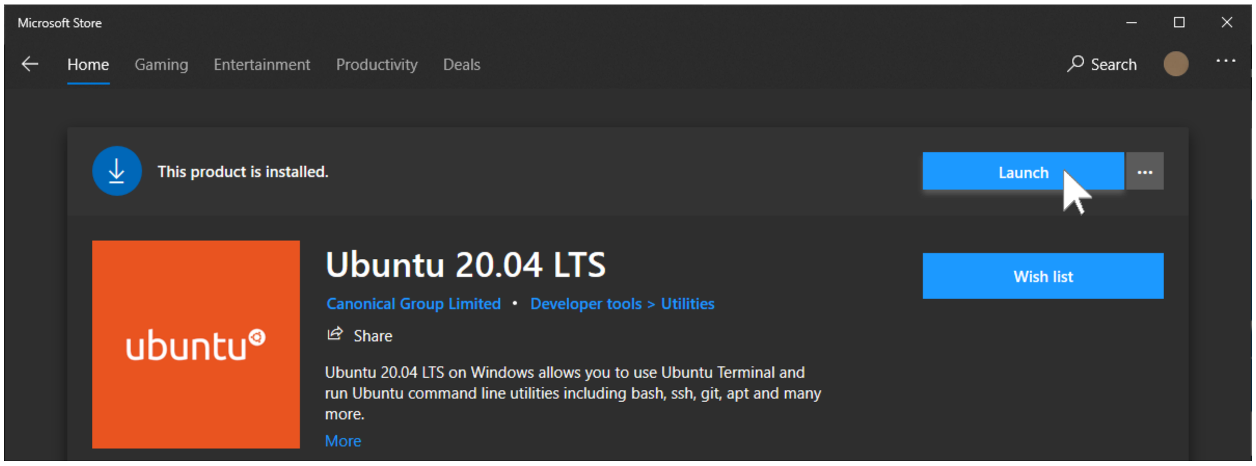
Available on

Description

Ubuntu 20.04 LTS on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

Please note that Windows 10 S does not support running this app.

When it has finished downloading, click `Launch`.



Ubuntu 20.04 LTS

Canonical Group Limited • Developer tools > Utilities

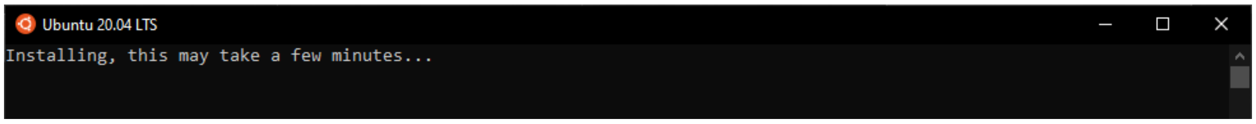
Share

Ubuntu 20.04 LTS on Windows allows you to use Ubuntu Terminal and run Ubuntu command line utilities including bash, ssh, git, apt and many more.

More

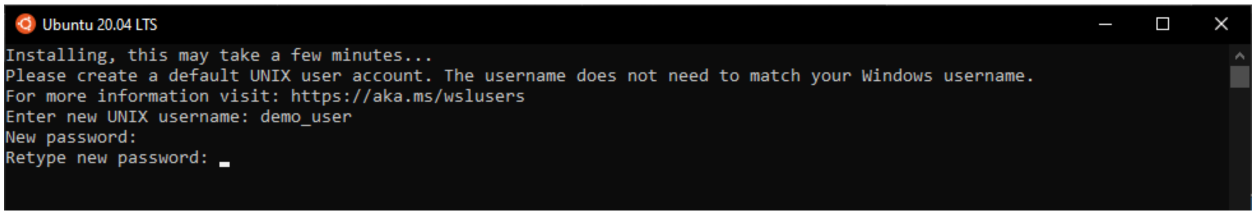
Set up Ubuntu in WSL

Wait for the Ubuntu OS to install.



```
Installing, this may take a few minutes...
```

Select a username and password. These do not need to be the same as your Windows or Microsoft Account login.

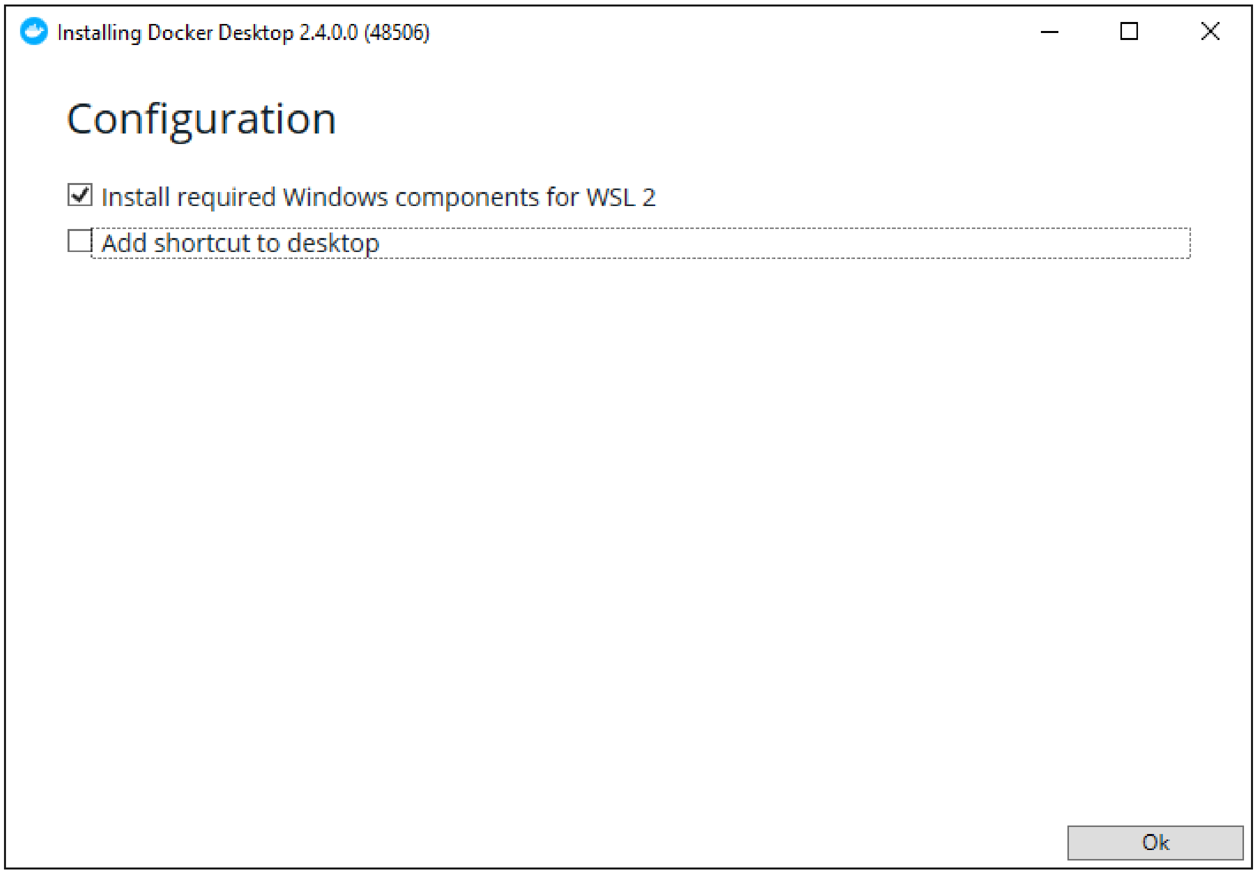


```
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows username.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: demo_user
New password:
Retype new password:
```

1.1.5 Install Docker for Windows

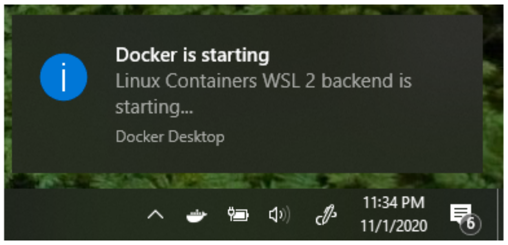
Download and run **Docker Desktop for Windows** from Docker Hub

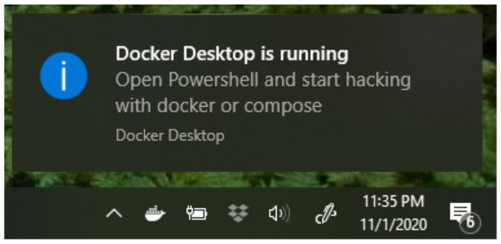
Be sure to select "**Install required components for WSL2**"



☑ Install required Windows components for WSL 2
☐ Add shortcut to desktop

After restarting your machine, Docker should start automatically, and you will see a notification stating "**Linux WSL2 containers are starting**"

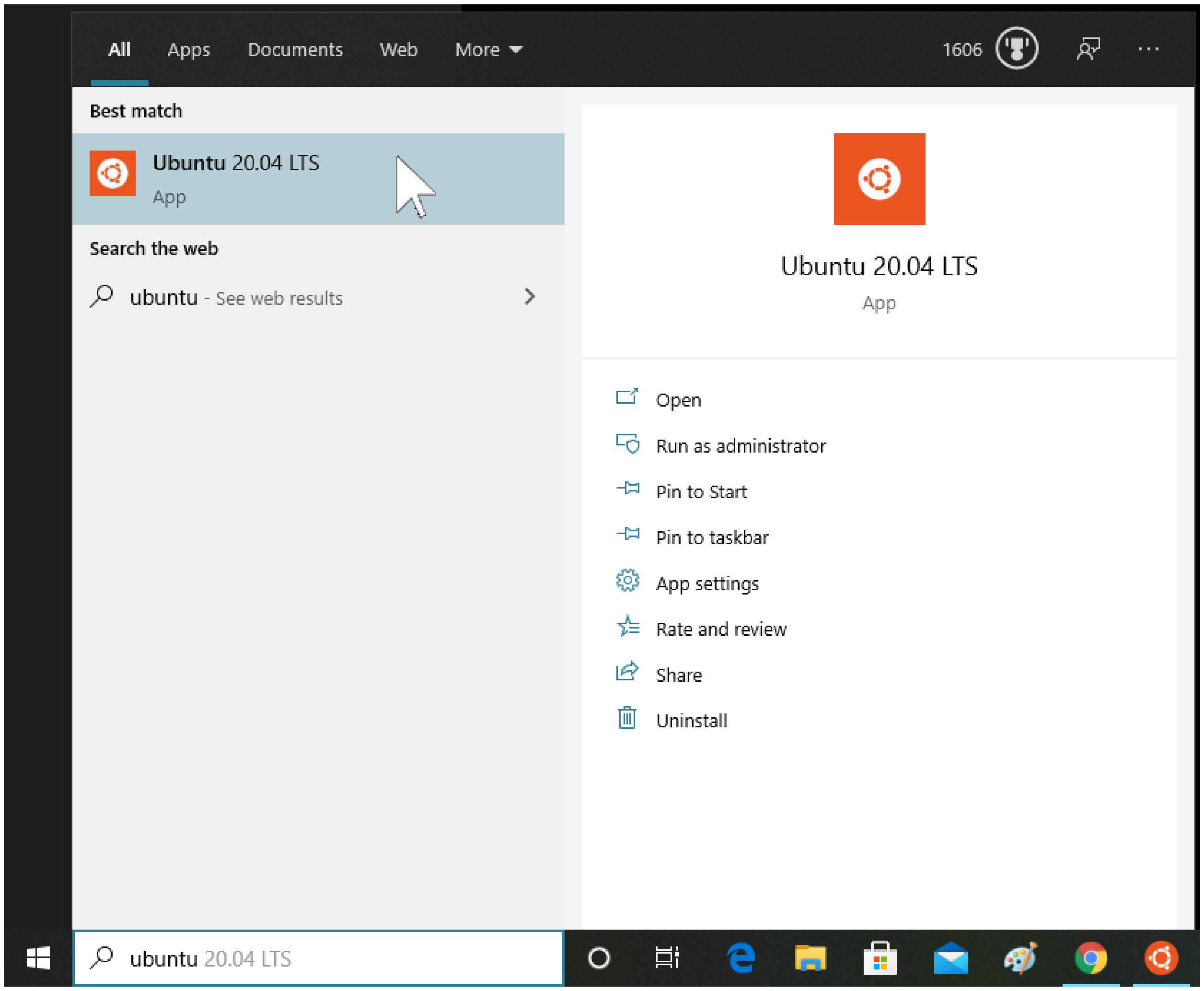



Docker Desktop is running
Open Powershell and start hacking with docker or compose
Docker Desktop
11:35 PM
11/1/2020

1.2 Installing GridAPPS-D

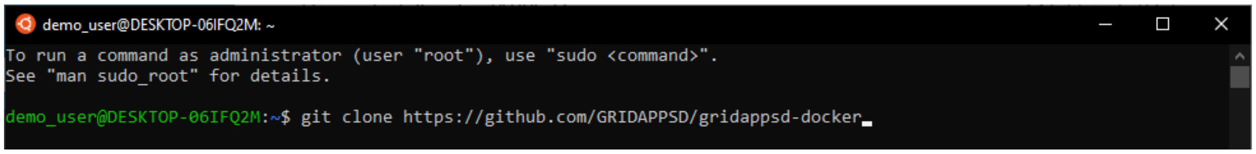
1.2.1 Clone the GridAPPS-D Docker repository

Disconnect from your corporate/laboratory VPN (if applicable) and open the Ubuntu terminal:



Clone the GridAPPS-D repository:

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

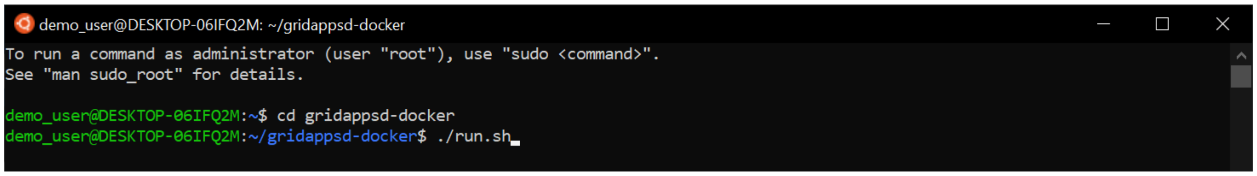


```
demo_user@DESKTOP-06IFQ2M: ~
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

demo_user@DESKTOP-06IFQ2M:~$ git clone https://github.com/GRIDAPPSD/gridappsd-docker
```

1.2.2 Install the GridAPPS-D Docker Containers

Change directories into the **gridappsd-docker** folder and start the latest stable release of the GridAPPS-D platform.

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

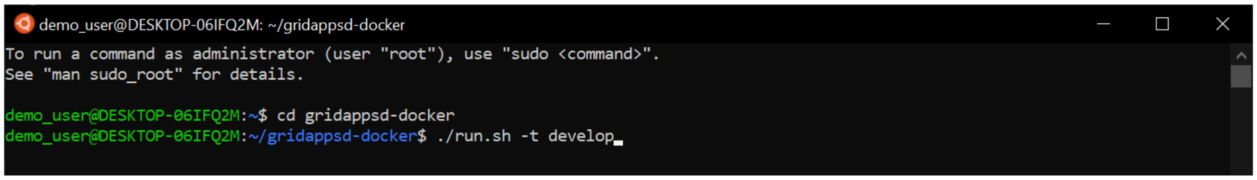


```
demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

demo_user@DESKTOP-06IFQ2M:~$ cd gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./run.sh
```

It is possible to specify a particular release tag using the -t option and the release tag

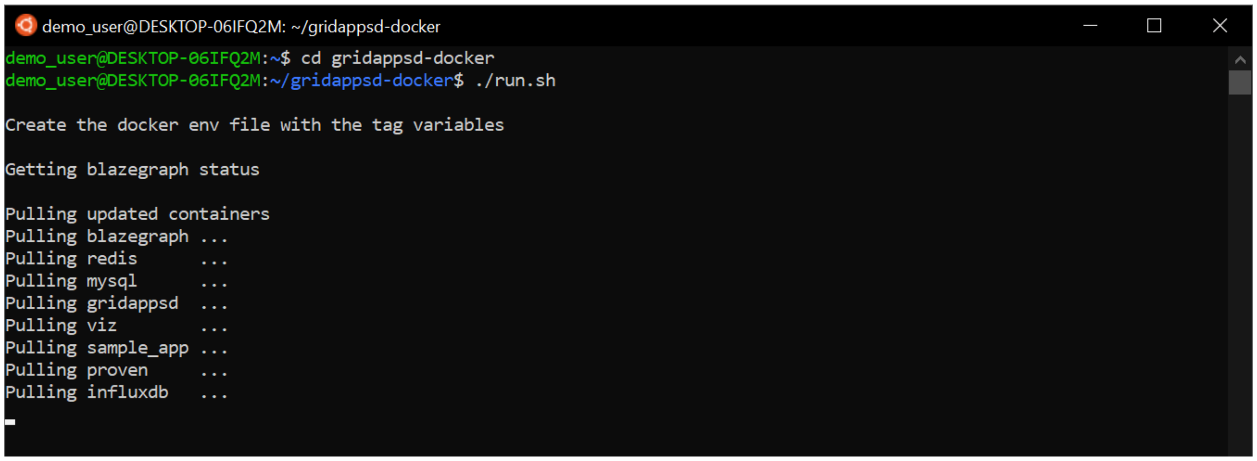
- `./run.sh -t develop` - use the `develop` branch with latest beta features
- `./run.sh -t releases_2021.04.0` - use the April 2021 release
- `./run.sh -t releases_2020.09.0` - use the September 2020 release

A complete set of releases is available in the *Platform Release History*



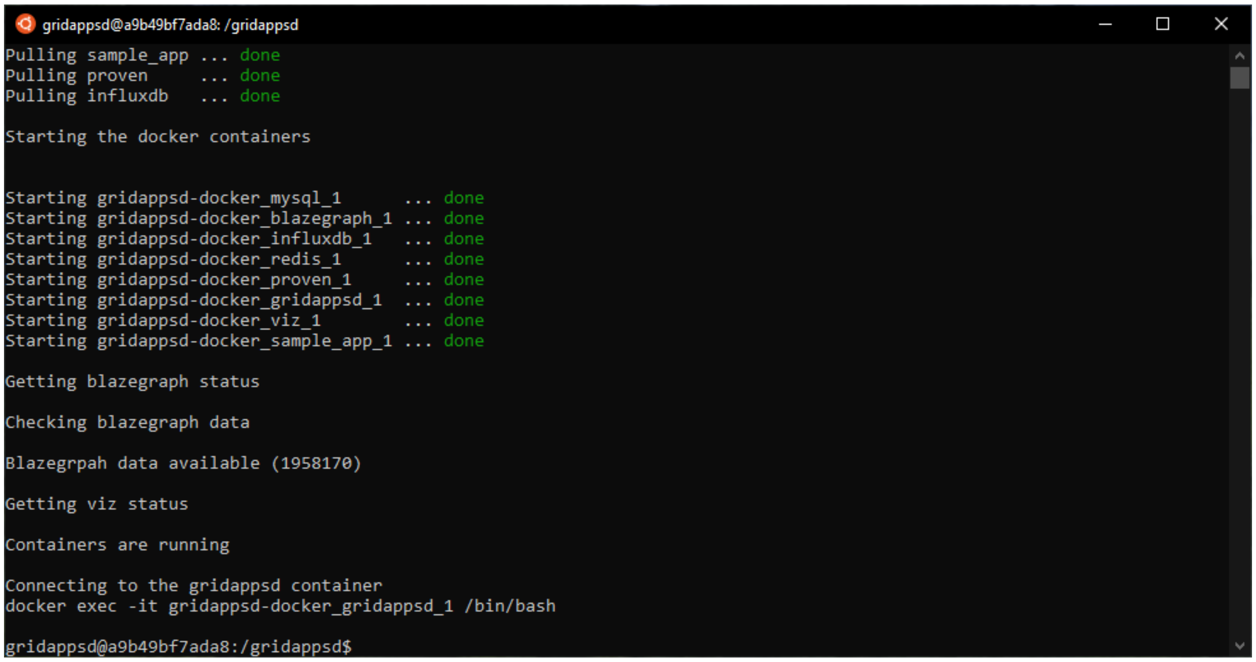
```
demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

demo_user@DESKTOP-06IFQ2M:~$ cd gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./run.sh -t develop
```

Wait for the platform to download the required docker containers. This is a very large package and will take several minutes.



```
demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~$ cd gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./run.sh

Create the docker env file with the tag variables

Getting blazegraph status

Pulling updated containers
Pulling blazegraph ...
Pulling redis      ...
Pulling mysql      ...
Pulling gridappsd  ...
Pulling viz        ...
Pulling sample_app ...
Pulling proven     ...
Pulling influxdb   ...
```

After the containers have finished downloading, they will automatically be created and then launched:



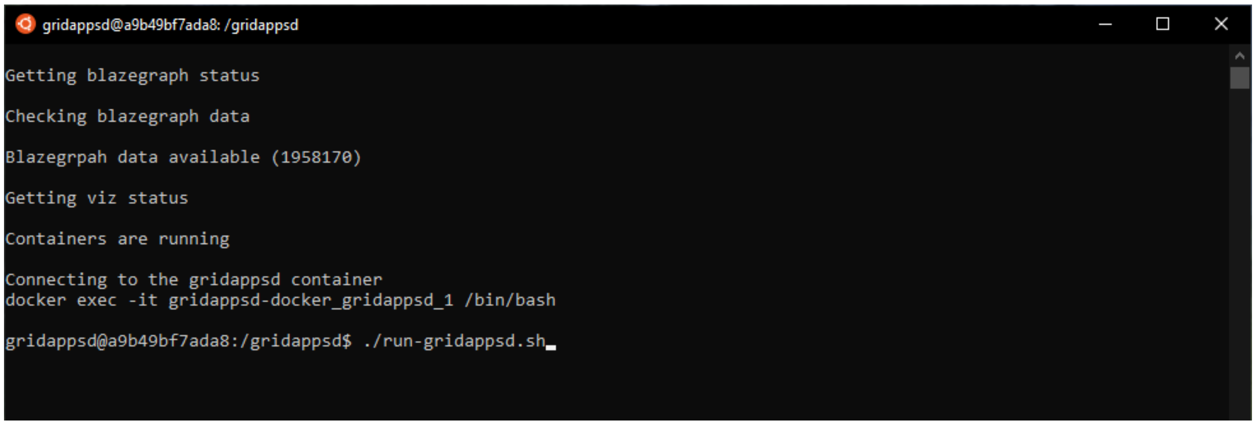
```
Pulling sample_app ... done
Pulling proven     ... done
Pulling influxdb   ... done

Starting the docker containers


Starting gridappsd-docker_mysql_1      ... done
Starting gridappsd-docker_blazegraph_1 ... done
Starting gridappsd-docker_influxdb_1   ... done
Starting gridappsd-docker_redis_1      ... done
Starting gridappsd-docker_proven_1     ... done
Starting gridappsd-docker_gridappsd_1  ... done
Starting gridappsd-docker_viz_1        ... done
Starting gridappsd-docker_sample_app_1 ... done

Getting blazegraph status

Checking blazegraph data

Blazegrpah data available (1958170)

Getting viz status

Containers are running

Connecting to the gridappsd container
docker exec -it gridappsd-docker_gridappsd_1 /bin/bash

gridappsd@a9b49bf7ada8:/gridappsd$
```

1.2.3 Launch the GridAPPS-D Platform

When all the containers are running, the terminal will move inside the docker enviroment, which has its own internal directories and path.

Start the GridAPPS-D platform inside the docker container by running

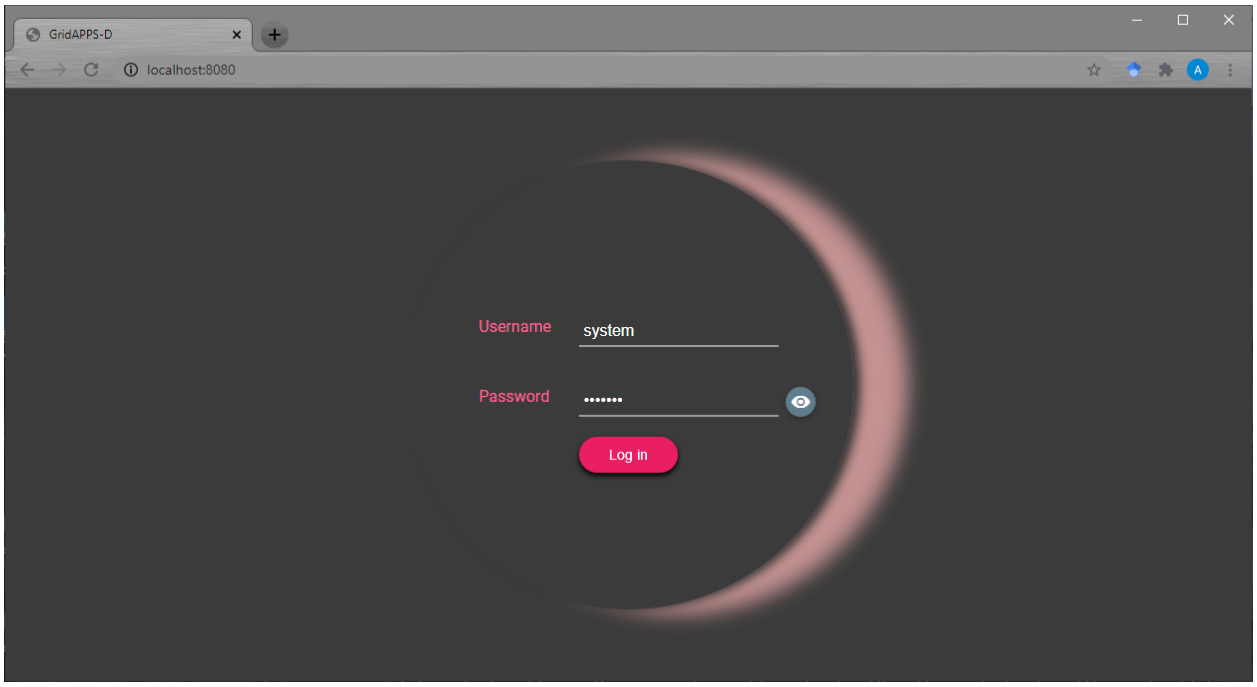
```
./run-gridappsd.sh
```



```
Getting blazegraph status

Checking blazegraph data

Blazegrpah data available (1958170)

Getting viz status

Containers are running

Connecting to the gridappsd container
docker exec -it gridappsd-docker_gridappsd_1 /bin/bash

gridappsd@a9b49bf7ada8:/gridappsd$ ./run-gridappsd.sh
```

The GridAPPS-D platform is now installed and running.

To confirm, open localhost:8080 to access the GridAPPS-D Visualization App:



1.2.4 Known Issues around Corporate VPN Connectivity

There is a known issue around WSL2 and Virtualbox VM compatibility with corporate VPNs.

If your machine was connected to a corporate VPN during setup, the Doman Name Server (DNS) lookup address is set to that of your corporate intranet. To reset it, open an ubuntu session and edit the `resolv.conf` file

- `sudo nano /etc/resolv.conf`
- comment out existing nameserver address
- add new line with `nameserver 8.8.8.8`
- save file and restart terminal

GridAPPS-D

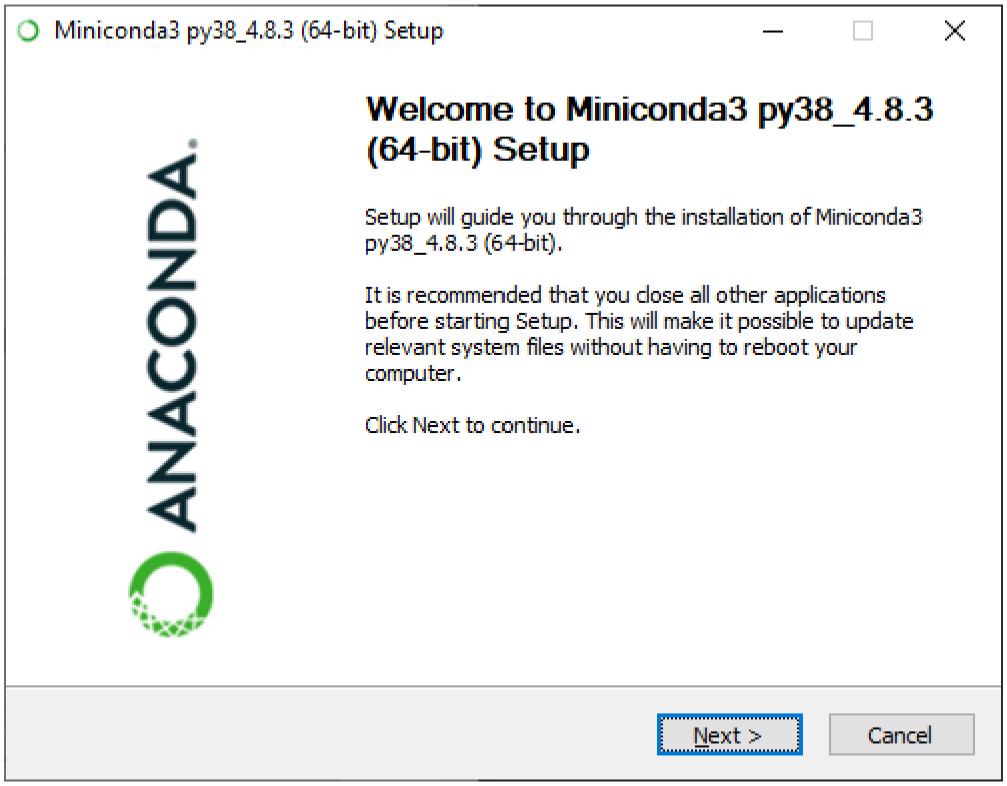
1.3 Installing GridAPPSD-Python and Notebook Tutorials

1.3.1 Install Anaconda or Miniconda

Download the latest version of the Miniconda from the Conda.io website:

- Python 3.8 for 64-bit Windows
- Python 3.8 for 32-bit Windows

Use the installation wizard with the recommended settings to complete installation.



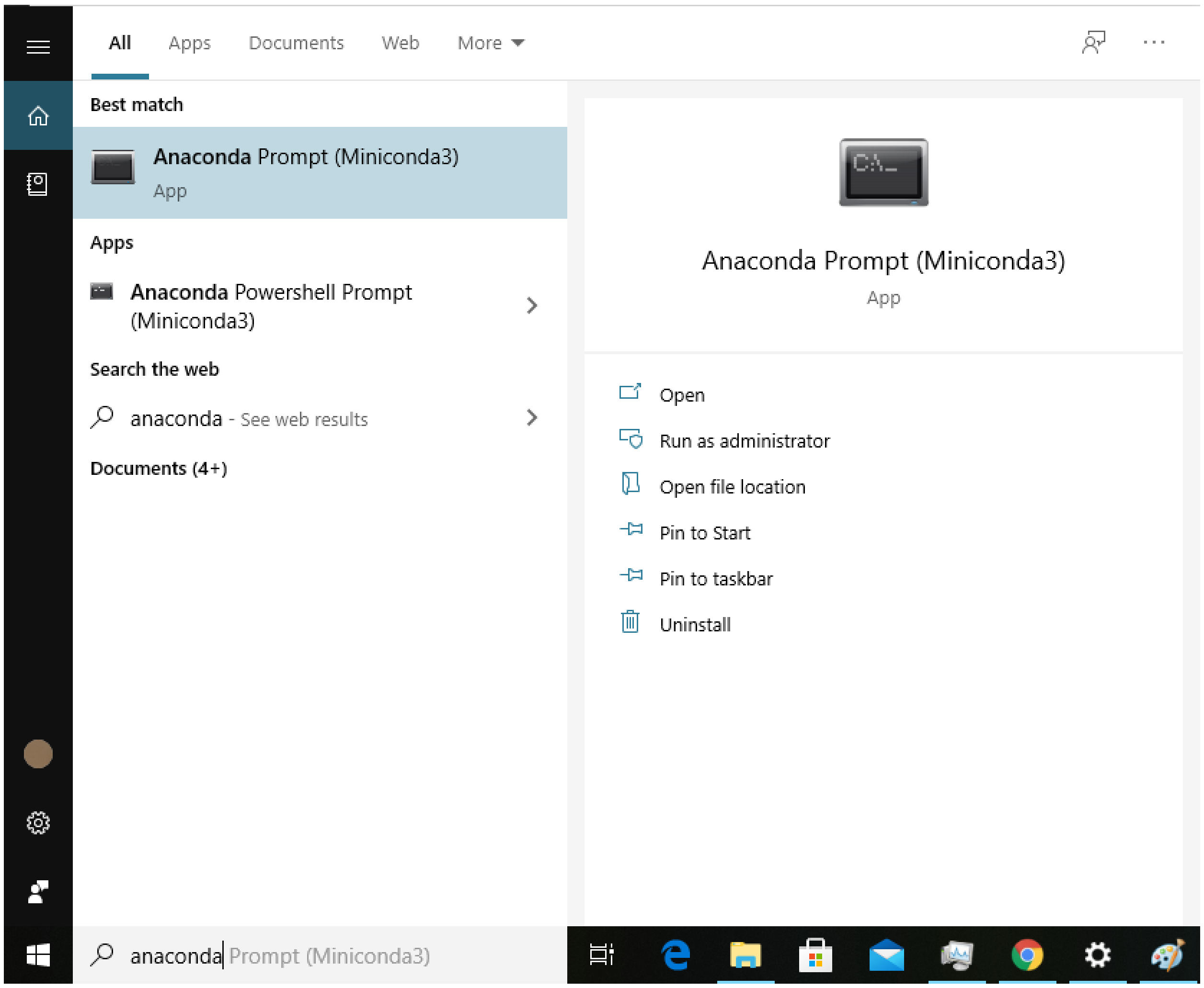
Welcome to Miniconda3 py38_4.8.3 (64-bit) Setup

Setup will guide you through the installation of Miniconda3 py38_4.8.3 (64-bit).

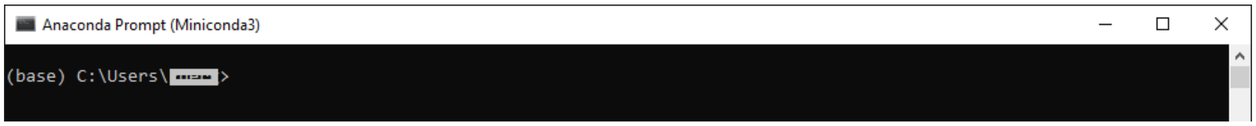
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

After installation is complete, launch the **Anaconda Prompt (Miniconda3)** from the Start Menu or by typing `anaconda` in the Cortana toolbar



The miniconda terminal window will open



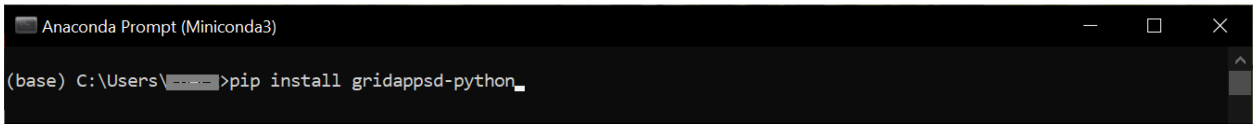
```
(base) C:\Users\[redacted]>
```

1.3.2 Install GridAPPSD-Python

In the Miniconda terminal window, download and install GridAPPSD-Python by running

```
pip install gridappsd-python
```

to download the GridAPPSD-Python library and required packages.



```
(base) C:\Users\[redacted]>pip install gridappsd-python
```

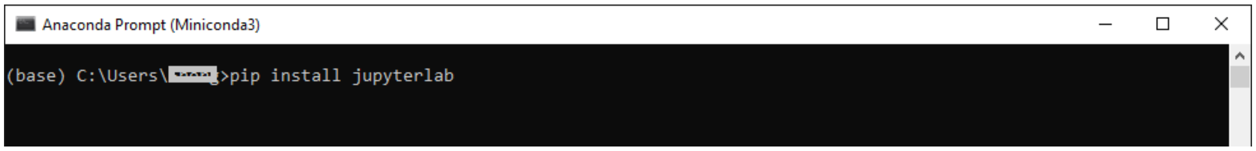
GridAPPSD-Python and all dependencies should have been automatically added to your anaconda path after completion.

1.3.3 Install Jupyter Lab

In the miniconda terminal window, run

`pip install jupyterlab`

to install the Jupyter environment for executing the python notebooks. It may take a couple minutes to collect and install all the required packages.



Anaconda Prompt (Miniconda3)

```
(base) C:\Users\[redacted]>pip install jupyterlab
```

1.3.4 Python Training Notebooks

The Jupyter / iPython training notebooks are the source materials for the GridAPPS-D ReadTheDocs website.

The notebooks include all the code examples and sample app materials in a format that can connect to a local GridAPPS-D platform session and interact in real-time with simulations in real-time.

Download Python Training Notebooks

In the miniconda terminal window, clone the python notebooks by running `git clone https://github.com/GRIDAPPSD/gridappsd-training` to download the python training notebooks.

|win_setup_install_notebooks.png|

By default, the notebooks will be saved in the directory `C:\Users\username\gridappsd-training`

Close the miniconda terminal

Running Python Training Notebooks

Start the Jupyter notebooks running on port 8890 (to avoid port sharing conflict with the GridAPPS-D Blazegraph database container):

`jupyter notebook --port 8890`

If running on a remote server (e.g. AWS cloud or university / laboratory server farm), start the notebooks by running

`jupyter notebook --port 8890 --no-browser --ip='0.0.0.0'`

Port Sharing between GridAPPS-D and Jupyter

By default, both Jupyter and the GridAPPS-D Blazegraph database use port 8889. If a Jupyter notebook is already running on port 8889, the Blazegraph database container will fail to start.

It is recommended to specify manually that Jupyter run on a different port:

```
jupyter notebook --port 8890
```

CHAPTER

TWO

UBUNTU LINUX INSTALLATION

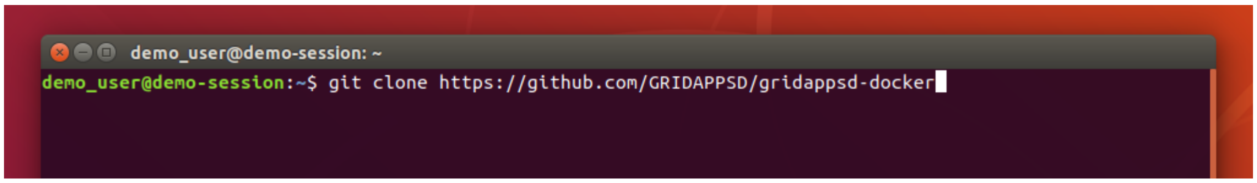
This section contains detailed installation instructions and runtime environment tips for running GridAPPS-D and its dependencies on an Ubuntu Linux machine.

2.1 Installing GridAPPS-D

2.1.1 Clone the GridAPPS-D Repository

Clone the GridAPPS-D GitHub repository

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



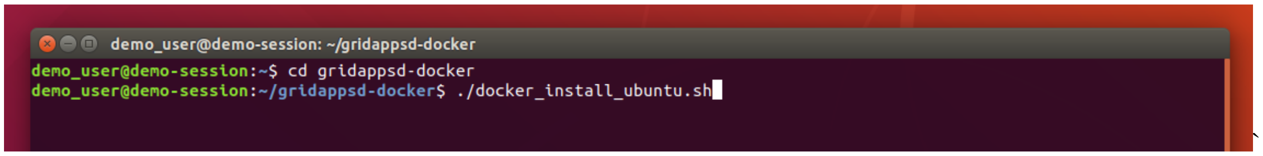
```
demo_user@demo-session: ~
demo_user@demo-session:~$ git clone https://github.com/GRIDAPPSD/gridappsd-docker
```

2.1.2 Install Docker

The GridAPPS-D repository includes a Docker installation script. This script only works for native linux environments (not WSL2).

Change directories into gridapps-docker and run the Docker installation script

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



```
demo_user@demo-session: ~/gridappsd-docker
demo_user@demo-session:~$ cd gridappsd-docker
demo_user@demo-session:~/gridappsd-docker$ ./docker_install_ubuntu.sh
```

2.1.3 Install GridAPPS-D

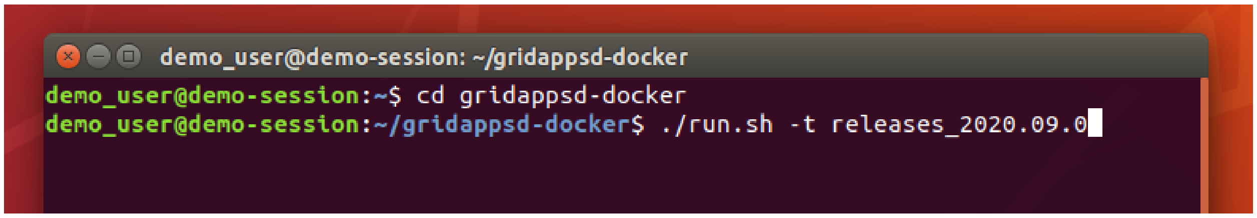
After Docker finishes installing, log out or restart the Ubuntu session.

After logging back in, change directories into gridappsd-docker and start the latest stable version of the GridAPPS-D platform, which will automatically download the required docker containers.

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

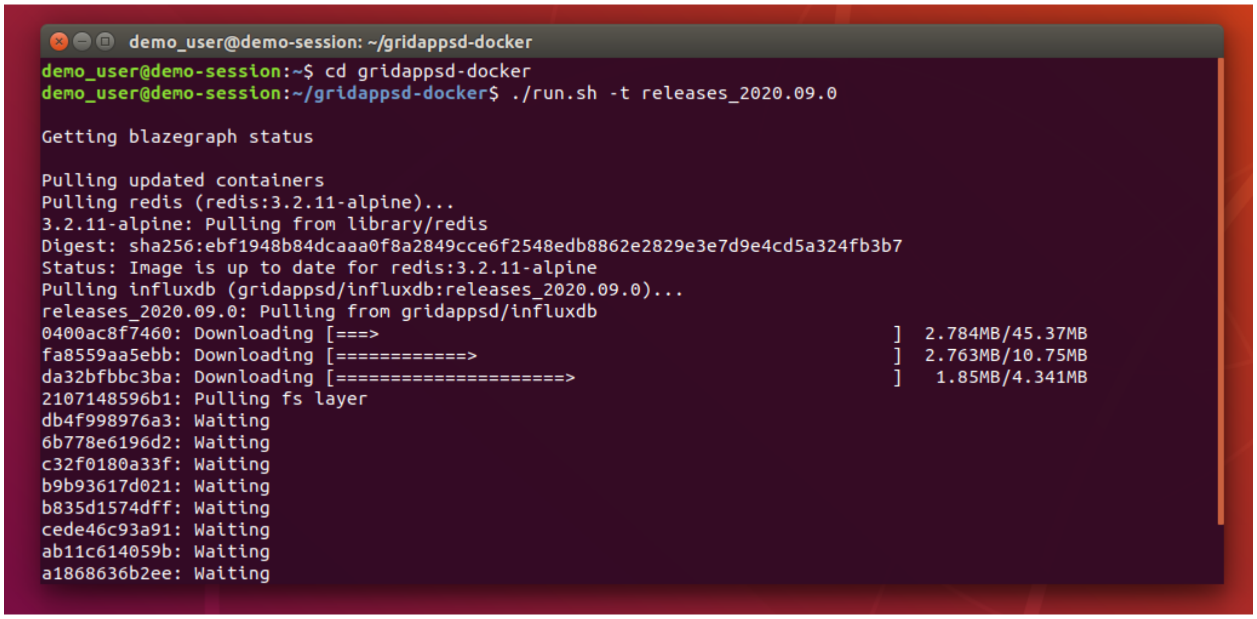
To install a particular release, specify the release tag using the `-t` option:

- `./run.sh -t develop` – Install latest develop version with beta features
- `./run.sh -t releases_2021.04.0` – Install April 2021 release
- `./run.sh -t releases_2020.09.0` – Install September 2020 release



```
demo_user@demo-session:~$ cd gridappsd-docker
demo_user@demo-session:~/gridappsd-docker$ ./run.sh -t releases_2020.09.0
```

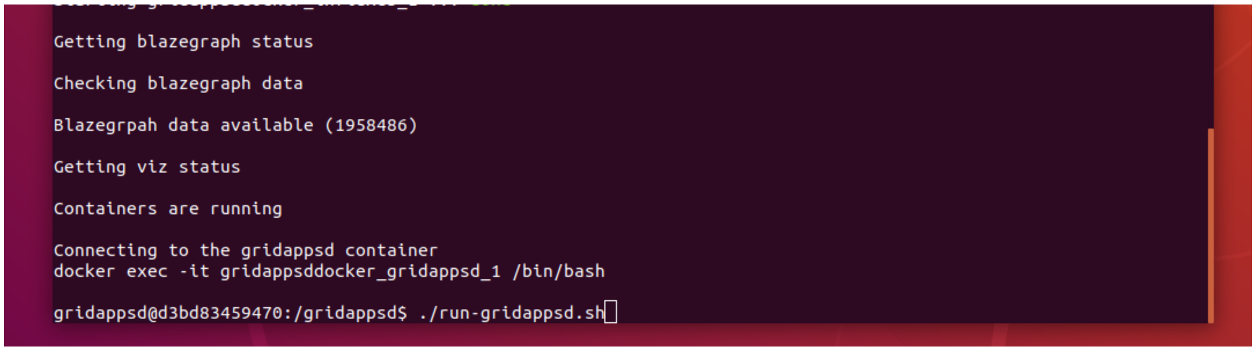
Wait for the docker containers to finish downloading. This will take a while due to the package size.



```
demo_user@demo-session:~$ cd gridappsd-docker
demo_user@demo-session:~/gridappsd-docker$ ./run.sh -t releases_2020.09.0

Getting blazegraph status

Pulling updated containers
Pulling redis (redis:3.2.11-alpine)...
3.2.11-alpine: Pulling from library/redis
Digest: sha256:ebf1948b84dcaaa0f8a2849cce6f2548edb8862e2829e3e7d9e4cd5a324fb3b7
Status: Image is up to date for redis:3.2.11-alpine
Pulling influxdb (gridappsd/influxdb:releases_2020.09.0)...
releases_2020.09.0: Pulling from gridappsd/influxdb
0400ac8f7460: Downloading [===>                                               ]  2.784MB/45.37MB
fa8559aa5ebb: Downloading [============>                                      ]  2.763MB/10.75MB
da32bfbbc3ba: Downloading [=====================>                             ]   1.85MB/4.341MB
2107148596b1: Pulling fs layer
db4f998976a3: Waiting
6b778e6196d2: Waiting
c32f0180a33f: Waiting
b9b93617d021: Waiting
b835d1574dff: Waiting
cede46c93a91: Waiting
ab11c614059b: Waiting
a1868636b2ee: Waiting
```

When the containers have finished downloading and installing, start the GridAPPS-D Platform

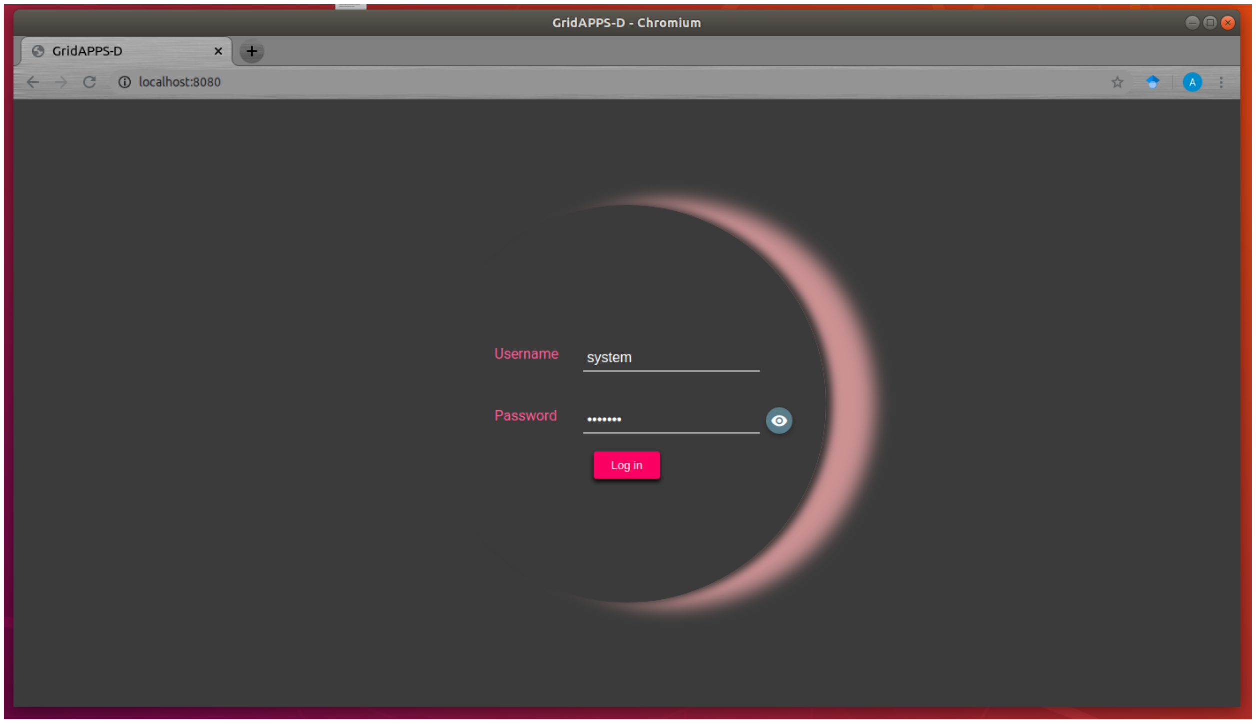
- `./run-gridappsd.sh`



```
Getting blazegraph status

Checking blazegraph data

Blazegrpah data available (1958486)

Getting viz status

Containers are running

Connecting to the gridappsd container
docker exec -it gridappsddocker_gridappsd_1 /bin/bash

gridappsd@d3bd83459470:/gridappsd$ ./run-gridappsd.sh
```

The GridAPPS-D platform is now installed and running.

To confirm, open localhost:8080 to access the GridAPPS-D Viz



2.2 Installing GridAPPSD-Python and Notebook Tutorials

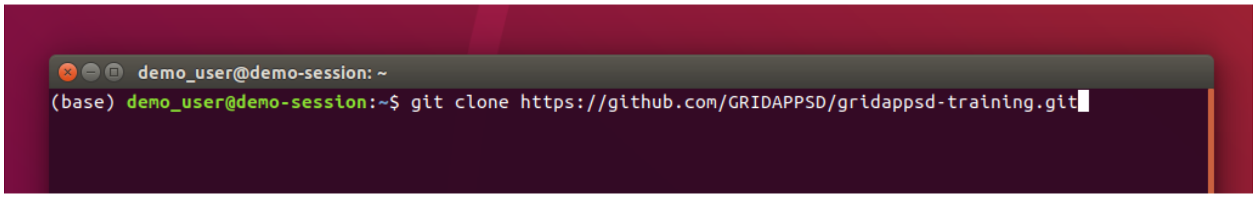
2.2.1 Quick Installation

Clone the GridAPPSD-Training Repository and run the `./install.sh` script

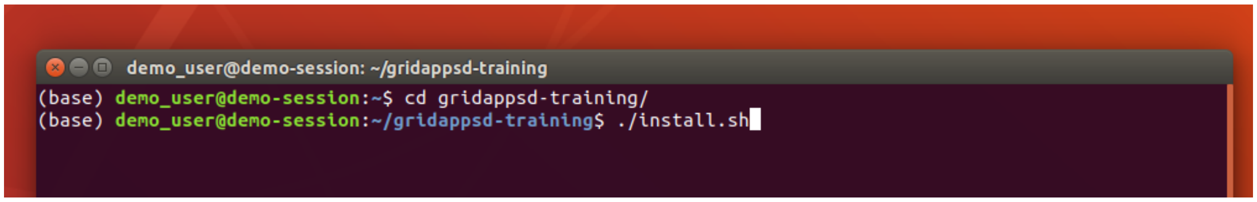
- `git clone https://github.com/GRIDAPPSD/gridappsd-training.git`
- `cd gridappsd-training`
- `./install.sh`

Accept the user terms for Miniconda and Jupyterlab.

After completion, the JupyterLab server will be running in a virtual environment with the training notebooks



```
(base) demo_user@demo-session:~$ git clone https://github.com/GRIDAPPSD/gridappsd-training.git
```



```
(base) demo_user@demo-session:~$ cd gridappsd-training/
(base) demo_user@demo-session:~/gridappsd-training$ ./install.sh
```

To start the jupyter notebooks at a later time, change directories into gridappsd-training and run the `./run.sh` script:

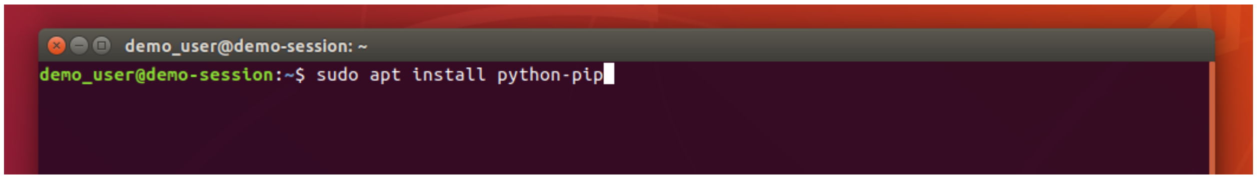
- `cd gridappsd-training`
- `./run.sh`

2.2.2 Manual Installation

Install Anaconda or Miniconda

If not **pip** is not installed, use `apt-get` to install it.

- `sudo apt-get install python-pip`



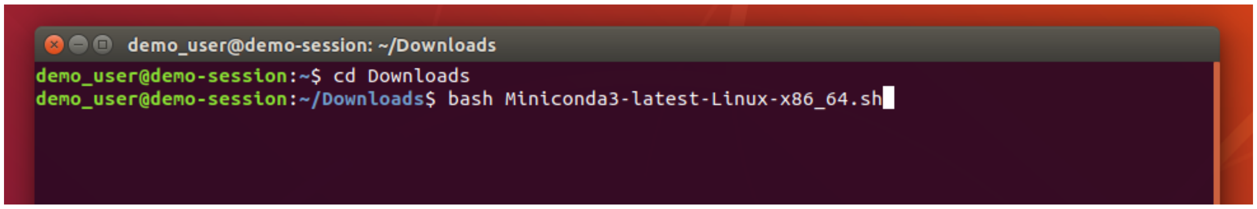
```
demo_user@demo-session:~$ sudo apt install python-pip
```

Download the latest version of Anaconda or Miniconda and save it in the `/Downloads` folder:

- Use Python 3.8 install for 64-bit systems from the Conda.io website.
- Use Python 3.7 install for 32-bit systems from the Conda.io website

Install Miniconda using bash

- `cd /Downloads`
- `bash Miniconda3-latest-Linux-x86_64.sh`



```
demo_user@demo-session: ~/Downloads
demo_user@demo-session:~$ cd Downloads
demo_user@demo-session:~/Downloads$ bash Miniconda3-latest-Linux-x86_64.sh
```

Follow the prompts on the installer screens. If you are unsure about any setting, accept the defaults. You can change them later. To make the changes take effect, close and then re-open your terminal window.

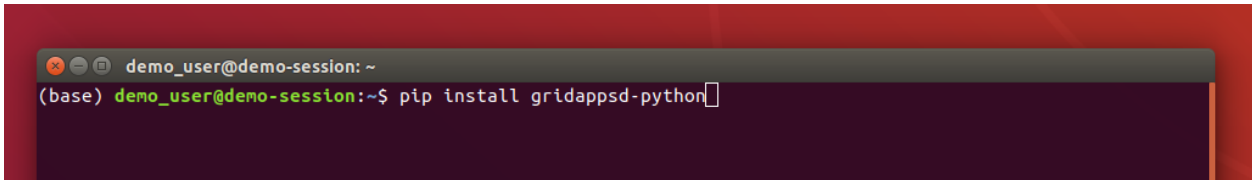
Test your installation. In your terminal window or Anaconda Prompt, run the command `conda list`. A list of installed packages appears if it has been installed correctly

---

Install GridAPPSD-Python

Use **pip** to install GridAPPSD-Python, which is need to pass API Calls to GridAPPS-D platform using the GridAPPSD-Python library methods:

- `pip install gridappsd-python`

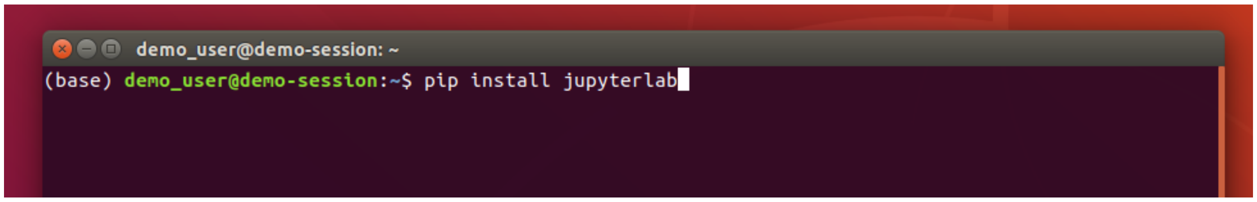


```
demo_user@demo-session: ~
(base) demo_user@demo-session:~$ pip install gridappsd-python
```

Install Jupyter Lab

Use **pip** to install Jupyter Lab, which is need to open and execute the Python Training Notebooks:

- `pip install jupyterlab`



```
demo_user@demo-session: ~
(base) demo_user@demo-session:~$ pip install jupyterlab
```

Download Python Notebooks

The Jupyter / iPython training notebooks are the source materials for the GridAPPS-D ReadTheDocs website.

The notebooks include all the code examples and sample app materials in a format that can connect to a local GridAPPS-D platform session and interact in real-time with simulations in real-time.

Clone the python notebooks in the GridAPPSD-Training repository by running

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

By default, the notebooks will be saved in the directory `gridappsd-training`

Clone the python notebooks in the GridAPPSD-Training repository by running

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

Start the Jupyter notebooks running on port 8890 (to avoid port sharing conflict with the GridAPPS-D Blazegraph database container):

```
jupyter notebook --port 8890
```

If running on a remote server (e.g. AWS cloud or university / laboratory server farm), start the notebooks by running

```
jupyter notebook --port 8890 --no-browser --ip='0.0.0.0'
```

Port Sharing between GridAPPS-D and Jupyter

By default, both Jupyter and the GridAPPS-D Blazegraph database use port 8889. If a Jupyter notebook is already running on port 8889, the Blazegraph database container will fail to start.

It is recommended to specify manually that Jupyter run on a different port:

```
jupyter notebook --port 8890
```

CHAPTER

THREE

RUNNING GRIDAPPS-D

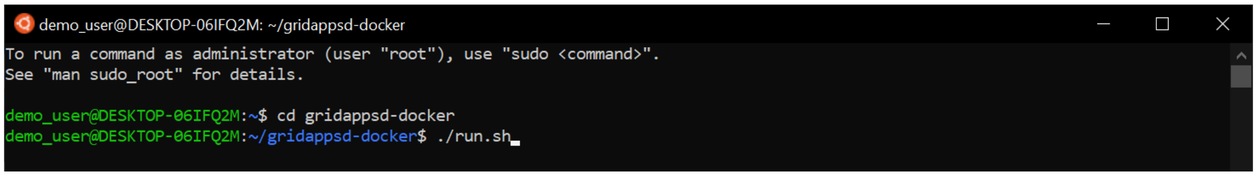
3.1 Starting the GridAPPS-D Platform

If you are accessing this section after completing the installation steps in the previous procedure, then the GridAPPS-D Platform is already running.

When you start your machine next time, you will need to start the GridAPPS-D Platform again. To do this, change directories into `gridappsd-docker` and run the `./run.sh` script

- `cd gridappsd-docker`
- `./run.sh` or `./run.sh -t release_tag`

A complete set of releases of the GridAPPS-D Platform is available under *Platform Release History*



demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

demo_user@DESKTOP-06IFQ2M:~$ cd gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./run.sh
```

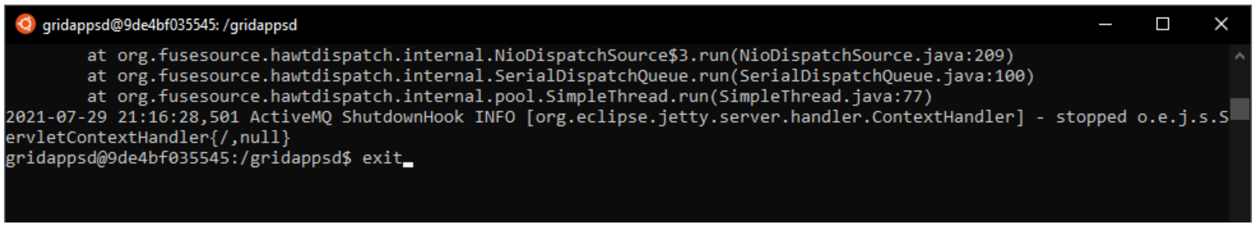
3.2 Stopping the Platform

3.2.1 Stopping the Platform from Inside the Docker Container

If you are currently inside the `./run-gridappsd.sh` script inside the docker container, use `Ctrl+C` to stop the platform. Some error messages may be displayed as the platform services are stopped.

|gapps_cntrl_c|

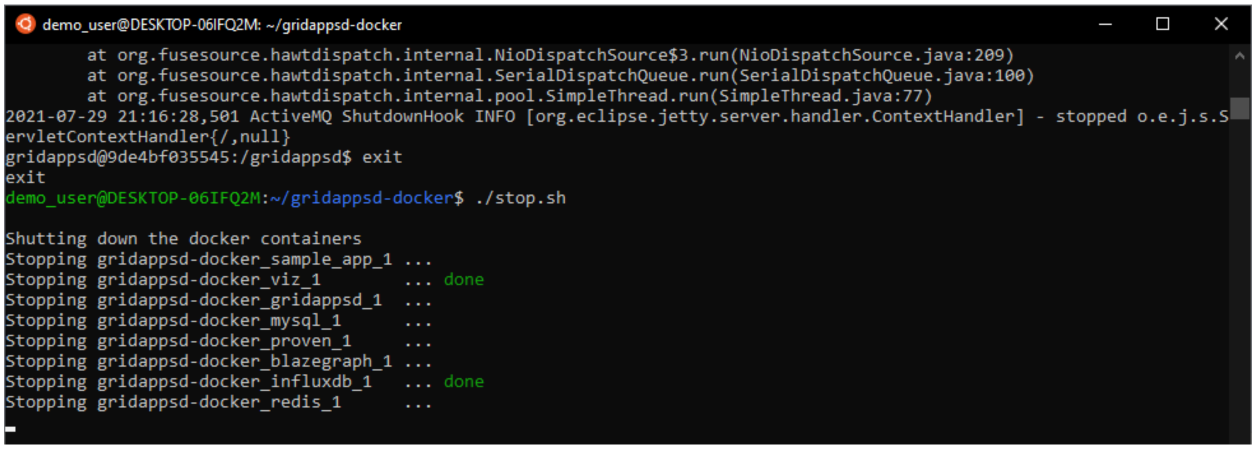
Then exit the docker container using `exit`. This will return to the main ubuntu bash command line:



gridappsd@9de4bf035545: /gridappsd

```
        at org.fusesource.hawtdispatch.internal.NioDispatchSource$3.run(NioDispatchSource.java:209)
        at org.fusesource.hawtdispatch.internal.SerialDispatchQueue.run(SerialDispatchQueue.java:100)
        at org.fusesource.hawtdispatch.internal.pool.SimpleThread.run(SimpleThread.java:77)
2021-07-29 21:16:28,501 ActiveMQ ShutdownHook INFO [org.eclipse.jetty.server.handler.ContextHandler] - stopped o.e.j.s.S
ervletContextHandler{/,null}
gridappsd@9de4bf035545:/gridappsd$ exit
```

Then run the `./stop.sh` script to shut down all the docker containers and free RAM used by the Platform:

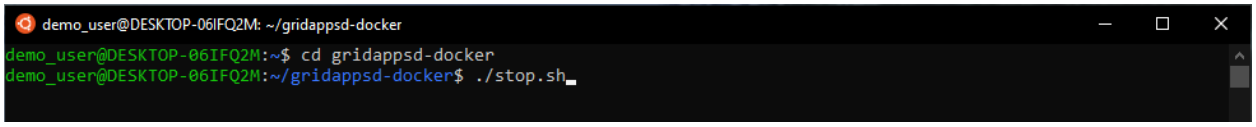


```
demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker
        at org.fusesource.hawtdispatch.internal.NioDispatchSource$3.run(NioDispatchSource.java:209)
        at org.fusesource.hawtdispatch.internal.SerialDispatchQueue.run(SerialDispatchQueue.java:100)
        at org.fusesource.hawtdispatch.internal.pool.SimpleThread.run(SimpleThread.java:77)
2021-07-29 21:16:28,501 ActiveMQ ShutdownHook INFO [org.eclipse.jetty.server.handler.ContextHandler] - stopped o.e.j.s.S
ervletContextHandler{/,null}
gridappsd@9de4bf035545:/gridappsd$ exit
exit
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./stop.sh

Shutting down the docker containers
Stopping gridappsd-docker_sample_app_1 ...
Stopping gridappsd-docker_viz_1        ... done
Stopping gridappsd-docker_gridappsd_1  ...
Stopping gridappsd-docker_mysql_1      ...
Stopping gridappsd-docker_proven_1     ...
Stopping gridappsd-docker_blazegraph_1 ...
Stopping gridappsd-docker_influxdb_1   ... done
Stopping gridappsd-docker_redis_1      ...
```

3.2.2 Stopping the Platform from a New Terminal

If your terminal was reset or closed, you can stop the platform and shut down all docker containers by changing directories into `gridappsd-docker` and running the `./stop.sh` script:

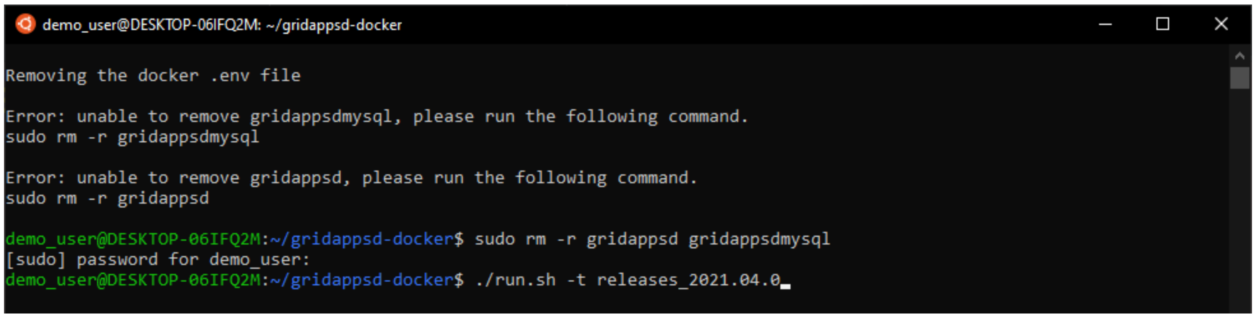
- `cd gridappsd-docker`
- `./stop.sh`



```
demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~$ cd gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./stop.sh
```

3.3 Restarting the Platform

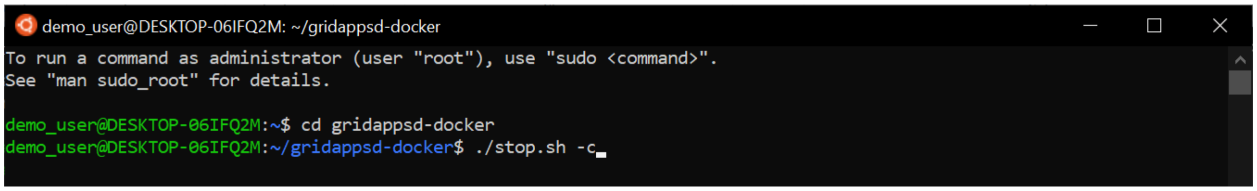
After the Platform has been stopped, it can be restarted by running the `./run.sh` script again from within the `gridappsd-docker` directory



```
demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker

Removing the docker .env file

Error: unable to remove gridappsdmysql, please run the following command.
sudo rm -r gridappsdmysql

Error: unable to remove gridappsd, please run the following command.
sudo rm -r gridappsd

demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ sudo rm -r gridappsd gridappsdmysql
[sudo] password for demo_user:
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./run.sh -t releases_2021.04.0
```

3.4 Changing Release Tags

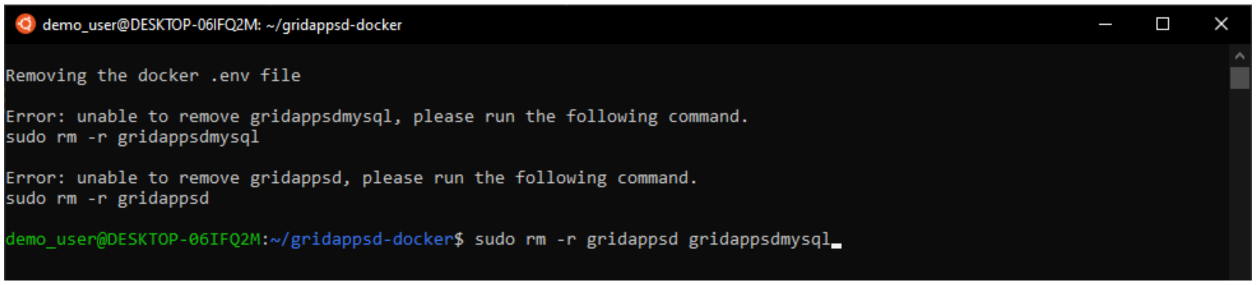
To change the Platform to run on a different release, run the `./stop.sh` script using the `-c` option to remove the current containers.



demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

demo_user@DESKTOP-06IFQ2M:~$ cd gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./stop.sh -c
```

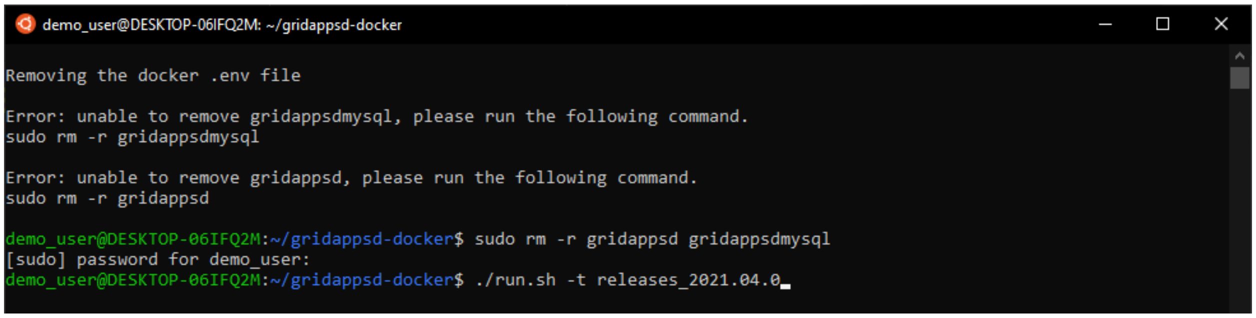
Remove the gridappsd directory using `sudo rm -r gridappsd gridappsdmysql`



demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker

```
Removing the docker .env file

Error: unable to remove gridappsdmysql, please run the following command.
sudo rm -r gridappsdmysql

Error: unable to remove gridappsd, please run the following command.
sudo rm -r gridappsd

demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ sudo rm -r gridappsd gridappsdmysql
```

Then start the platform again specifying the particular release tag desired. A complete list of platform releases is available in *Platform Release History*



demo_user@DESKTOP-06IFQ2M: ~/gridappsd-docker

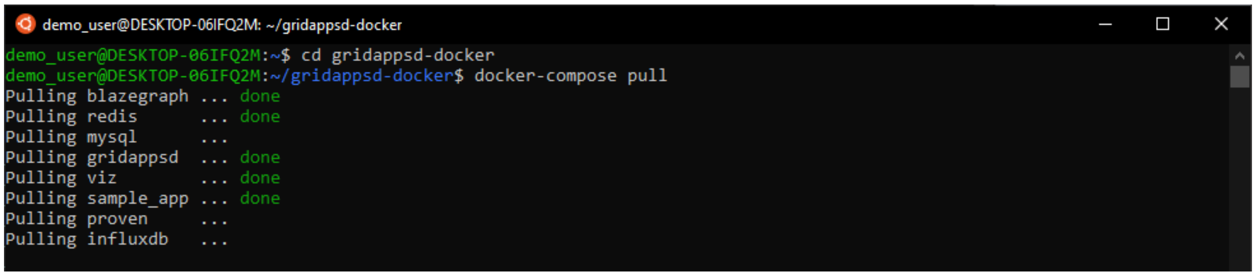
```
Removing the docker .env file

Error: unable to remove gridappsdmysql, please run the following command.
sudo rm -r gridappsdmysql

Error: unable to remove gridappsd, please run the following command.
sudo rm -r gridappsd

demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ sudo rm -r gridappsd gridappsdmysql
[sudo] password for demo_user:
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ ./run.sh -t releases_2021.04.0
```

3.5 Pulling Updated Containers

The GridAPPS-D platform should automatically check for and pull updated containers each time the `./run.sh` script is run.

However, it is sometimes necessary to force docker to pull new containers (e.g. if using a custom set of containers as specified by modifying the docker-compose.yml file).

New containers can be pulled by running `docker-compose pull` from within the `gridappsd-docker` directory



```
demo_user@DESKTOP-06IFQ2M:~$ cd gridappsd-docker
demo_user@DESKTOP-06IFQ2M:~/gridappsd-docker$ docker-compose pull
Pulling blazegraph ... done
Pulling redis      ... done
Pulling mysql      ...
Pulling gridappsd  ... done
Pulling viz        ... done
Pulling sample_app ... done
Pulling proven     ...
Pulling influxdb   ...
```

CHAPTER

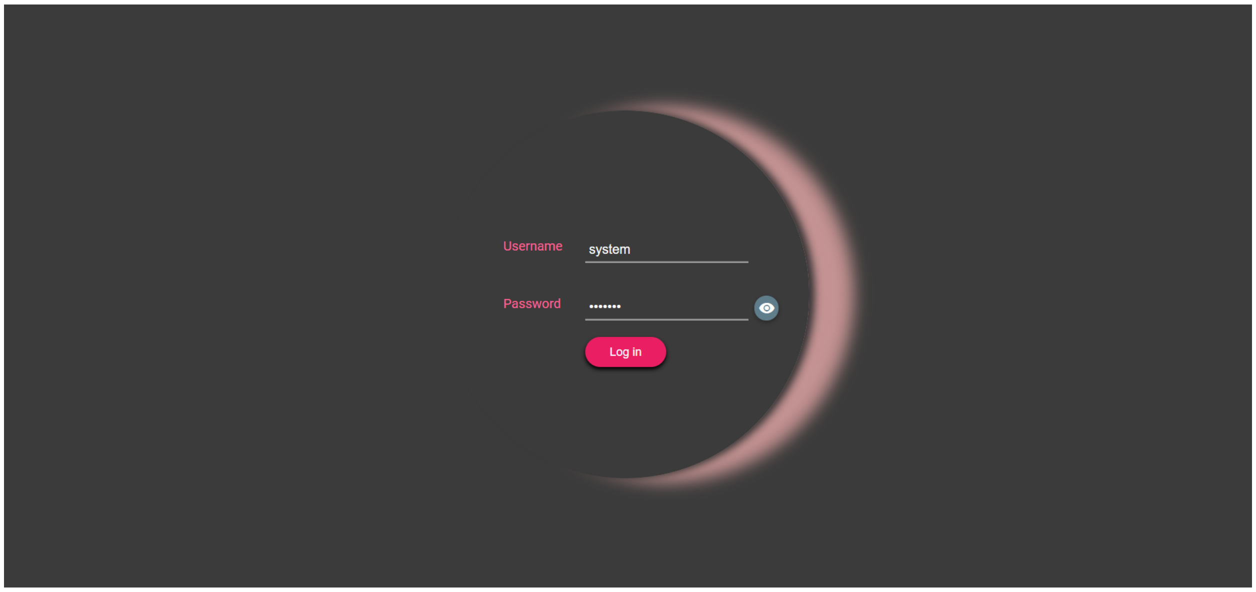
FOUR

USING THE GRIDAPPS-D VIZ

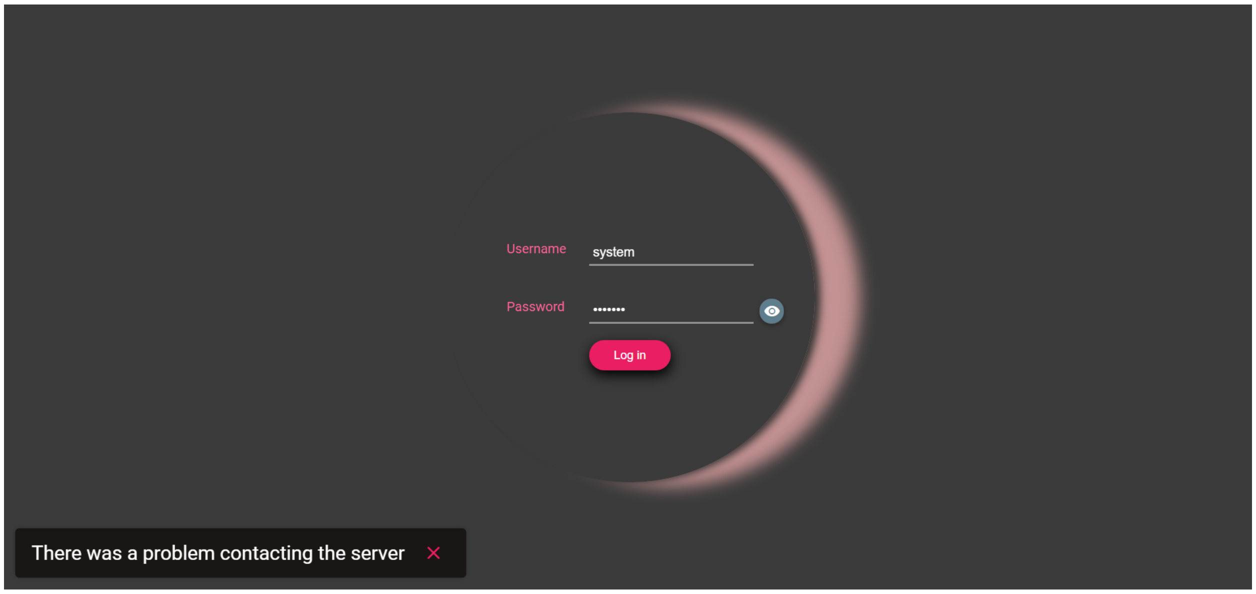
4.1 Accessing the GridAPPS-D Viz

Start the Platform using the `./run.sh` and `./run-gridappsd.sh` scripts as explained in *Running GridAPPS-D*. After the Platform is running, open your browser and navigate to localhost:8080 for desktop installations and `your.server.ip.address:8080` for cloud installations.

You should see a splash screen with login options for various user roles. The default username and password combinations are available in the GOSS core security config file.

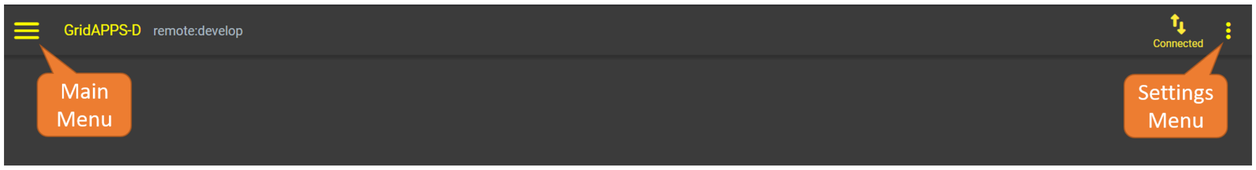


If the `./run-gridappsd.sh` script inside the GridAPPS-D docker container has not been started or has been interrupted, you will see a server connection error when attempting to log in:

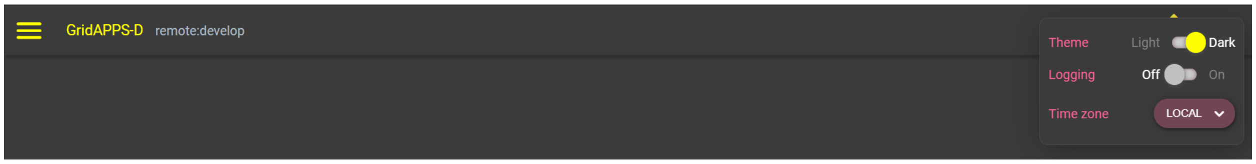


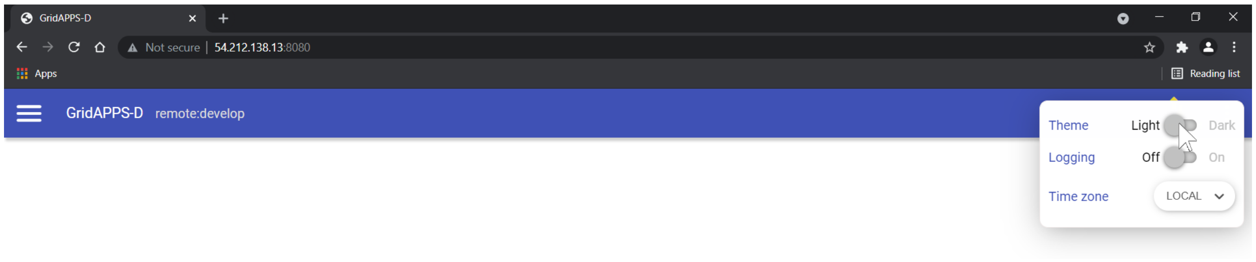
There was a problem contacting the server

After logging in, there are two menus in the top left and right corners. On the left is the main menu for using the VIZ features. On the right is the settings menu.

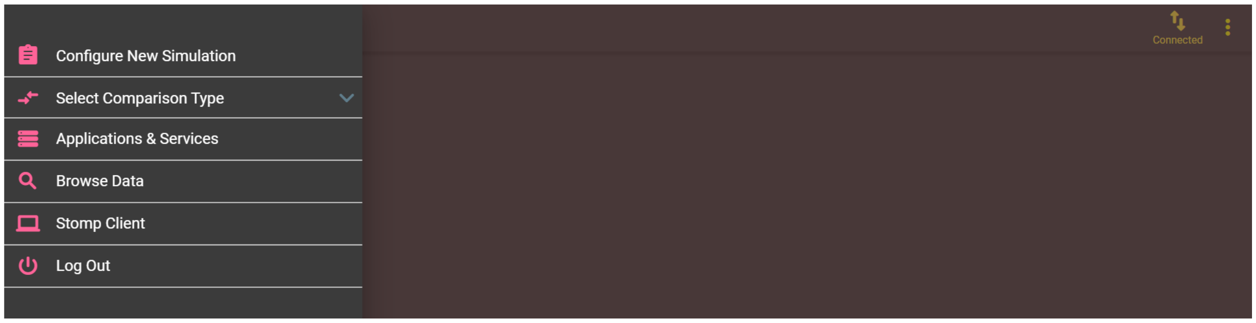


The settings menu contains options for changing the color theme from dark to light, the time zone used, and level of logging detail.



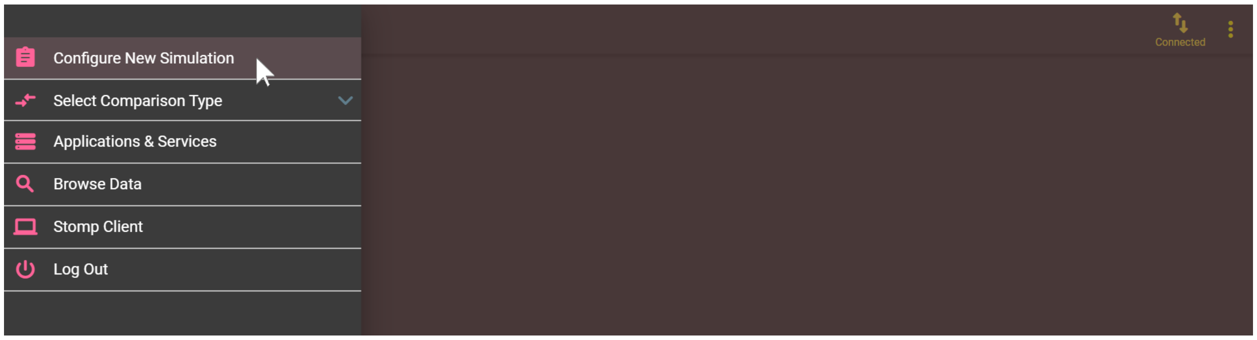


On the left is the main menu, which has options for creating simulations, comparing simulations, exploring the databases, and passing API calls through the STOMP client. Each of these will be explained in detail below.



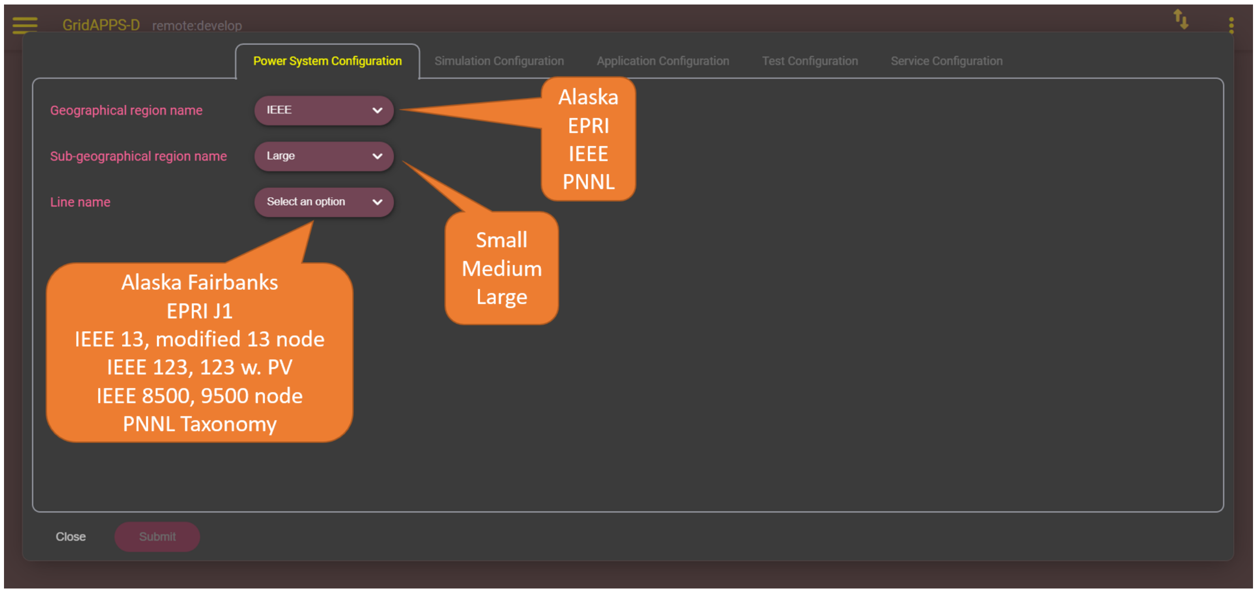
4.2 Creating a Simulation

To create a simulation from the GridAPPS-D VIZ, select `Configure New Simulation` from the main menu:



4.2.1 Power System Configuration

The first tab provides a set of menu options to select the desired distribution feeder on which the simulation will run. Detailed descriptions of the available test feeders inlcuded by default are provided in *Available Models in Default Installation*.

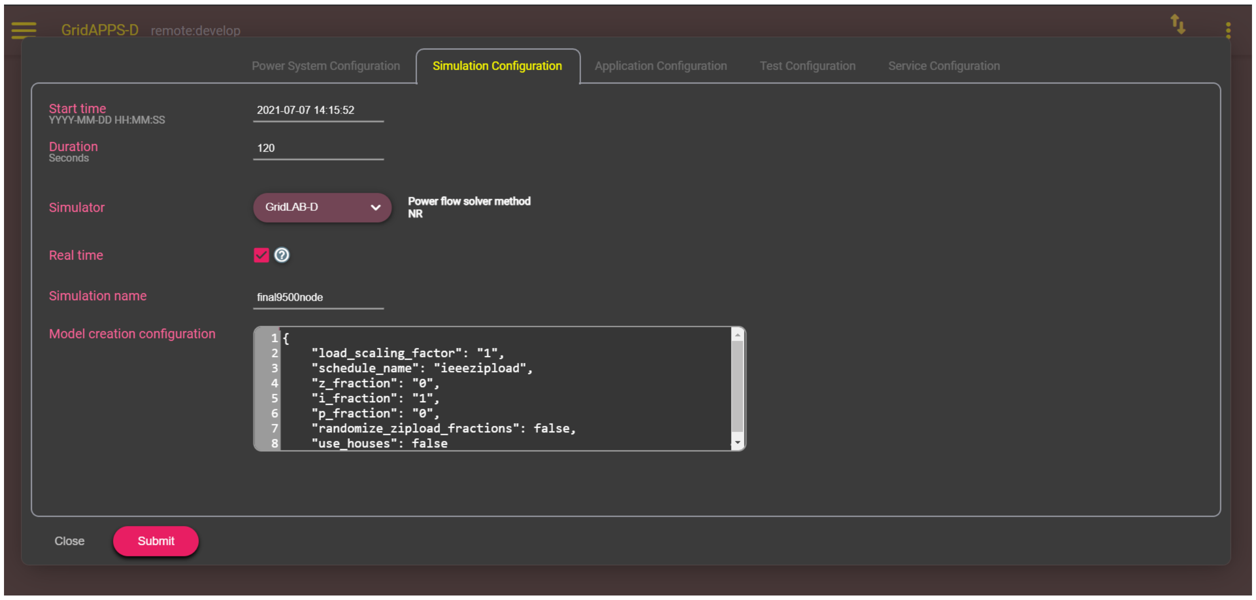


4.2.2 Simulation Configuration

The next tab provides several options for specifying

- simulation start time (which determines weather and load data used)
- simulation duration in seconds
- simulator used
- real-time
  - If checked, 1 sec of clock time equals 1 sec of simulation time
  - If unchecked, the simulation runs as fast as possible (used for generating AI training data)
- simulation name
- model configuration

The model creation configuration textbox enables modification of the model from the default configuration, inlcuding opening/closing switches, changing DER setpoints, and changing the load profile. Details of the syntax for custom configurations is explained in *Simulation Configuration Settings*

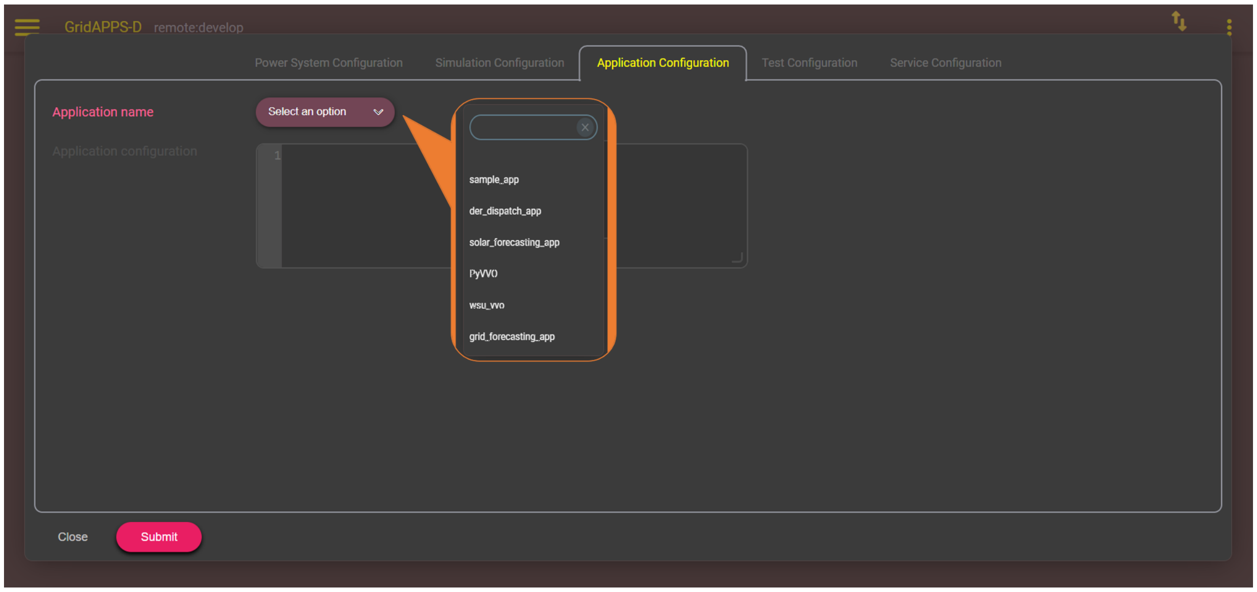

GridAPPS-D remote:develop
Power System Configuration
Simulation Configuration
Application Configuration
Test Configuration
Service Configuration
Close
Submit

Start time YYYY-MM-DD HH:MM:SS: 2021-07-07 14:15:52
Duration Seconds: 120
Simulator: GridLAB-D
Power flow solver method NR
Real time: ✓
Simulation name: final9500node
Model creation configuration:

```
{
    "load_scaling_factor": "1",
    "schedule_name": "ieeezipload",
    "z_fraction": "0",
    "i_fraction": "1",
    "p_fraction": "0",
    "randomize_zipload_fractions": false,
    "use_houses": false
```

4.2.3 Application Configuration

The application configuration tab enables selection of which app should run during the simulation. The GridAPPS-D VIZ allows for selection of just one app at a time. Two or more apps may be run simultaneously by starting the simulation through the Simulation API and specifying the desired apps using the *Application Configuration Settings*

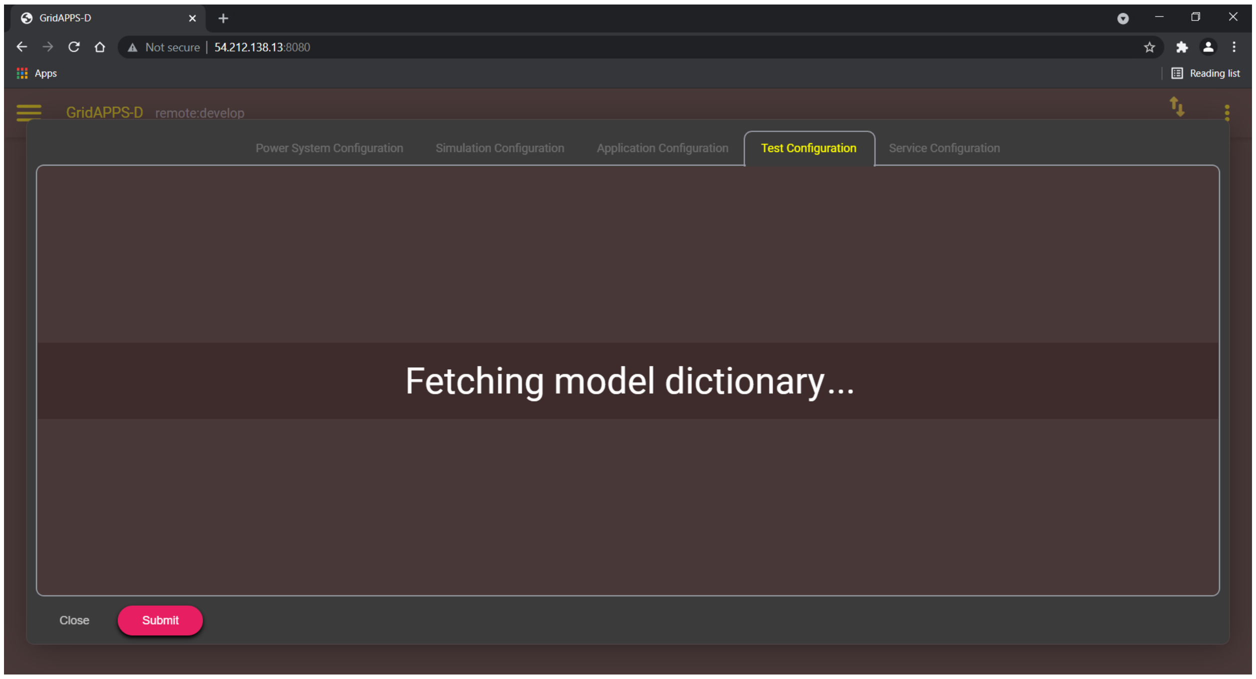


4.2.4 Test Manager Configuration

The Test Manager is used to create realistic operational events during the course of the simulation, including faults, communication outages, and custom event scripts.

The GridAPPS-D Viz provides a graphic interface for building TestManager event scripts, which are described in detail in *Test Manager Configuration*.

For large models (such as the IEEE 8500 node and 9500 node test systems), it may take several seconds for the model dictionary to load. While the power system model is being queried, a splashscreen will be displayed.


GridAPPS-D
Not secure | 54.212.138.13:8080
Apps
Reading list
GridAPPS-D remote:develop
Power System Configuration
Simulation Configuration
Application Configuration
Test Configuration
Service Configuration
Fetching model dictionary...
Close
Submit

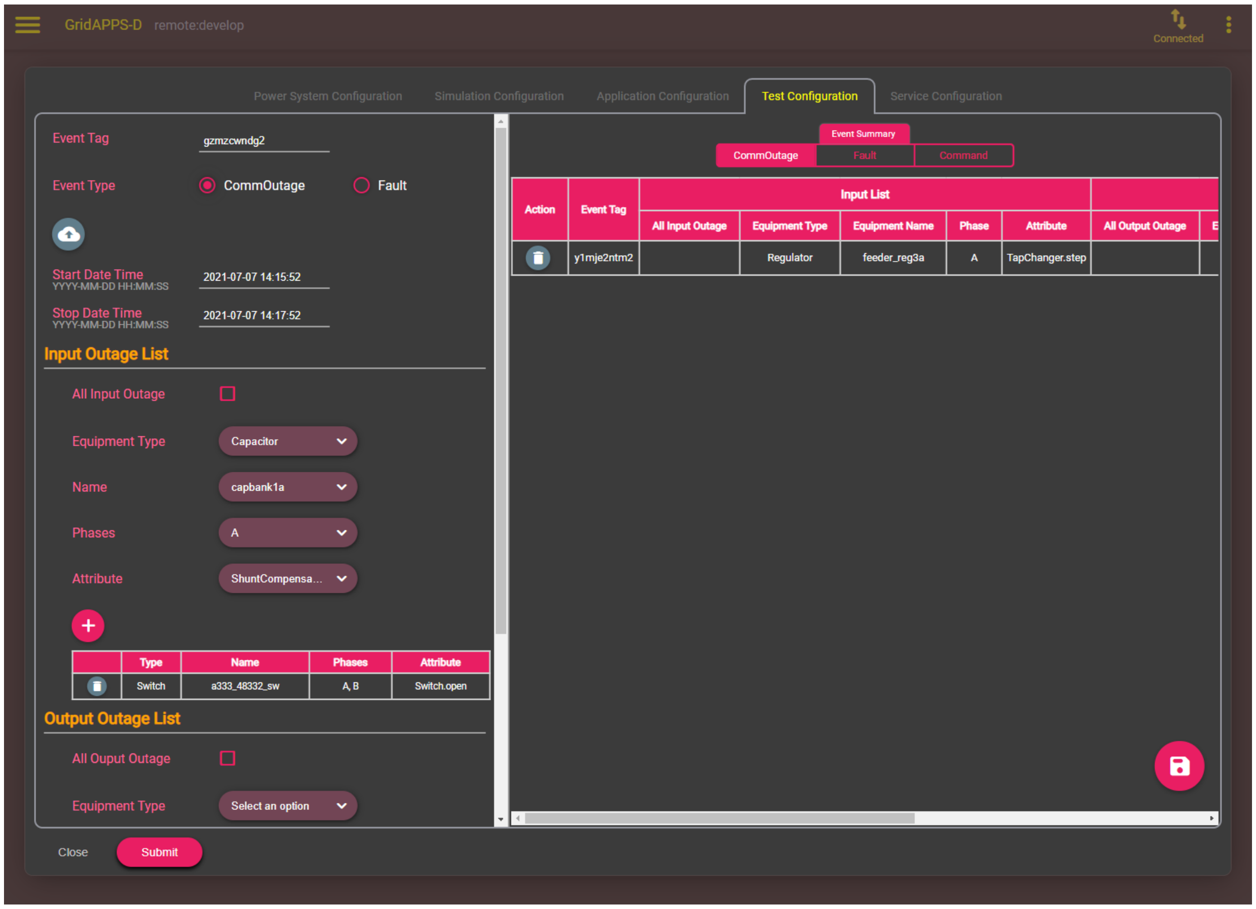
Communication outages

The first option is to create communication outages for both SCADA measurements and equipment control commands by clicking the `CommOutage` radio button.

Equipment control commands can be blocked by specifying an Input Outage with options for

- start time
- end time
- all equipment (outages all commands for all equipment if checked)
- equipment type
- equipment name
- affected phases
- CIM control attribute

To add an Input Outage, specify the desired options, click the + button, and then click the `Add event` button at the bottom of the page. The speficied outages will then be displayed on the right side under the `CommOutage` tab.



Event Tag gzmzcwndg2

Event Type [✓ CommOutage] Fault

Start Date Time YYYY-MM-DD HH:MM:SS 2021-07-07 14:15:52

Stop Date Time YYYY-MM-DD HH:MM:SS 2021-07-07 14:17:52

Input Outage List

All Input Outage ☐

Equipment Type Capacitor

Name capbank1a

Phases A

Attribute ShuntCompensa...

	Type	Name	Phases	Attribute
	Switch	a333_48332_sw	A, B	Switch.open

Output Outage List

All Ouput Outage ☐

Equipment Type Select an option

Event Summary

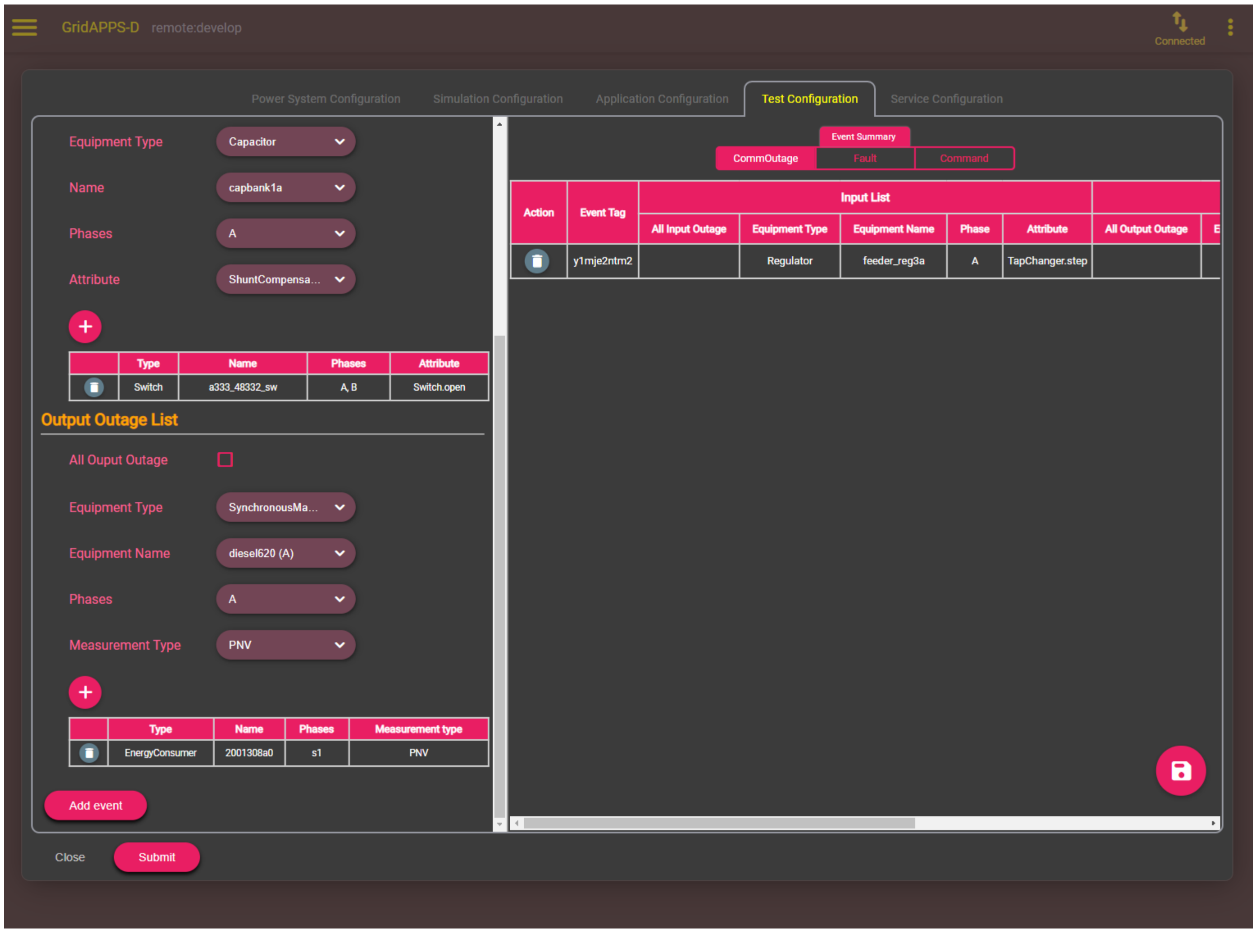
[✓ CommOutage] Fault Command

Action	Event Tag	Input List						
		All Input Outage	Equipment Type	Equipment Name	Phase	Attribute	All Output Outage	E
	y1mje2ntm2		Regulator	feeder_reg3a	A	TapChanger.step		

Measurement sensor outages can be created with an Output Outage with options for

- all measurements (all sensors outages if checked)
- equipment type
- equipment name
- affected phases
- measurement type (voltage, apparent power, position)

To add an Output Outage, specify the desired options, click the + button, and then click the `Add event` button at the bottom of the page. The speficied outages will then be displayed on the right side under the `CommOutage` tab.



Equipment Type: Capacitor
Name: capbank1a
Phases: A
Attribute: ShuntCompensa...

	Type	Name	Phases	Attribute
🗑	Switch	a333_48332_sw	A, B	Switch.open

Output Outage List

All Ouput Outage: ☐
Equipment Type: SynchronousMa...
Equipment Name: diesel620 (A)
Phases: A
Measurement Type: PNV

	Type	Name	Phases	Measurement type
🗑	EnergyConsumer	2001308a0	s1	PNV

Action	Event Tag	Input List						
		All Input Outage	Equipment Type	Equipment Name	Phase	Attribute	All Output Outage	E
🗑	y1mje2ntm2		Regulator	feeder_reg3a	A	TapChanger.step		

Equipment Faults

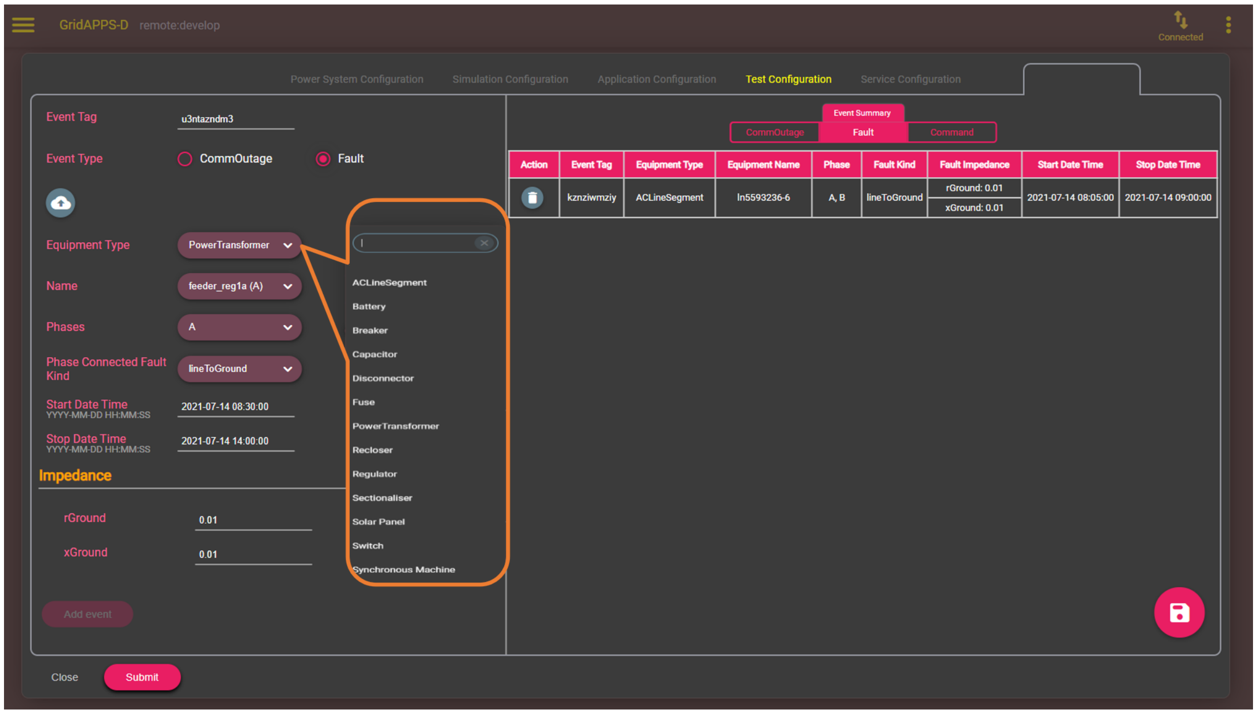
The next option is to create a fault on any class of equipment in the model by clicking the `Fault` radio button. Options will appear to select the

- equipment type
- equipment name
- phases to be faulted
- type of fault
- start time
- end time
- fault impedance

After setting the values, click the `Add event` button. The fault will be displayed on the right side of the screen under the `Fault` tab.

When the simulation reaches the event start time, the logical fault simulator traces the fault up to first upstream switch and opens that switch to clear the fault.

Note: The GridLAB-D logical fault simulator currently may not process faults in meshed networks correctly and fail to open the appropriate set of switches to clear the fault. This is a known bug with a workaround to open the appropriate switches using a custom Test Manager event file.



Event Tag	u3ntazndm3
Event Type	CommOutage [✓ Fault]
Equipment Type	PowerTransformer
Name	feeder_reg1a (A)
Phases	A
Phase Connected Fault Kind	lineToGround
Start Date Time YYYY-MM-DD HH:MM:SS	2021-07-14 08:30:00
Stop Date Time YYYY-MM-DD HH:MM:SS	2021-07-14 14:00:00

Impedance

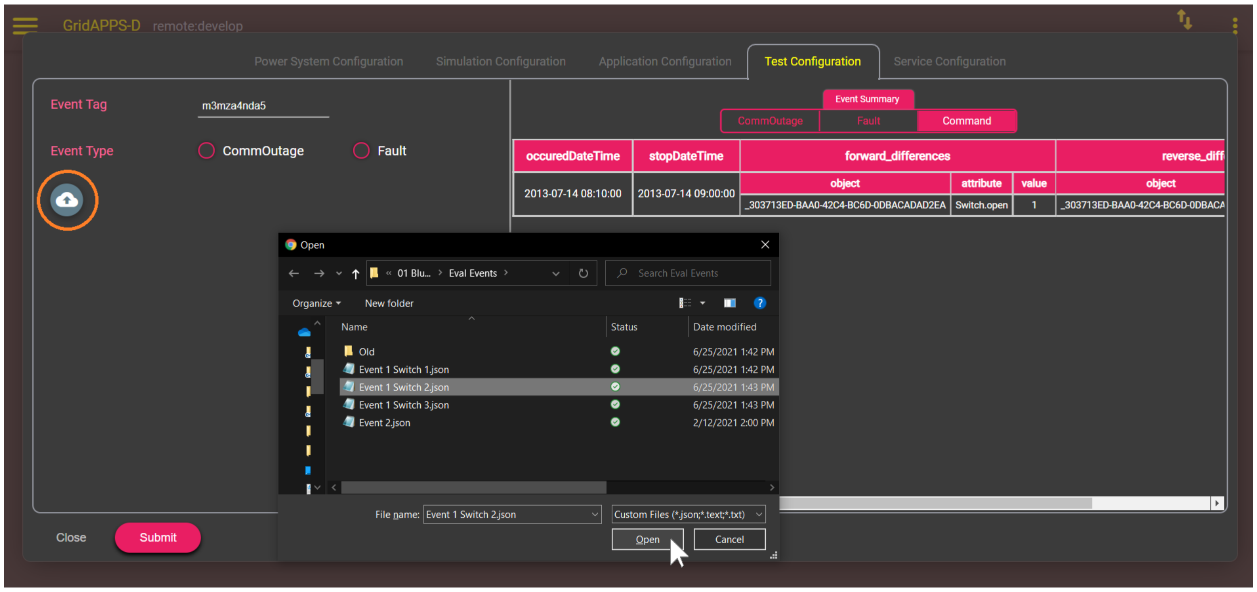
rGround	0.01
xGround	0.01

Action	Event Tag	Equipment Type	Equipment Name	Phase	Fault Kind	Fault Impedance	Start Date Time	Stop Date Time
	kznziwmziy	ACLineSegment	ln5593236-6	A, B	lineToGround	rGround: 0.01 xGround: 0.01	2021-07-14 08:05:00	2021-07-14 09:00:00

Custom Event Files

Custom event messages can be written in format of a CIM Difference Message. The format of custom event files is explained in *Scheduled Command Events*.

Event files can be uploaded by clicking the cloud-shaped button and selecting the desired files to be uploaded from your local machine. Click `Open` to upload each file. The custom event will then be displayed under the `Command` tab.



Event Tag	m3mza4nda5
Event Type	CommOutage Fault

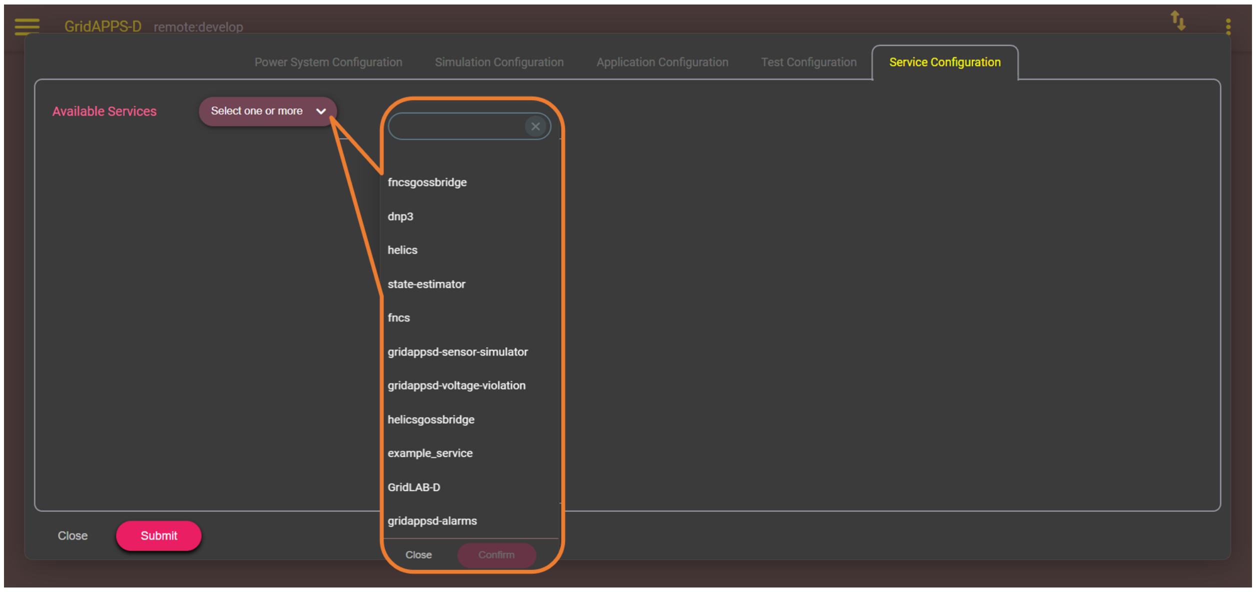
occuredDateTime	stopDateTime	forward_differences			reverse_diff
		object	attribute	value	object
2013-07-14 08:10:00	2013-07-14 09:00:00	_303713ED-BAA0-42C4-BC6D-0DBACADAD2EA	Switch.open	1	_303713ED-BAA0-42C4-BC6D-0DBACA

Name	Status	Date modified
Old	✓	6/25/2021 1:42 PM
Event 1 Switch 1.json	✓	6/25/2021 1:42 PM
Event 1 Switch 2.json	✓	6/25/2021 1:43 PM
Event 1 Switch 3.json	✓	6/25/2021 1:43 PM
Event 2.json	✓	2/12/2021 2:00 PM

File name: Event 1 Switch 2.json

4.2.5 2.5. Service Configuration

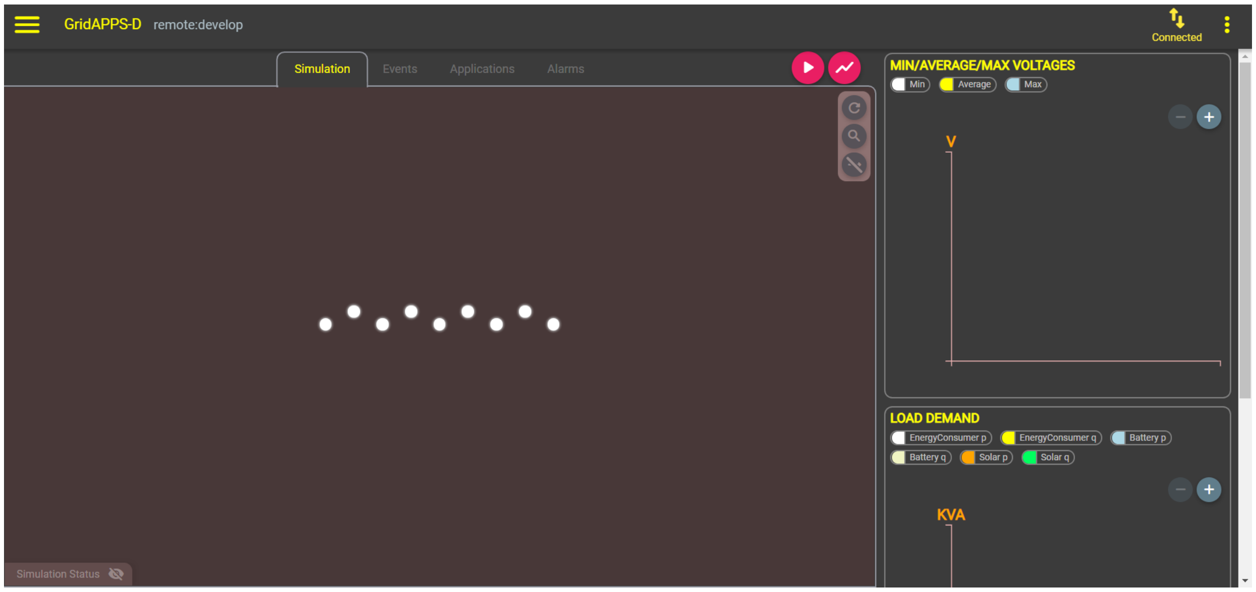
The last tab is the GridAPPS-D service configuration. This enables selection of particular services that should be run along with the simulation, such as the Sensor Simulator or DNP3 service.



After selecting all desired simulation parameters, click `Submit` to start the simulation.

4.3 Running a Simulation with VIZ

After clicking `Submit`, the GridAPPS-D VIZ will generate a new oneline diagram for the desired feeder. Large models, such as the IEEE 8500 and 9500 node test systems, will take several seconds to load. A splashscreen will be displayed while the oneline diagram is being created.

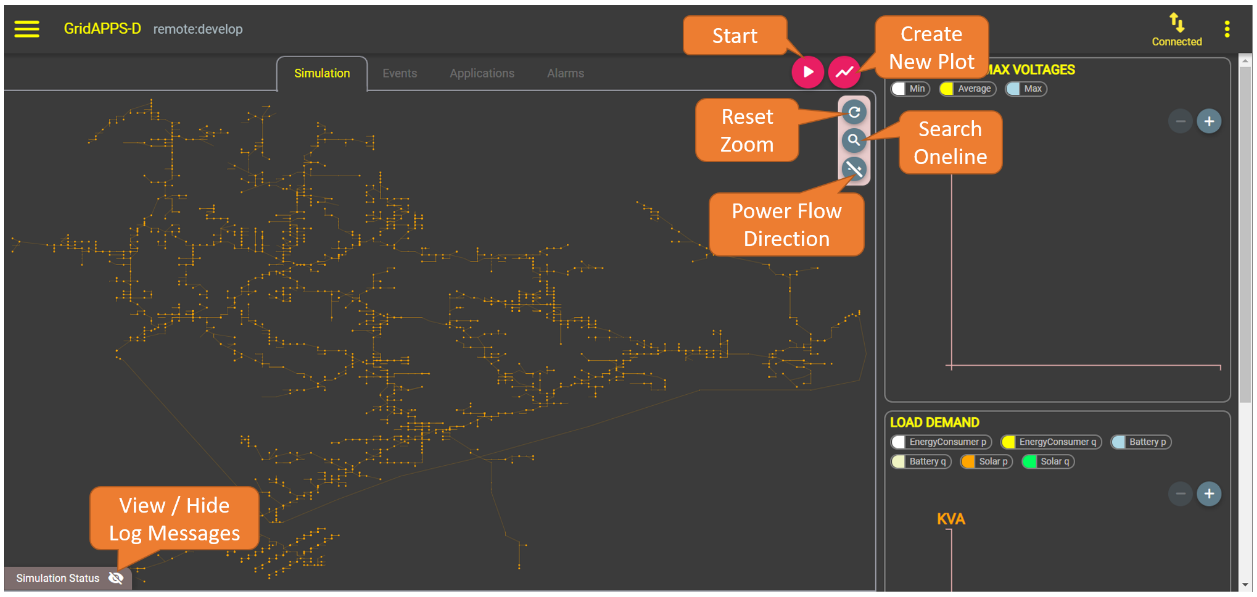


The oneline diagram will then appear in the main window under the `Simulation` tab.

Across the tab are three more tabs for events loaded into the simulation, applications configured to run, and SCADA alarms received during the simulation.

4.3.1 Simulation Tab

The Simulation Tab contains options for viewing and controlling equipment in the feeder model.

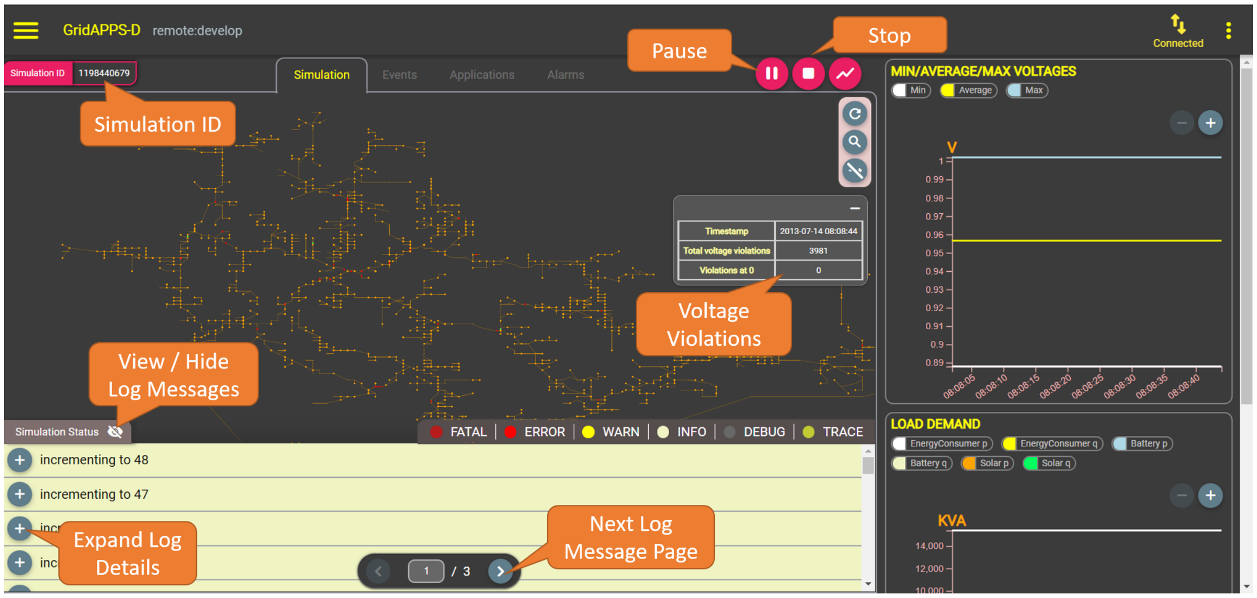


Starting, Pausing, and Stopping a Simulation

To start the simulation, click the start button shown above. A "Simulation is starting" message will be displayed while the CIM model is imported, converted, and configured to run with the specified simulation parameters.

After the simulation starts, a unique simulation ID will be created and displayed in the top left corner. This ID is used by applications to communicate with the correct simulation (as multiple simulations may be running simulateously).

The simulation may be paused or stopped using the pause and stop button in the top right corner of the simulation window.



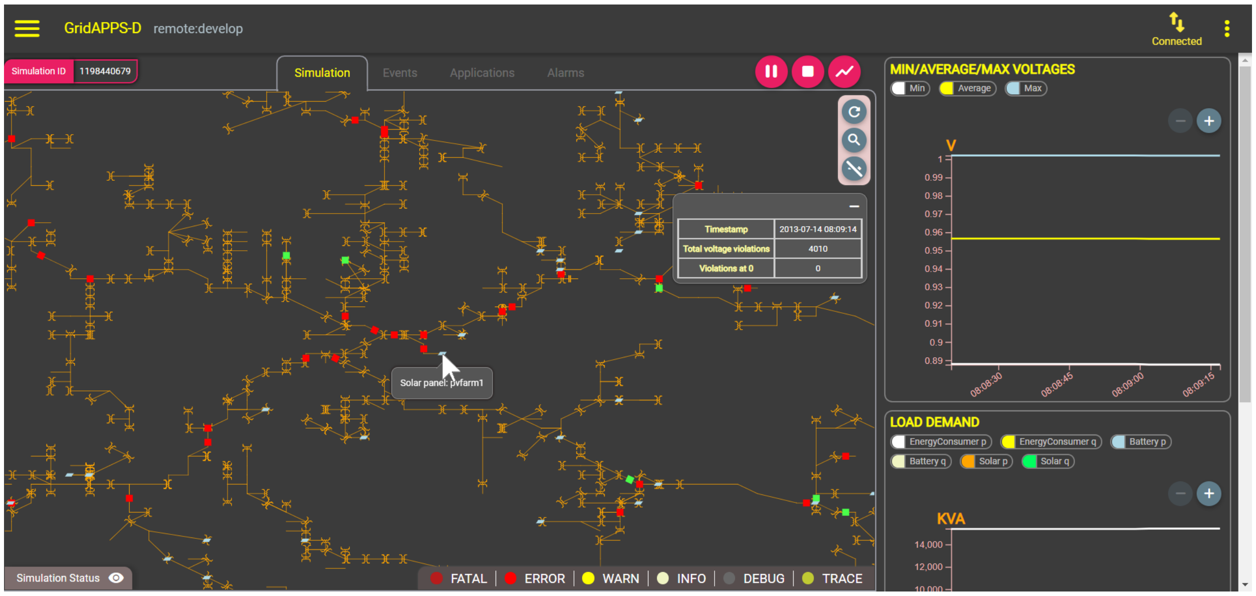
Timestamp	2013-07-14 08:08:44
Total voltage violations	3981
Violations at 0	0

incrementing to 48

incrementing to 47

Viewing equipment names

To view the names of devices in the feeder, zoom in using the mouse wheel and hover over the desired piece of equipment. Its name will be displayed below.

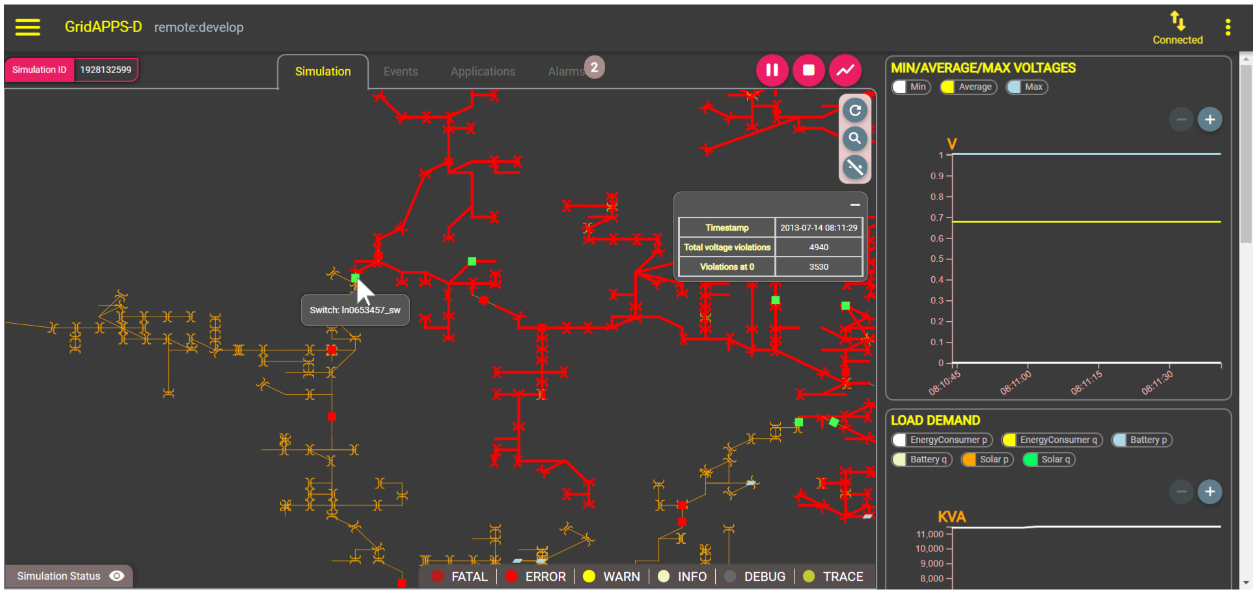


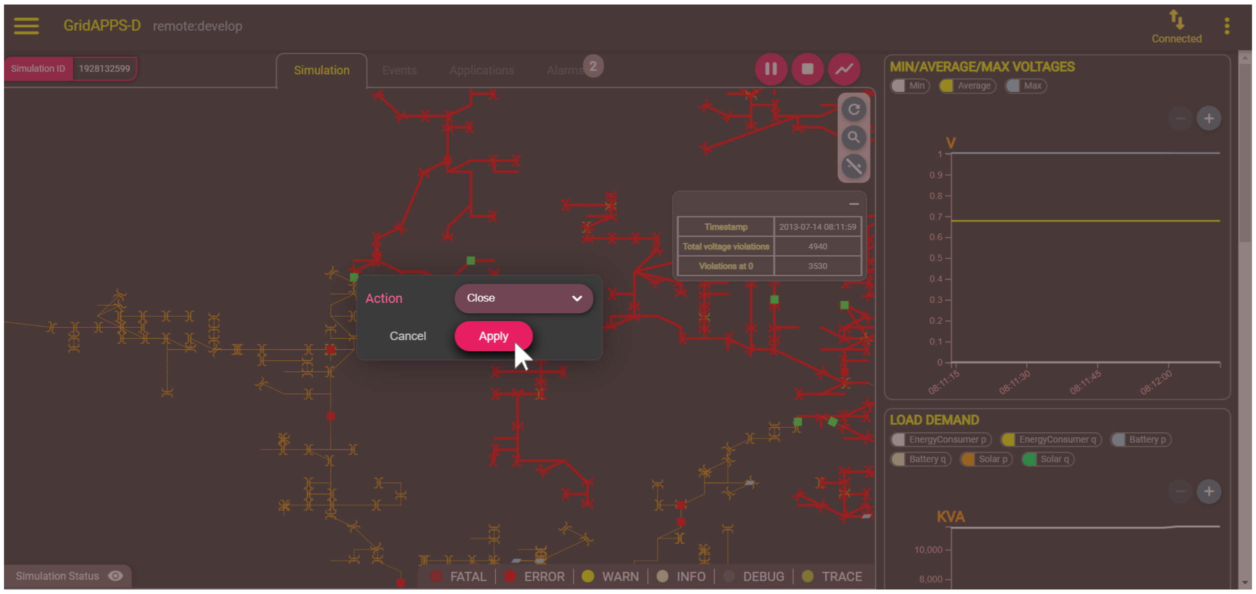
Timestamp	2013-07-14 08:09:14
Total voltage violations	4010
Violations at 0	0

Opening and Closing Switches

Closed switches are displayed as a red breaker square. Open switches are green.

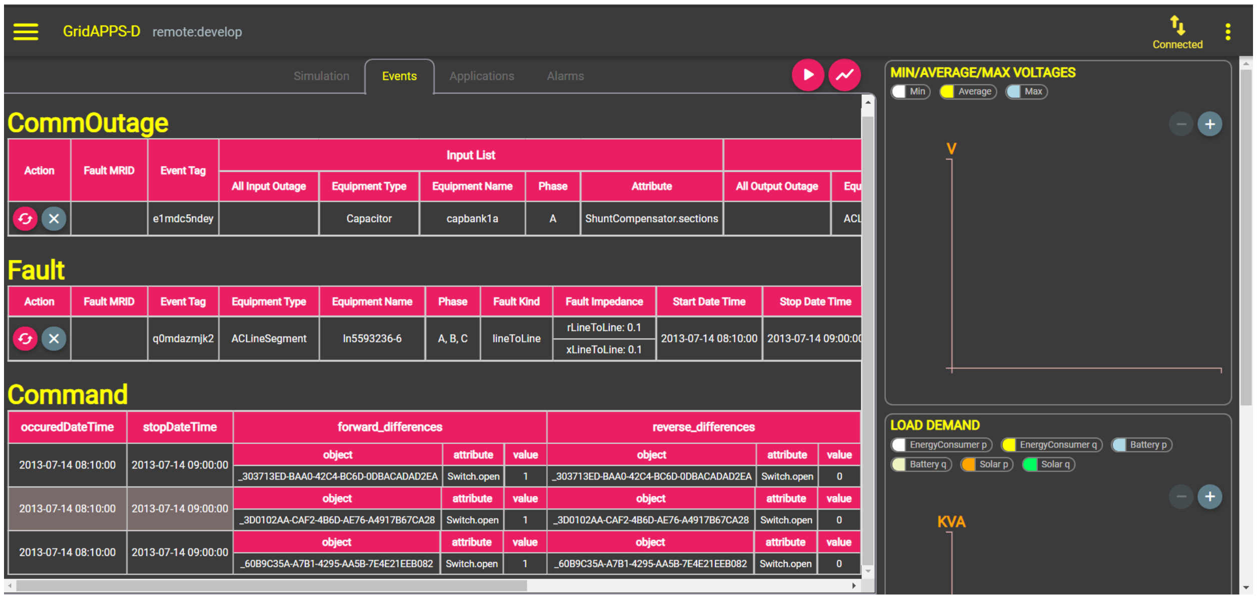
To open or close a switch, left-click on the square and select the desired state from the menu. Click `Apply`





4.3.2 Events Tab

The Events Tab shows events that were created using the TestManager and are scheduled to occur over the course of the simulation.



CommOutage

Action	Fault MRID	Event Tag	Input List						
			All Input Outage	Equipment Type	Equipment Name	Phase	Attribute	All Output Outage	Equ
		e1mdc5ndey		Capacitor	capbank1a	A	ShuntCompensator.sections		ACL

Fault

Action	Fault MRID	Event Tag	Equipment Type	Equipment Name	Phase	Fault Kind	Fault Impedance	Start Date Time	Stop Date Time
		q0mdazmjk2	ACLineSegment	ln5593236-6	A, B, C	lineToLine	rLineToLine: 0.1 xLineToLine: 0.1	2013-07-14 08:10:00	2013-07-14 09:00:0

Command

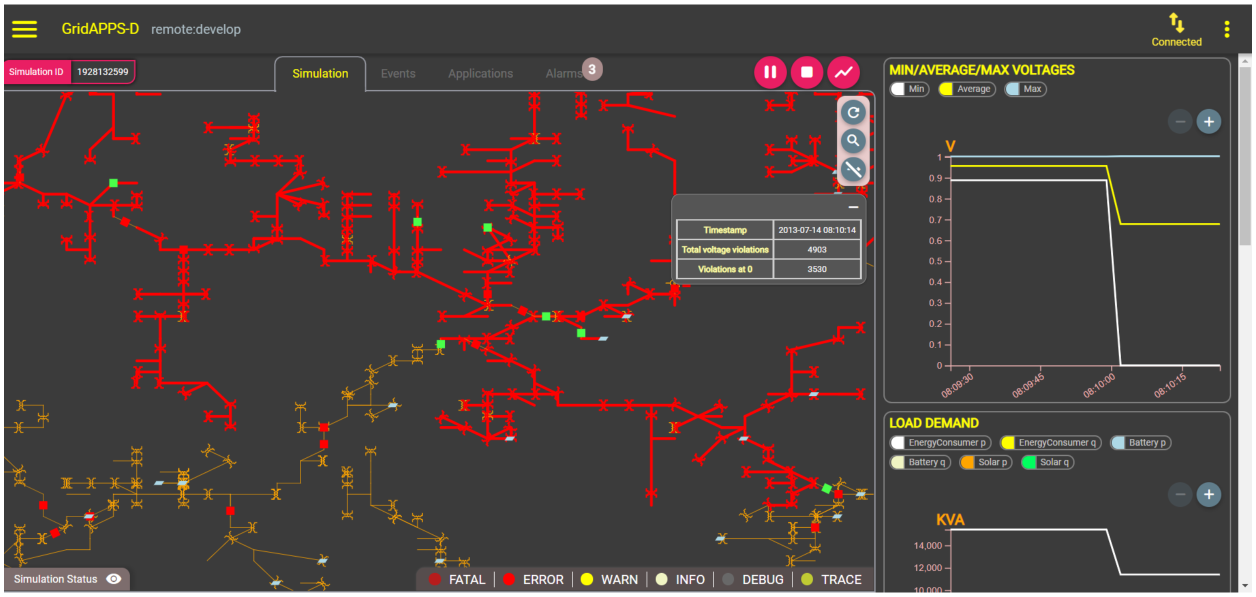
occuredDateTime	stopDateTime	forward_differences			reverse_differences		
		object	attribute	value	object	attribute	value
2013-07-14 08:10:00	2013-07-14 09:00:00	_303713ED-BAA0-42C4-BC6D-0DBACADAD2EA	Switch.open	1	_303713ED-BAA0-42C4-BC6D-0DBACADAD2EA	Switch.open	0
		object	attribute	value	object	attribute	value
2013-07-14 08:10:00	2013-07-14 09:00:00	_3D0102AA-CAF2-4B6D-AE76-A4917B67CA28	Switch.open	1	_3D0102AA-CAF2-4B6D-AE76-A4917B67CA28	Switch.open	0
		object	attribute	value	object	attribute	value
2013-07-14 08:10:00	2013-07-14 09:00:00	_60B9C35A-A7B1-4295-AA5B-7E4E21EEB082	Switch.open	1	_60B9C35A-A7B1-4295-AA5B-7E4E21EEB082	Switch.open	0

4.3.3 Alarms Tab

The GridAPPS-D Alarms service monitors the GOSS Message Bus and generates alarms for common equipment control actions, including

- Switch open/close
- Capacitor open/close
- Regulator tap change

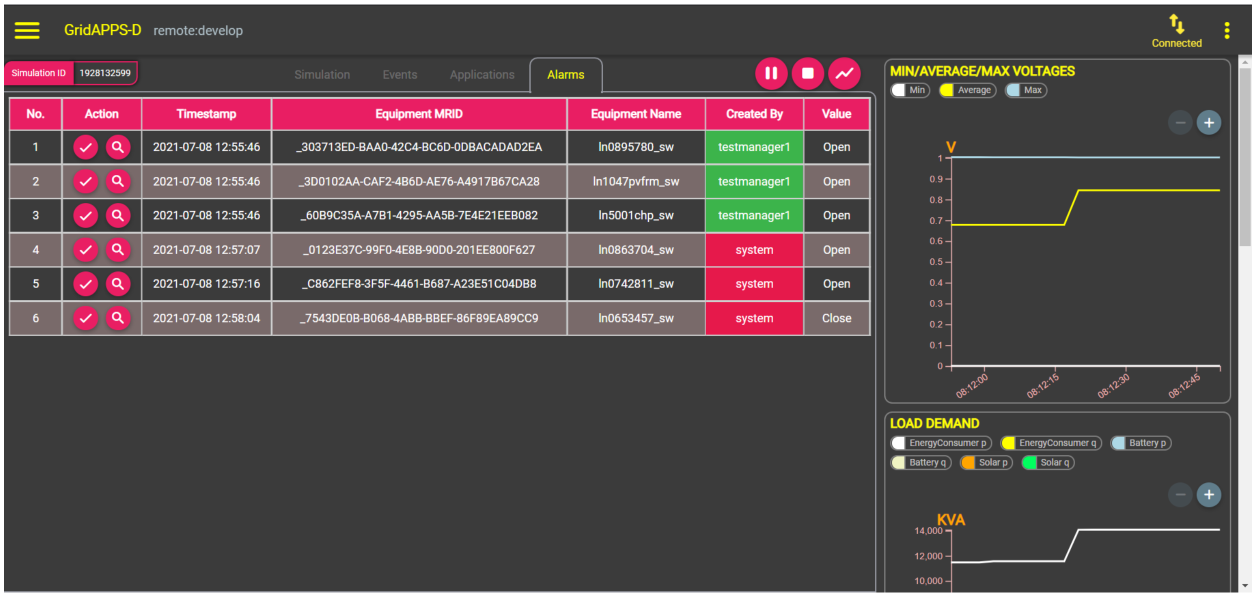
The Alarms Tab is the last one in the top ribbon and diplays a notification with the number of new alarms that have not been viewed:



Timestamp	2013-07-14 08:10:14
Total voltage violations	4903
Violations at 0	3530

Under the Alarms Tab is a table indicating the piece of equipment for which the alarm occured, the time of the alarm, and source of the action.

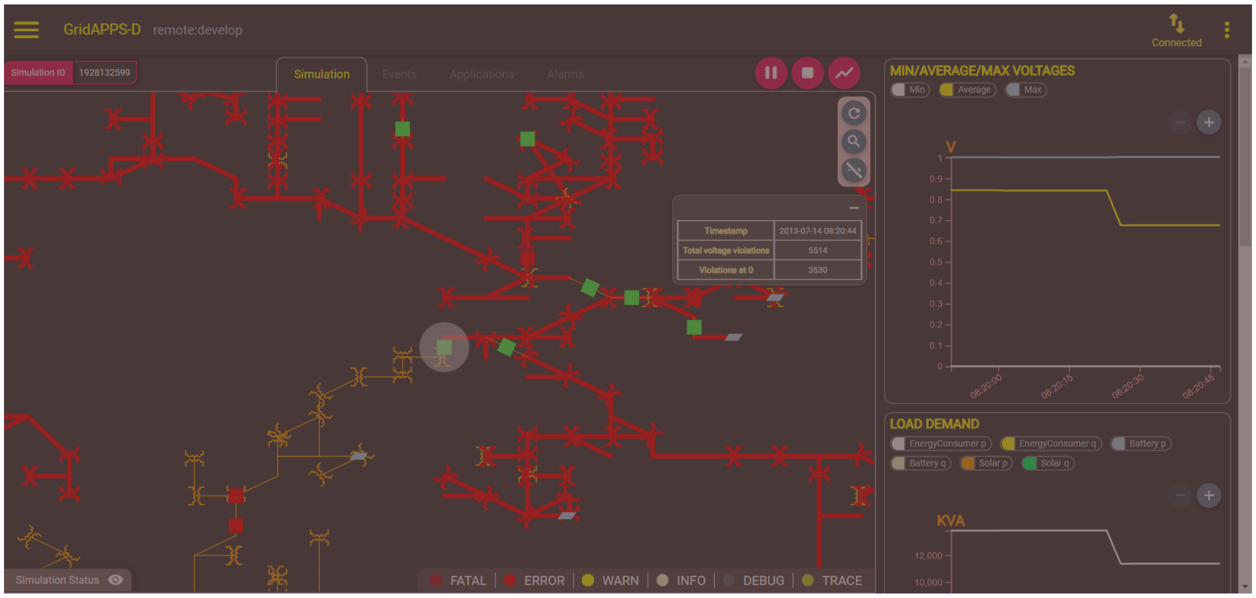
Alarms created by events defined in the TestManager (such as fault events and scheduled command events) appear as being created by testmanager1:



No.	Action	Timestamp	Equipment MRID	Equipment Name	Created By	Value
1	✓ 🔍	2021-07-08 12:55:46	_303713ED-BAA0-42C4-BC6D-0DBACADAD2EA	ln0895780_sw	testmanager1	Open
2	✓ 🔍	2021-07-08 12:55:46	_3D0102AA-CAF2-4B6D-AE76-A4917B67CA28	ln1047pvfrm_sw	testmanager1	Open
3	✓ 🔍	2021-07-08 12:55:46	_60B9C35A-A7B1-4295-AA5B-7E4E21EEB082	ln5001chp_sw	testmanager1	Open
4	✓ 🔍	2021-07-08 12:57:07	_0123E37C-99F0-4E8B-90D0-201EE800F627	ln0863704_sw	system	Open
5	✓ 🔍	2021-07-08 12:57:16	_C862FEF8-3F5F-4461-B687-A23E51C04DB8	ln0742811_sw	system	Open
6	✓ 🔍	2021-07-08 12:58:04	_7543DE0B-B068-4ABB-BBEF-86F89EA89CC9	ln0653457_sw	system	Close

Alarms can be acknowledged and removed from the table by clicking the check mark button on the far left of each row.

The alarm location can be viewed by clicking on the magnifying glass button. This will switch to the Simulation tab and the particular piece of equipment will be highlighted with an expanding circle.

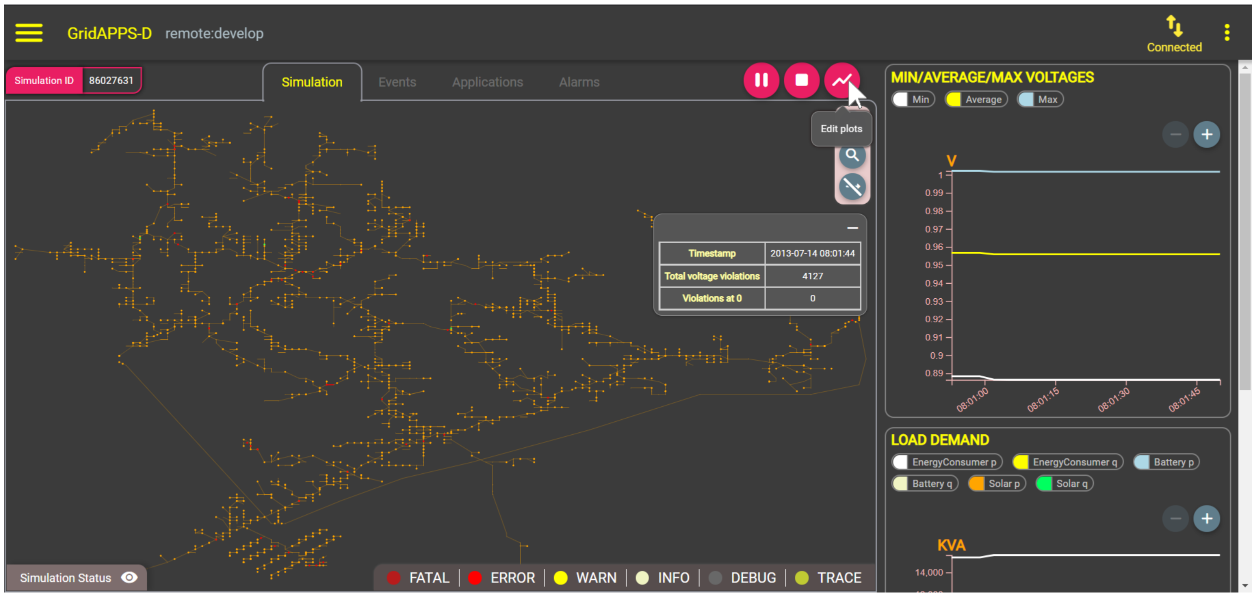


Timestamp	2013-07-14 08:20:44
Total voltage violations	5514
Violations at 0	3530

4.3.4 Creating Stripchart Plots

GridAPPS-D contains a set of basic plotting features for common power system parameters

New plots can be added by cliking the trendline button to the right of the pause and stop buttons.



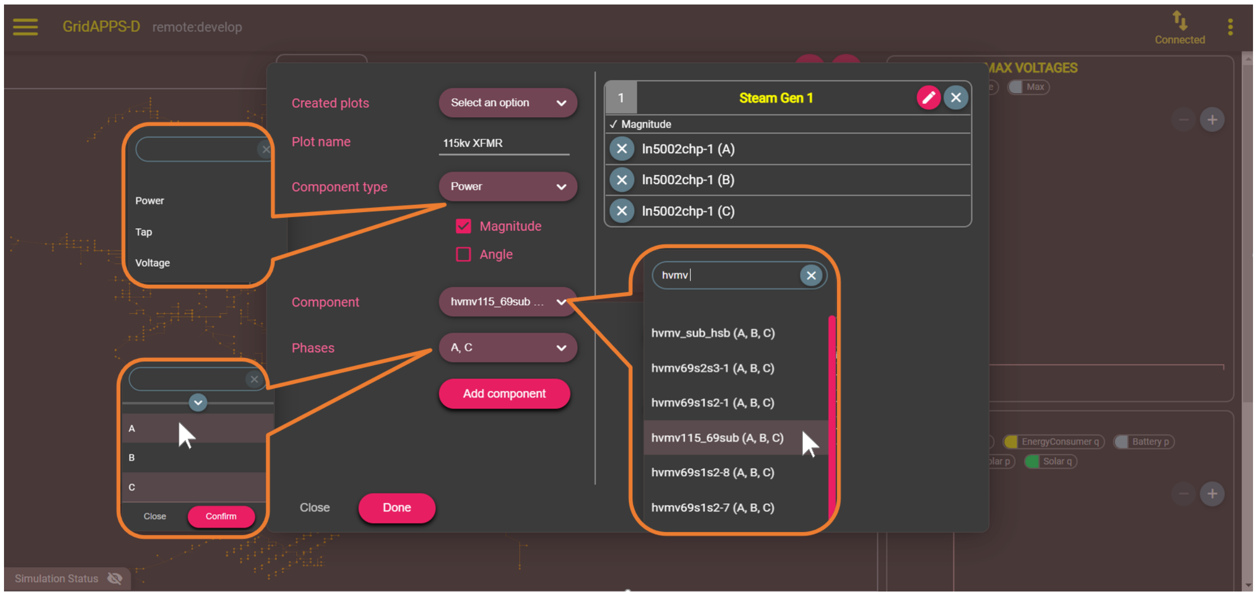
Timestamp	2013-07-14 08:01:44
Total voltage violations	4127
Violations at 0	0

A pop-up dialog will appear to select the desired plots with several options:

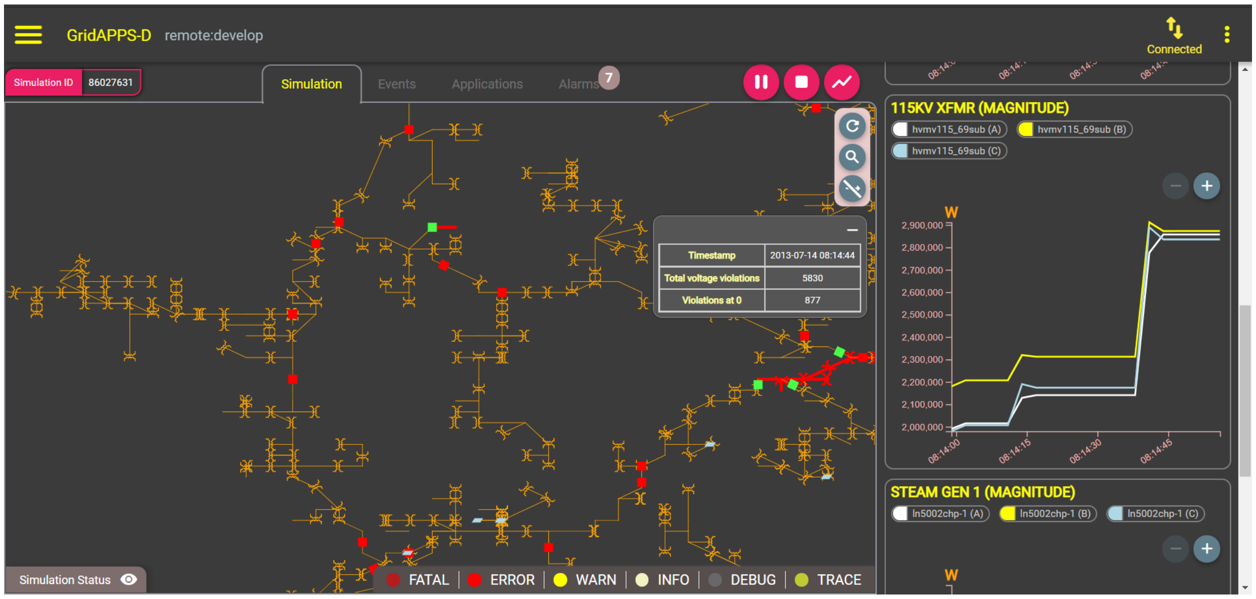
- Created plots – edit custom plots that have already been created
- Plot name – custom text string that will be displayed as plot title
- Component type – Power (`VA`), Tap (`Pos`), or Voltage (`PNV`) measurement type
- Magnitude / Angle

  - Apparent power & acos(power factor) for power measurements
  - Voltage magnitude & phase angle for voltage measurements
- Component – Component name, type to search
- Phases – Any combination of A, B, and C phases to plot

Be sure to click the `Add component` button before clicking `Done`

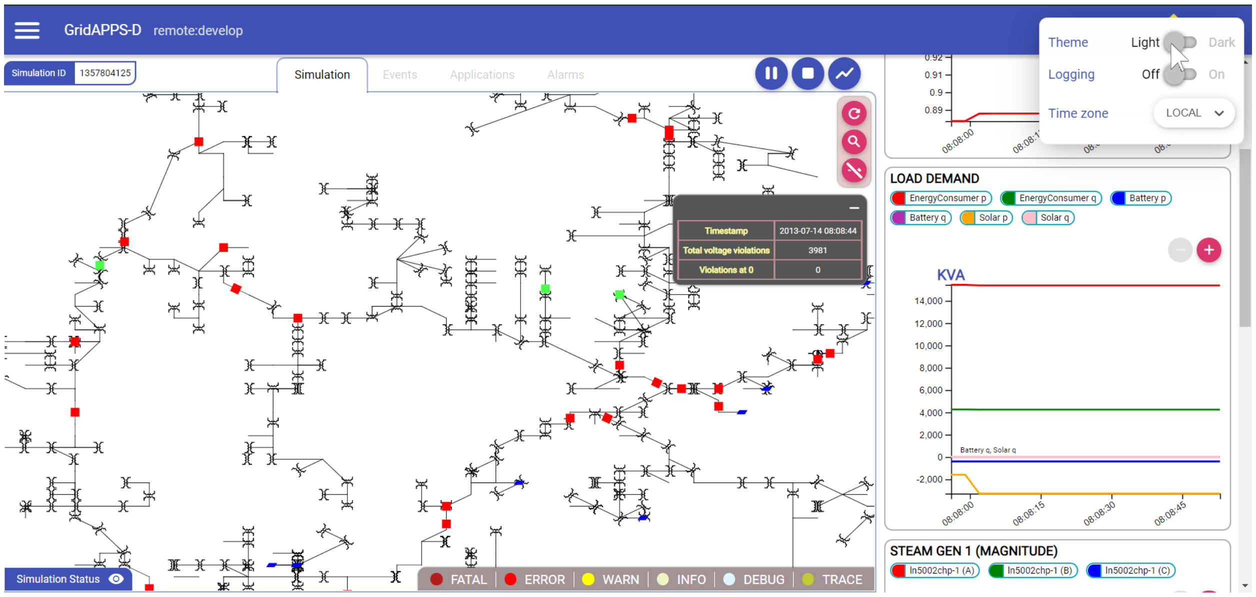


After clicking the `Done` button, the new custom plots will appear in the right-hand pane of the window.



Timestamp	2013-07-14 08:14:44
Total voltage violations	5830
Violations at 0	877

Alternate plot colors are available by switching from the theme from dark to light:



Timestamp	2013-07-14 08:08:44
Total voltage violations	3981
Violations at 0	0

Theme Light Dark

Logging Off On

Time zone LOCAL

CHAPTER

FIVE

DOCKER SHORTCUTS

CHAPTER

SIX

CLOUD SERVER CONFIGURATION

6.1 Configuring Remote GridAPPS-D VIZ

To access the GridAPPS-D VIZ from a local client, it is necessary to specify the IP address of the remote server in the `viz.config` file located in the `~/gridappsd-docker/conf/` directory.

The configuration file is formatted as a JSON string specifying the version of the VIZ used and the IP address and port to be used:

```
{
    "version": "remote:develop",
    "host": "my.server.ip.address:61614"
}
```

CHAPTER

SEVEN

GRIDAPPS-D PLATFORM RELEASE HISTORY

7.1 Version 2021.04.0

Install using `./run.sh -t releases_2021.04.0`

1. New Features
   - Platform added the capability for Applications to send Ochre commands to the simulations.
   - Powergrid-Models added IEEE 13 OCHRE model update.
   - Resolved issue with Proven write simultion input
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2021.04.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2021.04.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2021.04.0
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2021.04.0
   - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.8.1
   - 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/2021.04.0
   - fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
   - gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2021.04.0

7.2 Version 2021.03.0

Install using `./run.sh -t releases_2021.03.0`

1. New Features
   - Platform switched from using Cim2Glm to CimHub library for power grid APIs.
   - Powergrid-Models repository refactored to contianer only models.
   - CimHub repository refactored to contain CimHub library and related utility functions
   - Resolved issue regarding clean exit of HELICS on simulation stopped by user

- Resolved issue with Proven storing test manager input messages
- Viz: added timezome support
- Viz: Resolved issues with powerflow direction arrows
- Updated documentaiton on CimHub
- Added integration tests for configuration and alarm API

2. Source Code

- goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2021.03.0
- gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2021.03.0
- gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2021.03.0
- cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2021.03.0
- proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.7.4
- 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/2021.03.0
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2021.03.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2021.03.0

7.3 Version 2021.02.0

1. New Features

- Added HELICS as a simulator. This included adding HELICS in GridAPPS-D docker container and adding HELICS service configuration file
- Added HELICS-GOSS bridge to translate control and measurement messages between HELICS and platform.
- Generated HELICS configurations for GridLAB-D
- Added unit testing framework for the HELICS-GOSS bridge
- Added SoC support to the HELICS-GOSS bridge.
- Updated Test Manager to publish results as they are processed intead of at the end of the simulation.
- Updated services to use platform log level
- Added test users to test various roles
- Updated to use username/password from environment variables instead of hardcoded in soruce code
- Fixed APIs where response was not correct when selecting response format as XML
- Added OCHRE house simulator in the GridAPPSD docker container. Tested it with GridLAB-d and HELICS.
- Corrected naming of S2 69kV breaker names to match sub-transmission diagram

- Updated IEEE 13Assets model. Line length issue was fixed in opendsscmd 1.2.15
- Viz: Switched the arrow directions of selected lines and switches to display power flow direction.
- Viz: Plotting the Time Series Simulation Vs. Time Series Simulation results.
- Sample app updated to use token-based authentication.
- Updated gridappsd-python library for token-based authentication
- Added tests for configuration and power grid APIs

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/v1.3.7.3
   - 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

7.4 Version 2020.12.0

1. New Features
   - Increase AMQ topic permissions for all users until more specific permissions have been defined
   - Update configs to support the token based authentication
   - Updated to new version of cim2glm
   - Updated to support change in goss-core where it makes the decision to use a token in the gossclient a variable that must be set
   - Fixed sendError change that hadn't been updated in ProcessEvent
   - Updated log api to include process type
   - Viz: Updated to use token-based authentication
   - Viz: Added functionality to automatically reconnect to the platform when it is restarted
   - Viz: Fixed partial powerflow highlighting of lines
   - Viz: Corrected the values of capacitor for open and close
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.12.0

- gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.12.0
- gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.12.0
- cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.12.0
- 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/2020.12.0
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.12.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.12.0

7.5 Version 2020.11.0

1. New Features
   - Querying simulation file to use weather data for startime-1 minute
   - Moved state-estimator to gridappsd base container
   - Integration tests added for APIs
   - Viz: Changes made to notifications UI
   - Viz: Updated rendering positions for reverse arrows for transformers
   - Viz: Added buttons to zoom in and out on plots
   - gridappsd-python: Updates made for integration test runs
   - Cim2glm: Added repeatable randomization and reusable mRID for houses
   - Cim2glm: Saved JSON files with all node coordinates
   - Cim2glm: added missing s2 phase
   - Cim2glm: Made the SoC meaurement mRID persistent
   - Cim2glm: Fixes made for maven builds
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.11.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.11.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.11.0
   - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.11.0
   - 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/2020.11.0
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.11.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.11.0

7.6 Version 2020.09.0

1. New Features
    - Reduced log published based on log level
    - Changed default log level to INFO
    - Added additional code for SoC measurement translations
    - Publishing simulation started message as log level INFO
    - Fixed type for SoC measurement translation in fncs bridge.
    - Updated proven version for storing simulationid and current time
    - Added support for SoC measurement
    - Viz: Fixed code that detects whether the response body can be converted to CSV or not
    - Viz: Changed how simulation statuses STARTED and PAUSED are detected
    - Viz: Add a button to upload simulation configuration object
    - Viz: Attaching Magnitude or Angle to plot name if it doesn't have those suffixes already
    - Viz: Rendering min/average/max voltages and load demand plots
    - Viz: Rendering power flow direction indicators for edges/switches/capacitors/regulators during a simulation
    - Viz: Plotting percentages of nominal voltage by taking the average of Alo and Ahi then divide by sqrt(3)
    - Cim2glm: Support added for battery SoC measurement insertion and dictionary
    - Cim2glm: Added a query to list XY coordinates for buses
    - Cim2glm: Added support to insert synchronous machine
    - Cim2glm: Updated cim2glm version to 19.1.1
    - GridAPPS-D docker: Updated proven version to 1.3.7
2. Source Code
    - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.09.0
    - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.09.0
    - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.08.0
    - cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.09.0
    - proven-cluster - https://github.com/pnnl/proven-cluster/releases/tag/v1.3.7
    - proven-client - https://github.com/pnnl/proven-client/releases/tag/v1.3.7

- proven-message - https://github.com/pnnl/proven-message/releases/tag/v1.3.7
- proven-docker - https://github.com/GRIDAPPSD/proven-docker/tree/releases/2020.09.0
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridappsd-docker-build - https://github.com/GRIDAPPSD/gridappsd-docker-build/tree/releases/2020.08.0
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.09.0

7.7 Version 2020.08.0

1. New Features
   - Storing alarms data in timeseries data store InfluxDB.
   - Converted all simulation id to string
   - Updated version of Cim2glm library
   - Viz: Upgraded dependency to fix security alert reported by GitHub
   - Viz: Added an input box to change the response topic for stomp client UI
   - Viz: Implemented expected results view
   - Cim2glm: Wrote VLL for primary_voltge and secondary_voltage of 3-phase transformers
   - Cim2glm: added bus name and coordinates to the voltage limit dictionary
   - Cim2glm: Fixed a case sensitivity for Ubuntu
   - Cim2glm: Filled missing coordinates on transactive123. Optimized the XY coordinates in voltage limit dictionary
   - Cim2glm: Created script that inserts DER from a text file. Able to insert, drop and re-insert DER
   - Cim2glm: Fixed bug in adding a DER terminal with wrong mRID
   - Cim2glm: Added documentation to insert DER
   - Cim2glm: Fixed the conversion of open switches. Fixed the shorting of fuses
   - Cim2glm: Added temporary fix for two-phase transformers that are missing one phase'stransformer code
   - Cim2glm: Added method to support buildlimitmaps with just two parameters
   - Cim2glm: Added bus name and coordinates to the voltage limit dictionary
   - Cim2glm: Fixed capacitor naming - no impact on power flow - previously lines / switches numbered 1-3 but caps numbered 0-2.
   - Cim2glm: Renamed Loads.dss to BalancedLoads.dss
   - Sample app: Calling get_message function with simulation timesatamp instead of current time.
2. Source Code
   - goss-gridapps-d - https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/tree/releases/2020.08.0
   - gridappsd-viz - https://github.com/GRIDAPPSD/gridappsd-viz/tree/releases/2020.08.0
   - gridappsd-python - https://github.com/GRIDAPPSD/gridappsd-python/tree/releases/2020.08.0

- cim2glm - https://github.com/GRIDAPPSD/Powergrid-Models/tree/releases/2020.08.0
- 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
- fncs - https://github.com/GRIDAPPSD/fncs/tree/develop
- gridlab-d - https://github.com/GRIDAPPSD/gridlab-d/tree/develop
- sample-app - https://github.com/GRIDAPPSD/gridappsd-sample-app/tree/releases/2020.08.0

7.8 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

7.9 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

7.10 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

7.11 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

7.12 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

7.13 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

7.14 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

7.15 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 ieeezipload.player 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

7.16 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

7.17 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

7.18 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

7.19 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

7.20 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

7.21 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

7.22 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

7.23 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

7.24 Version: 2019.01.0

Release Date: January 2019

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:
    - 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

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

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:

- 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.

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

|GridAPPS-D_narrow.png|

CHAPTER

EIGHT

KNOWN VPN AND PROXY ISSUES

There are a few known issues around WSL2 and Virtualbox VM compatibility with corporate VPNs and proxies.

8.1 DNS Configuration

This issue affects the ability of the VM to reach GitHub, clone repositories, run `pip install` or open any websites from the browser.

If your machine was connected to a corporate VPN during setup, the Doman Name Server (DNS) lookup address is set to that of your corporate intranet. To reset it, open an ubuntu session and edit the `resolv.conf` file

- `sudo nano /etc/resolv.conf`
- comment out existing nameserver address
- add new line with `nameserver 8.8.8.8`
- save file and restart terminal

8.2 Proxy Server Configuration

This issue affects the ability of the server to run `curl` and other commands addressing various `localhost` ports

To fix this, add the NO_PROXY option by running in the Ubuntu command line

```
export NO_PROXY="localhost,127.0.0.1,local.home,*.yourcompany.org,*.
yourintranet.org"
```

Note: Be sure to change the proxy URLs to those of your organization.

CHAPTER

NINE

GRIDAPPS-D INTRODUCTION

9.1 What is GridAPPS-D?

GridAPPS-D™ is an open-source platform that accelerates development and deployment of portable applications for advanced distribution management and operations. It is built in a linux environment using Docker, which allows large software packages to be distributed as containers.

The GridAPPS-D™ project is sponsored by the U.S. DOE's Office of Electricity, Advanced Grid Research. Its purpose is to reduce the time and cost to integrate advanced functionality into distribution operations, to create a more reliable and resilient grid.

GridAPPS-D enables standardization of data models, programming interfaces, and the data exchange interfaces for:

- devices in the field
- distributed apps in the systems
- applications in the control room

The platform provides

- robust testing tools for applications
- distribution system simulation capabilities
- standardized research capability
- reference architecture for the industry
- application development kit

9.2 GridAPPS-D Platform Characteristics

9.2.1 Vendor / Vendor Platform Independent

The GridAPPS-D Platform and application development environment is independent of any specific vendor or vendor platform, in other words vendor neutral. The results of this effort are intended to be useful and available to any vendor or application developer who wishes to apply them or incorporate them into existing or future products.

9.2.2 Standards-based Architecture

GridAPPS-D is the first platform for energy and distribution management systems that is designed with standards for data integration, including data models, programming interfaces, and data exchange interfaces between grid devices in the field, distributed applications in utility systems, and applications in utility control rooms. This means that the applications developed using GridAPPS-D make them broadly applicable and interchangeable across utility systems, reducing the cost and time for utilities to integrate new functionality.

To the greatest extent possible, the GridAPPS-D Platform incorporates and supports industry standards, in particular interoperability standards, including the power system model representation using the Common Information Model (CIM) and communications with other platforms / physical equipment through DNP3, IEEE 2030.5, and the open field messaging bus (OpenFMB)

9.2.3 Replicable

As a reference implementation of a standards-based architecture, advanced applications and services developed with GridAPPS-D Platform should be replicable, with the ability to be deployed at multiple locations on different distribution feeders with almost no code customization.

9.2.4 Flexible Distribution Simulation

The GridAPPS-D Platform enables users to run real-time quasi-static simulations of large distribution network models with real-time load data, thermal co-simulation of houses, real-time weather data, and real-time operation of switches, DERs, and volt-var control equipment. The platform supports multiple distribution simulators through a co-simulation bridge that abstracts the simulation configuration details to a simple API.

9.3 Data Representation & Management

A key to GridAPPS-D is providing the distribution system application developer with a standardized approach to data. The intent is to allow the developer to make logical references to data referencing standard data models and interfaces without concern for how the data is physically made available. This standardized, logical data interface is based on existing standards to the greatest extent possible.

9.3.1 Standards-based Data Representation

The Common Information Model (CIM) is used for all power system models, which enables rapid exchange of power system models across compliant applications and services. Using the set of standardized model queries provided by the PowerGrid Models API, a GridAPPS-D application is able to scale seamlessly across different network models with no modifications to the application code.

9.3.2 Standards-based Data Interfaces

The GridAPPS-D Platform and GridAPPS-D APIs provide a standardized method for interfacing with power system model data, real-time simulation data, historical data, and log data. Each of these APIs abstract the database specifics, and enable simple queries through a set of standardized messages formatted as JSON strings.

9.3.3 Data Translation to Non-standardized Elements

CIM Hub and the Configuration File API allow conversion of the power system model data from the standards-based CIM XML format used by the GridAPPS-D Platform to model formats used by other software packages, such as GridLAB-D and OpenDSS. This model conversion process can be performed with a simple set of standardized API calls.

9.3.4 Available Distribution Feeders

The GridAPPS-D platform comes pre-configured with a combination of IEEE Test Feeders, PNNL Taxanomoy feeders, and other realistic synthetic models. Additional models and actual utility feeder data can be uploaded easily as CIM XML files into the GridAPPS-D Platform, which can then be used for application testing and real-time simulation.

9.4 Real-Time Distribution Simulation

The GridAPPS-D Platform inlcudes a robust real-time distribution simulator with comparable capabilities to a Dispatcher Training Simulator. This environment enables application developers to test algorithms and application code on both the standard realistic sythetic feeders pre-configured in the GridAPPS-D Platform download and any other power system models that the user can upload through the CIM Hub package.

The distribution simulator is the source of data to the distribution system application developer enabling them to evaluate the performance of their application with ideal or realistic noisy data under different operating and performance conditions.

The GridAPPS-D platform currently supports only quasi-static simulation (i.e. simulation of electromechanical / electromagnetic transients, variable microgrid island frequency, synchro-check relays, etc. are not supported currently). These types of simulations can be performed with GridLAB-D outside of the the GridAPPS-D Platform and application development environment.

9.4.1 Real-Time & Faster-than-Real-Time Simulation

Simulations can be run in two modes:

1) Real-time mode: one second of computer clock time corresponds to one second of simulation time. The GridAPPS-D Platform runs the simulation in each time and publishes simulation data and sensor measurements every three seconds.
2) Faster-than-real-time mode: The GridAPPS-D runs the simulation as fast as possible and does not wait for three seconds of computer clock time to pass before it publishes the simulation data from the current time step. This mode is very useful for creating historical training data sets for AI/ML applications.

9.4.2 Controllable Power System Equipment

All of the power system equipment can be controlled in real-time through the Simulation API, allowing applications to open/close switches, dispatch DGs / DERs, adjust setpoints of rooftop PV, adjust regulator taps, and turn capacitor banks on or off.

9.4.3 Noisy / Bad Data Injection & Communication Failures

The GridAPPS-D Platform supports the Sensor Simulator Service, which is able to inject noise, bad measurements, and data packet losses into the simulation output. The frequency at which sensors publish can also be adjusted and aggregated, allowing realistic representation of real sensors, such as AMI meters that publish data every 15 minutes, rather than at each simulation time step. This allows the user to train and evaluate applications with realistic measurement for meters and sensors, rather than "pure" data created by the power flow solver.

The GridAPPS-D Platform also supports simulation of communication failures through the Test Manager during which data is not received from sensors, control commands are delivered to selected equipment, or both. This enables application developers to test algorithm performance under realistic conditions, during which physical equipment might not respond to control commands.

9.4.4 Reconfigurable Power System Topologies

The GridAPPS-D Platform supports simulation of both meshed and radial power system topologies, as well as reconfiguration of the power system network in real-time by opening / closing / tripping of various switching devices, such as breakers, reclosers, sectionalizers, and fuses. These switches can be controlled by an application through the Simulation API or through the GridAPPS-D Viz GUI

9.4.5 Real-Time Simulation Visualization

The GridAPPS-D Platform includes the Viz GUI application, which presents a simple graphic user interfaces with some of the basic functionalities found in an Dispatcher Training Simulator, inlcuding a one-line diagram of the feeder, colorized switch positions, outage locations, alarm messages, and customizable stripcharts of power flow, node voltage, and tap position.

9.5 Using the GridAPPS-D Platform

GridAPPS-D currently runs in a Linux virtual machine (VM). Although it can be built from sources, the primary form of distribution is as a set of Docker containers. Users can install the Docker infrastructure on their computer and then download the Docker containers. Several platform usage scenarios are then feasible:

1. Start and run the application through its browser interface. Utilities could use the platform this way to evaluate new applications, or to evaluate applications on their own circuits. The App Hosting Manager allows a user to install and configure new applications to run in the platform, by modifying configuration files but without having to write new code. GridAPPS-D will also be able to ingest any distribution circuit provided in CIM format.
2. Write scripted scenarios and responses using the Test Manager, and run those through GridAPPS-D. This mode can be used for a more rigorous evaluation, and also for operator training.
3. Write a new application, using one of the open-source examples as a template. This mode should provide a faster on-ramp for application developers to develop a standards-compliant product.

4. DMS vendors can use the platform to develop and test their own standards-compliant interfaces. Any GridAPPS-D code may be incorporated into a commercial product, pursuant to its BSD license terms. The goal is for an application to be deployable from one platform to another, simply by moving the program file(s) and updating local configuration files.

CHAPTER

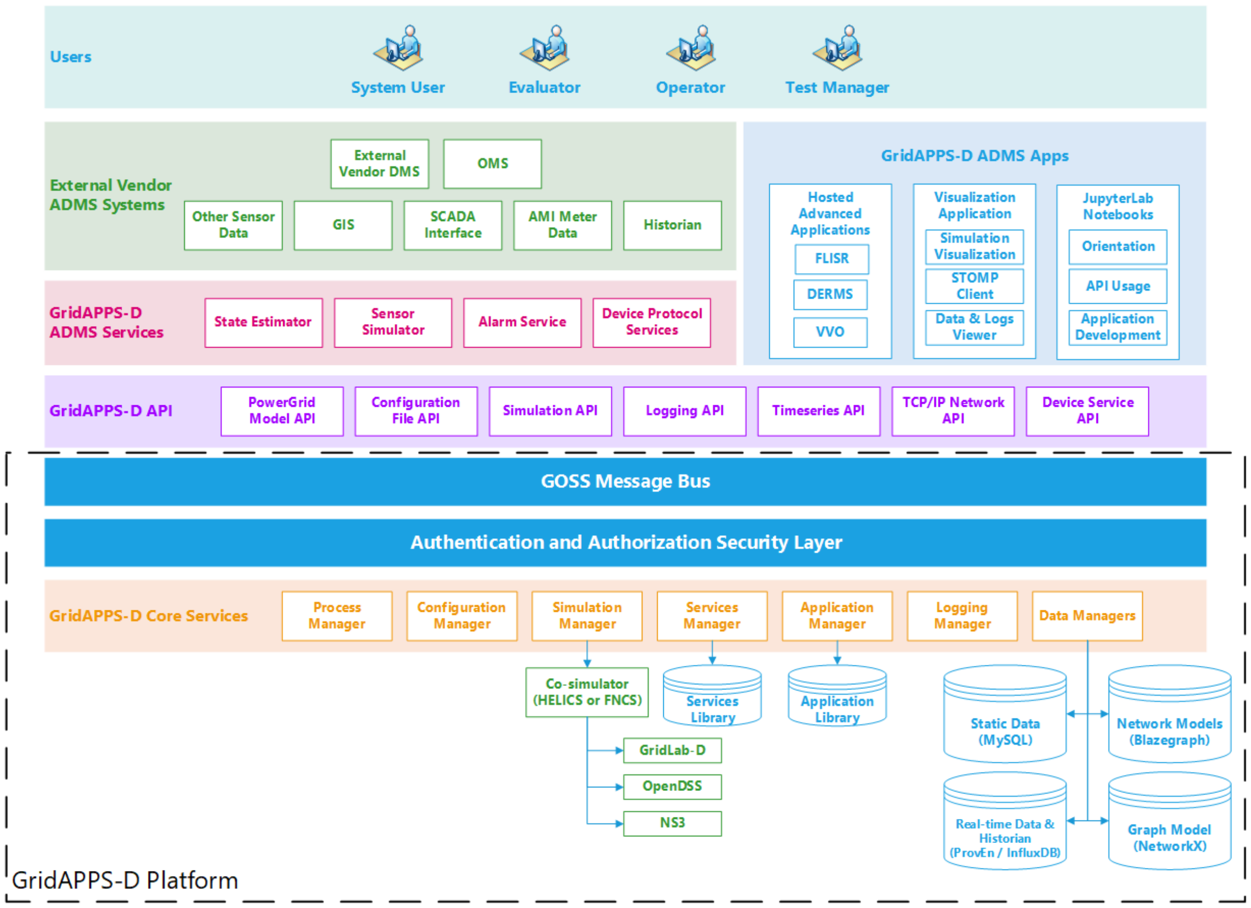
TEN

GRIDAPPS-D ARCHITECTURE

10.1 GridAPPS-D Architecture

GridAPPS-D offers a standards-based, open-source platform that enables rapid integration of advanced applications and services through a robust application programming interface (API).

The architecture of the development ecosystem is illustrated below.



10.2 GridAPPS-D User Roles

The GridAPPS-D platform contains several user roles with different permissions.

Currently, the permission are only implemented in the GridAPPS-D Viz. The permissions will be extended to platform authentication and API call execution in a future release.

The user roles can be associated with individual username/password logins. These are located in the GOSS core security configuration file

- `Admin`
  - This role is used by internal GridAPPS-D Platform functionalities.
  - All permissions are granted, inlcuding
    - Starting, pausing, stopping, and joining simulations
    - Registering / deleting applications and services
    - Publishing and subscribing to all API communication channels
    - All Viz tools (Simulation Control, STOMP Client, Data Comparison, Data Browser, Application Viewer)
    - Managing users (future release)
    - Managing permissions of applications and services (future release)
- `Evaluator`
  - This role is used by observers of application evaluation activities
  - Permissions inlcude
    - Joining existing simulations
    - STOMP Client, Data Comparison, Data Browser
- `Operator`
  - This role is used by operators for application evaluation activities
  - Permissions inlcude
    - Joining existing simulations
    - STOMP Client, Data Comparison, Data Browser
- `Test Manager`
  - This role is used to create simulations with TestManager events
  - Permissions inlcude
    - Starting, pausing, stopping, and joining simulations
    - All Viz tools (Simulation Control, STOMP Client, Data Comparison, Data Browser, Application Viewer)
- `Application`
  - This role is used by applications to authenticate with the GridAPPS-D Platform
  - Permissions inlcude
    - Controlling parallel simulations through Simulation API
    - Publishing and subscribing to all API communication channels
- `Service`

  - This role is used by services to authenticate with the GridAPPS-D Platform
  - Permissions inlcude
    * Controlling parallel simulations through Simulation API
    * Publishing and subscribing to all API communication channels

---

10.3 Integration with External Vendor Systems

External vendor systems are able to interface with GridAPPS-D compliant applications and services through two means.

The first is direct integration through the standards-based API and message bus. This enables products that comply with the GridAPPS-D™ platform to * reduce utility time and cost to integrate new functionality * give utilities more choice in technology providers * scale up or down for any size utility * expand market opportunities for developers and vendors

The second method is through the standards-based services, such as the DNP3 service, IEEE 2030.5 service, etc. that enable communication between GridAPPS-D compliant applications and external vendor systems through SCADA and other control center protocols.

10.4 GridAPPS-D Applications

The GridAPPS-D platform and API enable rapid development of advanced power applications that are able to operate in a real-time environment and interface with external software and systems. Multiple power applications have already been developed on the platform, including

- Volt-Var Optimization (VVO)
- Fault Location Isolation and Service Restoration (FLISR)
- Distributed Energy Resource Dispatch and Management (DERMS)
- Solar Forecasting, Load Forecasting, etc.
- and more

Applications can be containerized in Docker for direct integration into the platform or interface through the API. Applications can be written in any programming language, but API libraries are currently available in only Python and Java.

10.5 GridAPPS-D Services

The GridAPPS-D platform can host a multitude of services for processing both real-time simulation and control center data. These services can be called by any application through the GridAPPS-D API.

Some of the available services include

- **State Estimator**
- **Sensor Simulator**
- **Alarm Service**

- **DNP3 Protocol Service**
- **IEEE 2030.5 Protocol Service**

10.6 GridAPPS-D Application Programming Interface

GridAPPS-D offers a unique standards-based application programming interface (API) that will be the focus of the lessons in this set of tutorials. The API enables any application, service, or external vendor product to interface with each other, access control center data, run a real-time simulation, and issue equipment control commands.

GridAPPS-D has several APIs to serve different needs and objectives, inlcuding * **Powergrid Models API –** Allows apps and services to access the power system model data * **Configuration File API –** Allows apps to set equipment statuses and system conditions * **Simulation API –** Allows apps to start a real-time simulation and issue equipment commands * **Timeseries API –** Allows apps to pull real-time and historical data * **Logging API –** Allows apps to access logs and publish log messages

Additional APIs are currently under development to enable communication and control of field devices, as well as cyber-physical network co-simulation.

10.7 GOSS Message Bus

One of the unique features of GridAPPS-D is the GOSS Message Bus, which enables integration and communication between applications, services, and external software on a publish-subscribe basis.

The GridAPPS-D platform publishes SCADA and simulation data, alarms, and other real-time data. Applications subscribe to the types of messages relevant to their objectives and publish equipment commands and control settings.

10.8 GridAPPS-D Core Services

"Under the hood" of the GridAPPS-D platform are the core services and managers.

An application developer should not need a detailed understanding of the core services, as all interaction is performed through the various APIs, which will be dicussed in detail in the upcoming tutorial lessons.

The core services provide the key functionality offered by the GridAPPS-D platform, inlcuding database access, processing API calls, handling equipment commands, and running simulations.

Some of the core services included in the GridAPPS-D platform are * **Platform Manager –** Coordinates all of the other managers * **Process Manager –** Coordinates platform component interactions * **Application Manager –** Manages application registration, execution, and status reporting * **Configuration Manager –** Manages the setup and configuration of real-time simulations * **Simulation Manager –** Allows users and apps to create, start, stop, and pause co-simulations * **Data Manager –** Coordinates the integrated repository of model, timeseries data, and metadata * **Model Manager –** Loads and checks CIM-based power system models * **Logging Manager –** Supports logging for application development and execution * **Services Manager –** Coordinates all services available for users and apps * **Test Manager –** Enables creation of simulation events, faults, and network outages

---

10.9 Co-Simulation Framework

The co-simulation framework serves as the simulation context for the rest of GridAPPS-D. When a simulation is requested through the GridAPPS-D plaform the simulation manager instantiates a FNCS or HELICS co-simulation federation consisting of two applications. The first application is a powerflow simulator which can be either GridLAB-D or OpenDSS that simulates real world distribution feeder or feeders. The second is a custom application that serves as bridge between the FNCS/HELICS message bus and the GOSS message bus. The data that travels between the co-simulation federation and the rest of the platform are SCADA measurement, SCADA control, and simulation status and control messages.The bridge application subscribes to the simulation input topic to recieve any SCADA control, simulation control, and simulation event messages. The bridge forwards SCADA control commands and simulation events like faults and outages to the powerflow simulator. The bridge publishes SCADA measurements from the powerflow simulator on a simulation output topic that GridAPPS-D applications and other parts of the GridAPPS-D platform subscribe to.

10.10 Database Structures

Default installation of GridAPPS-D comes with following data stores:

- **MySQL:** It is used to store log data from platform, applications and services. For details, please see Logging API, which is covered in detail in Lesson 2.7.
- **Blazegraph:** It is used to store power grid model data. The data contains equipments, properties and their initial measurement values. It is a triplestore that supports complex graph representation and class structure for CIM standard data model.
- **InfluxDB:** InfluxDB is a time series data store and is used to store simulation output, simulation input, weather and load data. It also store output from services line sensor service and alarms service. For the purposes of the GridAPPS-D project, InfluxDB is managed by Proven. Proven is a database software suite supporting disclosure, collection, and management of modeling and simulation data.

For the purpose of developing applications, the data stores used should be transparent to the application as long the data model and standardized API is used.

CHAPTER

ELEVEN

GRIDAPPS-D PYTHON LIBRARY

11.1 Intro to GridAPPSD-Python

GridAPPSD-Python is a Python library that wrapa API calls and passes them to the various GridAPPS-D APIs through the GOSS Message Bus.

The library has numerous shortcuts to help you develop applications faster and interface them with other applications, services, and GridAPPS-D compatible software packages.

The GridAPPSD-Python library requires a python version >= 3.6 and < 4 in order to work properly. (Note: no testing has been done with python 4 to date).

The GridAPPSD-Python library can be installed using `pip install gridappsd-python`.

For more information, see the GridAPPSD-Python GitHub Repo and PyPi site.

11.2 Connecting to GridAPPS-D Platform

Before starting any development in the GridAPPS-D environment, it is necessary to establish a connection to the GridAPPS-D Platform.

11.2.1 Specifying Environment Variables (Preferred)

The preferred method for establishing a connection with the GridAPPS-D Platform is to define a set of environment variables that specify the connection address, port, username, and password.

Specifying the Environment Variables in Python Script

This method is recommended for initial application development when running in a development environment, such as PyCharm or the Jupyter Notebook tutorials.

[ ]:
```python
# Establish connection to GridAPPS-D Platform:
from gridappsd import GridAPPSD

import os # Set username and password
os.environ['GRIDAPPSD_USER'] = 'tutorial_user'
os.environ['GRIDAPPSD_PASSWORD'] = '12345!'
os.environ['GRIDAPPSD_ADDRESS'] = 'localhost'
```

(continues on next page)

(continued from previous page)

```
os.environ['GRIDAPPSD_PORT'] = '61613'

# Connect to GridAPPS-D Platform
gapps = GridAPPSD()
assert gapps.connected
```

Specifying the Environment Variable in ~/.bashrc Script

This method is recommended for more complete applications scripts where all the application scripts are called from a single ~/.bashrc script. In that script, the environment variables can be defined and then will be available to all scripts that need to connect the GridAPPS-D Platform.

```
# export allows all processes started by this shell to have access to the global␣
↪variable

# address where the gridappsd server is running - default localhost
export GRIDAPPSD_ADDRESS=localhost

# port to connect to on the gridappsd server (the stomp client port)
export GRIDAPPSD_PORT=61613

# username to connect to the gridappsd server
export GRIDAPPSD_USER=app_user

# password to connect to the gridappsd server
export GRIDAPPSD_PASSWORD=1234App

# Note these should be changed on the server in a cyber secure environment!
```

11.2.2 Specifying Connection Parameters Manually

An older method of connecting to the GridAPPS-D Platform is manually specifying the connection parameters. This method is still supported, but may be deprecated in future releases.

This method is less flexible and has an in-built portability issues associated with hard-coded platform passwords.

```
[ ]: gapps = GridAPPSD("('localhost', 61613)", username='system', password='manager')
```

11.2.3 GridAPPSD-utils Deprecated

GridAPPS-D Platform releases prior to 2021 used a library called `utils` to establish a connection with the platform. This library has been deprecated and replaced with Java Token Authentication using the environment variable method shown above.

The authentication method below will work with 2019-2020 versions of the GridAPPS-D Platform and GridAPPSD-Python, but not with any newer releases.

```
# DEPRECATED authentication method
from gridappsd import GridAPPSD, utils
gapps = GridAPPSD(address=utils.get_gridappsd_address(),
          username=utils.get_gridappsd_user(), password=utils.get_gridappsd_pass())
```

`utils` – **DEPRECATED** A set of utilities to assist with common commands, inlcuding

- `utils.validate_gridappsd_uri()` – Checks if GridAPPS-D is hosted on the correct port
- `utils.get_gridappsd_address()` – Returns the platform address such that response can be passed directly to a socket or the STOMP library
- `utils.get_gridappsd_user()` – Returns the login username
- `utils.get_gridappsd_pass()` – Returns the login password
- `utils.get_gridappsd_application_id()` – Only applicable if the environment variable 'GRIDAPPSD_APPLICATION_ID' has been set
- `utils.get_gridappsd_simulation_id()` – Retrieves the simulation id from the environment.

It is strongly recommended that applications that previously used this method replace any connection objects with environment variables to ensure compatibility with subsequent releases of the GRIDAPPS-D platform

11.3 Passing API calls with GridAPPSD-Python

There are three methods used in GridAPPSD-Python Library to pass API calls to the GridAPPS-D platform:

- `.get_response(self, topic, message, timeout)` – Pass a database query, response expected before timeout
- `.subscribe(self, topic, callback)` – Subscribe to a data stream
- `.send(self, topic, message)` – Send a command to a simulation, no response expected

Each are explained in more detail below

11.3.1 .get_response(topic, message)

This is the most commonly used method for passing API calls to the GridAPPS-D Platform. This method is used when a response is expected back from the GridAPPS-D platform within a particular timeout period. It is used for all database queries using

- *PowerGrid Models API* – queries for model info, object mRIDs, measurement mRIDs
- *Configuration File API* – queries to convert the model into other file format versions
- *Timeseries API* – queries for weather data and historical data from prior simulations

The syntax used when calling this method is `gapps.get_response(topic, message)` or alternatively, `gapps.get_response(topic, message, timeout = 30)`, where

- `topic` is the GridAPPS-D topic for the particular API (as described in *API Communication Channels*.
- `message` is the query message specifying what information the API should return
- `timeout =` is optional and gives the number of seconds given for the API to respond. Model conversion queries using the Configuration File API may take 30 - 60 seconds for very large models. Most other queries do not need a timeout specification.

11.3.2 .subscribe(topic, message)

This method is used for subscribing to the real-time data stream generated by the GridAPPS-D platform while running a simulation. It is used to subscribe to information published at each time step by the

- *Simulation API* – simulated SCADA data and measurements created by the simulation
- *Logging API* – log messages published by the Platform, applications, and simulation

The `.subscribe()` method is also used to subscribe to streaming data generated by some of the GridAPPS-D services.

The syntax used when calling this method is `gapps.subscribe(topic, message)`, where

- `topic` is the GridAPPS-D simulation output topic, log output topic, or service output topic for the particular real-time data stream the application needs to subscribe to, (as described in *API Communication Channels*.
- `message` is the subscription message. For simulation and log outputs, it is a method or class definition, as described in Comparison of Subscription Approaches.

11.3.3 .send(topic, message)

This method is used for sending equipment command and simulation input messages to the GridAPPS-D platform while running a simulation. It is used to send difference messages to the *Simulation API* and for other generic publishing needs, such as sending a command input to a GridAPPS-D Service.

The syntax used when calling this method is `gapps.send(topic, message)`, where

- `topic` is the simulation or service input topic(as described in *API Communication Channels*.
- `message` is the API call message to be published. The most commonly used simulation input message is a *Difference Message* used to control equipment settings.

11.3.4 .unsubscribe(conn_id)

This method is used to unsubscribe from a simulation or service that was previously subscribed to using the `.subscribe` method.

The syntax of this method is `gapps.unsubscribe(conn_id)`, where `conn_id` is the connection id obtained when previously subscribing using the `conn_id = gapps.subscribe(topic, message)`.

11.4 Importing Required Python Libraries

A typical GridAPPS-D application will require several libraries to be imported from GridAPPSD-Python as well as from other Python libraries.

11.4.1 Required GridAPPS-D Libraries

The GridAPPS-Python API contains several libraries, which are used to query for information, subscribe to measurements, and publish commands to the GOSS message bus. These inlcude

- `GridAPPSD` – This is the primary library that contains numerous methods for passing API calls, connecting to the GridAPPS-D platform, and other common tasks
- `topics` – This library contains methods for constructing the correct API channel strings
- `Simulation` – This library contains shortcut methods for subscribing and controlling simulations
- `Logger` – This library contains logging methods. It is recommended to invoke those methods using the `gapps.get_logger` method rather than importing the library
- `GOSS` – This library contains methods for passing API calls to the GOSS Message Bus. It is imported automatically when importing the `GridAPPSD` library
- `Houses` – This library populates a feeder with thermal house model loads. It is imported automatically when importing the `GridAPPS` library
- `utils` – Deprecated

Each of the libraries can be imported using `from gridappsd import library_name`. For example,

```
[ ]: from gridappsd import GridAPPSD
```

```
[ ]: from gridappsd import topics as t
```

Each of the libraries are discussed in detail in the next section.

11.4.2 Other Required Python Libraries

Below is a list of some of the additional libraries that you may need to import.

You may not need all of these additional libraries, depending on the needs of your application

- `argparse` – This is the recommended command-line parsing module in Python.(Online Documentation)
- `json` – Encoder and decoder for JavaScript Object Notation (JSON). (Online Documentation)
- `logging` – This module defines classes and functions for event logging. (Online Documentation
- `sys` – Python module for system specific parameters. (Online Documentation)
- `time` – Time access and conversions. (Online Documentation)
- `pytz` – Library to enable resolution of cross-platform time zones and ambiguous times. (Online Documentation
- `stomp` – Python client for accessing messaging servers using the Simple Text Oriented Messaging Protocol (STOMP). (Online Documentation)
- `os` – Miscellaneous operating system interface. Needed to set environment variables for the GridAPPS-D connection if working from a single Python script or notebook. (Online Documentation)

```
[ ]: import argparse
     import json
     import logging
     import sys
     import time
```

(continues on next page)

(continued from previous page)

```
import pytz
import stomp
import os
```

11.5 GridAPPSD-Python GridAPPSD Library

This library contains the most commonly used methods needed for building GridAPPS-D applications and services.

All of these methods are for the GridAPPS-D connection object defined using `gapps = GridAPPSD()`

11.5.1 Get Methods

This group of methods are used to get information and statuses about the GridAPPS-D platform and simulations:

- `.get_application_status()` – Returns the current status of an application
- `.get_application_id()` – Returns the unique ID of an application registered with the Platform
- `.get_houses()` – Returns houses populated in the feeder
- `.get_logger()` – Returns a log instance for interacting with logs within the Platform
- `.get_platform_status()` – Returns the current status of the Platform
- `.get_service_status()` – Returns the current status of a service
- `.get_simulation_id()` – Returns the simulation ID for the current GridAPPSD connection

11.5.2 Set / Send Methods

This group of methods are used to set the status of applications and services:

- `.set_application_status()` – Set the status of an application
- `.set_service_status()` – Set the status of a service
- `.set_simulation_id(simulation_id)` – Set the simulation ID if none is defined
- `.send_simulation_status(status, message, log_level)` – Sets simulation + service status and writes to GridAPPS-D logs
- `.send_status(status, message, log_level)` – Sets application status and writes to GridAPPS-D logs

11.5.3 PowerGrid Models API Methods

This group of methods run pre-built PowerGrid Models API queries for simpler query types:

- `query_data(query, timeout)` – *Run a generic SPARQL Query*
- `query_model(model_id, object_type, object_id)` – *Query using full CIM100 prefix*
- `query_model_info()` – *Query for dictionary of all models*
- `query_model_names(model_id)` – *Query for mRIDs of all models*
- `query_object(object_id, model_id)` – *Query for CIM attributes of an object*
- `query_object_dictionary(model_id, object_type, object_id)` – *Query for object dictionary*
- `query_object_types(model_id)` – *Query for CIM classes in a model*

11.6 GridAPPSD-Python Topics Library

The GridAPPSD-Python topics library is used to obtain the correct *API Communication Channel*, which tells the GridAPPS-D platform to which database, application, or simulation a particular API call should be delivered.

Static GridAPPS-D topics (such as those for the *PowerGrid Models API*, *Configuration File API*, and *Timeseries API*) can be imported using

```
[ ]: from gridappsd import topics as t
```

Dynamic GridAPPS-D topics (such as those for the *Simulation API* and various GridAPPS-D services) can be imported using

```
[ ]: from gridappsd.topics import simulation_output_topic
```

```
[ ]: from gridappsd.topics import simulation_input_topic
```

```
[ ]: from gridappsd.topics import simulation_log_topic
```

Each of the specific methods available in the `topics` library are discussed in detail in *API Communication Channels*.

11.7 GridAPPSD-Python Simulation Library

The GridAPPSD-Python simulation library is used for starting, running, and controlling parallel digital twin simulations. For more details on specific usage, see

- *Starting a Parallel Simulation*
- *Pausing, Resuming, and Stopping Parallel Simulations*
- *Subscribing to Parallel Simulations*

The Simulation library can be imported using

```
[ ]: from gridappsd.simulation import Simulation
```

Available methods in the `Simulation` library are

- `.start_simulation()` – Start the simulation
- `.pause()` – Pause the simulation
- `.resume()` – Resume the simulation
- `.resume_pause_at(pause_time)` – Resume the simulation, and then pause it in so many seconds
- `.stop()` – Stop the simulation
- `.run_loop()` – Loop the entire simulation until interrupted
- `.simulation_id` – Returns the Simulation ID of the simulation
- `.add_ontimestep_callback(myfunction1)` – Run the desired function on each timestep
- `.add_onmesurement_callback(myfunction2)` – Run the desired function when a measurement is received.
- `.add_oncomplete_callback(myfunction3)` – Run the desired function when simulation is finished
- `.add_onstart_callback(myfunction4)` – Run desired function when simulation is started

Note: method name ``.add_onmesurement`` is misspelled in the library definition!!

11.8 GridAPPSD-Python DifferenceBuilder

`DifferenceBuilder` is a GridAPPSD-Python library that is used to create and correctly format difference messages that used to create equipment control commands. The usage of difference builder is given in *Using DifferenceBuilder*.

The `DifferenceBuilder` library can be imported using

```
[ ]: from gridappsd import DifferenceBuilder

     my_diff_build = DifferenceBuilder(simulation_id)
```

CHAPTER

TWELVE

GRIDAPPS-D APPLICATION STRUCTURE

12.1 Application Structure

The structure of a GridAPPS-D application can be broken into nine sections:

- Querying for the power system model
- Querying for measurement MRIDs
- Querying for weather data (if needed)
- Configuring parallel simulations (if needed)
- App core algorithm & measurement processing
- Subscribing to simulation output
- Publishing equipment commands
- Querying historical & timeseries data
- Subscribing and publishing to logs

Each of these task sections within an application are explained below with sample code that can interact with a simulation running on the GridAPPS-D platform.

12.2 Connecting to GridAPPS-D Platform

Prior to running any of the API calls or other core application code, the application needs to establish a secure connection with the GridAPPS-D platform.

[ ]:
```
# Import GridAPPSD-Python Library:
from gridappsd import GridAPPSD

# When developing locally, paste Simulation ID into this variable
# When running inside docker, this is passed automatically by platform
viz_simulation_id = "242488458"

# Simulation running on IEEE 123 node model:
model_mrid = "_C1C3E687-6FFD-C753-582B-632A27E28507"
```

```
[ ]: # Set environment variables - when developing, put environment variable in ~/.bashrc␣
     ↪file or export in command line
     # export GRIDAPPSD_USER=system
     # export GRIDAPPSD_PASSWORD=manager

     import os # Set username a
     os.environ['GRIDAPPSD_USER'] = 'tutorial_user'
     os.environ['GRIDAPPSD_PASSWORD'] = '12345!'

     # Connect to GridAPPS-D Platform
     gapps = GridAPPSD(viz_simulation_id)
     assert gapps.connected
```

12.3 Querying for the Power System Model

The first portion of a GridAPPS-D application is series of queries to the PowerGrid Models API to obtain information about the power system model.

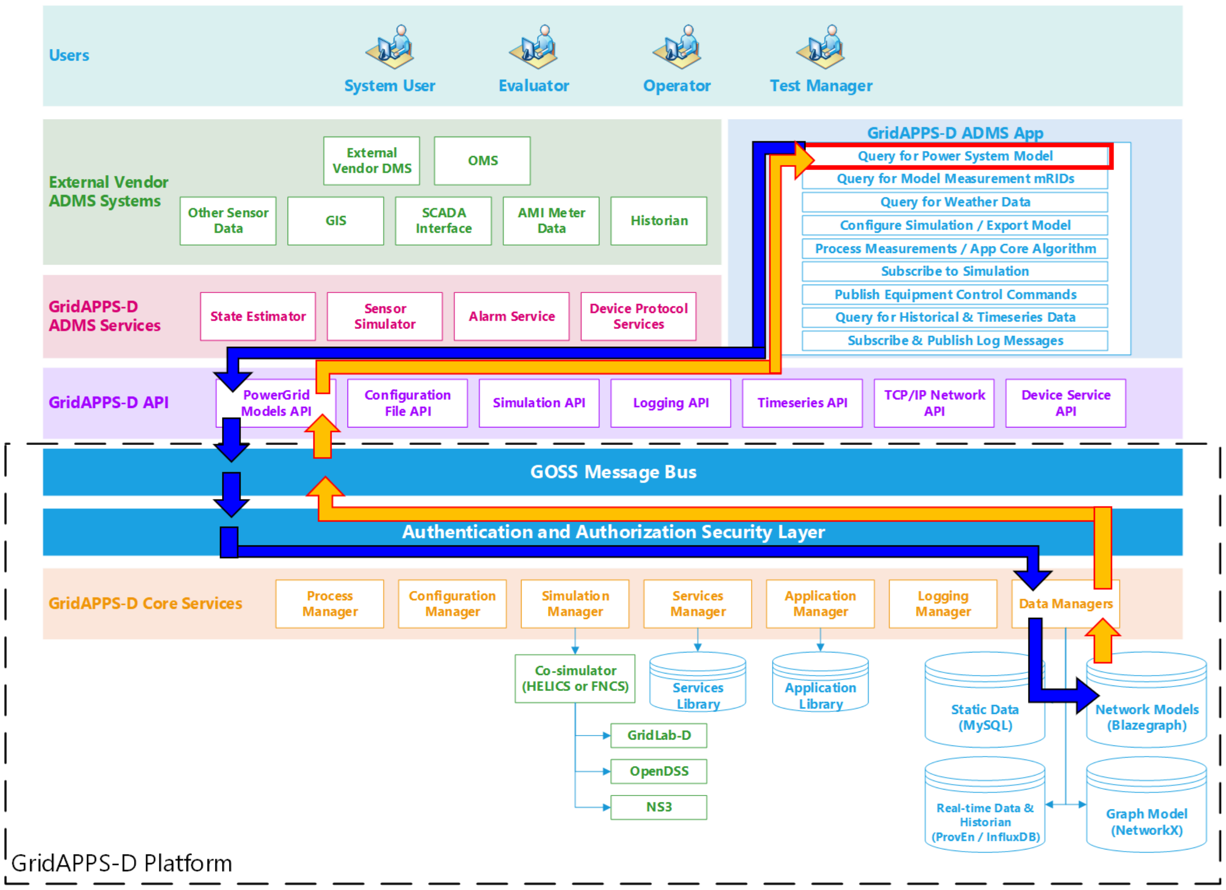
Because GridAPPS-D applications are designed to be portable across numerous power system models without any code modification, the application must query the Blazegraph database and create a set of local variables that contain the information needed by the app to run its internal code.

An application will query for the various pieces of power system equipment relevant to its objective (e.g. a VVO app will be interested in regulators and capacitors, while a FLISR app will be interested in switches and reclosers present in the model). The query will typically include requests for information about the names, location, mRIDS, and electrical parameters for the various pieces of equipment needed by the application..

12.3.1 Model Query Information flow

The figure below shows the information flow involved in making a query for the power system model.

The query is sent using `gapps.get_response(topic, message)` on a queue channel (explained in *API Communication Channels*) with a response expected back from the platform within the specified timeout period.



Application passes query to GridAPPS-D Platform

First, the application creates a query message for requesting information about the desired power system components in the format of a JSON string or equivalant Python dictionary object. The syntax of this message is explained in detail in *Using the PowerGrid Models API*.

The application then passes the query through the PowerGrid Models API to the GridAPPS-D Platform, which publishes it to a queue channel on the GOSS Message Bus. If the app is authenticated and authorized to pass queries, the query message is delivered to the data managers, which obtain the desired information from the Blazegraph Database.

GridAPPS-D Platform responds to Application query

The data managers then publish the response from the Blazegraph Database to the appropriate queue channel. The PowerGrid Models API then returns the desired information back to the application as a JSON message or equivalant Python dictionary object.

12.3.2 Model Query Sample App Code

Below is a sample query of how the application will use the PowerGrid Models API to query for the details associated for all the switches in the feeder.

```
[ ]: from gridappsd import topics as t

message = {
    "modelId": model_mrid,
    "requestType": "QUERY_OBJECT_DICT",
    "resultFormat": "JSON",
    "objectType": "LoadBreakSwitch"
}

response_obj = gapps.get_response(t.REQUEST_POWERGRID_DATA, message)
switch_dict = response_obj["data"]

# Filter to get mRID for switch SW2:
for index in switch_dict:
    if index["IdentifiedObject.name"] == 'sw2':
        sw_mrid = index["IdentifiedObject.mRID"]

print(switch_dict[0]) # Print dictionary for first switch

print('mRID of sw2 is ',sw_mrid)
```

12.4 Querying for Measurement mRIDs

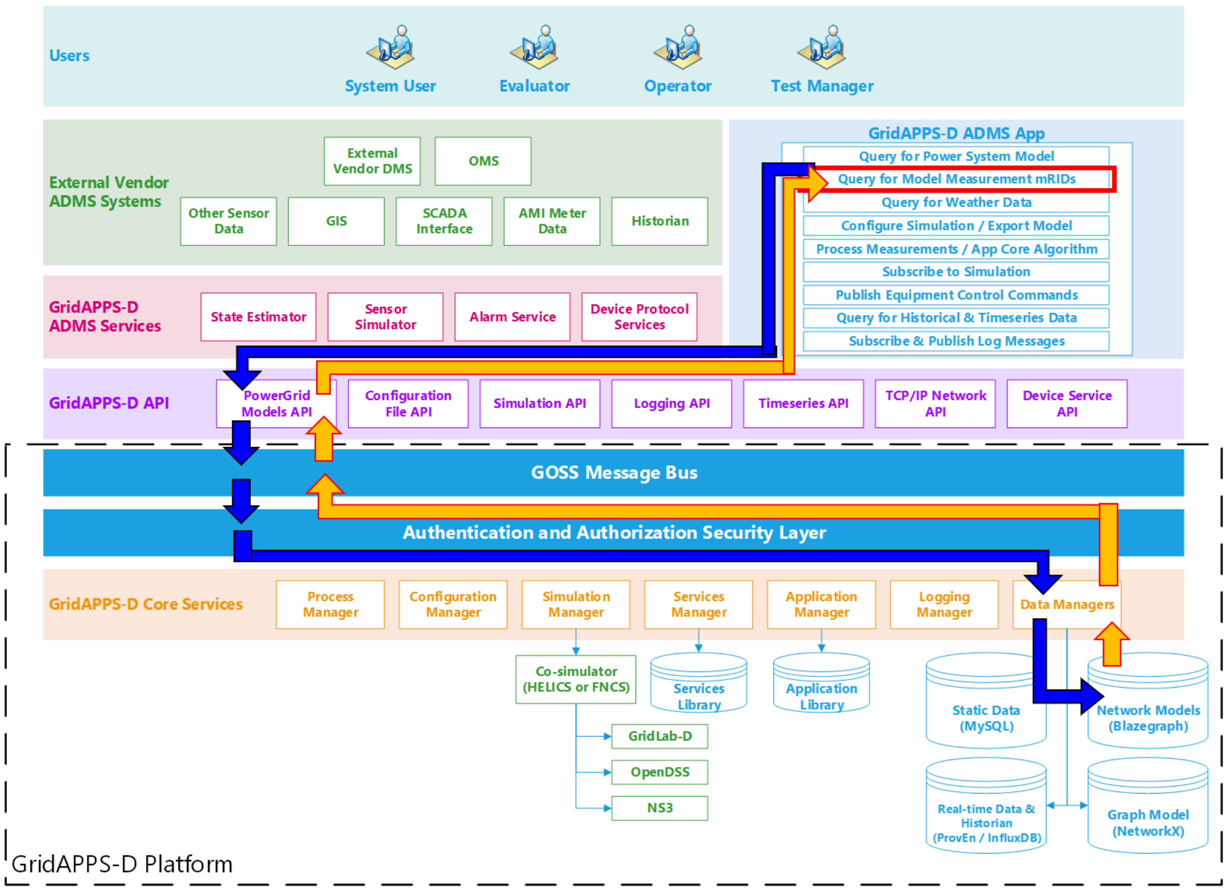
The next portion of a GridAPPS-D application is series of queries to the PowerGrid Models API to obtain information about the measurements associated with various pieces of equipment the application is interested in. Due to structure of the Common Information Model (introduced in *Intro to Common Information Model*), there exist a separate set of objects associated with the positive-neutral-voltage (PNV), volt-ampere (VA), and position measurements (POS) for each line, transformer, switch, etc.

Because GridAPPS-D applications are designed to be portable across numerous power system models without any code modification, the application must query the Blazegraph Database and create a set of local variables that contain the unique mRIDS of each measurement needed by the app to run its internal code. In a subsequent step, the app will use these measurement mRIDs to subscribe to the live streaming data issued by the simulation.

12.4.1 Measurement Query Information Flow

The figure below shows the information flow involved in making a query for the power system model.

The query is sent using `gapps.get_response(topic, message)` on a queue channel (explained in *API Communication Channels*) with a response expected back from the platform within the specified timeout period.



The figure below shows the information flow involved in making a query for the power system model.

Application passes query to GridAPPS-D Platform

First, the application creates a query message for requesting information about the desired power system components in the format of a JSON string or equivalant Python dictionary object. The syntax of this message is explained in detail in *Using PowerGrid Models API*.

The application then passes the query through the PowerGrid Models API to the GridAPPS-D Platform, which publishes it to a queue channel on the GOSS Message Bus. If the app is authenticated and authorized to pass queries, the query message is delivered to the data managers, which obtain the desired information from the Blazegraph Database.

GridAPPS-D Platform responds to Application query

The data managers then publish the response from the Blazegraph Database to the appropriate queue channel. The PowerGrid Models API then returns the desired information back to the application as a JSON message or equivalant Python dictionary object.

12.4.2 Measurement Query Sample App Code

Below is a sample query of how the application will use the PowerGrid Models API to query for the measurement mRIDs of all switches in the power system model

```
[ ]: message = {
         "modelId": model_mrid,
         "requestType": "QUERY_OBJECT_MEASUREMENTS",
         "resultFormat": "JSON",
         "objectType": "LoadBreakSwitch"
     }

     response_obj = gapps.get_response(t.REQUEST_POWERGRID_DATA, message) # Pass query to␣
     ↪PowerGrid Models API
     measurements_obj = response_obj["data"]

     global Pos_obj # Define global python dictionary of position measurements
     Pos_obj = [k for k in measurements_obj if k['type'] == 'Pos'] # Filter measurements␣
     ↪to just switch positions

     print(Pos_obj[0]) # Print switch position measurement mRID for first switch
```

12.5 Querying for Weather Data

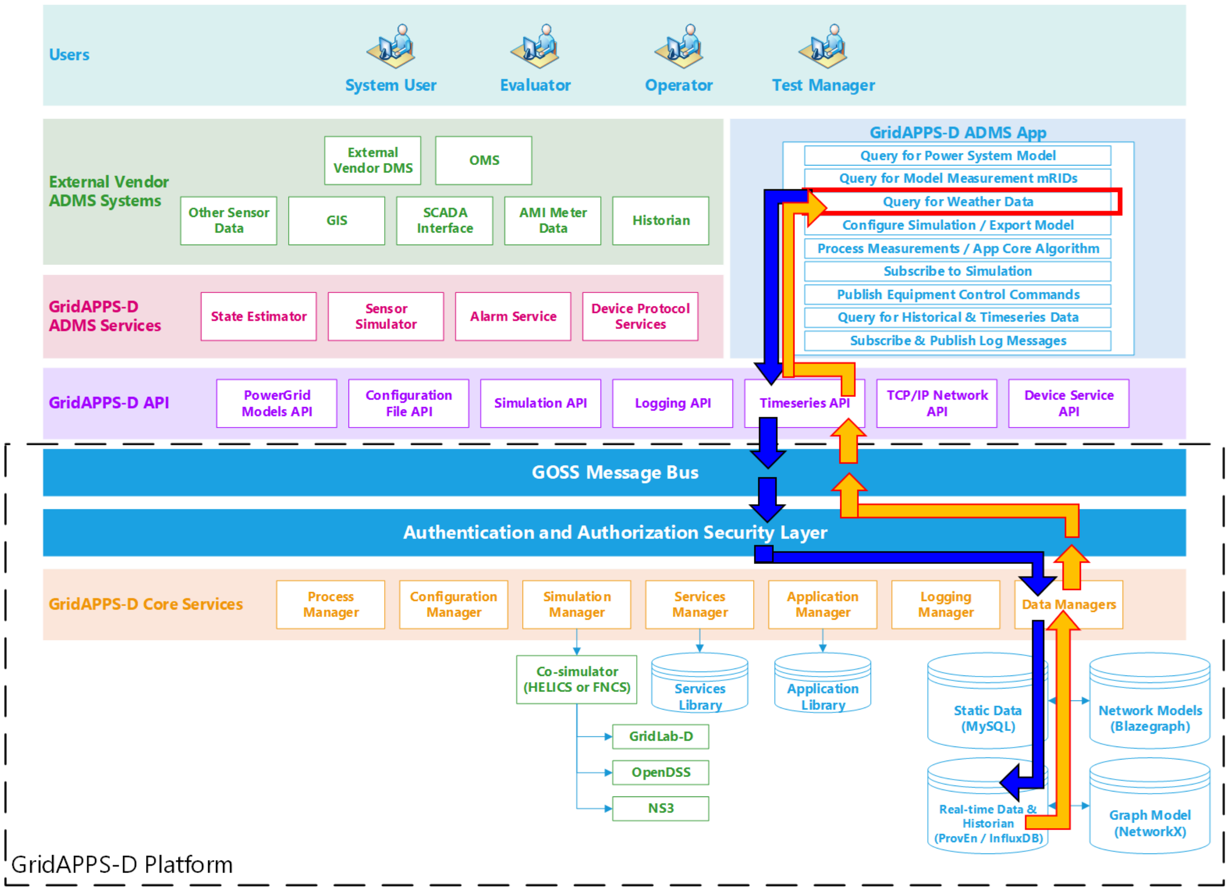
The next portion of a GridAPPS-D application is series of queries to the Timeseries API to obtain information about the weather data for the current time, including irradiation, temperature, etc. This information can be used for solar forecasting, load forecasting, etc.

Because GridAPPS-D applications are designed to be portable across numerous power system models without any code modification, the application must query the Timeseries Influx Database and create a set of local variables that contain the weather data needed by the app to run its internal code.

12.5.1 Weather Query Information Flow

The figure below shows the information flow involved in making a query for the power system model.

The query is sent using `gapps.get_response(topic, message)` on the Timeseries queue channel (explained in *API Communication Channels*) with a response expected back from the platform within the specified timeout period.



Application passes query to GridAPPS-D Platform

First, the application creates a query message for requesting information about the desired power system components in the format of a JSON string or equivalant Python dictionary object. The syntax of this message is explained in detail in *Using the Timeseries API*.

The application then passes the query through the Timeseries API to the GridAPPS-D Platform, which publishes it to a queue channel on the GOSS Message Bus. If the app is authenticated and authorized to pass queries, the query message is delivered to the Data Managers, which obtain the desired information from the Timeseries Influx Database.

GridAPPS-D Platform responds to Application query

The Data Managers then publish the response from the Timeseries Influx Database to the appropriate queue channel. The Timeseries API then returns the desired information back to the application as a JSON message or equivalant Python dictionary object.

12.5.2 Weather Query Sample App Code

Below is a sample query to the Timeseries API requesting all weather data between a certain startTime and endTime (given in unix absolute time). The application can then use that weather data to feed its internal forecasting algorithm.

```
[ ]: # Use queryFilter of "startTime" and "endTime"
message = {
    "queryMeasurement":"weather",
    "queryFilter":{"startTime":"1357048800000000",
                   "endTime":"1357048860000000"},
    "responseFormat":"JSON"
}

response_obj = gapps.get_response(t.TIMESERIES, message) # Pass query to Timeseries␣
↪API
weather_obj = response_obj["data"]

print(weather_obj[1]) # Print first line of weather data
```

12.6 Configuring a Parallel Simulation

Some applications may choose to run parallel simulations (similar to a digital twin), either within the GridAPPS-D platform or by exporting the model to OpenDSS, GridLAB-D, etc. This is accomplished through one or more queries to the Configuration File API to create a simulation configuration file and/or exported power system model.

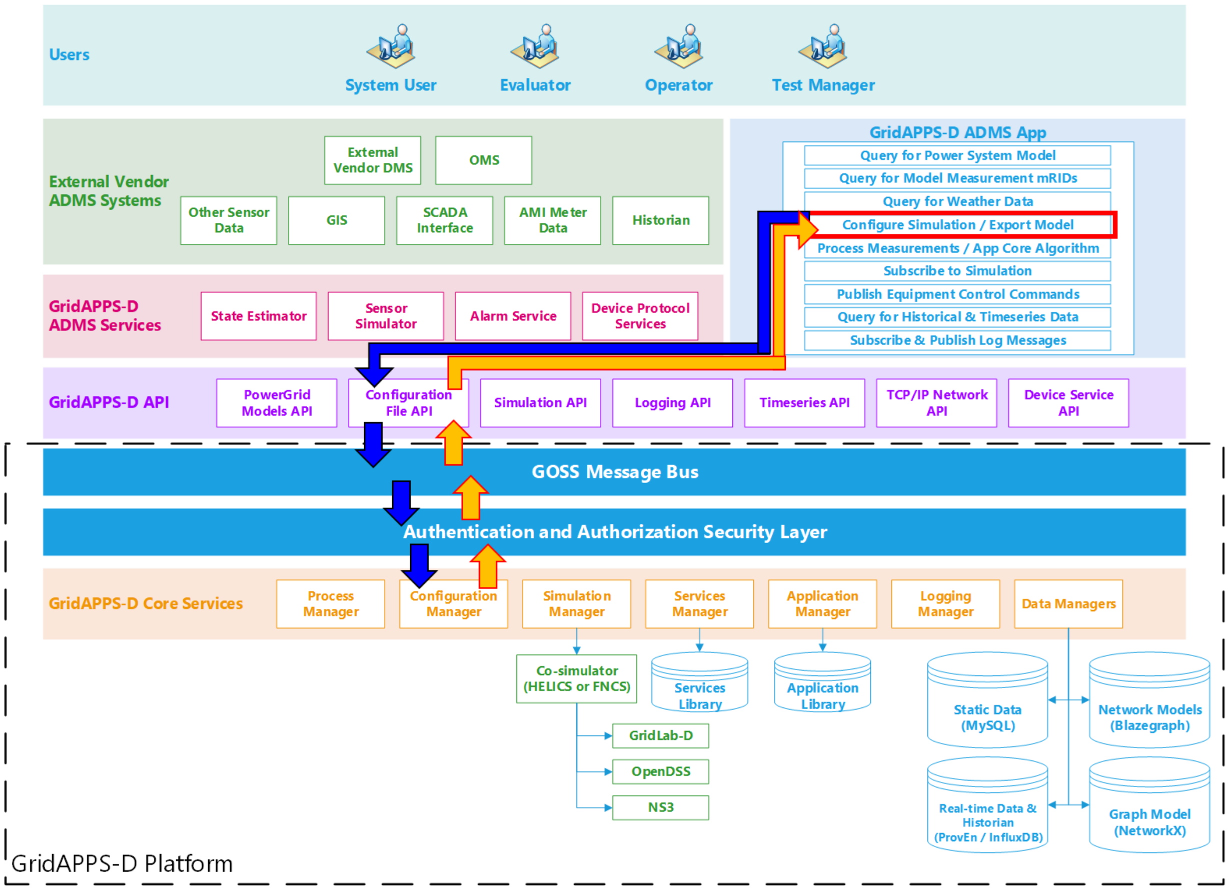
The simulation configuration file contains all the necessary info to create a new simulation, including the power system model, date/time, and variations from the default basecase (i.e. re-dispatched DERs and switches that have been opened/closed).

The exported power system model is the entire model as a set of GLM or DSS that can be saved to an external file and then solved with a different power flow solver outside of the GridAPPS-D Platform.

12.6.1 Configuration Query Information Flow

The figure below shows the information flow involved in making a query for the power system model.

The query is sent using `gapps.get_response(topic, message)` on the Configuration File queue channel (explained in *API Communication Channels*) with a response expected back from the platform within the specified timeout period.



Application passes query to GridAPPS-D Platform

First, the application creates a query message for requesting information about the desired power system configuration in the format of a JSON string or equivalant Python dictionary object. The syntax of this message is explained in detail in *Using the Configuration File API*

The application then passes the query through the Configuration File API to the GridAPPS-D Platform, which publishes it to a queue channel on the GOSS Message Bus. If the app is authenticated and authorized to pass queries, the query message is delivered to the Configuration Manager.

GridAPPS-D Platform responds to Application query

The Configuration Manager obtains the CIM XML file for the desired power system model and then converts it to the desired output format with all of the requested changes to the model. The Configuration File API then returns the desired information back to the application as a JSON message (for Y-Bus or partial models) or export the files to the directory specified in the

12.6.2 Configuration Query Sample App Code

Below is a sample query showing how an application would make a query through the Configuration File API to change all loads to constant current loads, convert the power system model to a set of OpenDSS files, and export them to the directory `/tmp/dsssimulation`.

[ ]:
```
topic = "goss.gridappsd.process.request.config"

message = {
    "configurationType": "DSS All",
    "parameters": {
        "directory": "/tmp/dsssimulation/",
        "model_id": model_mrid,
        "simulation_id": "12345678",
        "simulation_name": "ieee123",
        "simulation_start_time": "1518958800",
        "simulation_duration": "60",
        "simulation_broker_host": "localhost",
        "simulation_broker_port": "61616",
        "schedule_name": "ieeezipload",
        "load_scaling_factor": "1.0",
        "z_fraction": "0.0",
        "i_fraction": "1.0",
        "p_fraction": "0.0",
        "solver_method": "NR" }
}

gapps.get_response(topic, message)
```

12.7 Processing Measurements & App Core Algorithm

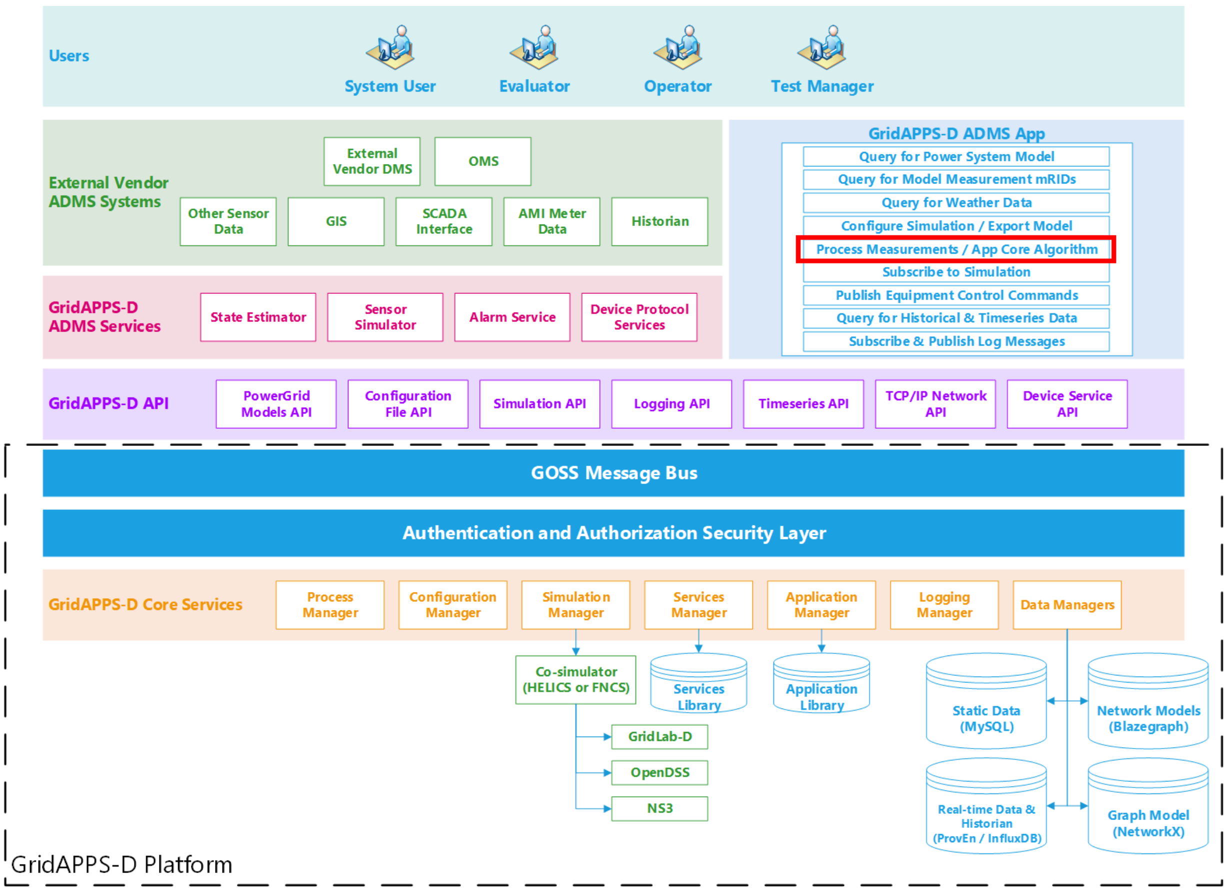
The next portion of a GridAPPS-D application is the measurement processing and core algorithm section. This section is built as either a class or function definition with prescribed arguments. Each has its advantages and disadvantages:

- The function-based approach is simpler and easier to implement. However, any parameters obtained from other APIs or methods to be used inside the function currently need to be defined as global variables.
- The class-based approach is more complex, but also more powerful. It provides greater flexibility in creating additional methods, arguments, etc.

12.7.1 App Core Information Flow

This portion of the application does not communicate directly with the GridAPPS-D platform.

Instead, the next part of the GridAPPS-D application (*Subscribing to Simulation Output*) delivers the simulated SCADA measurement data to the core algorithm function / class definition. The core algorithm processes the data to extract the desired measurements and run its optimization / control agorithm.



No message from core algorithm to GridAPPS-D Platform

The core algorithm does not send any API messages to the platform

No response to core algorithm from GridAPPS-D Platform

The core algorithm receives its measurement data and other imputs from the subscription object defined next, rather than directly from the GridAPPS-D platform.

12.7.2 App Core Sample App Code

Below is a very simple core algorithm that determines the number of open switches in the model and prints the result for each simulation timestep. The syntax of the function / class definition is described in detail in

```
[ ]: def demoSubscription1(header, message):
         # Extract time and measurement values from message
         timestamp = message["message"]["timestamp"]
         meas_value = message["message"]["measurements"]

         meas_mrid = list(meas_value.keys()) #obtain list of all mrid from message

         # Filter to measurements with value of zero
         open_switches = []
         for index in Pos_obj:
             if index["measid"] in meas_value:
```

(continues on next page)

(continued from previous page)

```
            mrid = index["measid"]
            power = meas_value[mrid]
            if power["value"] == 0:
                open_switches.append(index["eqname"])

    # Print message to command line
    print("............")
    print("Number of open switches at time", timestamp, ' is ', len(set(open_
↪switches)))
```

12.8 Subscribing to Simulation Output

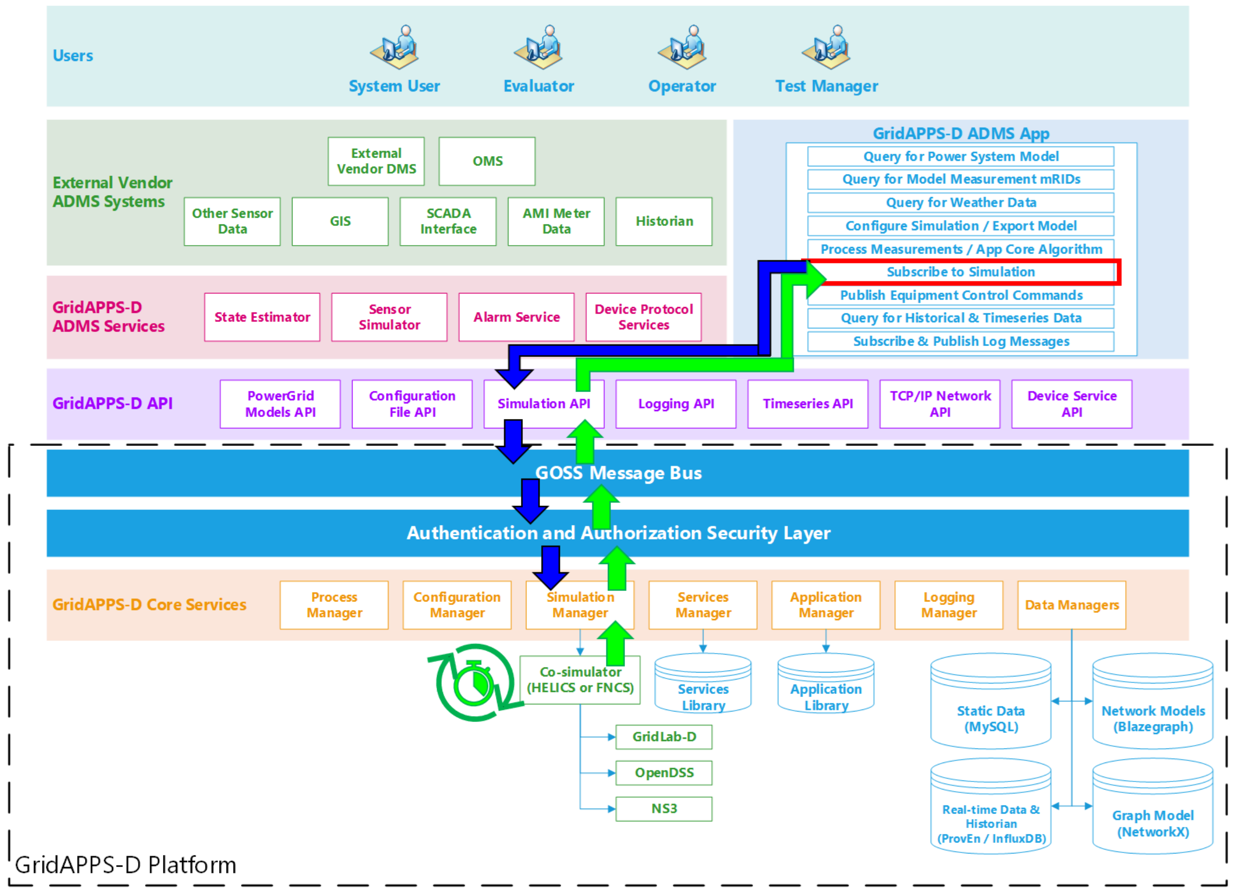
The next portion of a GridAPPS-D application is series of queries to the Timeseries API to obtain information about the weather data for the current time, including irradiation, temperature, etc. This information can be used for solar forecasting, load forecasting, etc.

Because GridAPPS-D applications are designed to be portable across numerous power system models without any code modification, the application must query the Timeseries Influx Database and create a set of local variables that contain the weather data needed by the app to run its internal code.

12.8.1 Simulation Subscription Information Flow

The figure below shows the information flow involved in subscribing to the simulation output.

The subscription request is sent using `gapps.subscribe(topic, class/function object)` on the specific Simulation topic channel (explained in *API Communication Channels*). No immediate response is expected back from the platform. However, after the next simulation timestep, the Platform will continue to deliver a complete set of measurements back to the application for each timestep until the end of the simulation.



Application passes subscription request to GridAPPS-D Platform

The subscription request is perfromed by passing the app core algorithm function / class definition to the `gapps.subscribe` method. The application then passes the subscription request through the Simulation API to the topic channel for the particular simulation on the GOSS Message Bus. If the application is authorized to access simulation output, the subscription request is delivered to the Simulation Manager.

GridAPPS-D Platform delivers published simulation output to Application

Unlike the previous queries made to the various databases, the GridAPPS-D Platform does not provide any immediate response back to the application. Instead, the Simulation Manager will start delivering measurement data back to the application through the Simulation API at each subsequent timestep until the simulation ends or the application unsubscribes. The measurement data is then passed to the core algorithm class / function, where it is processed and used to run the app's optimization / control algorithms.

12.8.2 Simulation Subscription Sample App Code

Below is an example of how an application subscribes to the GridAPPS-D simulation output using the function or class definition created as part of the *Measurement Processing / App Core*

```
[ ]: from gridappsd.topics import simulation_output_topic

output_topic = simulation_output_topic(viz_simulation_id)

gapps.subscribe(output_topic, demoSubscription1)
```

12.9 Publishing Equipment Commands

The next portion of a GridAPPS-D App is publishing equipment control commands based on the optimization results or objectives of the app algorithm.

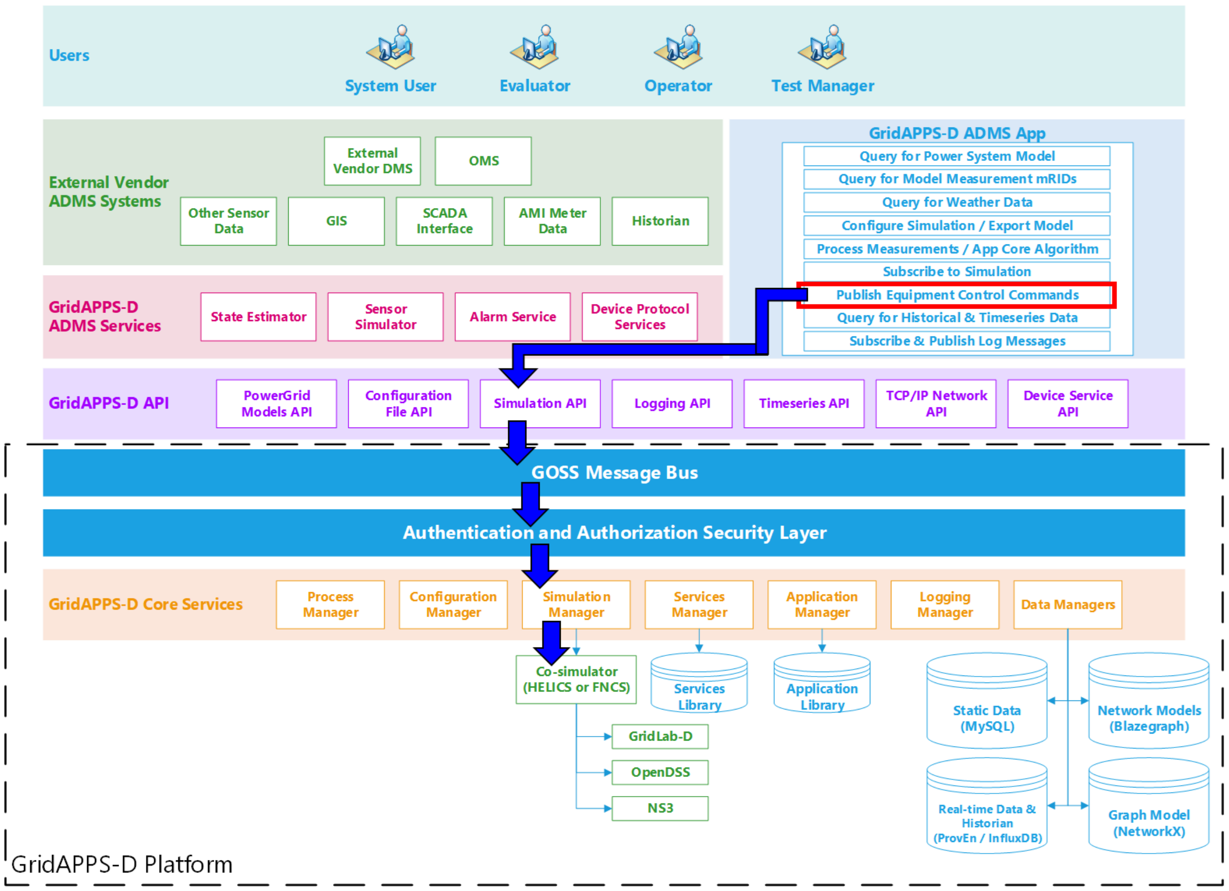
Depending on the preference of the developer, this portion can be a separate function definition, or included as part of the main class definition as part of the *Measurement Processing / App Core* class definition described earlier.

12.9.1 Equipment Command Information Flow

The figure below outlines information flow involved in publishing equipment commands to the simulation input.

Unlike the various queries to the databases in the app sections earlier, equipment control commands are passed to the GridAPPS-D API using the `gapps.send(topic, message)` method. No response is expected from the GridAPPS-D platform.

If the application desires to verify that the equipment control command was received and implemented, it needs to do so by 1) checking for changes in the associated measurements at the next timestep and/or 2) querying the Timeseries Database for historical simulation data associated with the equipment control command.



Application sends difference message to GridAPPS-D Platform

First, the application creates a difference message containing the current and desired future control point / state of the particular piece of power system equipment to be controlled. The difference message is a JSON string or equivalant Python dictionary object. The syntax of a difference message is explained in detail in *Publishing Equipment Commands*.

The application then passes the query through the Simulation API to the GridAPPS-D Platform, which publishes it on the topic channel for the particular simulation on the GOSS Message Bus. If the app is authenticated and authorized to control equipment, the difference message is delivered to the Simulation Manager. The Simulation Manager then passes the command to the simulation through the Co-Simulation Bridge (either FNCS or HELICS).

No response from GridAPPS-D Platform back to Application

The GridAPPS-D Platform does not provide any response back to the application after processing the difference message and implementing the new equipment control setpoint.

12.9.2 Equipment Command Sample App Code

Below is an example of an app code block

```
[ ]: import time
     from gridappsd import DifferenceBuilder
     from gridappsd.topics import simulation_input_topic

     input_topic = simulation_input_topic(viz_simulation_id)

     my_open_diff = DifferenceBuilder(viz_simulation_id)
     my_open_diff.add_difference(sw_mrid, "Switch.open", 1, 0) # Open switch given by sw_
     ↪mrid
     open_message = my_open_diff.get_message()

     my_close_diff = DifferenceBuilder(viz_simulation_id)
     my_close_diff.add_difference(sw_mrid, "Switch.open", 0, 1) # Close switch given by sw_
     ↪mrid
     close_message = my_close_diff.get_message()

     while True:
         time.sleep(5)
         gapps.send(input_topic, open_message)
         time.sleep(5)
         gapps.send(input_topic, close_message)
```

12.9.3 Viewing Application Results in GridAPPS-D Viz

Return to the browser tab in which the GridAPPS-D Simulation is currently running. Switch `sw5` will now be opening and closing every 5 seconds, with the downstream portion of the feeder being de-energized and reconnected with each switch operation.

The core application algorithm will also reflect this with the printed response alternating between two and three open switches every few timesteps.

12.10 Querying Historical & Timeseries Data

The next portion of a GridAPPS-D application is querying historical data from the current and/or previous simulations.

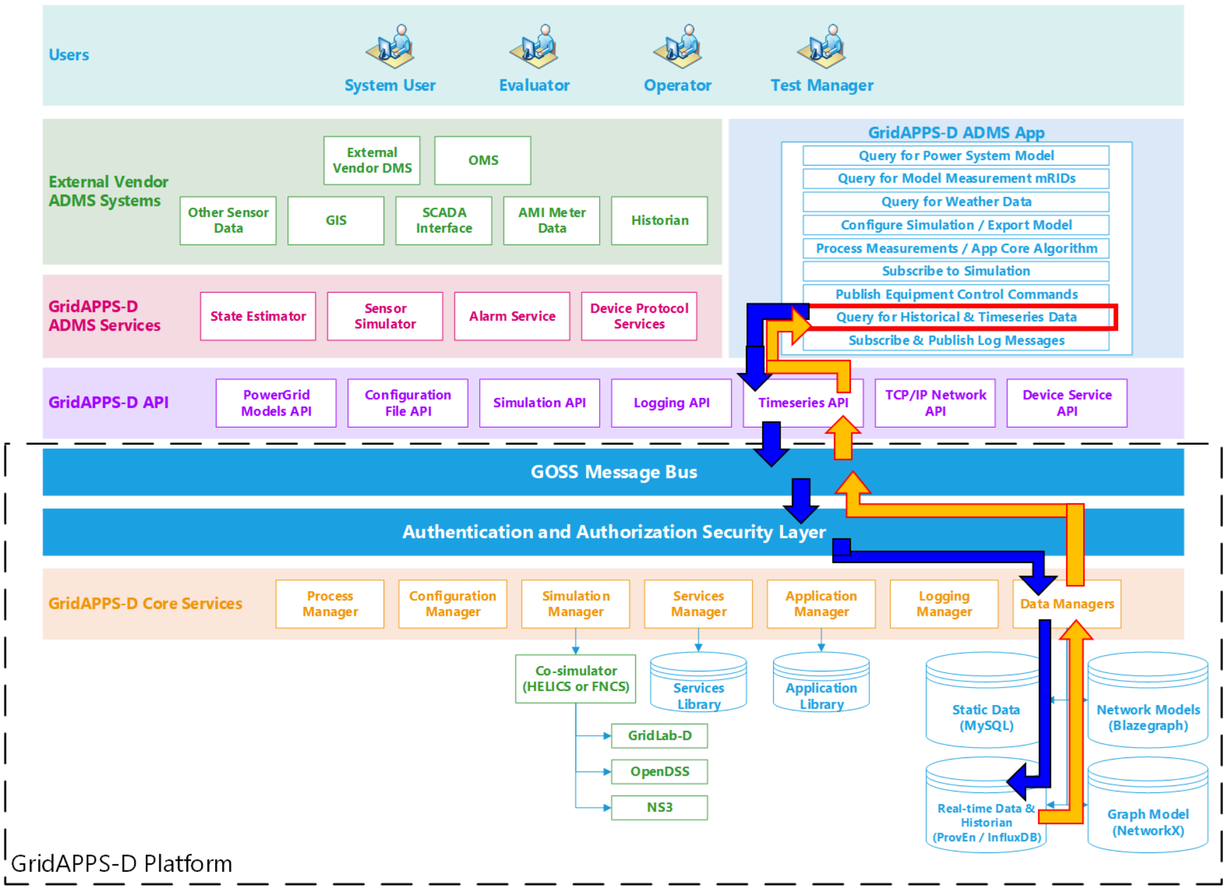
All simulation output and commands from the current and previous simulations are stored in the Timeseries Database, and can be queried to provide AI/ML training data, verify processing of equipment commands, or

Note that Timeseries Database data is cleared when the GridAPPS-D Platform is shut down with the ./stop.sh script. It is recommended to copy historical / training data to an external persistent directory using the `docker cp` command, as given in *Docker Shortcuts*.

12.10.1 Historical Data Query Information Flow

The figure below outlines the information flow involved in querying for historical and timeseries data.

The query is sent using the `gapps.get_response(topic, message)` method on the Timeseries queue channel with a response expected back from the GridAPPS-D platform within the specified timeout period.



Application passes query to GridAPPS-D Platform

First, the application creates a query message for requesting information about the desired power system components in the format of a JSON string or equivalant Python dictionary object. The syntax of this message is explained in detail in *Querying Timeseries Data*.

The application then passes the query through the Timeseries API to the GridAPPS-D Platform, which publishes it to a queue channel on the GOSS Message Bus. If the app is authenticated and authorized to pass queries, the query message is delivered to the Data Managers, which obtain the desired information from the Timeseries Influx Database.

GridAPPS-D Platform responds to Application query

The Data Managers then publish the response from the Timeseries Influx Database to the appropriate queue channel. The Timeseries API then returns the desired information back to the application as a JSON message or equivalant Python dictionary object.

12.10.2 Historical Data Query Sample App Code

[ ]:
```
import time

start_time = str(int(time.time())-10) # Start query from 10 sec ago
end_time = str(int(time.time()))

# Query for a particular set of measurments
message = {
    "queryMeasurement": "simulation",
    "queryFilter":{"simulation_id": simulation_id,
                   "startTime": start_time,
                   "endTime": end_time,
                   "measurement_mrid": pos_obj},
    "responseFormat":"JSON"
}

gapps.get_response(t.TIMESERIES, message) # Pass API call
```

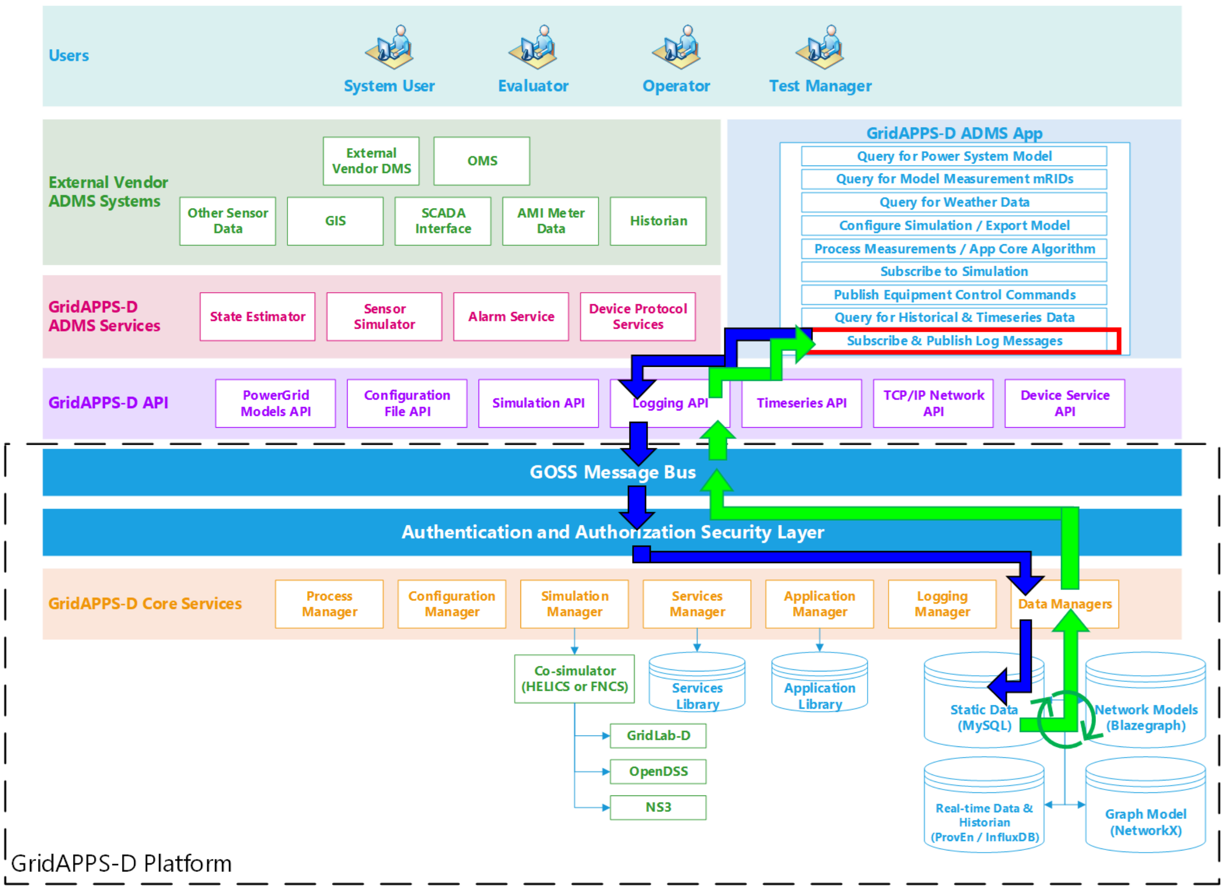
12.11 Subscribing and Publishing to Logs

The last portion of an application is subscribing and publishing to logs. This step is extremely useful for 1) informing end users of application behavior and 2) application debugging during development and demonstration.

The GridAPPS-D Logging API provides an extension of the standard Python logging library and enables applications to subscribe to real-time log messages from a simulation, query previously logged messages from the MySQL database, and publish messages to their either own log or their GridAPPS-D logs.

12.11.1 Logging Information Flow

The figure below shows the information flow involved in subscribing and publishing to logs.



12.11.2 Log Message Sample App Code

```
[ ]: from gridappsd.topics import simulation_log_topic
     log_topic = simulation_log_topic(viz_simulation_id)

     def demoLogFunction(header, message):
         timestamp = message["timestamp"]
         log_message = message["logMessage"]

         print("Log message received at timestamp ", timestamp, "which reads:")
         print(log_message)
         print("........................")

     gapps.subscribe(log_topic, demoLogFunction)
```

CHAPTER

THIRTEEN

GRIDAPPS-D SERVICE STRUCTURE

[ ]:

```

```

CHAPTER

FOURTEEN

INTRODUCTION TO THE COMMON INFORMATION MODEL

This section introduces the CIM as a model format that is used for power system data and information exchange across applications, platforms, and services. The CIM is used for all power system models in GridAPPS-D, and it is important to have an understanding of the concepts and implementation of CIM for describing power systems using unique mRIDs for each piece of equipment and associated modeling objects.

14.1 Introduction

14.1.1 What is the Common Information Model?

The Common Information Model (CIM) is an abstract information model that can be used to model an electrical network and the various equipment used on the network.

CIM is widely used for data exchange of bulk transmission power systems, and is now beginning to find increasing use for distribution modeling and analysis.

By using a common model, utilities, vendors, and researches from both academia and industry can reduce the effort and cost of data integration, and instead focus on developing increased functionality for managing and optimizing the smart grid of the future.

14.1.2 Why is Data Integration Important?

In a typical distribution utility there are hundreds and even in some cases thousands of software solutions and applications that are managed by the IT department. These applications are used and operated independently by the various groups, departments, and organizations within the utility. Whenever a business process requires data from one system or application to be transferred to another system or application, the data needs to be manually extracted from the first database and then converted to the format of the other application's database.

Two strategies exist for dealing with extreme level of effort needed to manage, update, export, convert, and import data formats between different applications and databases.

1) Reduce the number of databases by purchasing a large software suite from a single vendor using a single proprietary data format that is internally-integrated and compatible with all the applications needed by utility
2) Adopt a common data integration platform that allows external integration between multiple software packages using a shared data format

14.1.3 What does CIM Provide?

CIM is an information model, that is an abstract, formal representation of objects, their attributes, the relationships between them, and the operations that can be performed on them. It is NOT a database structure or physical data store. It is a technology-agnostic model for describing the properties of physical power system equipment, power flow data, and messages that can be exchanged between various platforms and applications.

To describe various power system objects, CIM uses **Class Diagrams** and **Sequence Diagrams** created using the Unified Modeling Language (UML). It also uses the Resource Description Framework (RDF) to describe classes and attributes in an eXtensible Markup Language (XML) file format. The details of what is covered in each part of the CIM is described in detail below.

14.2 Background and Structure of the CIM

14.2.1 UML Class Diagrams

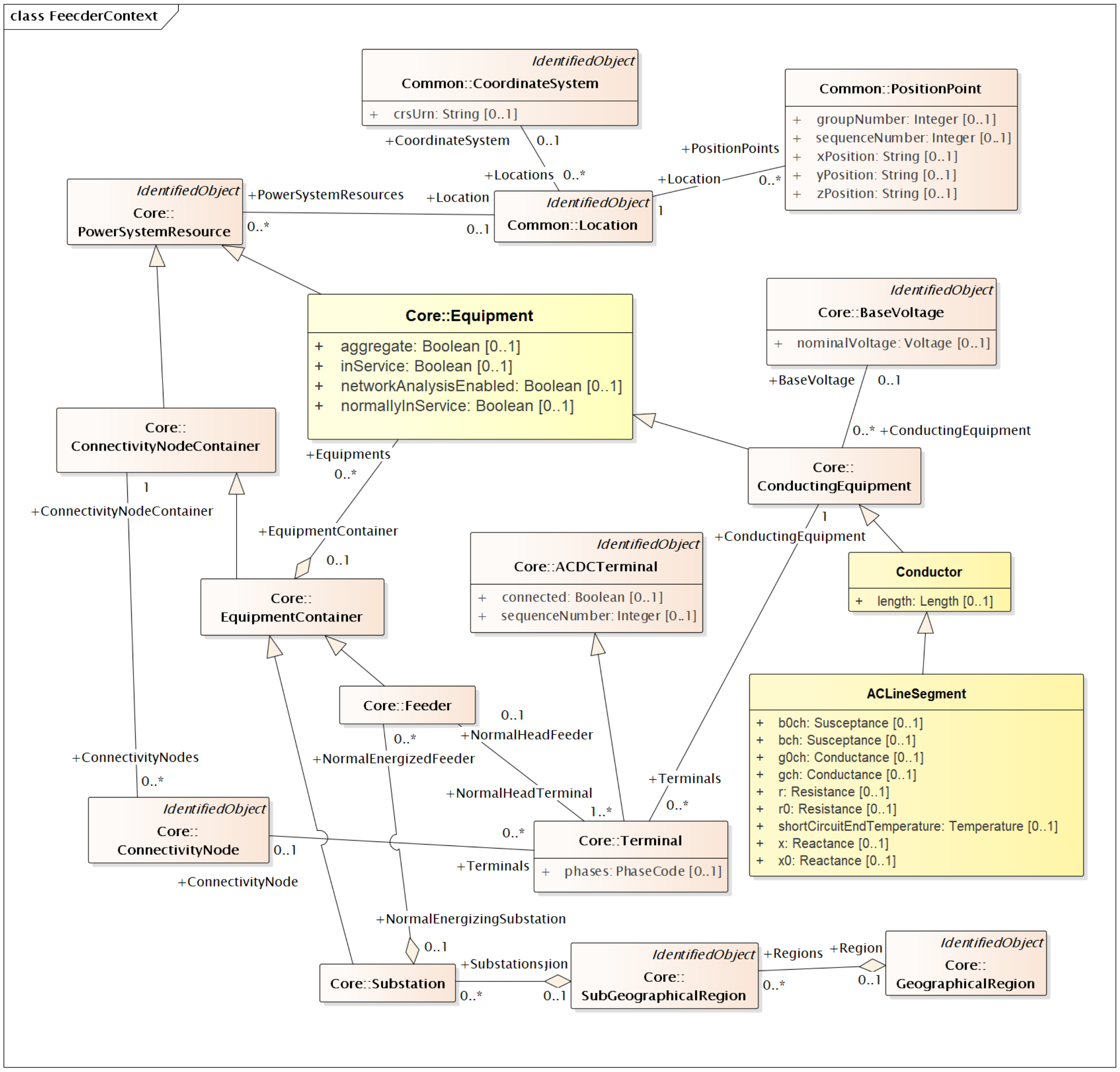
The Unified Modeling Language (UML) provides 13 types of diagrams to define software architecture. One of the is the **UML Class Diagram**, which visually represents object hierarchies and relationships.

First a review of basic concepts and terminology related to class diagrams:

- An **object** is any thing that we want to describe.
- A **class** represents a specific type of object.
- A **class hierarchy** is a model of the system showing every component as a separate class. The class hierarchy should represent the real-world structure of the system.
- A **package** is a group of classes. Think of folders in a computer file explorer.
- **Inheritance** allows us to define very general "parent classes" and very specific "child classes".
- **Attributes** are the properties that describe what type of thing the class represents.
- **Associations** are the relationships between various objects and how they are connected to each other.

Class diagrams show all the attributes and associations of various classes in a particular package in a single picture. To read a class diagram, remember that

- Lines with an arrowhead indicate class inheritance. For example, in the figure below, *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.
- Lines with a diamond indicate composition. For example, *Substations* make up a *SubGeographicalRegion*, which then make up a *GeographicRegion*.
- Lines without a terminating symbol are associations. For example, *ACLineSegment* has (through inheritance) a *BaseVoltage*, *Location* and one or more *Terminals*.
- 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, *PowerSystemResource* inherits from *IdentifiedObject*.



A complete set of UML Class Diagrams is provided in the Advanced CIM Modeling section. This section contains class diagrams for all the objects used in GridAPPS-D and tables of properties to help you create and pass your own custom SPARQL queries to the Blazegraph Database.

14.2.2 UML Sequence Diagrams

UML sequence diagrams are used to model the flow of messages, events, and actions between the entities of a system. Time is represented vertically—showing the time sequence of interactions in the system. Displayed horizontally at the top of the diagram are the applications or entities in the system.

CIM uses UML diagrams to represent work flow, operations processes, and other utility use-cases. For the purposes of application development within GridAPPS-D, a detailed understanding of UML sequence diagrams is not required.

14.2.3 Resource Description Framework (RDF)

The Resource Description Framework (RDF) is a method of defining information models that is specified by the World Wide Web Consortium (the W3C). Detailed documentation is available on the W3C website.

RDF focuses on making statements about objects in a subject-predicate-object expression. Each expression is commonly called a "triple" in RDF terminology. The subject is defined by naming a resource, the object denotes traits or attributes associated with the subject, and the predicate expresses the relationship between the subject and the object.

The subject, or resource, in an RDF model is expressed as a Uniform Resource Identifier (URI). URIs are similar to the Uniform Resource Locators (URLs) used as web addresses but are more general because they are not limited to accessible data on the web. The predicate and object are also technically URIs and so also are just identifiers. The subject-predicate-object triplets takes the form of expressing syntactical constructs like "a substation has a name".

RDF Schema (RDFS) files describe the classes, attributes, and relationships of an information model and typically use an .rdfs file format. RDF instance files describe object instances and typically use an .xml extension. RDF incremental files describe changes to a set of object instances as described by an instance file, and typically use an .xml extension.

CIM uses RDF instance files to define power system models with unique master resource identifier (**mRID**) issued by a model authority. The mRID is globally unique within an exchange context. Global uniqeness is easily achived by using a UUID for the mRID. It is strongly recommended to do this. For CIM XML data files in RDF syntax, the mRID is mapped to rdf:ID or rdf:about attributes that identify CIM object elements.

Key Concepts & Terminology from RDF

- **URI References** – CIM and GridAPPS-D use two URI references to identify properties and resources. These identify the RDF format and the CIM classes used.
  - `<http://www.w3.org/1999/02/22-rdf-syntax-ns#>`
  - `<http://iec.ch/TC57/CIM100#>`
- __

14.3 Summary of CIM XML Classes

This section provides a brief look at the classes of equipment modeled in CIM XML and used in GridAPPS-D.

Details of each package, the class diagram, and attributes of each class are provided in the relevant sections of the reference guide to this lesson.

14.3.1 Names, Nodes, Terminals

The *Core* package provides very high level information of the distribution feeder model

IdentifiedObject

The *Core* package contains a class called *IdentifiedObject*. This class is very abstract and only contains attributes used to reference the object either by a user or in software. The attributes of *IdentifiedObject* include ***mRID***, which is the master resource identifier that should be a globally 3-18 unique identifier of objects; the *mRID* does not have to be human-readable. This identifier is generally intended to be used by software systems.

The attributes *name*, *description*, *aliasName*, and *pathName* are intended for providing identifiers that are human-readable. It is common for names of objects within a utility to not be unique due to historical naming conventions, the results of mergers and acquisitions, and the inability of other software systems to manage uniqueness. For these reasons, there are no constraints on these names requiring them to be unique.

PowerSystemResource

The *PowerSystemResource* class inherits from *IdentifiedObject* and provides another relatively abstract class used in the CIM. The *PowerSystemResource* class supports an association to a *Company* class. This relationship identifies the company that operates the resource.

ConnectivityNode

The *ConnectivityNode* class has a relationship to the *Terminal* class. Each *ConductingEquipment* object has *Terminals,* which are then connected to *ConnectivityNodes.* The terminals can be thought of as being closely related to the conducting equipment, and the connectivity nodes are the glue that defines what equipment is connected to what other equipment.

CIM also includes the *TopologicalNode* class, which is used to convert breaker-switch oriented power system models to bus-branch models. This object is not used in GridAPPS-D, which does not feature transmission substation configurations (e.g. breaker-and-a-half, main-and-transfer-bus, ring-bus, etc.) that require topological processing of breaker and switch positions to determine network topology and line connectivity.

14.3.2 Power System Equipment

CIM XML provides a number of classes for defining physical power system equipment, including lines, switches, transformers, regulators, capacitors, and reactors.

Equipment and *ConductingEquipment*

The *ConductingEquipment* class inherits from an *Equipment* class which inherits from *PowerSystemResource*. This is the parent class for most of the physical equipment that are used to model the power system.

Conductor and *ACLineSegment*

Directly inheriting from *ConductingEquipment* is the *Conductor* class. This class specifies the length of the conductor.

Each segment of a distribution line is defined in a CIM model as an *ACLineSegment*. This class contains the electrical attributes commonly associated with a line needed for steady state analysis, including the positive-sequence and zero-sequence resistance, reactance, conductance, and susceptance.

More details are available in the **`LineModel class diagram <>`__** and **`list of attributes <>`__**

PowerTransformer, *TransformerWindings*, and *TapChanger*

These three classes specify the portions of a step-down transformer and regulator.

The *PowerTransformer* class inherits from *Equipment* (not ConductingEquipment) and has associations to the TransformerWinding class.

The majority of the electrical characteristics associated with the transformer are actually associated with the *TransformerWinding* objects.

An association from the *TransformerWinding* class to the *TapChanger* class is used when the transformer has a tap changer. The *TapChanger* class has as attributes for things like the tap steps and nominal setting. The *TapChanger* class inherits from the *PowerSystemResource* class instead of the *Equipment* class, so it has few inherited attributes and associations.

14.4 References

Portions of this tutorial have reproduced verbatim text and information from the EPRI report An Introduction to the CIM for Integrating Distribution Applications and System and the CIM Ontology Diagrams

[ ]:

```
```

CHAPTER

FIFTEEN

API COMMUNICATION CHANNELS

When communicating with the GridAPPS-D Platform through API, it is necessary to specify a communication channel, which tells the GridAPPS-D platform on which channel to communicate with the application and through which API the message should be directed.

15.1 `/queue/` vs `/topic/` Channels

GridAPPS-D uses two types of communication channels to determine the visibility of the API call to other applications and services.

15.1.1 Queue Channels

`/queue/` is used for communication channels where only the GridAPPS-D Platform is listening to the API call. These API calls are processed on a first-in, first-out basis. There is only one subscriber to the communication channel.

API calls to the Blazegraph database, Logs, Timeseries database, Config files, and Platform status are all queue channels. All the GridAPPS-D Topics for queue channels typically do not change over the course of an application or simulation run.

In the GridAPPSD-Python library, it is assumed that a topic is a queue channel if not otherwise specified. These two GridAPPS-D Topic definitions are equivalent:

```
topic = '/queue/goss.gridappsd.process.request.data.powergridmodel'
```

```
topic = 'goss.gridappsd.process.request.data.powergridmodel'
```

15.1.2 Topic Channels

`/topic/` is used for communication channels where the API call is to broadcast to all subscribers through the GOSS Message Bus, inlcuding other applications, services, FNCS Bridge, etc.

API calls to the Simulation, services, and active applications use topic channels to communicate and typically need to the specify the Simulation ID, Service ID, and Application ID. The particular topic for such an API call will change between simulations and instances, and so shortcut functions are provided in GridAPPSD-Python library to assist in generating the correct Topic.

In GridAPPSD-Python, it is necessary to specify if a GridAPPS-D Topic is a `/topic/` channel broadcasting to all subscribers:

```
topic = "/topic/goss.gridappsd.simulation.input."+simulation_id
```

15.2 Static GridAPPS-D Topics

Below are a list of the most common topics and where they are used. The appropriate topic for each API call will also be listed again in the subsequent sections on each GridAPPS-D API. The list below can serve as an additional convenient reference.

These topics remain the remain the same between platform, application, and simulation instances. The GridAPPSD-Python Library shortcuts use all uppercase naming to indicate that these are static topic names.

15.2.1 Importing the Topics Library

When using topics in GridAPPSD-Python, it is recommended to import the `topics` library from `gridappsd`. This enables you to rapidly call the correct topic without needing to search for the correct topic string. This also protects your code from any changes inside the GridAPPS-D Platform if particular topic strings are deprecated or replaced – the python library names will stay persistent between all Platform releases.

For static GridAPPS-D topics, import the library by running

```
[ ]: from gridappsd import topics as t
```

15.2.2 Request PowerGrid Model Data

This `/queue/` channel is used to communicate with PowerGrid Models API to pull power system model info from the the Blazegraph Database. The PowerGrid Model API is covered in detail in *Using the PowerGrid Models API*.

The base static string used is `goss.gridappsd.process.request.data.powergridmodel`, which can be called using the `.REQUEST_POWERGRID_DATA` or `.BLAZEGRAPH` methods from the topics library

A sample message that would be passed with this topic is

```
[ ]: from gridappsd import topics as t

     # Sample PowerGrid Model message
     message = '{"requestType": "QUERY_MODEL_NAMES", "resultFormat": "JSON"}'

     gapps.get_response(t.REQUEST_POWERGRID_DATA, message)
```

```
[ ]: from gridappsd import topics as t

     # Sample PowerGrid Model message
     message = '{"requestType": "QUERY_MODEL_NAMES", "resultFormat": "JSON"}'

     gapps.get_response(t.BLAZEGRAPH, message)
```

15.2.3 Request Timeseries Data

This `/queue/` channel is used to communicate with the Timeseries API and Timeseries database, which stores real-time and historical data, such as weather information and AMI meter readings. The Timeseries database is covered in detail in *Using the Timeseries API*. A sample message that would be passed with this topic is

Text String: The topic can be specified as a static string:

- `topic = "goss.gridappsd.process.request.data.timeseries"`
- `gapps.get_response(topic, message)`

GridAPPSD-Python Library Method: The correct topic can also be imported from the GridAPPSD-Python topics library:

- `from gridappsd import topics as t`
- `gapps.get_response(t.TIMESERIES, message)`

15.2.4 Request Platform Status

This topic is used to check that status of the GridAPPS-D Platform.

Text String: The topic can be specified as a static string:

- `topic = "/queue/goss.gridappsd.process.request.status.platform"`
- `gapps.get_response(topic, message)`

GridAPPSD-Python Library Method: The correct topic can also be imported from the GridAPPSD-Python topics library.

- `from gridappsd import topics as t`
- `gapps.get_response(t.PLATFORM_STATUS, message)`

15.2.5 Querying Log Data

This topic is used to query log data in the MySQL Database using the Logging API

Note: This topic is different from the one used to subscribe to real-time log data being published by an ongoing simulation. This topic is used for querying data already stored in the database.

Text String: The topic can be specified as a static string:

- `topic = "goss.gridappsd.process.request.data.log"`
- `gapps.get_response(topic, message)`

GridAPPSD-Python Library Method: The correct topic can also be imported from the GridAPPSD-Python topics library:

- `from gridappsd import topics as t`
- `gapps.get_response(t.LOGS, message)`

15.2.6 Subscribing to Platform Logs

This topic is used to subscribe the to logs created by the GridAPPS-D Platform, such as which managers and core services have been started and are running.

Text String: The topic can be specified as a static string:

- `topic = "goss.gridappsd.process.request.data.timeseries"`
- `gapps.get_response(topic, message)`

GridAPPSD-Python Library Function: The correct topic can also be imported from the GridAPPSD-Python topics library. Note that this is a python function similar to the dynamic topics presented in the next section.

- `from gridappsd.topics import platfor_log_topic
- `topic = platform_log_topic()`
- `gapps.get_response(topic, message)`

[*Return to Top*]

15.3 Dynamic GridAPPS-D Topics

Several GridAPPS-D topics are unique to each application, simulation, or service instance. These topics are dynamic and will change from instance to instance.

The GridAPPS-D Platform will require that the topic specify the particular instance so that the API call can be delivered to the correct simulation or service.

To assist with the task of creating a dynamic topic that automatically updates between instances, several function are available in the GridAPPSD-Python topics library.

The available GridAPPSD-Python functions for dynamic topics are

- `simulation_input_topic(simulaton_id)` – Gets the topic to write data to for the simulation
- `simulation_output_topic(simulation_id)` – Gets the topic for subscribing to output from the simulation
- `simulation_log_topic(simulation_id)` – Topic for the subscribing to the logs from the simulation
- `service_input_topic(service_id, simulation_id)` – Utility method for getting the input topic for a specific service
- `service_output_topic(service_id, simulation_id)` – Utility method for getting the output topic for a specific service
- `application_input_topic(application_id, simulation_id)` – Utility method for getting the input topic for a specific application
- `application_output_topic(application_id, simulation_id)` – Utility method for getting the output topic for a specific application

---

15.3.1 Subscribe to Simulation Output

This topic is used to communicate with the Simulation API, which is covered in detail in *Controlling Simulations with Simulation API*. The Simulation Output Topic is used to subscribe to the simulation output, enabling applications to listen to switching actions, obtain equipment measurements, and so on.

The GridAPPSD-Python shortcut function for generating the correct topic is

`simulation_output_topic(simulation_id)`

There are two ways to use the function. The first is to call the library function directly. The second is to use it as part of a class definition.

1) Call the topic function directly

```
[ ]: # Import GridAPPS-D Topic Function:
from gridappsd.topics import simulation_output_topic

# Call GridAPPSD-Python Topic Function
topic = simulation_output_topic(simulation_id)

# Print to Notebook Kernel:
print(topic)
```

2) Use the topic function in a class definition

```
[ ]: # Import GridAPPS-D Topic Function:
from gridappsd.topics import simulation_output_topic

# Define Subscription Class
class MySubscription(object):
    def __init__(self,simulation_id):
        self._subscribe_to_topic = simulation_output_topic(simulation_id)

# Define Main Function:
def _main():
    subscription = MySubscription(simulation_id)
    print(subscription._subscribe_to_topic)

# Call Main Function:
_main()
```

15.3.2 Publish to Simulation Input

This topic is used to communicate with the Simulation API, which is covered in detail in *Controlling Simulations with Simulation API*. The Simulation Input Topic is used to publish commands to the GOSS Message Bus, which are then broadcast to all applications, services, and simulations that are listening. Examples of actions that will use this topic include taking switching actions, adjusting DER setpoints, and changing regulator taps.

The GridAPPSD-Python shortcut function for generating the correct topic is

`simulation_input_topic(simulation_id)`

There are two ways to use the function. The first is to call the library function directly. The second is to use it as part of a class definition.

1) Call the topic function directly

[ ]:
```python
# Import GridAPPS-D Topic Function:
from gridappsd.topics import simulation_input_topic

# Call GridAPPSD-Python Topic Function
topic = simulation_output_topic(simulation_id)

# Print to Notebook Kernel:
print(topic)
```

2) Use the topic function in a class definition

[ ]:
```python
# Import GridAPPS-D Topic Function:
from gridappsd.topics import simulation_input_topic

# Define Subscription Class
class MySimulationPublisher(object):
    def __init__(self,simulation_id):
        self._publish_to_topic = simulation_input_topic(simulation_id)

# Define Main Function:
def _main():
    subscription = MySimulationPublisher(simulation_id)
    print(subscription._publish_to_topic)

# Call Main Function:
_main()
```

15.3.3 Subscribe to Simulation Logs

This topic is used to communicate with the Simulation API, which is covered in detail in Lesson XX. The Simulation Output Topic is used to subscribe to the simulation output, which applications use to * Listen to switching actions * Obtaining equipment measurements * ***GET FULL LIST***

The GridAPPSD-Python shortcut function for generating the correct topic is

`simulation_output_topic(simulation_id)`

There are two ways to use the function. The first is to call the library function directly. The second is to use it as part of a class definition.

1) Call the topic function directly

[ ]:
```python
# Import GridAPPS-D Topic Function:
from gridappsd.topics import simulation_output_topic

# Call GridAPPSD-Python Topic Function
topic = simulation_output_topic(simulation_id)

# Print to Notebook Kernel:
print(topic)
```

2) Use the topic function in a class definition

```
[ ]: # Import GridAPPS-D Topic Function:
from gridappsd.topics import simulation_output_topic

# Define Subscription Class
class MySubscription(object):
    def __init__(self,simulation_id):
        self._subscribe_to_topic = simulation_output_topic(simulation_id)

# Define Main Function:
def _main():
    subscription = MySubscription(simulation_id)
    print(subscription._subscribe_to_topic)

# Call Main Function:
_main()
```

CHAPTER

SIXTEEN

API MESSAGE STRUCTURE

This section introduces the format used for passing messages to the GridAPPS-D API and how to wrap those messages using the GridAPPSD-Python Library.

16.1 Python Dictionaries VS JSON Strings

One of the confusing aspects of passing messages to and from the GridAPPS-D Platform and APIs is the difference between Python Dictionaries and JSON scripts, which look identical.

JSON is a *serialization format*. That is, JSON is a way of representing structured data in the form of a textual string.

A **Python Dictionary** is a *data structure*. That is, it is a way of storing data in memory that provides certain abilities to the code: in the case of dictionaries, those abilities include rapid lookup and enumeration.

It is possible to convert between the two by importing the JSON library: `import json`. Full documentation of JSON-Python interoperability and usage is available in Python Docs.

Use the `json.dumps()` method to serialize a dictionary as a JSON string. Use the `json.loads()` to import a JSON file and convert it into a dictionary. But the two are not the same: dictionaries are for working with data in your program, and JSON is for storing it or sending it around between programs.

With the GridAPPSD-Python Library, it is possible to pass query arguments as either a python dictionary or as a string. Both approaches will provide the same results.

1) Format API call message as a dictionary

This is the most direct approach, and will be used most often throughout this set of notebook tutorials. The format and structure of the python dictionary is explained in the next section.

[ ]:
```
model_mrid = "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62" # IEEE 13 Node used for all␣
↪example queries

# Format message as python dictionary
message = {
    "requestType": "QUERY_OBJECT_IDS",
    "resultFormat": "JSON",
    "modelId": model_mrid,
    "objectType": "LoadBreakSwitch"
}
```

```
[ ]: # Specify correct topic
     topic = "goss.gridappsd.process.request.data.powergridmodel"

     # Pass API Call to GridAPPS-D Platform
     gapps.get_response(topic, message)
```

2) Format API call message as a string

This approach uses quotations (either `' '` or `" "`) to wrap the API call (identical to the python dictionary) as JSON-formatted text, concatenated into a string.

```
[ ]: model_mrid = "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62" # IEEE 13 Node used for all␣
     ↪example queries

     # Format message as JSON text wrapped as a string
     message = """
     {
         "requestType": "QUERY_OBJECT_IDS",
         "resultFormat": "JSON",
         "modelId": "%s",
         "objectType": "LoadBreakSwitch"
     }
     """ % model_mrid
```

```
[ ]: # Specify correct topic
     topic = "goss.gridappsd.process.request.data.powergridmodel"

     # Pass API Call to GridAPPS-D Platform
     gapps.get_response(topic, message)
```

16.2 Structure of a GridAPPS-D Message

The structure of messages in GridAPPS-D follows that of a Python Dictionary using a data structure that is more generally known as an associative array. An excellent tutorial on advanced usage of the python dictionary structure is available on Real Python.

A dictionary consists of a collection of **key-value pairs**. Each key-value pair maps the **key** to its associated **value**.

- A dictionary is defined by enclosing a comma-separated list of key-value pairs in curly braces (**{ }**).
- A colon (**:**) separates each key from its associated value.
- Square brackets (**[ ]**) are used for a list of values associated to a particular key.
- Additional curly braces (**{ }**) can be used for cases where multiple key-value pairs (e.g. equipment setpoints) are associated with a particular key (e.g. an equipment class).

The general dictionary format used for GridAPPS-D messages is

```
message = {
    "key1": "value1",
    "key2": ["value21", "value22"],
    "key3": {
        "key31": "value31",
        "key32": "value32"
```

(continues on next page)

(continued from previous page)

```
        },
      .
      .
      .
    "key": "value"
}
```

Important: Be sure to pay attention to placement of commas (**,**) at the end of each line. Commas are placed at the end of each line *except* the last line. Incorrect comma placement will result in a syntax exception.

The particular set of key-value pairs for each GridAPPS-D API is covered in detail in Lessons 2.1 through 2.7.

16.3 Parsing Returned Data

After passing an API call, the GridAPPS-D Platform returns a JSON string that is subsequently converted into a python dictionary by the GridAPPSD-Python Library. This section will outline how to parse the data returned.

For this example, we are going to use a simple query from the PowerGrid Model API (covered in Lesson 2.2.) to obtain the details of a piece of equipment using its unique mRID (introduced in the next lesson).

```
[ ]: model_mrid = "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62" # IEEE 13 Node used for all␣
     ↪example queries

     # Specify correct topic
     topic = "goss.gridappsd.process.request.data.powergridmodel"

     message = {
             "modelId": model_mrid,
             "requestType": "QUERY_OBJECT_DICT",
             "resultFormat": "JSON",
             "objectType": "LinearShuntCompensator",
     }

     # Pass API Call to GridAPPS-D Platform
     response = gapps.get_response(topic, message)

     import json
     with open("foo.txt", 'w') as out:
         out.write(json.dumps(response, indent=2))
```

The structure of the python dictionary returned by the API is three key-value pairs for the keys of

- `'data'` – this is the data you requested
- `'responseComplete'` – true or false
- `'id'` – unique id associated with the API response dictionary

A typical API response will the structure below:

```
response = {
    'data': [{'key1': 'value1',
        'key2': ['value21', 'value22']},
        {'key1': 'value1',
```

(continues on next page)

(continued from previous page)

```
        'key2': ['value21', 'value22']}],
    'responseComplete': True,
    'id': '12345678'
}
```

The first step is to filter the dictionary to just the data requested: `response['data']`. The result will be a list object.

Note: *some* API calls will also need to additional filters of `[results][bindings]`. The STOMP Client presented in the next section is very helpful for previewing the structure of the dictionary returned by GridAPPS-D.

```
[ ]: response = gapps.get_response(topic, message)
     response_obj = response['data']
```

As `response_obj` is of the python type `list` rather than `dict`, it is necessary to use numerical indices instead of keys to access the values. A simple `for` loop is very helpful here.

In this example, we want to filter the results to create a list that contains just the name and mRID of the capacitor banks in the model.

```
[ ]: capacitors = []
     for index in response_obj:
         cap_name = index['IdentifiedObject.name']
         cap_mrid = index['id']
         message = dict(name = cap_name,
                        mrid = cap_mrid)
         capacitors.append(message)
```

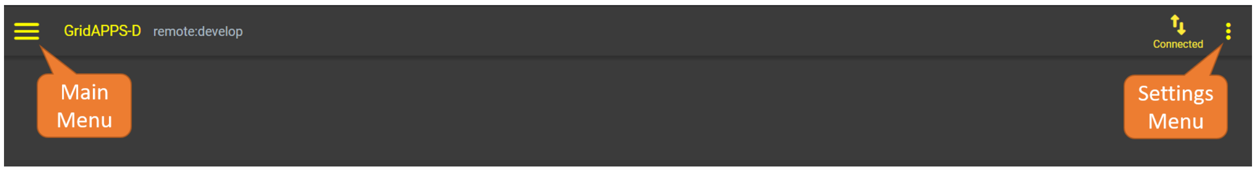
[*Return to Top*]

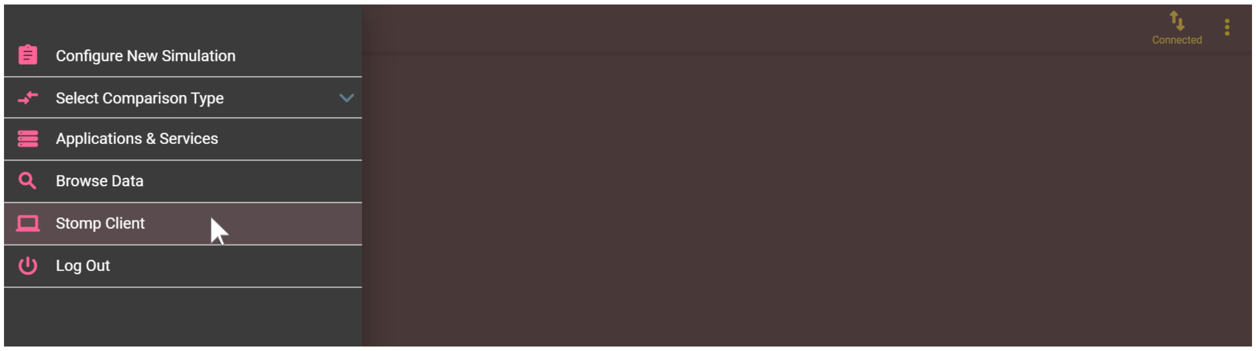
16.4 Using the STOMP Client

The GridAPPS-D Visualization App includes a feature to pass API call messages through the GUI using the Simple Text Oriented Messaging Protocol (STOMP).

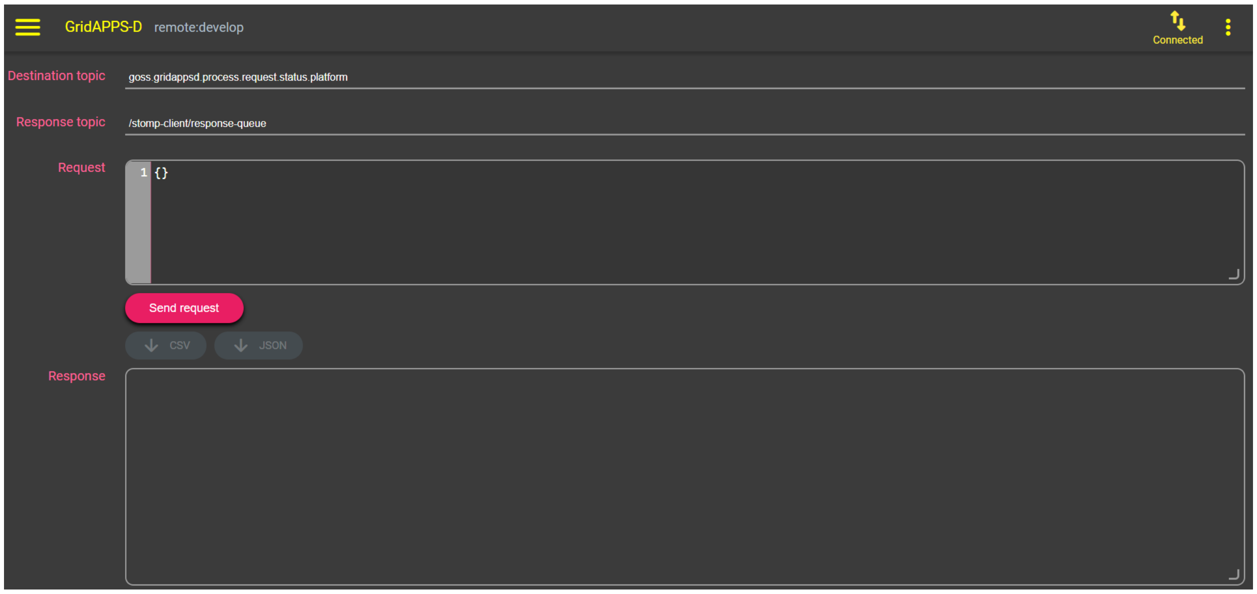
Open the Viz App, which is hosted on localhost:8080 (note: cloud-hosted installations will use the IP address of the server).

Select **Stomp Client** from the main drop-down menu:





This opens the STOMP Client, which can be used to pass a message to any of the GridAPPS-D APIs to preview results or debug the API call message.



Destination topic: goss.gridappsd.process.request.status.platform

Response topic: /stomp-client/response-queue

Request:

```
{}
```

16.4.1 Specifying the Topic

The appropriate GridAPPS-D topic needs to be copied and pasted into the **Destination Topic** box at the top of the window. The topic specifies on which channel the STOMP Client will communicate with the GridAPPS-D Platform and to which API the message needs to be delivered.

A complete list of GridAPPS-D topics was provided in API Communication Channels and will also be provided in context for each of the API calls detailed in subsequent lessons.

IMPORTANT: Remember to remove the python wrapping quotations at the beginning and end of the topic. For example, if the python-wrapped topic was

```
topic = "goss.gridappsd.process.request.data.powergridmodel" # Specify the
topic
```

then the topic that is entered in the Stomp Client **Destination Topic** box is simply

```
goss.gridappsd.process.request.data.powergridmodel
```

IMPORTANT: The GridAPPSD-Python shortcut functions will not work in the STOMP Client. The full text string versions must be used.

16.4.2 Entering the Request Message

The **Request** box accepts an API call message identical to those provided in these notebook lessons.

IMPORTANT: Remember to remove the python wrapping at the beginning and end of the message. For example, if the python-wrapped message was

`message = "{"requestType": "QUERY_MODEL_NAMES", "resultFormat": "JSON"}" # Sample PowerGrid Model API Call`

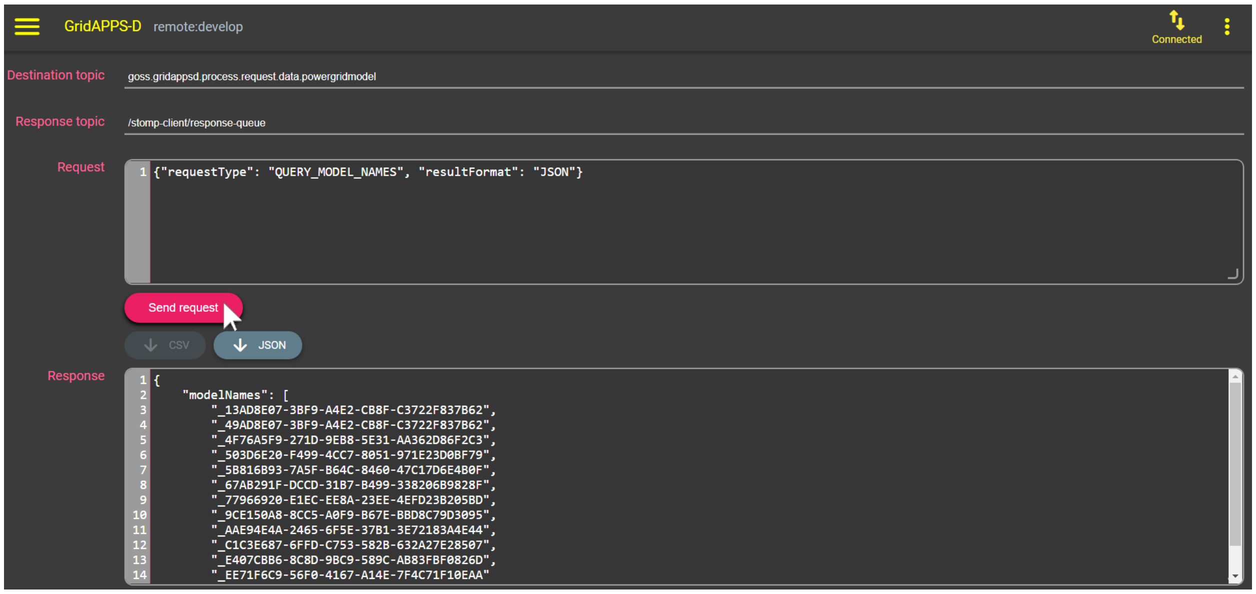
then the message that is entered in the Stomp Client **Request** box is simply

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

The STOMP client will automatically flag any errors in the JSON message.

16.4.3 Submitting a Request

After entering the topic and message, click **Send request** to send the API call to the GridAPPS-D Platform. The response will be displayed in the box below.



goss.gridappsd.process.request.data.powergridmodel

/stomp-client/response-queue

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

```
{
    "modelNames": [
        "_13AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "_4F76A5F9-271D-9EB8-5E31-AA362D86F2C3",
        "_503D6E20-F499-4CC7-8051-971E23D0BF79",
        "_5B816B93-7A5F-B64C-8460-47C17D6E4B0F",
        "_67AB291F-DCCD-31B7-B499-338206B9828F",
        "_77966920-E1EC-EE8A-23EE-4EFD23B205BD",
        "_9CE150A8-8CC5-A0F9-B67E-BBD8C79D3095",
        "_AAE94E4A-2465-6F5E-37B1-3E72183A4E44",
        "_C1C3E687-6FFD-C753-582B-632A27E28507",
        "_E407CBB6-8C8D-9BC9-589C-AB83FBF0826D",
        "_EE71F6C9-56F0-4167-A14E-7F4C71F10EAA"
```

It can be seen that the response from the STOMP Client is identical to that obtained by passing the same topic and message using the GridAPPSD-Python Library:

[ ]:
```python
from gridappsd import GridAPPSD # Import Libraries
gapps = GridAPPSD("('localhost', 61613)", username='system', password='manager') #␣
↪Connect to Platform
topic = "goss.gridappsd.process.request.data.powergridmodel" # Specify correct Topic
message = {
    "requestType": "QUERY_MODEL_NAMES",
    "resultFormat": "JSON"
} # Sample PowerGrid Model API message
gapps.get_response(topic, message) # Pass API call to Platform
```

|GridAPPS-D_narrow.png|

CHAPTER

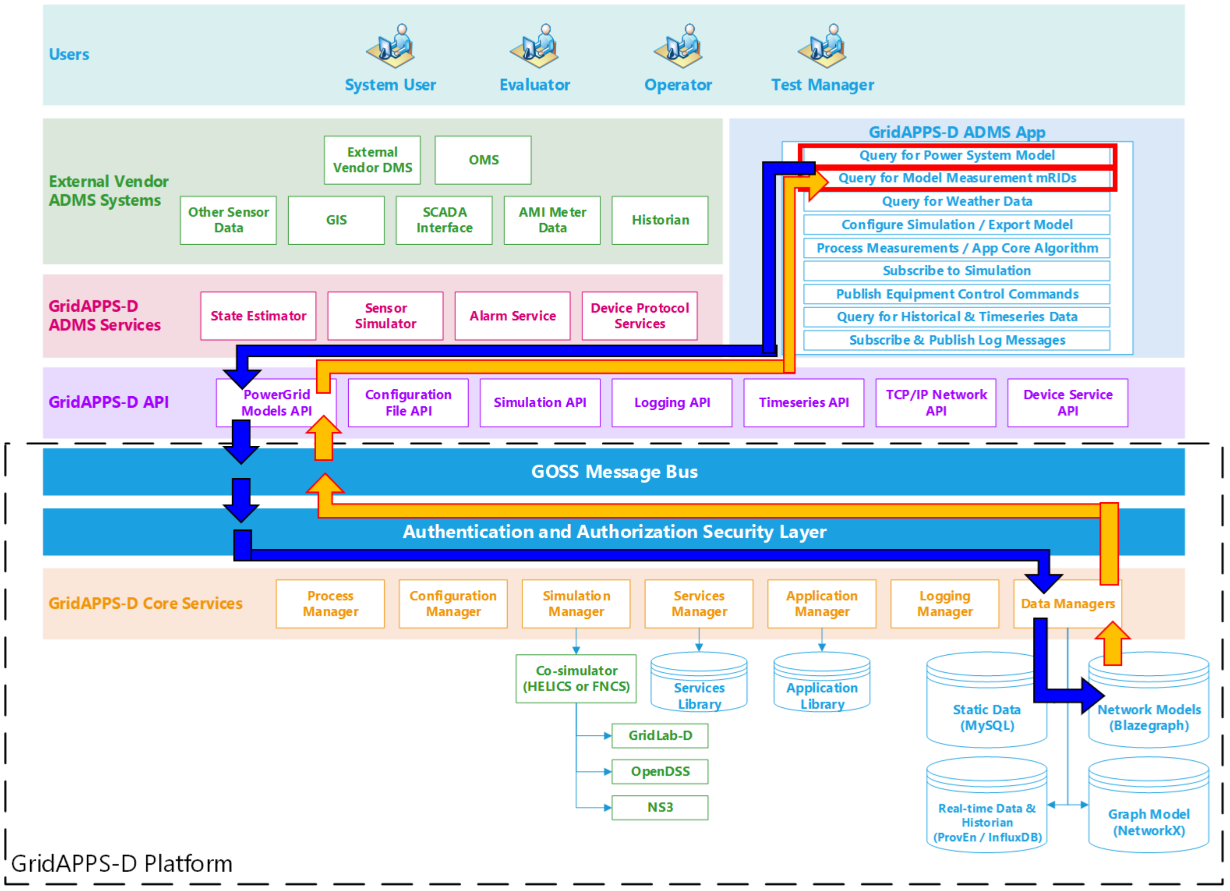
SEVENTEEN

USING THE POWERGRID MODELS API

17.1 Introduction to the PowerGrid Model API

The PowerGrid Models API is used to pull model information from the Blazegraph Database, inlcuding the names, mRIDs, measurements, and nominal values of power system equipment in the feeder (such as lines, loads, switches, transformers, and DERs).

In the Application Components diagram (explained in detail with sample code in *GridAPPS-D Application Structure*), the PowerGrid Models API is used for querying for the power system model and querying for model measurement MRIDs.



17.2 API Syntax Overview

Application passes query to GridAPPS-D Platform

First, the application creates a query message for requesting information about the desired power system components in the format of a JSON string or equivalant Python dictionary object. The syntax of this message is explained in detail below.

The query is sent using `gapps.get_response(topic, message)` with a response expected back from the platform within the specified timeout period.

The application then passes the query through the PowerGrid Models API to the GridAPPS-D Platform, which publishes it to the `goss.gridappsd.process.request.data.powergridmodel` queue channel on the GOSS Message Bus. If the app is authenticated and authorized to pass queries, the query message is delivered to the data managers, which obtain the desired information from the Blazegraph Database.

GridAPPS-D Platform responds to Application query

The data managers then publish the response from the Blazegraph Database to the appropriate queue channel. The PowerGrid Models API then returns the desired information back to the application as a JSON message or equivalant Python dictionary object.

17.2.1 API Communication Channel

All queries passed to the PowerGrid Models API need to use the correct communication channel, which is obtained using the *GridAPPS-D Topics library*.

The PowerGrid Model API uses a `/queue/` channel to pull power system model info from the the Blazegraph Database. The base static string used is `goss.gridappsd.process.request.data.powergridmodel`, which can be called using the `.REQUEST_POWERGRID_DATA` or `.BLAZEGRAPH` methods from the topics library.

When developing in python, it is recommended to use the `.REQUEST_POWERGRID_DATA` method. When using the STOMP client in GridAPPS-D VIZ, it is necessary to use the base static string.

[ ]:
```
from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA
```

17.2.2 Structure of a Query Message

Queries passed to PowerGrid Models API are formatted as python dictionaries or equivalent JSON scripts wrapped as a python string.

```
message = {
    "requestType": "INSERT QUERY HERE",
    "resultFormat": "JSON",
    "modelId": "OPTIONAL INSERT MODEL mRID HERE",
    "objectType": "OPTIONAL INSERT CIM CLASS HERE",
    "objectId": "OPTIONAL INSERT OBJECT mRID HERE",
    "filter": "OPTIONAL INSERT SPARQL FILTER HERE"
}
```

The components of the message are as follows:

- `"requestType":` – Specifies the type of query. Available requestType are listed in the next section.

- `"resultFormat":` – Specifies the format of the response, can be `"JSON"`, `"CSV"`, or `"XML"`. (CAUTION: the PowerGridModel API uses the key *resultFormat*, while the Timeseries API uses the key *reponseFormat*. Using the wrong key for either API will result in a java.lang error.)
- `"modelID":` – Optional. Used to filter the query to only one particular model whose mRID is specified. Be aware of spelling and capitalization differences between JSON query spelling `"modelId"` and Python Library spelling `model_id`.
- `"objectType":` – Optional. Used to filter the query to only one CIM class of equipment. Speciying the *objectID* will override any values specified for *objectType*.
- `"objectID":` – Optional. Used to filter the query to only one object whose mRID is specified. Specifying the *objectID* will override any values specified for *objectType*.
- `"filter":` – Optional. Used to filter the query using a SPARQL filter. SPARQL queries are covered in the next lesson.

The usage of each of these message components are explained in detail with code block examples below.

Important: Be sure to pay attention to placement of commas (**,**) at the end of each JSON line. Commas are placed at the end of each line *except* the last line. Incorrect comma placement will result in a JsonSyntaxException.

All of the queries are passed to the PowerGrid Model API using the `.get_response(topic, message)` method for the GridAPPS-D platform connection variable.

17.2.3 Specifying the requestType

Below are the possible `requestType` strings that are used to specify the type of each query. Executable code block examples are provided for each of the requests in the subsections below.

The first group of *requestType* key-value pairs are for queries for information related to the just the mRIDs of a set of feeders or set of equipment within a particular feeder:

- `"requestType": "QUERY_MODEL_NAMES"` – *Query for the list of all model name mRIDs*
- `"requestType": "QUERY_OBJECT_IDS"` – *Query for a list of all mRIDs for objects of a CIM class*

The second group of *requestType* key-value pairs are for queries for Python dictionaries containing all specifics of a set of feeders or set of equipment within a particular feeder:

- `"requestType": "QUERY_MODEL_INFO"` – *Query for the dictionary of all details for all feeders in Blazegraph*
- `"requestType": "QUERY_OBJECT_DICT"` – *Query for the dictionary of all details for an object using either its *objectType* OR its *objectID**

The third group of *requestType* key-value pairs are for obtaining information about CIM objects and attributes:

- `"requestType": "QUERY_OBJECT_TYPES"` – *Query for the types of CIM classes of objects in the model*
- `"requestType": "QUERY_OBJECT"` – *Query for CIM attributes of an object using its unique mRID*

One of the most important queries is for object measurements. Each piece of equipment has voltage, power, and/or position measurements associated with it. Each measurement has a unique mRID which is different from that of the equipment.

- `"requestType": "QUERY_OBJECT_MEASUREMENTS"` – *Query for all measurement types and mRIDs for an object using either its *objectType* OR its *ObjectID**

The last group of *requestType* key-value pairs are for queries based on SPARQL filters or complete SPARQL queries. Usage of these two *requestType* is made in conjunction with custom SPARQL queries given in Sample SPARQL Queries.

- `"requestType": "QUERY_MODEL"` – Query for all part of a specified model, filtered by object type using a SPARQL filter.
- `"requestType": "QUERY"` – Query using a complete SPARQL query.

17.2.4 CIM Objects Supported by PowerGrid Models API

Below is a list of CIM object classes that can be queried for using the PowerGrid Models API. Other classes and associated attributes need to be queried for using a custom SPARQL query. Sample SPARQL queries for must custom queries can be found in the CIMHub queries.txt file

CIM Classes supported by the PowerGrid Models API

- `ACLineSegment`
- `Breaker`
- `ConnectivityNode`
- `EnergyConsumer`
- `EnergySource`
- `Fuse`
- `LinearShuntCompensator`
- `LoadBreakSwitch`
- `PowerElectronicsConnection`
- `PowerTransformer`
- `Recloser`
- `SynchronousMachine`
- `TransformerTank`

CIM Classes requiring custom SPARQL queries

- `ACLineSegmentPhase`
- `Analog`
- `Asset`
- `BaseVoltage`
- `BatteryUnit`
- `ConcentricNeutralCableInfo`
- `CoordinateSystem`
- `CurrentLimit`
- `Discrete`
- `EnergyConsumerPhase`
- `Feeder`
- `GeographicalRegion`
- `House`

- `IEC61970CIMVersion`
- `LinearShuntCompensatorPhase`
- `LoadResponseCharacteristic`
- `Location`
- `NoLoadTest`
- `OperationalLimitSet`
- `OperationalLimitType`
- `OverheadWireInfo`
- `PerLengthPhaseImpedance`
- `PerLengthSequenceImpedance`
- `PhaseImpedanceData`
- `PhotovoltaicUnit`
- `PositionPoint`
- `PowerElectronicsConnectionPhase`
- `PowerTransformerEnd`
- `PowerTransformerInfo`
- `RatioTapChanger`
- `RegulatingControl`
- `ShortCircuitTest`
- `SubGeographicalRegion`
- `Substation`
- `SwitchPhase`
- `TapChangerControl`
- `TapChangerInfo`
- `TapeShieldCableInfo`
- `Terminal`
- `TopologicalIsland`
- `TopologicalNode`
- `TransformerCoreAdmittance`
- `TransformerEndInfo`
- `TransformerMeshImpedance`
- `TransformerTankEnd`
- `TransformerTankInfo`
- `VoltageLimit`
- `WirePosition`
- `WireSpacingInfo`

17.3 Querying for Model mRIDS

Every piece of equipment has a unique mRID, as explained in *Intro to Common Information Model*. These mRIDs are used to identify and communicate with equipment in GridAPPS-D.

The set of queries below provide just the mRIDs of equipment matching the query filters. If the full details of equipment are desired (e.g. name and properties), use the *Query for Equipment Dictionaries* API calls in the next section.

This section outlines the pre-built JSON queries that can be passed to the PowerGrid Model API to obtain mRIDs and other information for all models and feeders stored in the Blazegraph Database.

17.3.1 Query for mRIDs of all Models

This query obtains a list of all the model MRIDs stored in the Blazegraph database.

Query requestType:

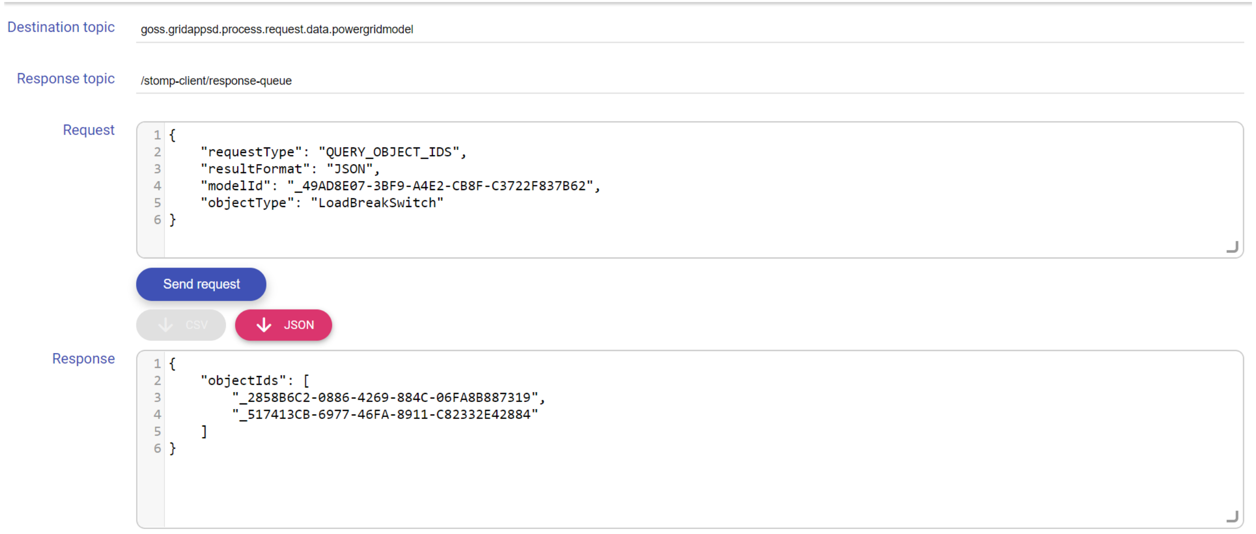
- `"requestType": "QUERY_MODEL_NAMES"`

Allowed parameters:

- `"resultFormat":` – "XML" / "JSON" / "CSV" – Optional. Will return results as a list in the format selected.

[ ]:
```python
from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA

message = {
    "requestType": "QUERY_MODEL_NAMES",
    "resultFormat": "JSON"
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:



Destination topic: goss.gridappsd.process.request.data.powergridmodel

Response topic: /stomp-client/response-queue

Request

```
{
    "requestType": "QUERY_OBJECT_IDS",
    "resultFormat": "JSON",
    "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
    "objectType": "LoadBreakSwitch"
}
```

Response

```
{
    "objectIds": [
        "_2858B6C2-0886-4269-884C-06FA8B887319",
        "_517413CB-6977-46FA-8911-C82332E42884"
    ]
}
```

Python Library Method

The GridAPPSD-Python library contains a pre-built method for obtaining the mRIDs of all models, providing identical results to those obtained above.

The `.query_model_names` method is associated with the GridAPPSD connection object and returns a list of all the CIM XML classes of objects present in the Blazegraph for a particular model or all models in the database.

```
[ ]: gapps.query_model_names()
```

17.3.2 Query for mRIDs of Objects in a Feeder

This query is used to obtain all the mRIDs of objects of a particular CIM class in the feeder.

Query responseType is

- `"requestType": "QUERY_OBJECT_IDS"`

Allowed parameters are:

- `"modelId"`: "model name mRID" – When specified it searches against that model, if empty it will search against all models
- `"objectType"`: "CIM Class" – Optional. Specifies the type of objects you wish to return details for.
- `"resultFormat"`: – "XML" / "JSON" / "CSV" – Will return results as a list in the format selected.

Within a particular feeder, it is possible to query for objects of all the CIM classes supported by PowerGrid Models API (discussed above in *CIM Objects Supported by the API*). Other types of equipment require custom SPARQL queries.

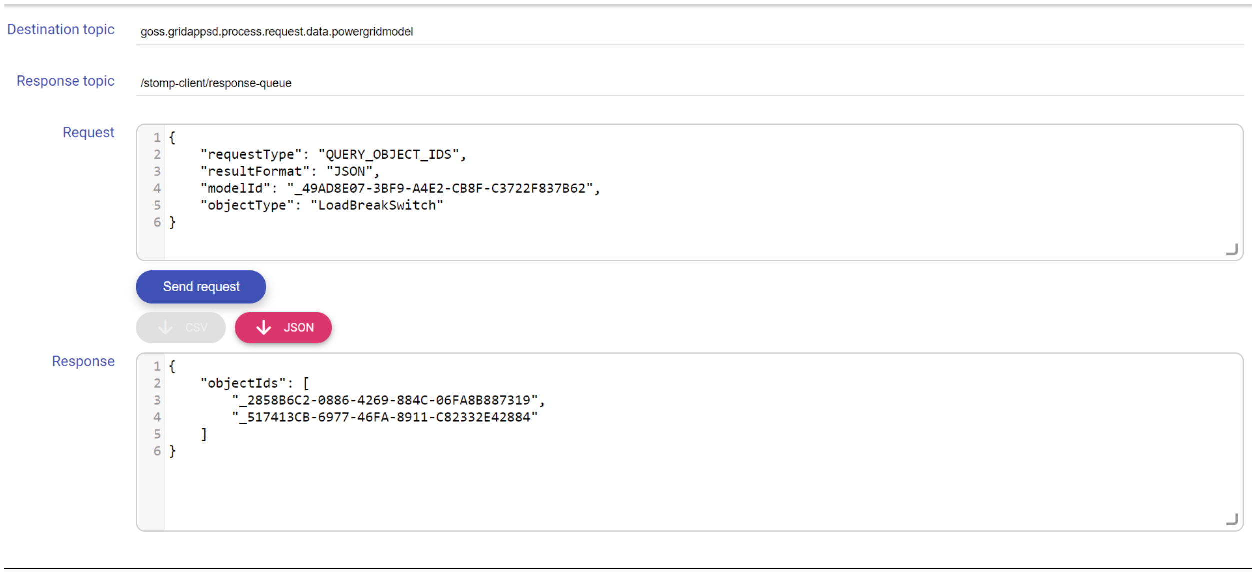
Note that the RDF URI is not included in the query, only the name of the class, such as `"objectType": "ACLineSegment"` or `"objectType": "LoadBreakSwitch"`.

```
[ ]: from gridappsd import topics as t
     topic = t.REQUEST_POWERGRID_DATA

     message = {
         "requestType": "QUERY_OBJECT_IDS",
         "modelId": model_mrid,
         "objectType": "LoadBreakSwitch",
         "resultFormat": "JSON"
     }

     gapps.get_response(topic, message)
```

It is possible to then filter the response to just a list of the mRIDs:

```
[ ]: response_obj = gapps.get_response(topic, message)
     switch_mrids = response_obj['data']['objectIds']
     print(switch_mrids)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:


Destination topic
goss.gridappsd.process.request.data.powergridmodel
Response topic
/stomp-client/response-queue
Request
Send request
CSV
JSON
Response

```
{
    "requestType": "QUERY_OBJECT_IDS",
    "resultFormat": "JSON",
    "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
    "objectType": "LoadBreakSwitch"
}
```

```
{
    "objectIds": [
        "_2858B6C2-0886-4269-884C-06FA8B887319",
        "_517413CB-6977-46FA-8911-C82332E42884"
    ]
}
```

17.4 Querying for Equipment Dictionaries

This section outlines the pre-built JSON queries that can be passed to the PowerGrid Model API to obtain mRIDs and other information for a particular object or a class of objects for one or more feeders stored in the Blazegraph Database.

All of the examples in this section use the IEEE 13 node model.

```
[ ]: model_mrid = "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62" # IEEE 13 Node used for all␣
     ↪example queries
```

17.4.1 Query for Dictionary of all Models

This query returns a list of names and MRIDs for all models, substations, subregions, and regions for all available feeders stored in the Blazegraph database.

Query requestType:

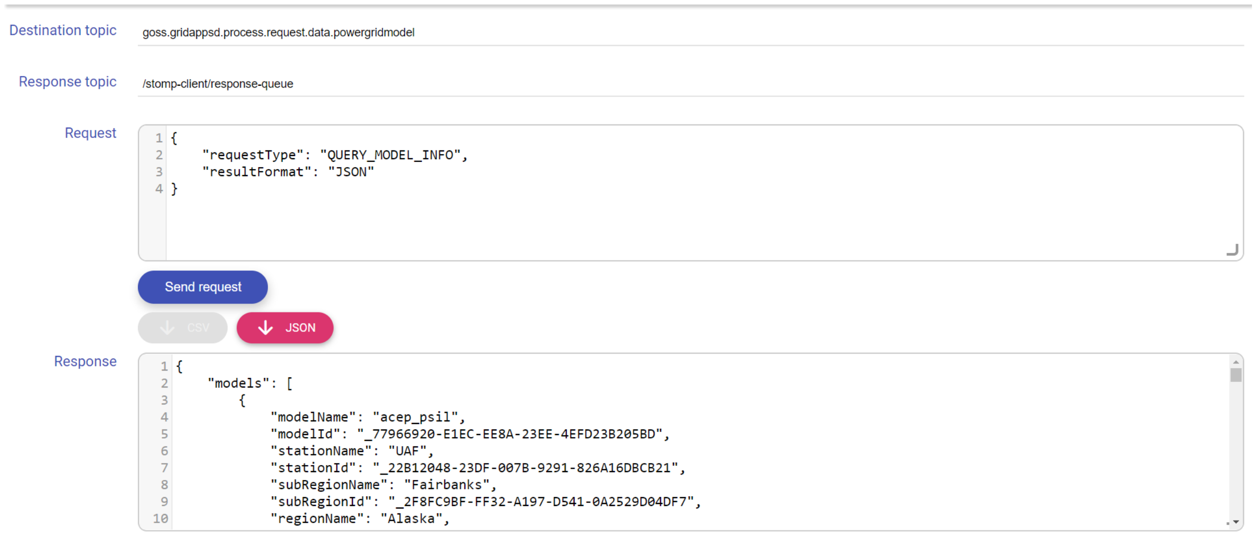
- `"requestType": "QUERY_MODEL_INFO"`

Allowed parameters:

- `"resultFormat":` – "XML" / "JSON" / "CSV" – Will return results as a list in the format selected.

```
[ ]: from gridappsd import topics as t
     topic = t.REQUEST_POWERGRID_DATA

     message = {
         "requestType": "QUERY_MODEL_INFO",
         "resultFormat": "JSON"
     }

     gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:


Destination topic
goss.gridappsd.process.request.data.powergridmodel
Response topic
/stomp-client/response-queue
Request
Send request
CSV
JSON
Response

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

```
{
    "models": [
        {
            "modelName": "acep_psil",
            "modelId": "_77966920-E1EC-EE8A-23EE-4EFD23B205BD",
            "stationName": "UAF",
            "stationId": "_22B12048-23DF-007B-9291-826A16DBCB21",
            "subRegionName": "Fairbanks",
            "subRegionId": "_2F8FC9BF-FF32-A197-D541-0A2529D04DF7",
            "regionName": "Alaska",
```

Python Library Method

The GridAPPSD-Python library contains a pre-built method for requesting the dictionary of all models.

The `.query_model_info()` method is associated with the GridAPPSD connection object and runs the same query as above:

[ ]:
```
gapps.query_model_info()
```

17.4.2 Query for Object Dictionary

This query returns a python dictionary of all the equipment attributes and mRIDs. The query can be for 1) all objects of a particular `objectType` or 2) for those connected to a particular object based on the `objectId`.

If neither `objectType` or `objectId` is provided, the query will provide all measurements belonging to the model. Query responseType is

- `"requestType": "QUERY_OBJECT_DICT"`

Allowed parameters are:

- `"modelId"`: "model name mRID" – When specified it searches against that model, if empty it will search against all models
- `"objectId"`: "object mRID" – Optional. Specifies the type of objects you wish to return details for.
- `"objectType"`: "CIM Class" – Optional. Specifies the type of objects you wish to return details for.
- `"resultFormat"`: "XML" / "JSON" / "CSV" – Will return results as a list in the format selected.

Speciying the `objectID` will override any values specified for `objectType`.

Example 1: Querying for model dictionary for an objectID

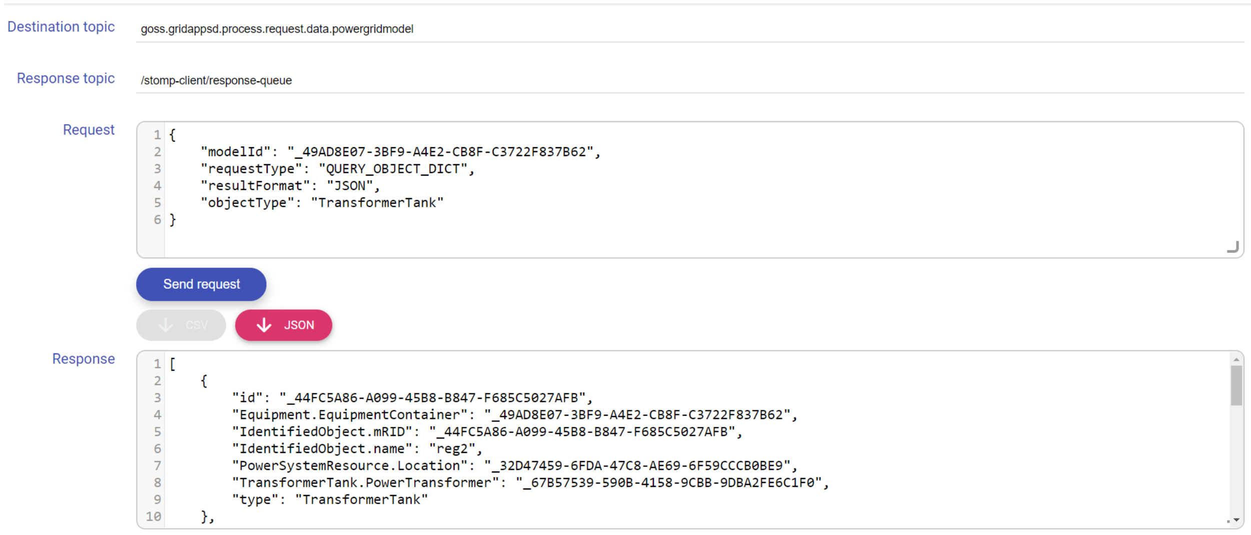
[ ]:
```
from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA

message = {
    "requestType": "QUERY_OBJECT_DICT",
```

(continues on next page)

(continued from previous page)

```
    "modelId": model_mrid,
    "objectId": switch_mrids[1],
    "resultFormat": "JSON"
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:



Destination topic goss.gridappsd.process.request.data.powergridmodel

Response topic /stomp-client/response-queue

Request

```
{
    "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
    "requestType": "QUERY_OBJECT_DICT",
    "resultFormat": "JSON",
    "objectType": "TransformerTank"
}
```

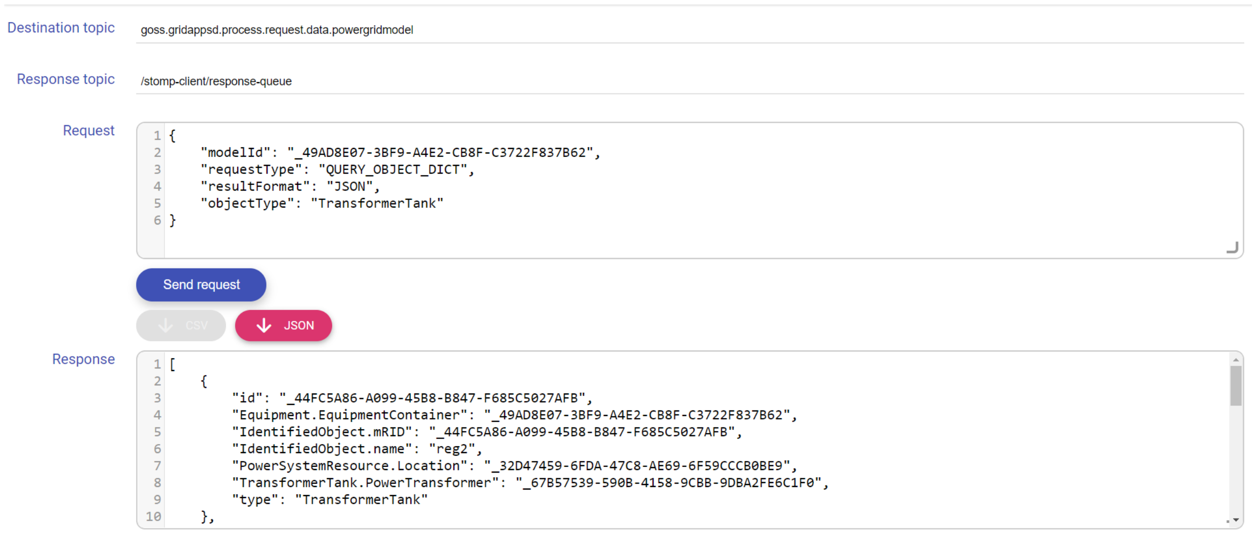
Response

```
[
    {
        "id": "_44FC5A86-A099-45B8-B847-F685C5027AFB",
        "Equipment.EquipmentContainer": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "IdentifiedObject.mRID": "_44FC5A86-A099-45B8-B847-F685C5027AFB",
        "IdentifiedObject.name": "reg2",
        "PowerSystemResource.Location": "_32D47459-6FDA-47C8-AE69-6F59CCCB0BE9",
        "TransformerTank.PowerTransformer": "_67B57539-590B-4158-9CBB-9DBA2FE6C1F0",
        "type": "TransformerTank"
    },
```

Example 2: Querying for model dictionary for an objectType

[ ]:
```
from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA

message = {
    "requestType": "QUERY_OBJECT_DICT",
    "modelId": model_mrid,
    "objectType": "TransformerTank",
    "resultFormat": "JSON"
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:



Destination topic: goss.gridappsd.process.request.data.powergridmodel

Response topic: /stomp-client/response-queue

Request:

```
{
    "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
    "requestType": "QUERY_OBJECT_DICT",
    "resultFormat": "JSON",
    "objectType": "TransformerTank"
}
```

Response:

```
[
    {
        "id": "_44FC5A86-A099-45B8-B847-F685C5027AFB",
        "Equipment.EquipmentContainer": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "IdentifiedObject.mRID": "_44FC5A86-A099-45B8-B847-F685C5027AFB",
        "IdentifiedObject.name": "reg2",
        "PowerSystemResource.Location": "_32D47459-6FDA-47C8-AE69-6F59CCCB0BE9",
        "TransformerTank.PowerTransformer": "_67B57539-590B-4158-9CBB-9DBA2FE6C1F0",
        "type": "TransformerTank"
    },
```

Python Library Method

The GridAPPSD-Python library contains a pre-built method for requesting the object dictionary

The `.query_object_dictionary(model_id, object_type, object_id)` method is associated with the GridAPPSD connection object and runs the same query as above to return the object dictionary

```
[ ]: gapps.query_object_dictionary(model_id = model_mrid, object_id = switch_mrids[1])
```

```
[ ]: gapps.query_object_dictionary(model_id = model_mrid, object_type = "TransformerTank")
```

17.5 Querying for CIM Attributes

17.5.1 Query for CIM Classes of Objects in Model

This query is used to query for a list of all the CIM XML classes of objects present in the Blazegraph for a particular model or all models in the database.

Query requestType is

- `"requestType": "QUERY_OBJECT_TYPES"`

Allowed parameters are

- `"modelId"`: "model name mRID" – Optional. Searches only the particular model identified by the given unique mRID
- `"resultFormat"`: – "XML" / "JSON" / "CSV" – Will return results as a list in the format selected.

1) Query entire Blazegraph database

Omit the `"modelId"` parameter to search the entire blazegraph database.

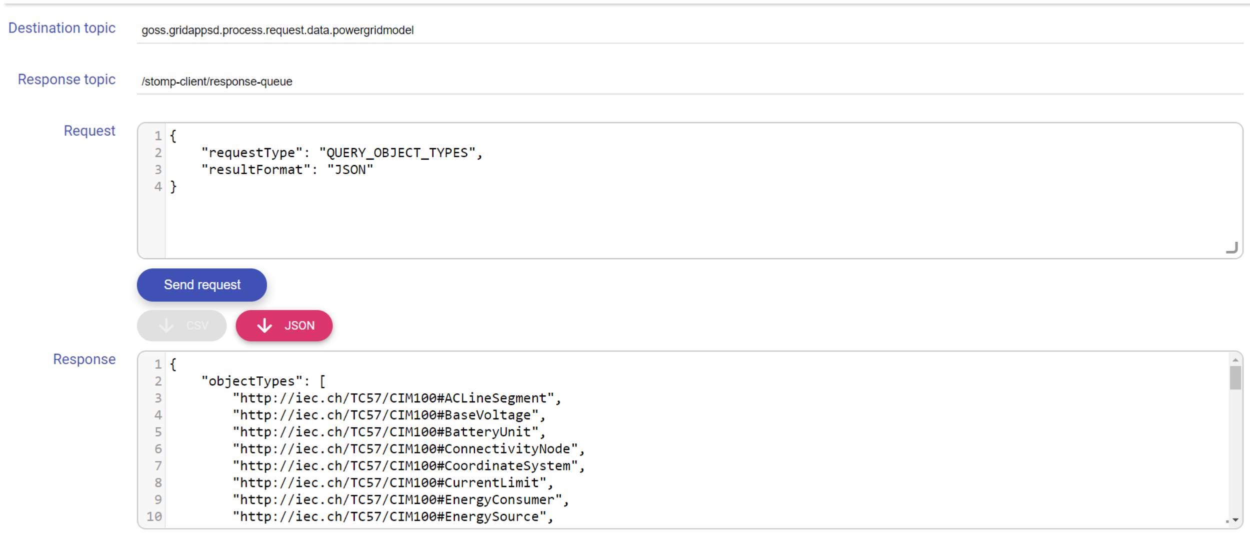
```
[ ]: from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA

message = {
```

(continues on next page)

(continued from previous page)

```
    "requestType": "QUERY_OBJECT_TYPES",
    "resultFormat": "JSON"
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:



goss.gridappsd.process.request.data.powergridmodel

/stomp-client/response-queue

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

```
{
    "objectTypes": [
        "http://iec.ch/TC57/CIM100#ACLineSegment",
        "http://iec.ch/TC57/CIM100#BaseVoltage",
        "http://iec.ch/TC57/CIM100#BatteryUnit",
        "http://iec.ch/TC57/CIM100#ConnectivityNode",
        "http://iec.ch/TC57/CIM100#CoordinateSystem",
        "http://iec.ch/TC57/CIM100#CurrentLimit",
        "http://iec.ch/TC57/CIM100#EnergyConsumer",
        "http://iec.ch/TC57/CIM100#EnergySource",
```

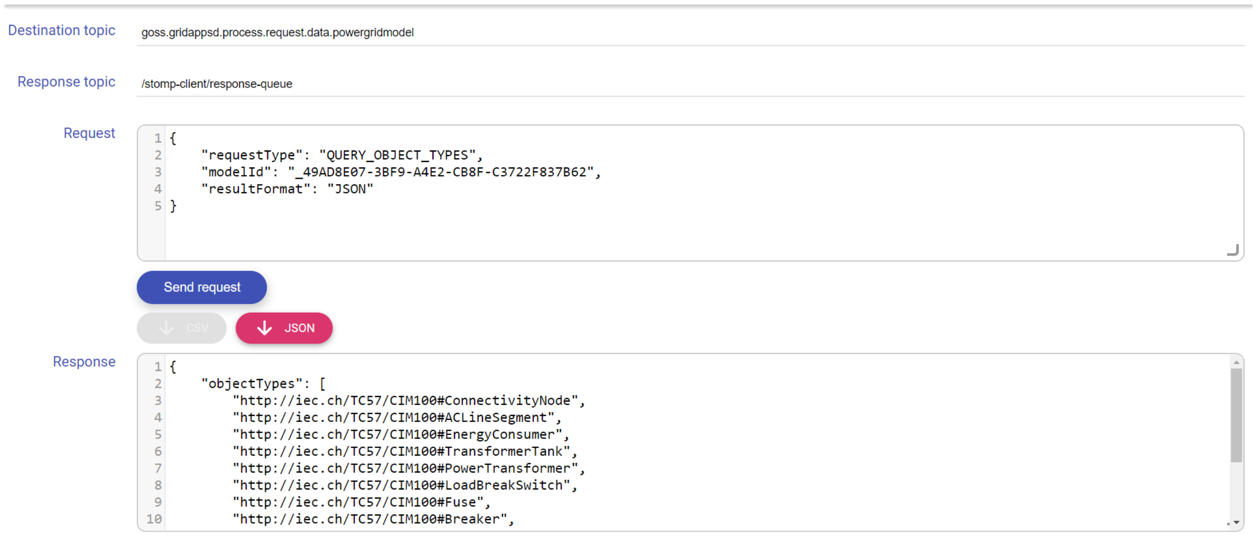
2) Query for only a particular model

Specify the model MRID as a python string and pass it as a parameter to the method to return only the CIM classes of objects in that particular model.

Be aware of spelling and capitalization differences between JSON query spelling `"modelId"` and Python Library spelling `model_id`.

```
[ ]: from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA

message = {
    "requestType": "QUERY_OBJECT_TYPES",
    "modelId": model_mrid,
    "resultFormat": "JSON"
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:



goss.gridappsd.process.request.data.powergridmodel

/stomp-client/response-queue

```
{
    "requestType": "QUERY_OBJECT_TYPES",
    "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
    "resultFormat": "JSON"
}
```

```
{
    "objectTypes": [
        "http://iec.ch/TC57/CIM100#ConnectivityNode",
        "http://iec.ch/TC57/CIM100#ACLineSegment",
        "http://iec.ch/TC57/CIM100#EnergyConsumer",
        "http://iec.ch/TC57/CIM100#TransformerTank",
        "http://iec.ch/TC57/CIM100#PowerTransformer",
        "http://iec.ch/TC57/CIM100#LoadBreakSwitch",
        "http://iec.ch/TC57/CIM100#Fuse",
        "http://iec.ch/TC57/CIM100#Breaker",
```

Python Library Method

The GridAPPSD-Python library contains a pre-built method for obtaining all the CIM classes in the model, providing identical results to those obtained above.

The `.query_object_types` method is associated with the GridAPPSD connection object and returns a list of all the CIM XML classes of objects present in the Blazegraph for a particular model or all models in the database.

Allowed parameters are

- model_id (optional) - when specified, it searches only the particular model identified by the given unique mRID

1) Query entire Blazegraph database

Leave the arguments blank to search all models in the Blazegraph database

```
[ ]: gapps.query_object_types()
```

2) Query for only a particular model

Specify the model MRID as a python string and pass it as a parameter to the method to return only the CIM classes of objects in that particular model

```
[ ]: gapps.query_object_types(model_mrid)
```

17.5.2 Query for CIM Attributes of an Object

This query is used to obtain all the attributes and mRIDs of those attributes for a particular object whose mRID is specified.

Query responseType is

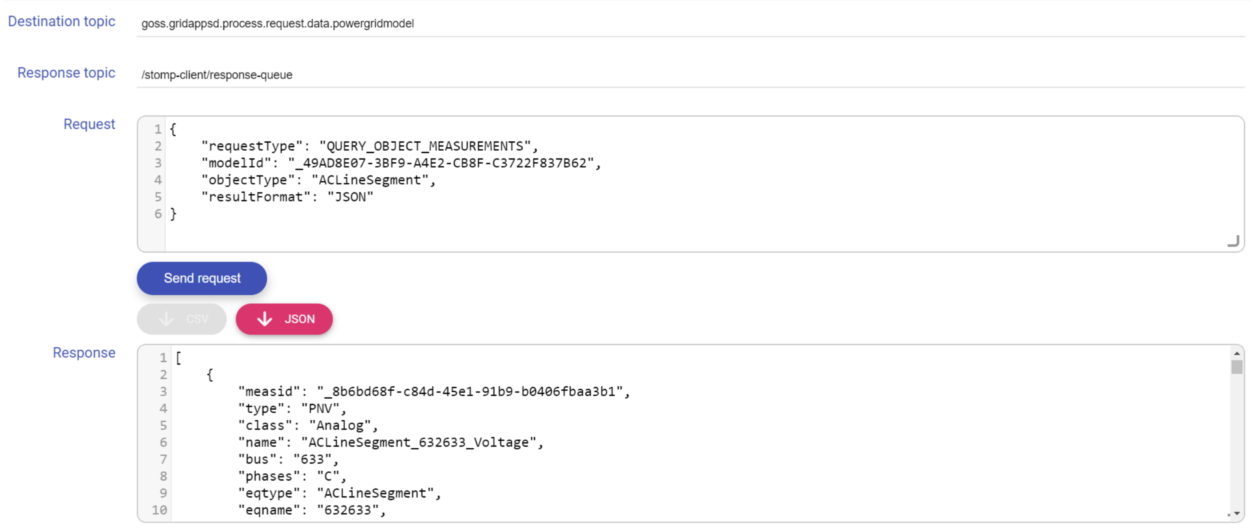
- `"requestType": "QUERY_OBJECT"`

Allowed parameters are:

- `"modelId"`: "model name mRID" – When specified it searches against that model, if empty it will search against all models
- `"objectId"`: "object mRID" – Optional. Specifies the type of objects you wish to return details for.
- `"resultFormat"`: – "XML" / "JSON" / "CSV" – Will return results as a list in the format selected.

```
[ ]: from gridappsd import topics as t
     topic = t.REQUEST_POWERGRID_DATA

     object_mrid = "_2858B6C2-0886-4269-884C-06FA8B887319"

     message = {
         "requestType": "QUERY_OBJECT",
         "resultFormat": "JSON",
         "modelId": model_mrid,
         "objectId": object_mrid
     }

     gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:


Destination topic
goss.gridappsd.process.request.data.powergridmodel
Response topic
/stomp-client/response-queue
Request
Send request
CSV
JSON
Response

```
{
    "requestType": "QUERY_OBJECT_MEASUREMENTS",
    "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
    "objectType": "ACLineSegment",
    "resultFormat": "JSON"
}
```

```
[
    {
        "measid": "_8b6bd68f-c84d-45e1-91b9-b0406fbaa3b1",
        "type": "PNV",
        "class": "Analog",
        "name": "ACLineSegment_632633_Voltage",
        "bus": "633",
        "phases": "C",
        "eqtype": "ACLineSegment",
        "eqname": "632633",
```

Python Library Method

The GridAPPSD-Python library contains a pre-built method for obtaining the mRIDs of all models, providing identical results to those obtained above.

The `.query_object` method is associated with the GridAPPSD connection object and returns a list of all the CIM XML classes of objects present in the Blazegraph for a particular model or all models in the database.

```
[ ]: model_mrid = "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62" # IEEE 13 Node used for all␣
     ↪example queries
     object_mrid = "_2858B6C2-0886-4269-884C-06FA8B887319"

     gapps.query_object(model_mrid, object_mrid)
```

17.6 Querying for Object Measurements

17.6.1 Object mRIDs vs Measurement mRIDs

A key concept in GridAPPS-D and CIM XML power system models is the difference between the object mRID of a piece of equipment and multiple measurement mRIDs associated with its control settings and power flow values.

Measurements differ from the state variables (e.g. those obtained from State Estimator or a power flow calculation) in that the values are measured here and not calculated or estimated. Each Measurement is associated to a *PowerSystem-Resource*, 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 the following currently suppported:

- **PNV –** Phase to Neutral Voltage
- **VA –** Volt-Amperes (apparent power)
- **A –** Amperes (current)
- **POS –** Position for switches and transformer taps

Each measurement object 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.

17.6.2 Querying for Measurements

This query returns details for the measurements within a model. The query can be for 1) all objects of a particular `objectType` or 2) for those connected to a particular object based on the `objectId`.

If neither `objectType` or `objectId` is provided, the query will provide all measurements belonging to the model. Query responseType is

- `"requestType": "QUERY_OBJECT_MEASUREMENTS"`

Allowed parameters are:

- `"modelId"`: "model name mRID" – When specified it searches against that model, if empty it will search against all models
- `"objectId"`: "object mRID" – Optional. Specifies the type of objects you wish to return details for.
- `"objectType"`: "CIM Class" – Optional. Specifies the type of objects you wish to return details for.
- `"resultFormat"`: "XML" / "JSON" / "CSV" – Will return results as a list in the format selected.

Speciying the `objectID` will override any values specified for `objectType`.

Example 1: Querying for all measurements for an objectID

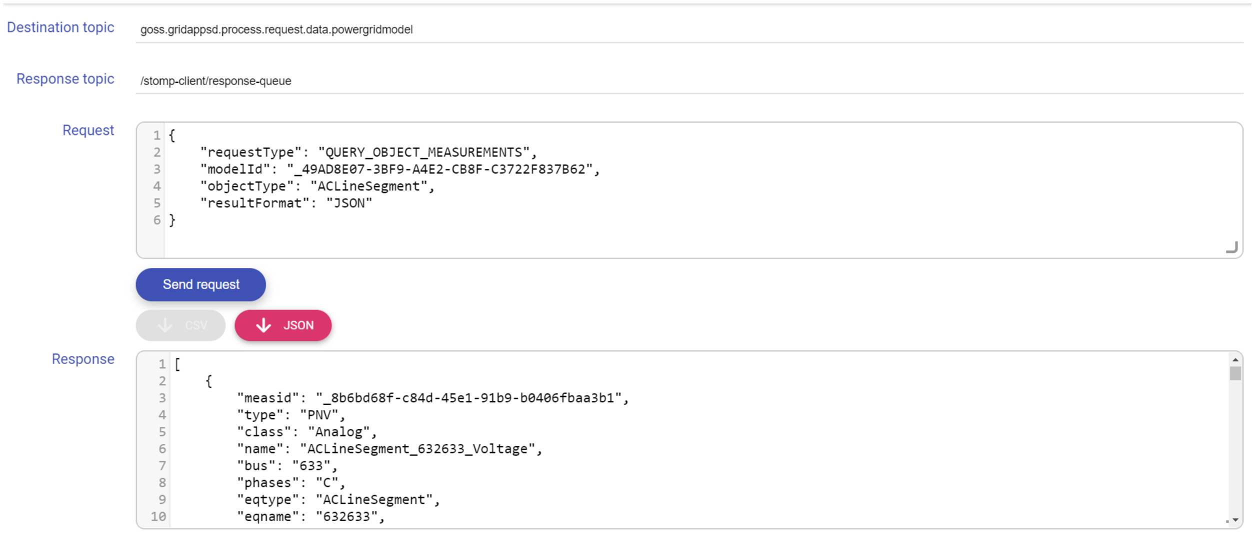
[ ]:
```
from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA

message = {
    "requestType": "QUERY_OBJECT_MEASUREMENTS",
```

(continues on next page)

(continued from previous page)

```
    "modelId": model_mrid,
    "objectId": switch_mrids[1],
    "resultFormat": "JSON"
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:



goss.gridappsd.process.request.data.powergridmodel

/stomp-client/response-queue

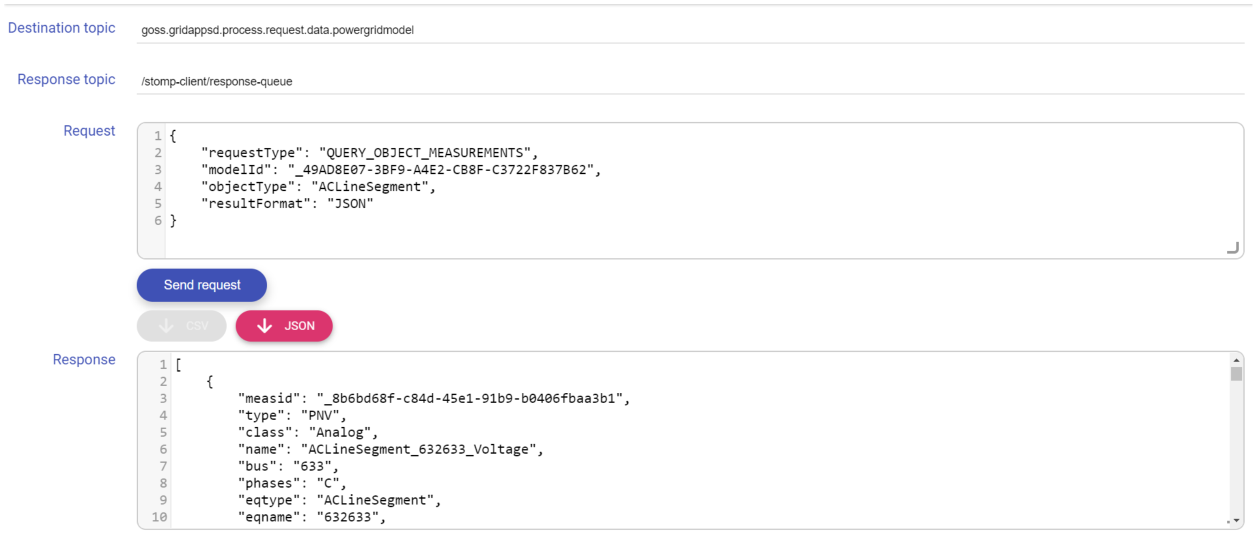
```
{
    "requestType": "QUERY_OBJECT_MEASUREMENTS",
    "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
    "objectType": "ACLineSegment",
    "resultFormat": "JSON"
}
```

```
[
    {
        "measid": "_8b6bd68f-c84d-45e1-91b9-b0406fbaa3b1",
        "type": "PNV",
        "class": "Analog",
        "name": "ACLineSegment_632633_Voltage",
        "bus": "633",
        "phases": "C",
        "eqtype": "ACLineSegment",
        "eqname": "632633",
```

Example 2: Querying for all measurements for an objectType

```
[ ]: from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA

message = {
    "requestType": "QUERY_OBJECT_MEASUREMENTS",
    "modelId": model_mrid,
    "objectType": "ACLineSegment",
    "resultFormat": "JSON"
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:



goss.gridappsd.process.request.data.powergridmodel

/stomp-client/response-queue

```
{
    "requestType": "QUERY_OBJECT_MEASUREMENTS",
    "modelId": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
    "objectType": "ACLineSegment",
    "resultFormat": "JSON"
}
```

```
[
    {
        "measid": "_8b6bd68f-c84d-45e1-91b9-b0406fbaa3b1",
        "type": "PNV",
        "class": "Analog",
        "name": "ACLineSegment_632633_Voltage",
        "bus": "633",
        "phases": "C",
        "eqtype": "ACLineSegment",
        "eqname": "632633",
```

17.6.3 Filtering Returned Data

After receiving the python dictionary of measurements, it will be necessary to parse it to inlcude just the desired set of measurements. This is done using the method presented in *Parsing Returned Data*

[ ]:
```
# Create query message to obtain measurement mRIDs for all switches
message = {
    "modelId": model_mrid,
    "requestType": "QUERY_OBJECT_MEASUREMENTS",
    "resultFormat": "JSON",
    "objectType": "LoadBreakSwitch"
}

# Pass query message to PowerGrid Models API
response_obj = gapps.get_response(t.REQUEST_POWERGRID_DATA, message)
measurements_obj = response_obj["data"]

# Switch position measurements (Pos)
Pos_obj = [k for k in measurements_obj if k['type'] == 'Pos']

# Switch terminal phase-neutral-voltage measurements (PNV)
PNV_obj = [k for k in measurements_obj if k['type'] == 'PNV']

# Switch volt-ampere apparent power measurements (VA)
VA_obj = [k for k in measurements_obj if k['type'] == 'VA']

# Switch current measurements (A)
A_obj = [k for k in measurements_obj if k['type'] == 'A']
```

17.7 Querying with a Custom SPARQL String

This section outlines how the PowerGrid Models API can be used to pass custom SPARQL queries for CIM objects and properties that do not have pre-built JSON string queries.

17.7.1 Query using a SPARQL filter

This query returns a dictionary of all objects matching the particular filter. The query can be for 1) all objects of a particular `objectType` or 2) those with attributes corresponding to a particular SPARQL filter, or both.

If neither `objectType` or `filter` is specified, the query will provide all objects belonging to the model.

Query responseType is

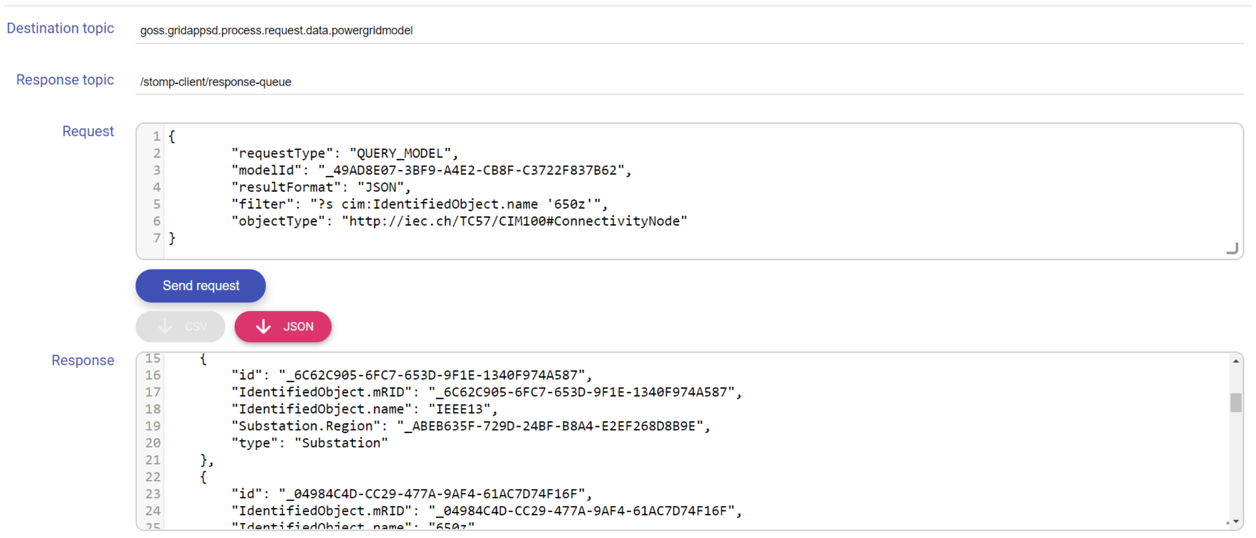
- `"requestType": "QUERY_MODEL"`

Allowed parameters are:

- `"modelId"`: "model name mRID" – Optional. When specified it searches against that model, if empty it will search against all models
- `"objectType"`: "CIM Class" – Optional. Specifies the type of objects you wish to return details for.
- `"filter"`: "SPARQL triple" – Optional. Applies the SPARQL triple filter to the query results
- `"resultFormat"`: "XML" / "JSON" / "CSV" – Will return results as a list in the format selected.

[ ]:
```
from gridappsd import topics as t
topic = t.REQUEST_POWERGRID_DATA

message = {
        "requestType": "QUERY_MODEL",
        "modelId": model_mrid,
        "resultFormat": "JSON",
        "filter": "?s cim:IdentifiedObject.name '650z'",
        "objectType": "http://iec.ch/TC57/CIM100#ConnectivityNode"
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.powergridmodel` topic and the same message without the python wrapping:



goss.gridappsd.process.request.data.powergridmodel

/stomp-client/response-queue

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

```
    {
        "id": "_6C62C905-6FC7-653D-9F1E-1340F974A587",
        "IdentifiedObject.mRID": "_6C62C905-6FC7-653D-9F1E-1340F974A587",
        "IdentifiedObject.name": "IEEE13",
        "Substation.Region": "_ABEB635F-729D-24BF-B8A4-E2EF268D8B9E",
        "type": "Substation"
    },
    {
        "id": "_04984C4D-CC29-477A-9AF4-61AC7D74F16F",
        "IdentifiedObject.mRID": "_04984C4D-CC29-477A-9AF4-61AC7D74F16F",
        "IdentifiedObject.name": "650z"
```

Python Library Method

The GridAPPSD-Python library contains a pre-built method for passing very simple queries. The SPARQL filter key is not supported.

The `.query_model(model_id, object_type, object_id)` method is associated with the GridAPPSD connection object and runs the generic SPARQL query on the Blazegraph database. `object_type` uses the full `http://iec.ch/TC57/CIM100#` prefix. `object_id` is the mRID of the desired object.

```
[ ]: gapps.query_model(model_id = model_mrid, object_type = "http://iec.ch/TC57/CIM100
     ↪#ConnectivityNode")
```

```
[ ]: gapps.query_model(model_id = model_mrid, object_id = "_7BEDDADD-0A14-429F-8601-
     ↪9EA8B892CA6E")
```

17.7.2 Query using a Generic SPARQL Query

This query is used to pass a generic SPARQL query to the Blazegraph database.

Query responseType is

- `"requestType": "QUERY"`

Allowed parameters are:

- `"modelId"`: "model name mRID" – Optinal. When specified it searches against that model, if empty it will search against all models
- `"queryString"`: "SPARQL query text" – Applies the SPARQL triple filter to the query results
- `"resultFormat"`: "XML" / "JSON" / "CSV" – Will return results as a list in the format selected.

```
[ ]: from gridappsd import topics as t
     topic = t.REQUEST_POWERGRID_DATA

     message = {
         "requestType": "QUERY",
         "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}",
```

(continues on next page)

(continued from previous page)

```
    "resultFormat": "JSON"
}

gapps.get_response(topic, message)
```

Although it is possible to check these queries using the STOMP client, it is recommended to validate generic SPARQL queries using the Blazegraph Workbench hosted on localhost:8889/bigdata/#query. The STOMP client contains a JSON validator which may struggle with the line breaks included in SPARQL queries.

Python Library Method

The GridAPPSD-Python library contains a pre-built method for passing generic SPARQL queries

The `.query_data(query, timeout)` method is associated with the GridAPPSD connection object and runs the generic SPARQL query on the Blazegraph database.

```
[ ]: sparql_message = "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}"

     gapps.query_data(query = sparql_message, timeout = 60)
```

17.8 Available Models in Default Installation

The GridAPPS-D Platform comes by default loaded with several distribution feeders ranging in size from 13 to 9500 nodes. Each feeder and recommended usage cases are summarized below.

Name	Features	Houses	Buses	Nodes	Branches	Load	Origin
ACEP_PSIL	480-volt microgrid with PV, wind and diesel	No	8	24	13	0.28	UAF
EPRI_DPV_J1	1800 kW PV in 11 locations	No	3434	4245	4901	9.69	EPRI DPV
IEEE13	Added CIM sampler	No	22	57	51	3.44	IEEE (mod)
IEEE13_Assets	Uses line spacings and wires	No	16	41	40	3.58	IEEE (mod)
IEEE13_OCHRE	Added 40 service transformers and triplex loads	Yes	74	160	75	0.24	IEEE (mod)
IEEE37	Delta system	No	39	117	73	2.59	IEEE
IEEE123	Includes switches for reconfiguration	No	130	274	237	3.62	IEEE
IEEE123_PV	Added 3320 kW PV in 14 locations	Yes	214	442	334	0.27	IEEE/NREL
IEEE8500	Large model, balanced secondary loads	Yes	4876	8531	6103	11.98	IEEE
IEEE9500	Added 2 grid sources and DER	Yes	5294	9499	6823	9.14	GridAPPS-D
R2_12_47_2	Supports approximately 4000 houses	Yes	853	1631	1086	6.26	PNNL
Transactive	Added 1281 secondary loads to IEEE123	Yes	1516	3051	2812	3.92	GridAPPS-D

17.8.1 IEEE 13 Node Model

This is a very small distribution test feeder operating at 4.16 kV voltage level. It consists of a single voltage regulator at the substation, overhead and underground lines, shunt capacitor, and an in-line transformer. This feeder is relatively highly loaded and provides a good test of the convergence of the problem for a very unbalanced system.

This model is recommended for debugging as the model is small enough that issues can be traced by hand.

17.8.2 IEEE 123 Node Model

This models a medium-sized unbalanced distribution system operating at the nominal voltage of 4.16 kV. It consists of overhead and underground lines with single, two and three-phase laterals, along with step regulators and shunt capacitors for voltage regulation. The feeder model is characterized by the unbalanced loading having all combinations of load types (constant current, impedance, and power). It also includes a few switches to allow for the alternate paths for the power flow via feeder reconfiguration.

This model is recommended for initial app testing and debugging thorugh the first stages of development.

17.8.3 IEEE 123 Node Model with PV

This model is a derivative of the IEEE 123 Node model and includes rooftop solar throughout the system.

17.8.4 IEEE 8500 Node Model

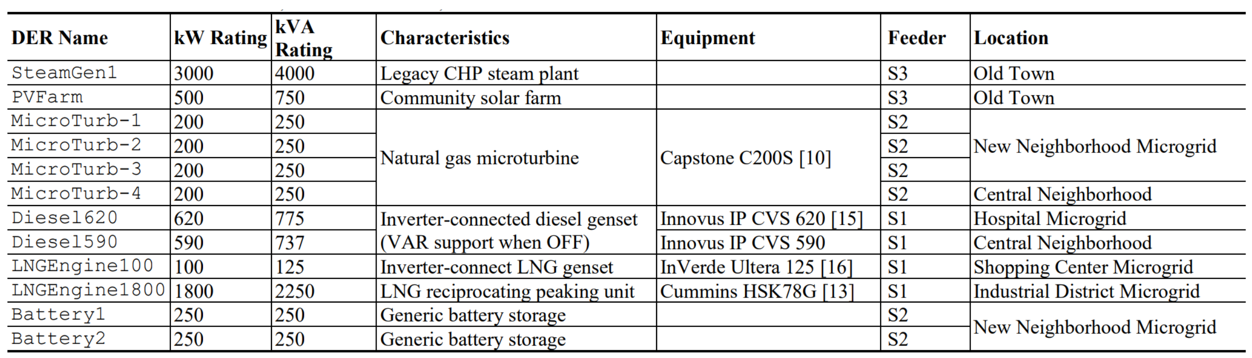
This is a relatively large and realistic radial distribution feeder consisting of MV and LV (secondary) circuits [21]. Unlike other test systems, this feeder also includes 120/240V center-tapped transformers that are commonly deployed in North American power distribution systems. Thus, it allows for users to interchange between the two versions of loading conditions: balanced (208 V) and unbalanced (120 V) in the secondary transformers. Voltage control is possible using a substation LTC transformer, as well as multiple poletop regulators and capacitor banks. The feeder was created to test scalability and convergence of power flow algorithms on a large unbalanced power distribution system.

- Length: 170 km
- Nominal voltage: 12.47 kV, 120/240V
- Topology: radial
- Service transformers: yes
- Customers: 1177
- Peak load: 11.1 MW
- Normally-open switches: no

17.8.5 9500 Node Test System

The 9500 Node Test System includes three radial distribution feeders with just over 12 MW of load, consisting of both medium voltage and low voltage equipment each supplied by a different distribution substation. The three distribution feeders are connected to each other through Normally-Open switches, which is representative of the way many utilities operate in North America. One feeder represents today's grid with low penetration of customer-side renewables. The second represents a potential future grid with microgrids and 100% renewable penetration. The third has no customer resources, a district steam plant, and a utility-scale PV farm. All three feeders have customers connected by low-voltage secondary triplex lines.

This is the recommended target feeder for development of all new applications



DER Name	kW Rating	kVA Rating	Characteristics	Equipment	Feeder	Location
SteamGen1	3000	4000	Legacy CHP steam plant		S3	Old Town
PVFarm	500	750	Community solar farm		S3	Old Town
MicroTurb-1	200	250	Natural gas microturbine	Capstone C200S [10]	S2	New Neighborhood Microgrid
MicroTurb-2	200	250			S2	
MicroTurb-3	200	250			S2	
MicroTurb-4	200	250			S2	Central Neighborhood
Diesel620	620	775	Inverter-connected diesel genset	Innovus IP CVS 620 [15]	S1	Hospital Microgrid
Diesel590	590	737	(VAR support when OFF)	Innovus IP CVS 590	S1	Central Neighborhood
LNGEngine100	100	125	Inverter-connect LNG genset	InVerde Ultera 125 [16]	S1	Shopping Center Microgrid
LNGEngine1800	1800	2250	LNG reciprocating peaking unit	Cummins HSK78G [13]	S1	Industrial District Microgrid
Battery1	250	250	Generic battery storage		S2	New Neighborhood Microgrid
Battery2	250	250	Generic battery storage		S2	

17.8.6 PNNL Taxonomy Feeder

17.8.7 EPRI J1 Feeder

17.8.8 UAF Microgrid

17.9 Adding New Models to GridAPPS-D

- `git clone https://github.com/GRIDAPPSD/CIMHub.git`
- `pip install SPARQLWrapper numpy pandas`

Copy CIM XML file and UUID file into CIMHub directory.

Open CIMHub directory in terminal:

- `cd CIMHub`

View the list of feeder models currently in the Blazegraph Database:

- `python3 ../CIMHub/utils/ListFeeders.py`

Upload your new model using `curl` to Blazegraph, specifying the name and MRID of your new feeder:

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

Create the set of txt files containing the measurable objects in your new model using the `ListMeasurables` script:

- `python3 ../CIMHub/utils/ListMeasureables.py cimhubconfig.json yournewmodel _YOUR-NEW-MODEL-FEEDER-MRID-123ABC456`

Insert the measurements into Blazegraph using the `InsertMeasurements` script. The measurement MRIDs will be saved into the file `uuidfile.json`:

- ``for f in `ls -1 *txt`; do python3 ../CIMHub/utils/InsertMeasurements.py cimhubconfig.json $f uuidfile.json; done``

Insert houses into Blazegraph using the `InsertHouses` script. The measurement MRIDs for the houses will be saved into the file `uuidfile2.json`: * `python3 ../CIMHub/utils/InsertHouses.py cimhubconfig.json _YOUR-NEW-MODEL-FEEDER-MRID-123ABC456 1 1 uuidfile2.json`

Open the GridAPPS-D Viz in a new window and you should be able to start a simulation using your new model

17.10 Adding New Models to the PowerGrid Models GitHub Repo

All models included by default in GridAPPS-D are stored in the PowerGrid Models GitHub repository

---

|GridAPPS-D_narrow.png|

CHAPTER

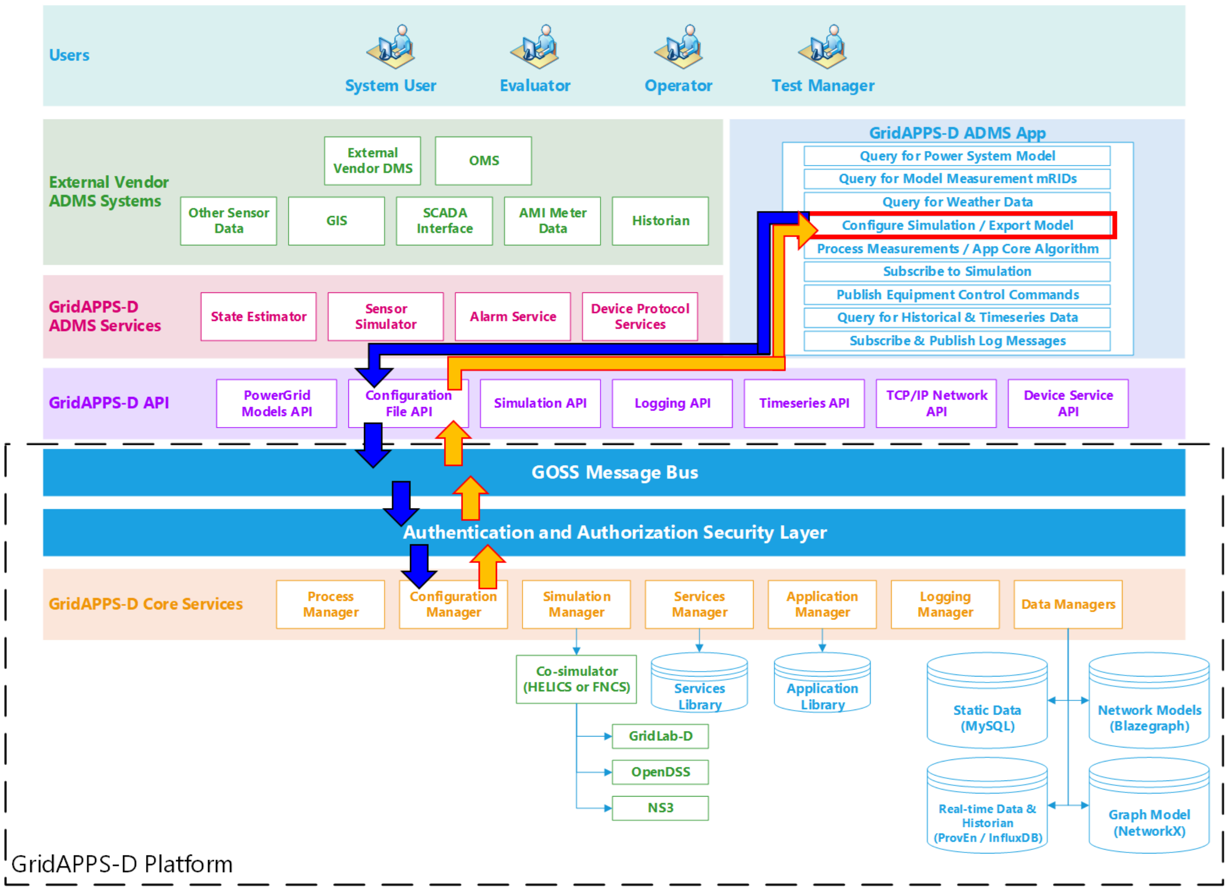
EIGHTEEN

USING THE CONFIGURATION FILE API

18.1 Introduction to the Configuration File API

The Configuration File API is used to generate power system models that can be solved in GridLab-D or OpenDSS based on the original CIM XML model. The load profile and ZIP parameters can be modified from the nominal values prior to model creation and export.

In the Application Components diagram (explained in detail with sample code in *GridAPPS-D Application Structure*), the Configuration File API is used for configuring parallel simulations and exporting the power system model.



18.2 API Syntax Overview

Application passes query to GridAPPS-D Platform

First, the application creates a query message for requesting information about the desired power system configuration in the format of a JSON string or equivalant Python dictionary object. The syntax of this message is explained in detail below.

The application then passes the query through the Configuration File API to the GridAPPS-D Platform, which publishes it to a queue channel on the GOSS Message Bus. If the app is authenticated and authorized to pass queries, the query message is delivered to the Configuration Manager.

GridAPPS-D Platform responds to Application query

The Configuration Manager obtains the CIM XML file for the desired power system model and then converts it to the desired output format with all of the requested changes to the model. The Configuration File API then returns the desired information back to the application as a JSON message (for Y-Bus or partial models) or export the files to the directory specified in the

---

18.2.1 API Communication Channel

All queries passed to the PowerGrid Models API need to use the correct communication channel, which is obtained using the *GridAPPS-D Topics library*.

The PowerGrid Model API uses a `/queue/` channel to pull power system model info from the the Blazegraph Database. The base static string used is `goss.gridappsd.process.request.config`, which can be called using the `.CONFIG` method from the topics library.

When developing in python, it is recommended to use the `.CONFIG` method. When using the STOMP client in GridAPPS-D VIZ, it is necessary to use the base static string.

[ ]:
```
from gridappsd import topics as t
topic = t.CONFIG
```

18.2.2 Structure of a Query Message

Queries passed to Configuration File API are formatted as python dictionaries or equivalent JSON scripts wrapped as a python string.

The accepted set of key-value pairs for the Configuration File API query message is

```
message = {
    "configurationType": "INSERT QUERY HERE",
    "parameters": {
        "key1": "value1",
        "key2": "value2"}
}
```

The components of the message are as follows:

- `"configurationType":` – Specifies the type of configuration file requested.

- `"parameters":` – Specifies any specific power system model parameters. Values depend on the particular configurationType.

The usage of each of these message components are explained in detail with code block examples below.

Important: Be sure to pay attention to placement of commas (**,**) at the end of each JSON line. Commas are placed at the end of each line *except* the last line. Incorrect comma placement will result in a JsonSyntaxException.

All of the queries are passed to the Configuration API using the `.get_response(topic, message)` method for the GridAPPS-D platform connection variable.

18.2.3 Specifying the configurationType

Below are the possible `configurationType` key-value pairs that are used to specify the type of each query. Executable code block examples are provided for each of the requests in the subsections below.

The first group of *configurationType* key-value pairs are for queries for information related to the GridLab-D GLM files and settings:

- `"configurationType": "GridLab-D All"` – *Export all GridLab-D files*
- `"configurationType": "GridLab-D Base GLM"` – *Query for GridLab-D base GLM file*
- `"configurationType": "GridLab-D Symbols"` – *Query for GridLab-D symbols file*
- `"configurationType": "GridLab-D Simulation Output"` – *Query for available measurement types*

The second group of *configurationType* key-value pairs are for queries for OpenDSS model files

- `"configurationType": "DSS All"` – *Export all OpenDSS model files*
- `"configurationType": "DSS Base"` – *Query for OpenDSS version of power system model*
- `"configurationType": "DSS Coordinate"` – *Query for list of OpenDSS XY coordinates*
- `"configurationType": "YBus Export"` – *Export Y-Bus matrix from OpenDSS*

The third group of *configurationType* are for queries for CIM dictionary and feeder index files:

- `"configurationType": "CIM Dictionary"` – *Query for python dictionary of CIM XML model*
- `"configurationType": "CIM Feeder Index"` – *Query for python dictionary of model mRIDs*

18.3 Querying for GridLab-D Configuration Files

This section outlines the details of key-value pairs for the possible queries associated with each value of the `queryMeasurement` key listed above for obtaining or exporting GridLAB-D Files

18.3.1 Export all GridLab-D Files

This API call generates all the GLM files necessary to solve the power system model in GridLab-D. The query returns a directory where the set of GLM files are located inside the GridAPPS-D docker container.

Configuration File request key-value pair:

- `"configurationType": "GridLab-D All"`

The `parameters` key has a set of required values as well as some optional values:

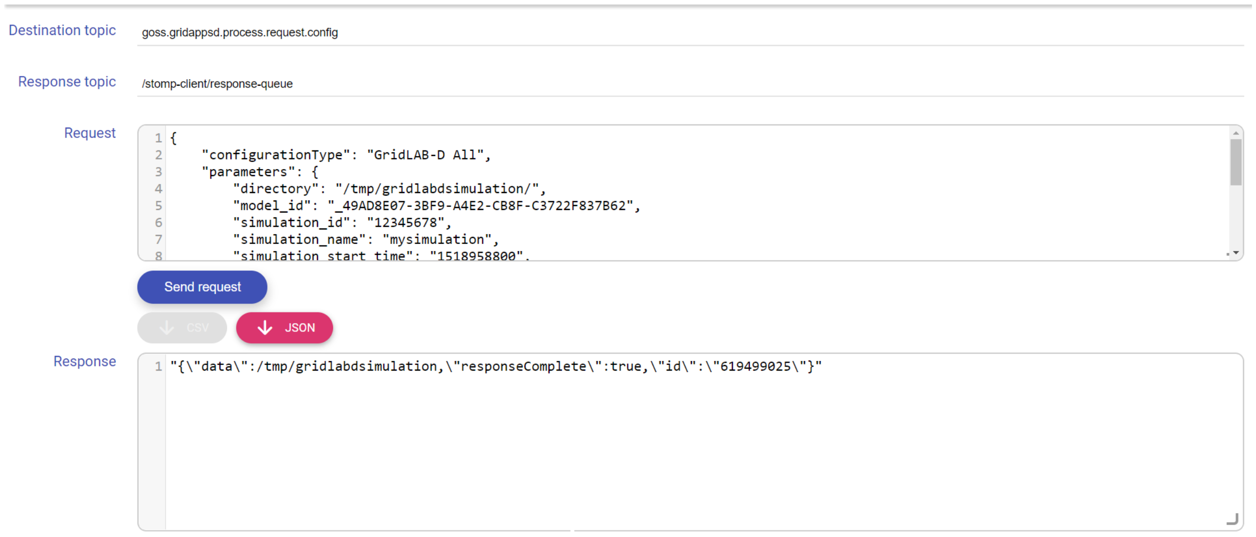
```
"parameters": {     REQUIRED KEYS                    REQUIRED VALUES
                "model_id":                             mRID as string ,
                "directory":                            output directory as string ,
                "simulation_name":                      string ,
                "simulation_start_time":                epoch time number ,
                "simulation_duration":                  number ,
                "simulation_id":                        number ,
                "simulation_broker_host":               string ,
                "simulation_broker_port":               number ,
                    OPTIONAL KEYS                    OPTIONAL VALUES
                "i_fraction":                           number ,
                "p_fraction":                           number ,
                "z_fraction":                           number ,
                "load_scaling_factor":                  number ,
                "schedule_name":                        string ,
                "solver_method":                        string }
```

The numeric values for the key-value pairs associated with `parameters` can be written as number or as strings. The key-value pairs can be specified in any order.

Example: Export IEEE 13 node model with constant current loads to GLM files :

```
[ ]: from gridappsd import topics as t
topic = t.CONFIG

message = {
    "configurationType": "GridLAB-D All",
    "parameters": {
        "directory": "/tmp/gridlabdsimulation/",
        "model_id": model_mrid,
        "simulation_id": "12345678",
        "simulation_name": "mysimulation",
        "simulation_start_time": "1518958800",
        "simulation_duration": "60",
        "simulation_broker_host": "localhost",
        "simulation_broker_port": "61616",
        "schedule_name": "ieeezipload",
        "load_scaling_factor": "1.0",
        "z_fraction": "0.0",
        "i_fraction": "1.0",
        "p_fraction": "0.0",
        "solver_method": "NR" }
}

gapps.get_response(topic, message, timeout = 120)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.data.config` topic and the same message without the python wrapping:


Destination topic: goss.gridappsd.process.request.config

Response topic: /stomp-client/response-queue

Request

```
{
    "configurationType": "GridLAB-D All",
    "parameters": {
        "directory": "/tmp/gridlabdsimulation/",
        "model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "simulation_id": "12345678",
        "simulation_name": "mysimulation",
        "simulation_start_time": "1518958800",
```

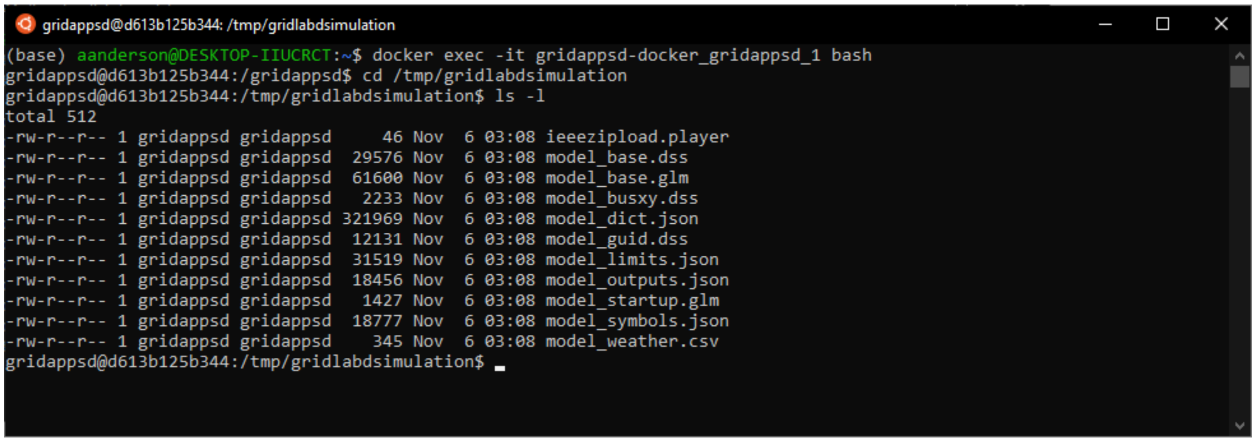
Response

```
"{\"data\":/tmp/gridlabdsimulation,\"responseComplete\":true,\"id\":\"619499025\"}"
```

Note: The output directory is inside the GridAPPS-D Docker Container, not in your Ubuntu or Windows environment. To access the files, it is necessary to change directories to those inside the docker container.

Open a new Ubuntu terminal and run: * `docker exec -it gridappsd-docker_gridappsd_1 bash` * `cd /tmp/gridlabdsimulation` * `ls -l`

To copy the files to your regular directory, use the `docker cp` command. For help using docker, see *Docker Shortcuts* on working with Docker containers.



```
(base) aanderson@DESKTOP-IIUCRCT:~$ docker exec -it gridappsd-docker_gridappsd_1 bash
gridappsd@d613b125b344:/gridappsd$ cd /tmp/gridlabdsimulation
gridappsd@d613b125b344:/tmp/gridlabdsimulation$ ls -l
total 512
-rw-r--r-- 1 gridappsd gridappsd     46 Nov  6 03:08 ieeezipload.player
-rw-r--r-- 1 gridappsd gridappsd  29576 Nov  6 03:08 model_base.dss
-rw-r--r-- 1 gridappsd gridappsd  61600 Nov  6 03:08 model_base.glm
-rw-r--r-- 1 gridappsd gridappsd   2233 Nov  6 03:08 model_busxy.dss
-rw-r--r-- 1 gridappsd gridappsd 321969 Nov  6 03:08 model_dict.json
-rw-r--r-- 1 gridappsd gridappsd  12131 Nov  6 03:08 model_guid.dss
-rw-r--r-- 1 gridappsd gridappsd  31519 Nov  6 03:08 model_limits.json
-rw-r--r-- 1 gridappsd gridappsd  18456 Nov  6 03:08 model_outputs.json
-rw-r--r-- 1 gridappsd gridappsd   1427 Nov  6 03:08 model_startup.glm
-rw-r--r-- 1 gridappsd gridappsd  18777 Nov  6 03:08 model_symbols.json
-rw-r--r-- 1 gridappsd gridappsd    345 Nov  6 03:08 model_weather.csv
gridappsd@d613b125b344:/tmp/gridlabdsimulation$ _
```

18.3.2 Query for GridLab-D Base GLM File

This API call generates a single GLM file that contains the entire power system model that can be solved in GridLab-D. The query returns the entire model GLM file wrapped in a python dictionary.

Configuration File request key-value pair:

- `"configurationType": "GridLab-D Base GLM"`

The `parameters` key has a set of required values as well as some optional values:

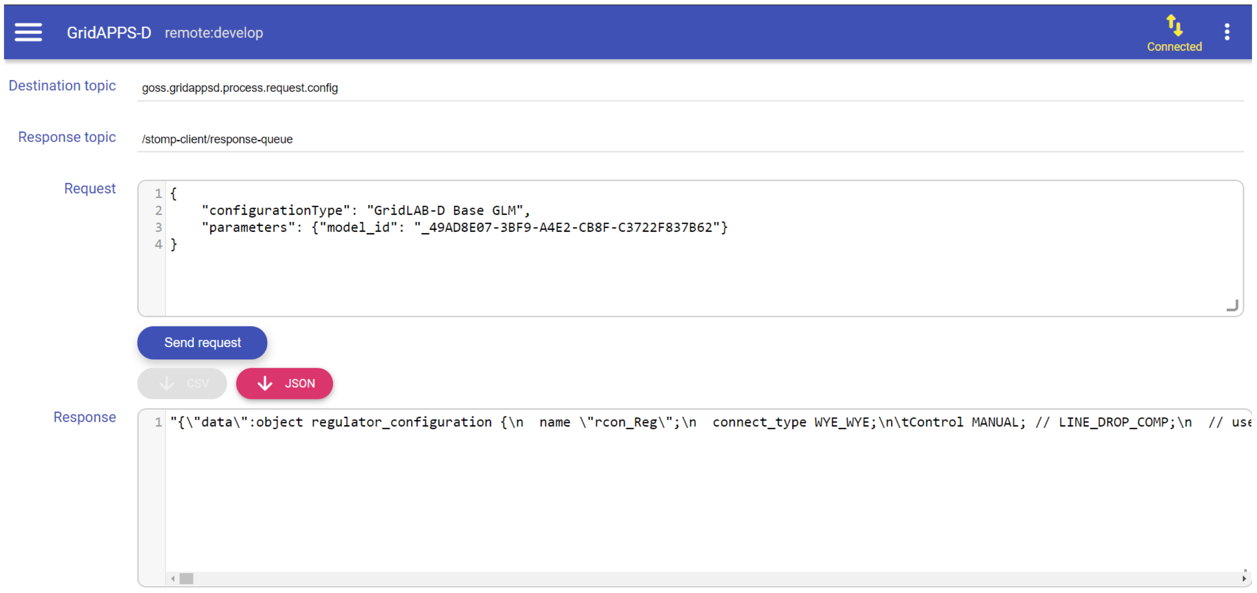
```
"parameters": {      REQUIRED KEYS                      REQUIRED VALUES
                "model_id":                                mRID as string ,
                    OPTIONAL KEYS                        OPTIONAL VALUES
                "i_fraction":                              number ,
                "p_fraction":                              number ,
                "z_fraction":                              number ,
                "load_scaling_factor":                     number ,
                "schedule_name":                           string }
```

The numeric values for the key-value pairs associated with `parameters` can be written as number or as strings. The key-value pairs can be specified in any order.

Example 1: Create GLM base file using nominal load values:

```
[ ]: from gridappsd import topics as t
     topic = t.CONFIG

     message = {
         "configurationType": "GridLAB-D Base GLM",
         "parameters": {"model_id": model_mrid}
     }

     gapps.get_response(topic, message, timeout = 60)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:



Destination topic goss.gridappsd.process.request.config

Response topic /stomp-client/response-queue

Request

```
{
    "configurationType": "GridLAB-D Base GLM",
    "parameters": {"model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62"}
}
```

Response

```
"{\"data\":object regulator_configuration {\n  name \"rcon_Reg\";\n  connect_type WYE_WYE;\n\tControl MANUAL; // LINE_DROP_COMP;\n  // use
```

Example 2: Create GLM base file using all constant current loads and hourly load curve:

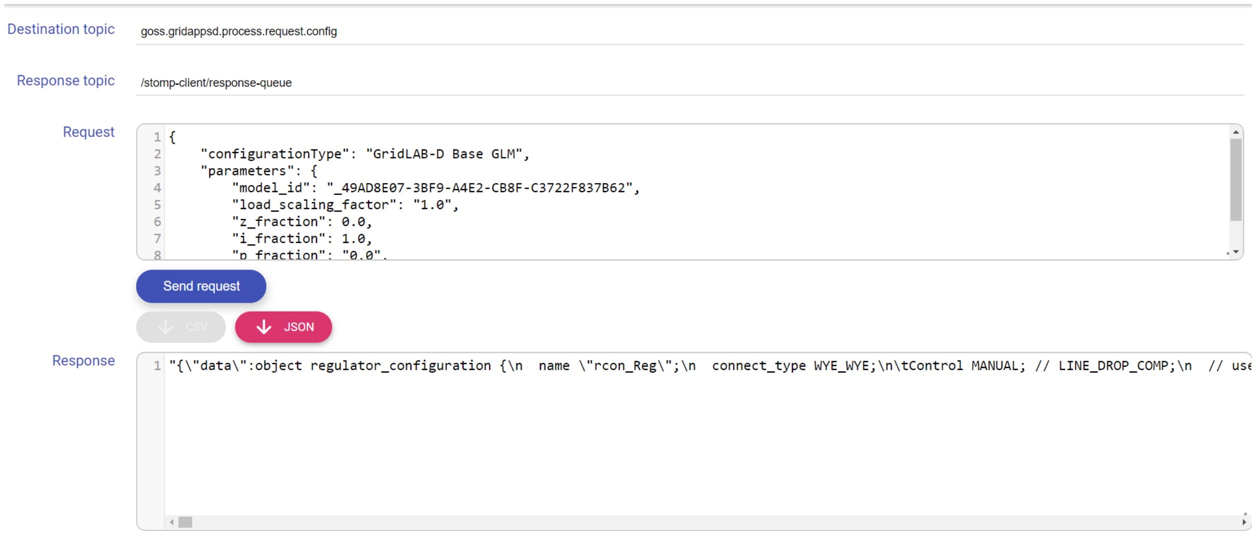
```
[ ]: from gridappsd import topics as t
     topic = t.CONFIG

     message = {
         "configurationType": "GridLAB-D Base GLM",
         "parameters": {
             "model_id": model_mrid,
             "load_scaling_factor": "1.0",
```

(continues on next page)

(continued from previous page)

```
        "z_fraction": 0.0,
        "i_fraction": 1.0,
        "p_fraction": "0.0",
        "schedule_name": "ieeezipload"}
}

gapps.get_response(topic, message, timeout = 60)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:



Destination topic goss.gridappsd.process.request.config

Response topic /stomp-client/response-queue

Request

```
{
    "configurationType": "GridLAB-D Base GLM",
    "parameters": {
        "model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "load_scaling_factor": "1.0",
        "z_fraction": 0.0,
        "i_fraction": 1.0,
        "p_fraction": "0.0",
```

Response

```
"{\"data\":object regulator_configuration {\n  name \"rcon_Reg\";\n  connect_type WYE_WYE;\n\tControl MANUAL; // LINE_DROP_COMP;\n  // use
```

18.3.3 Query for GridLab-D Symbols File

This API call generates a python dictionary with all the XY coordinates used by GridLab-D when running a simulation.

Configuration File request key-value pair:

- `"configurationType": "GridLab-D Symbols"`

The `parameters` key has a set of required values as well as some optional values:

```
"parameters": {      REQUIRED KEYS                      REQUIRED VALUES
                "model_id":                              mRID as string ,
                     OPTIONAL KEYS                     OPTIONAL VALUES
                "simulation_id":                         number }
```

The key-value pairs can be specified in any order.

```
[ ]: from gridappsd import topics as t
     topic = t.CONFIG

     message = {
         "configurationType": "GridLAB-D Symbols",
         "parameters": { "model_id": model_mrid }
     }
```

(continues on next page)

(continued from previous page)

```
gapps.get_response(topic, message, timeout = 60)
```

18.3.4 Query for GridLab-D Measurement Types

This API call returns a list of device names and types of available measurement in the GridLab-D format (**not** CIM classes or measurement mRIDs)

Configuration File request key-value pair:

- `"configurationType": "GridLab-D Simulation Output"`

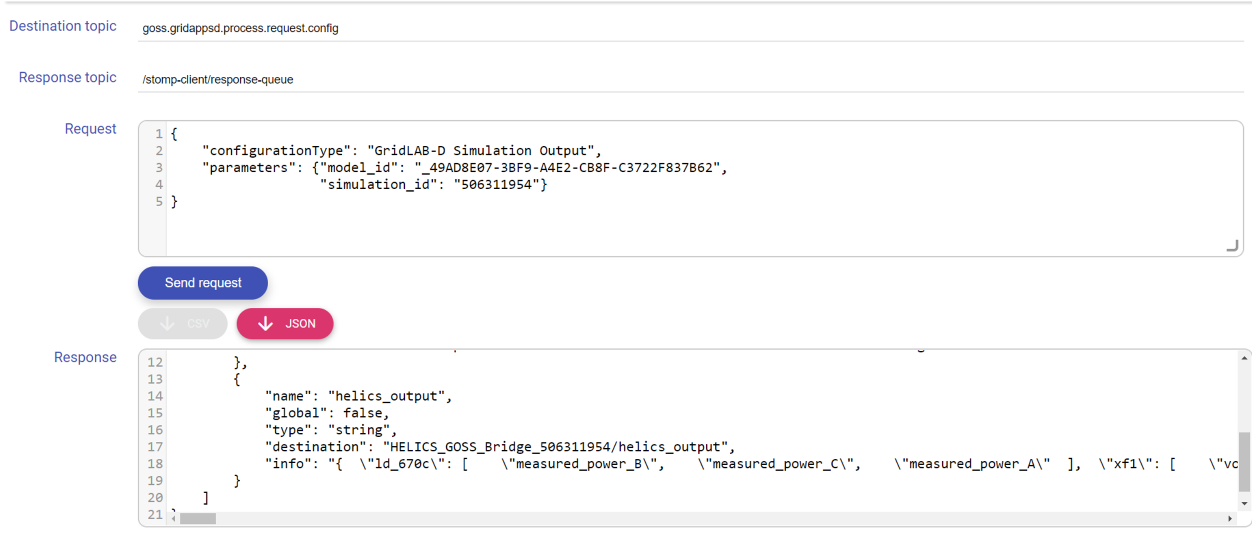
The `parameters` key has a set of required values as well as some optional values:

```
"parameters": {      REQUIRED KEYS                      REQUIRED VALUES
                 "simulation_id":                          number or string,
                    OPTIONAL KEYS                      OPTIONAL VALUES
                 "model_id":                               mRID as string }
```

The key-value pairs can be specified in any order.

[ ]:
```python
from gridappsd import topics as t
topic = t.CONFIG

message = {
    "configurationType": "GridLAB-D Simulation Output",
    "parameters": {"simulation_id": "paste sim_id here"}
}

gapps.get_response(topic, message, timeout = 60)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:



goss.gridappsd.process.request.config

/stomp-client/response-queue

```
{
    "configurationType": "GridLAB-D Simulation Output",
    "parameters": {"model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
                   "simulation_id": "506311954"}
}
```

```
        },
        {
            "name": "helics_output",
            "global": false,
            "type": "string",
            "destination": "HELICS_GOSS_Bridge_506311954/helics_output",
            "info": "{  \"ld_670c\": [    \"measured_power_B\",    \"measured_power_C\",    \"measured_power_A\"  ],  \"xf1\": [    \"vc
        }
    ]
}
```

18.4 Querying for OpenDSS Configuration Files

This section outlines the details of key-value pairs for the possible queries associated with each value of the `queryMeasurement` key listed above for obtaining or exporting OpenDSS Files

18.4.1 Export all OpenDSS Files

This API call generates all the OpenDSS files necessary to solve the power system model in OpenDSS. The query returns a directory where the set of DSS files are located.

Configuration File request key-value pair:

- `"configurationType": "DSS All"`

The `parameters` key has a set of required values as well as some optional values:

```
"parameters": {     REQUIRED KEYS                     REQUIRED VALUES
                "model_id":                             mRID as string ,
                "directory":                            desired output directory as␣
↪string ,
                "simulation_name":                       string ,
                "simulation_start_time":                epoch time number ,
                "simulation_duration":                  number ,
                "simulation_id":                        number ,
                "simulation_broker_host":               string ,
                "simulation_broker_port":               number ,
                    OPTIONAL KEYS                     OPTIONAL VALUES
                "i_fraction":                           number ,
                "p_fraction":                           number ,
                "z_fraction":                           number ,
                "load_scaling_factor":                  number ,
                "schedule_name":                        string ,
                "solver_method":                        string }
```

The numeric values for the key-value pairs associated with `parameters` can be written as number or as strings. The key-value pairs can be specified in any order.

Example: Export IEEE 13 node model with constant current loads to DSS files :

```
[ ]: from gridappsd import topics as t
     topic = t.CONFIG

     message = {
         "configurationType": "DSS All",
         "parameters": {
             "directory": "/tmp/dsssimulation/",
             "model_id": model_mrid,
             "simulation_id": "12345678",
             "simulation_name": "ieee13",
             "simulation_start_time": "1518958800",
             "simulation_duration": "60",
             "simulation_broker_host": "localhost",
             "simulation_broker_port": "61616",
             "schedule_name": "ieeezipload",
             "load_scaling_factor": "1.0",
             "z_fraction": "0.0",
             "i_fraction": "1.0",
```

(continues on next page)

(continued from previous page)

```
        "p_fraction": "0.0",
        "solver_method": "NR" }
}

gapps.get_response(topic, message, timeout = 60)
```

Note: The output directory is inside the GridAPPS-D Docker Container, not in your Ubuntu or Windows environment. To access the files, it is necessary to change directories to those inside the docker container.

Open a new Ubuntu terminal and run: * `docker exec -it gridappsd-docker_gridappsd_1 bash` * `cd /tmp/dssdsimulation` * `ls -l`

To copy the files to your regular directory, use the `docker cp` command. For help using docker, see *Docker Shortcuts* on working with Docker containers.

18.4.2 Query for OpenDSS Base File

This API call generates a single DSS file that contains the entire power system model that can be solved in OpenDSS. The query returns the entire model DSS file wrapped in a python dictionary.

Configuration File request key-value pair:

- `"configurationType": "DSS Base"`

The `parameters` key has a set of required values as well as some optional values:

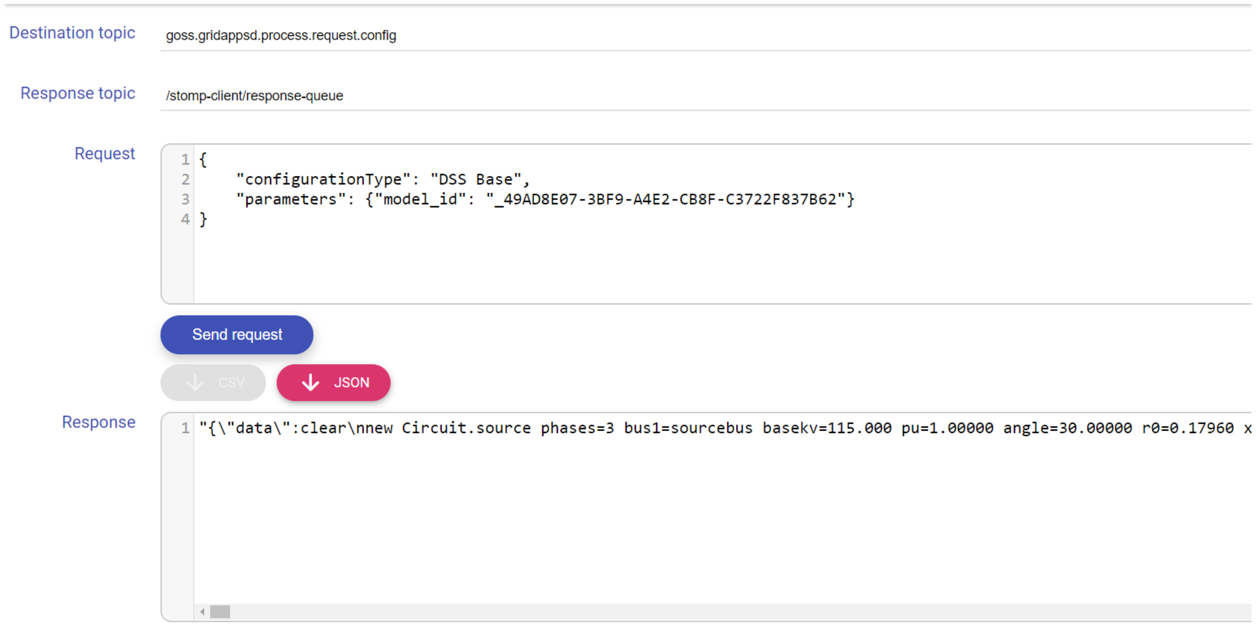
```
"parameters": {      REQUIRED KEYS                      REQUIRED VALUES
                "model_id":                               mRID as string ,
                    OPTIONAL KEYS                      OPTIONAL VALUES
                "i_fraction":                             number ,
                "p_fraction":                             number ,
                "z_fraction":                             number ,
                "load_scaling_factor":                    number ,
                "schedule_name":                          string }
```

The numeric values for the key-value pairs associated with `parameters` can be written as number or as strings. The key-value pairs can be specified in any order.

Example 1: Create OpenDSS base file using nominal load values:

```
[ ]: from gridappsd import topics as t
topic = t.CONFIG

message = {
    "configurationType": "DSS Base",
    "parameters": {"model_id": model_mrid}
}

gapps.get_response(topic, message, timeout = 60)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:


Destination topic goss.gridappsd.process.request.config

Response topic /stomp-client/response-queue

Request
```
{
    "configurationType": "DSS Base",
    "parameters": {"model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62"}
}
```
Send request

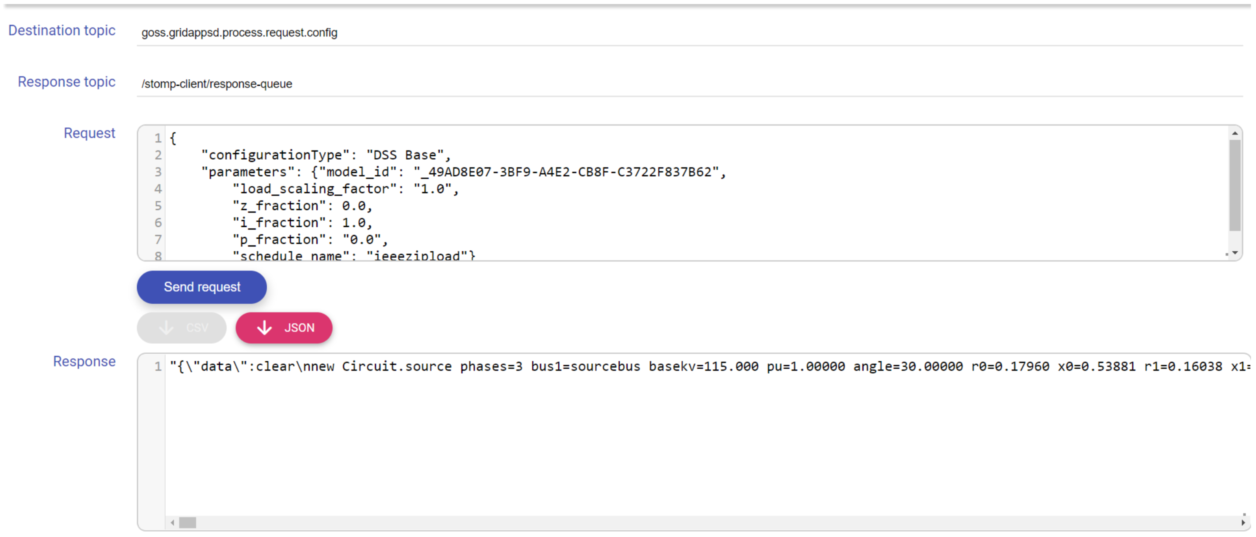
CSV JSON

Response
```
"{\"data\":clear\nnew Circuit.source phases=3 bus1=sourcebus basekv=115.000 pu=1.00000 angle=30.00000 r0=0.17960 x
```

Example 2: Create OpenDSS base file using all constant current loads and hourly load curve:

```
[ ]: message = {
         "configurationType": "DSS Base",
         "parameters": {
             "model_id": model_mrid,
             "load_scaling_factor": "1.0",
             "z_fraction": 0.0,
             "i_fraction": 1.0,
             "p_fraction": "0.0",
             "schedule_name": "ieeezipload"}
     }

     gapps.get_response(topic, message, timeout = 60)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:


Destination topic goss.gridappsd.process.request.config

Response topic /stomp-client/response-queue

Request
```
{
    "configurationType": "DSS Base",
    "parameters": {"model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
        "load_scaling_factor": "1.0",
        "z_fraction": 0.0,
        "i_fraction": 1.0,
        "p_fraction": "0.0",
        "schedule_name": "ieeezipload"}
```
Send request

CSV JSON

Response
```
"{\"data\":clear\nnew Circuit.source phases=3 bus1=sourcebus basekv=115.000 pu=1.00000 angle=30.00000 r0=0.17960 x0=0.53881 r1=0.16038 x1=
```

18.4.3 Query for OpenDSS Coordinate File

This API call generates a file with all the XY coordinates used by OpenDSS when plotting the feeder.

Configuration File request key-value pair:

- `"configurationType": "DSS Coordinate"`

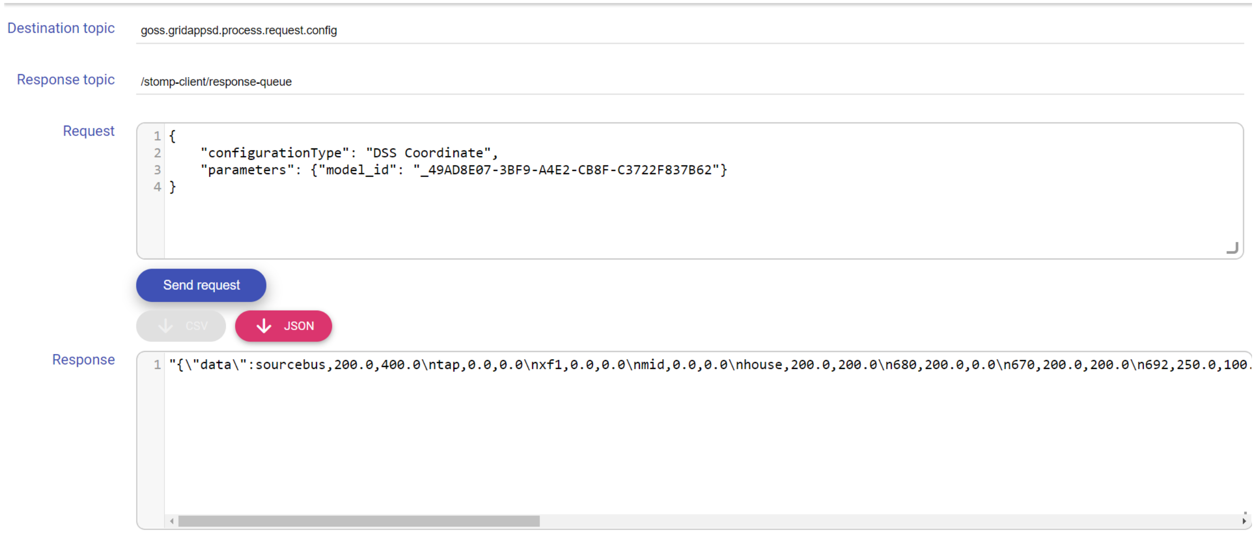
The `parameters` key has a set of required values as well as some optional values:

```
"parameters": {      REQUIRED KEYS                      REQUIRED VALUES
                "model_id":                            mRID as string ,
                    OPTIONAL KEYS                     OPTIONAL VALUES
                "simulation_id":                       number }
```

The key-value pairs can be specified in any order.

```
[ ]: from gridappsd import topics as t
     topic = t.CONFIG

     message = {
         "configurationType": "DSS Coordinate",
         "parameters": {"model_id": model_mrid}
     }

     gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:



Destination topic: goss.gridappsd.process.request.config

Response topic: /stomp-client/response-queue

Request

```
{
    "configurationType": "DSS Coordinate",
    "parameters": {"model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62"}
}
```

Response

```
"{\"data\":sourcebus,200.0,400.0\ntap,0.0,0.0\nxf1,0.0,0.0\nmid,0.0,0.0\nhouse,200.0,200.0\n680,200.0,0.0\n670,200.0,200.0\n692,250.0,100.
```

18.4.4 Query for Y-Bus Matrix

This API call generates a Y-Bus matrix from **either** the model mRID or the simulation id.

Note: The GridAPPS-D platform currently does not have an in-built topology processor, so the Y-Bus matrix is NOT updated during the simulation to reflect switching actions or transformer tap changes that happen in real time.

Real-time Y-bus matrix output will be available from the future Dynamic Y-bus Service from the GridAPPS-D App Toolbox.

Configuration File request key-value pair:

- `"configurationType": "YBus Export"`

The `parameters` key has a set of required values as well as some optional values:

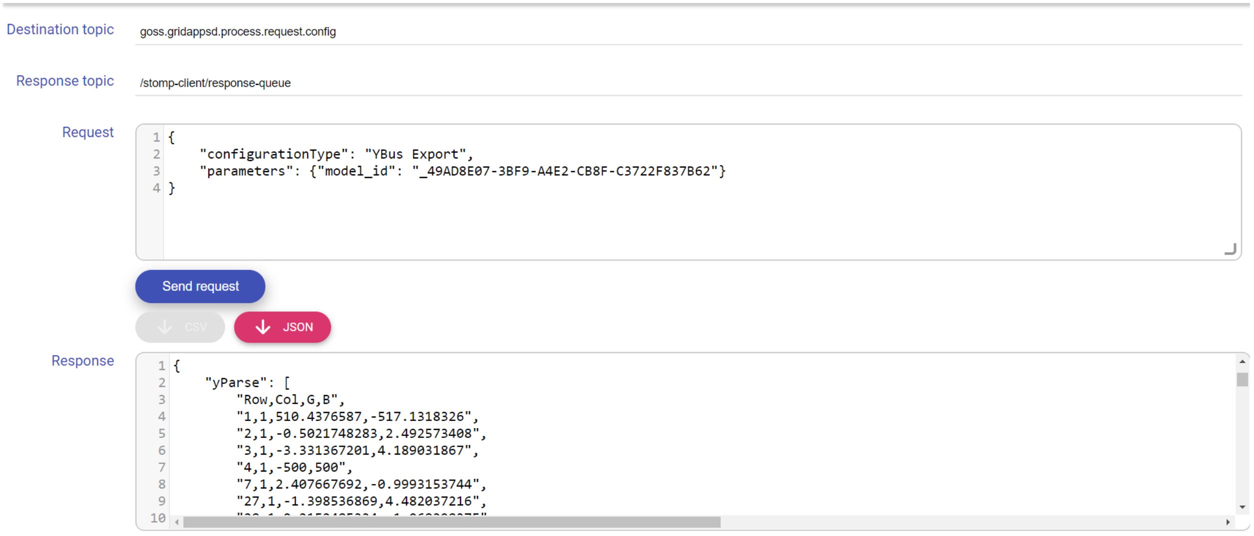
```
"parameters": {      REQUIRED KEYS                         REQUIRED VALUES
                "model_id":                                  mRID as string ,
                    OR
                "simulation_id":                             number }
```

The key-value pairs can be specified in any order.

Example 1: Request Y-Bus for IEEE 13 node model using model mRID:

[ ]:
```python
from gridappsd import topics as t
topic = t.CONFIG

message = {
    "configurationType": "YBus Export",
    "parameters": {"model_id": model_mrid}
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:



Destination topic: goss.gridappsd.process.request.config

Response topic: /stomp-client/response-queue

Request

```
{
    "configurationType": "YBus Export",
    "parameters": {"model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62"}
}
```

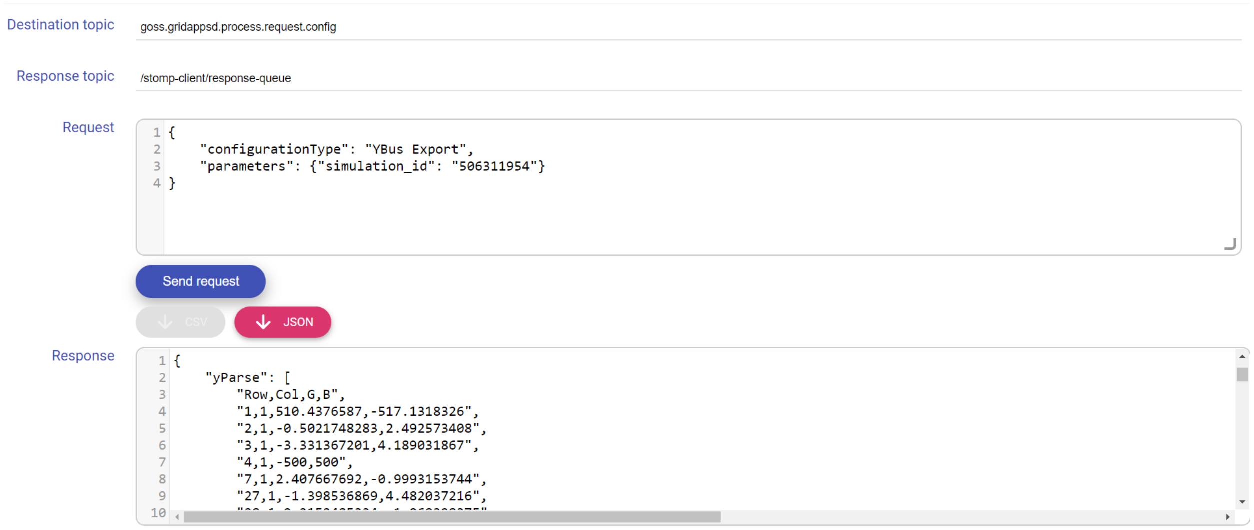
Response

```
{
    "yParse": [
        "Row,Col,G,B",
        "1,1,510.4376587,-517.1318326",
        "2,1,-0.5021748283,2.492573408",
        "3,1,-3.331367201,4.189031867",
        "4,1,-500,500",
        "7,1,2.407667692,-0.9993153744",
        "27,1,-1.398536869,4.482037216",
```

Example 2: Request Y-Bus for IEEE 13 node model with all loads set as constant current using model mRID:

[ ]:
```python
from gridappsd import topics as t
topic = t.CONFIG

message = {
    "configurationType": "YBus Export",
    "parameters": {
        "model_id": model_mrid,
        "load_scaling_factor": "2.0",
        "schedule_name": "ieeezipload",
        "z_fraction": "0.4",
        "i_fraction": "0.3",
        "p_fraction": "0.3" }
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:


Destination topic goss.gridappsd.process.request.config

Response topic /stomp-client/response-queue

Request
```
{
    "configurationType": "YBus Export",
    "parameters": {"simulation_id": "506311954"}
}
```

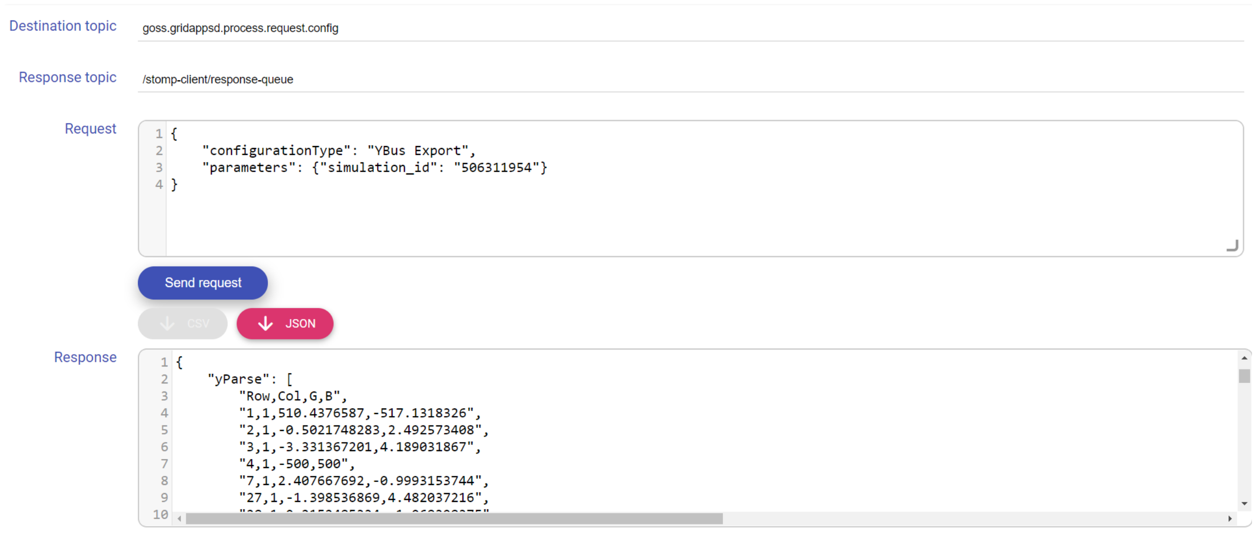
Response
```
{
    "yParse": [
        "Row,Col,G,B",
        "1,1,510.4376587,-517.1318326",
        "2,1,-0.5021748283,2.492573408",
        "3,1,-3.331367201,4.189031867",
        "4,1,-500,500",
        "7,1,2.407667692,-0.9993153744",
        "27,1,-1.398536869,4.482037216",
```

Example 3: Obtain Y-Bus from simulation_id:

[ ]:
```python
viz_simulation_id = "paste id here"
```

[ ]:
```python
topic = "goss.gridappsd.process.request.config"

message = {
    "configurationType":"YBus Export",
    "parameters":{"simulation_id": viz_simulation_id}
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:


Destination topic goss.gridappsd.process.request.config

Response topic /stomp-client/response-queue

Request
```
{
    "configurationType": "YBus Export",
    "parameters": {"simulation_id": "506311954"}
}
```

Response
```
{
    "yParse": [
        "Row,Col,G,B",
        "1,1,510.4376587,-517.1318326",
        "2,1,-0.5021748283,2.492573408",
        "3,1,-3.331367201,4.189031867",
        "4,1,-500,500",
        "7,1,2.407667692,-0.9993153744",
        "27,1,-1.398536869,4.482037216",
```

18.5 Querying for CIM Dictionary Files

This section outlines the details of key-value pairs for the possible queries associated with each value of the `queryMeasurement` key listed above for obtaining or dictionaries of CIM objects used for file conversion.

18.5.1 Query for Model Dictionary

This API generates a python dictionary which maps the CIM mRIDs of objects in the power system model to names of model objects used in other simulators.

Configuration File request key-value pair:

- `"configurationType": "CIM Dictionary"`

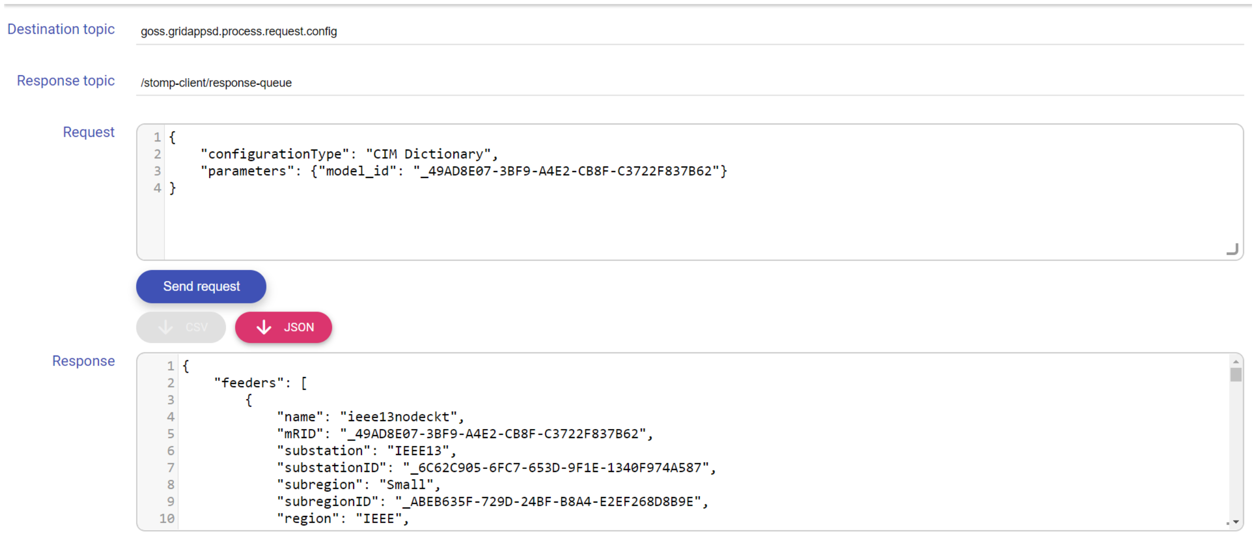
The `parameters` key has a set of required values as well as some optional values:

```
"parameters": {      REQUIRED KEYS                     REQUIRED VALUES
                "model_id":                              mRID as string ,
                    OPTIONAL KEYS                      OPTIONAL VALUES
                "simulation_id":                         number }
```

The key-value pairs can be specified in any order.

```
[ ]: topic = "goss.gridappsd.process.request.config"

     message = {
         "configurationType": "CIM Dictionary",
         "parameters":{"model_id": model_mrid}
     }

     gapps.get_response(topic, message, timeout = 30)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:



```
goss.gridappsd.process.request.config
```

```
/stomp-client/response-queue
```

```
{
    "configurationType": "CIM Dictionary",
    "parameters": {"model_id": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62"}
}
```

```
{
    "feeders": [
        {
            "name": "ieee13nodeckt",
            "mRID": "_49AD8E07-3BF9-A4E2-CB8F-C3722F837B62",
            "substation": "IEEE13",
            "substationID": "_6C62C905-6FC7-653D-9F1E-1340F974A587",
            "subregion": "Small",
            "subregionID": "_ABEB635F-729D-24BF-B8A4-E2EF268D8B9E",
            "region": "IEEE",
```

18.5.2 Query for CIM Feeder Index

This API call returns a python dictionary of the available feeders in the Blazegraph database of power system models.

Configuration File request key-value pair:

- `"configurationType": "CIM Feeder Index"`

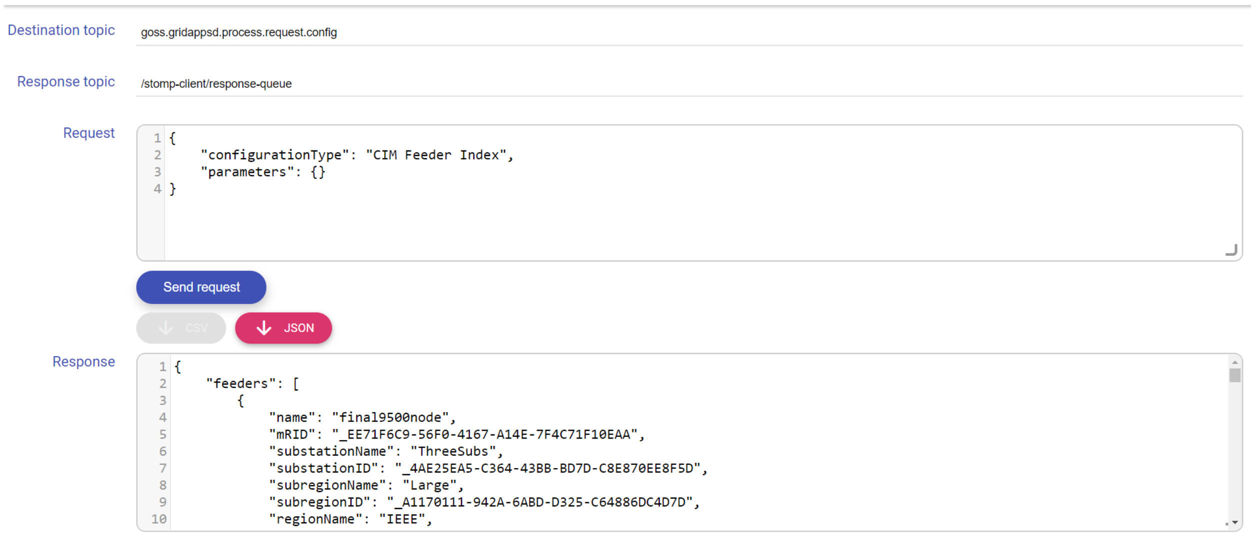
The `parameters` key has a set of required values as well as some optional values:

```
"parameters": {      REQUIRED KEYS                      REQUIRED VALUES
                "model_id":                               mRID as string ,
                    OPTIONAL KEYS                      OPTIONAL VALUES
                "simulation_id":                          number }
```

The key-value pairs can be specified in any order.

```
[ ]: from gridappsd import topics as t
topic = t.CONFIG

message = {
    "configurationType": "CIM Feeder Index",
    "parameters":{}
}

gapps.get_response(topic, message)
```

The same query can be passed through the STOMP client by specifying the `goss.gridappsd.process.request.config` topic and the same message without the python wrapping:



```
goss.gridappsd.process.request.config
```

```
/stomp-client/response-queue
```

```
{
    "configurationType": "CIM Feeder Index",
    "parameters": {}
}
```

```
{
    "feeders": [
        {
            "name": "final9500node",
            "mRID": "_EE71F6C9-56F0-4167-A14E-7F4C71F10EAA",
            "substationName": "ThreeSubs",
            "substationID": "_4AE25EA5-C364-43BB-BD7D-C8E870EE8F5D",
            "subregionName": "Large",
            "subregionID": "_A1170111-942A-6ABD-D325-C64886DC4D7D",
            "regionName": "IEEE",
```

|GridAPPS-D_narrow.png|

CHAPTER

NINETEEN

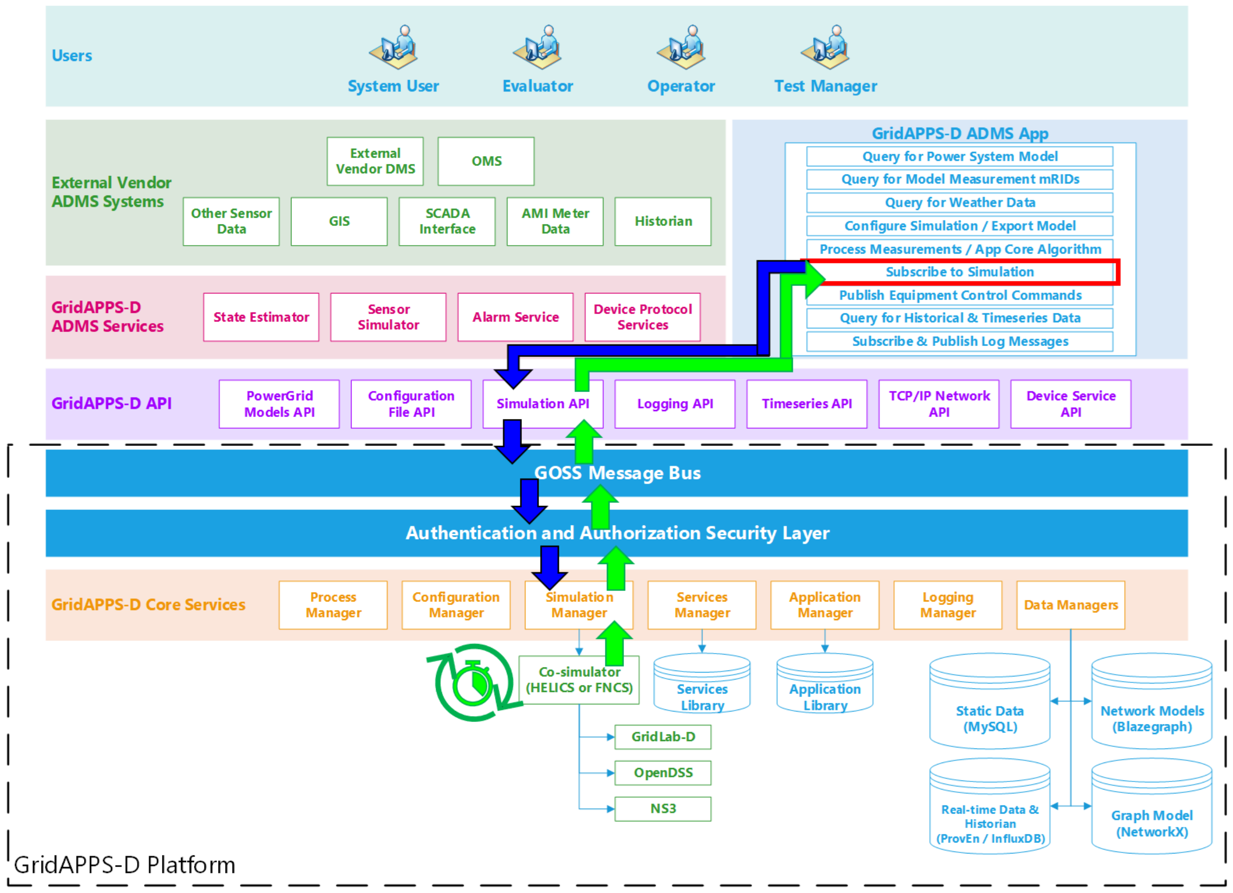
RUNNING SIMULATIONS WITH THE SIMULATION API

19.1 Introduction to the Simulation API

The Simulation API is used for all actions related to a power system simulation. It is used to start, pause, restart, and stop a simulation from the command line or inside an application. It is all used to subscribe to measurements of equipment (such as line flows, loads, and DG setpoints) and the statuses of switches, capacitors, transformer taps, etc. It is also used to publish equipment control and other simulation input commands.

In the Application Components diagram (explained in detail with sample code in *GridAPPS-D Application Structure*), the PowerGrid Models API is used for controlling the simulation, subscribing to measurement data, and controlling equipment.

This section covers only the portion of the API used for starting, stopping, and pausing simulations. Equipment control commands are covered in *Controlling Simulation with the Simulation API*



19.2 API Syntax Overview

Application passes subscription request to GridAPPS-D Platform

The subscription request is perfromed by passing the app core algorithm function / class definition to the `gapps.subscribe` method. The application then passes the subscription request through the Simulation API to the topic channel for the particular simulation on the GOSS Message Bus. If the application is authorized to access simulation output, the subscription request is delivered to the Simulation Manager.

GridAPPS-D Platform delivers published simulation output to Application

Unlike the previous queries made to the various databases, the GridAPPS-D Platform does not provide any immediate response back to the application. Instead, the Simulation Manager will start delivering measurement data back to the application through the Simulation API at each subsequent timestep until the simulation ends or the application unsubscribes. The measurement data is then passed to the core algorithm class / function, where it is processed and used to run the app's optimization / control algorithms.

19.2.1 API Communication Channel

The Simulation API is used to communicate with a broad range of subscribers through both static `/queue/` (the type of channel used for passing databse queries) and dynamic `/topic/` communication channel names.

Extreme care is needed to use the correct communication channel. For a review of GridAPPS-D topics, see *API Communication Channels*

The correct communication channel for each Simulation API call will be provided in the corresponding section for each API task below.

19.2.2 Structure of a Simulation Message

Due to the wide range of tasks accomplished by the Simualtion API, there is no single message format that is used across all API Calls.

Each message takes the form of a python dictionary or equivalent JSON script wrapped as a python string.

The structure of each Simulation API call message will be provided in the corresponding section for each API task below.

19.3 Starting a Simulation

This section explains the API call used to start simulations. For most application developers, typical use cases would be either

- starting a digital twin simulation from within an app for "what-if" analysis
- building an automated test script that would launch a simulation with exactly identical parameters each time.

The API message structure and start command are also used by the GridAPPS-D Viz when starting a simulation using the steps in *Creating a Simulation with GridAPPS-D Viz*. However, the API call and detail simulation start message are hidden from the user when using Viz.

19.3.1 GridAPPS-D Communication Channel

This is a static `/queue/` communication channel used to quest the simulation to start. The base static string used is `goss.gridappsd.process.request.simulation`, which can be called using the `.REQUEST_SIMULATION` method from the topics library.

When developing in python, it is recommended to use the `.REQUEST_SIMULATION` method. When using the STOMP client in GridAPPS-D VIZ, it is necessary to use the base static string.

[ ]:
```
from gridappsd import topics as t
topic = t.REQUEST_SIMULATION
```

19.3.2 Simulation Start Message Structure

To start a simulation from the command line or from inside an application, it is necessary to pass a message with a structure similar to that used by the *Configuration File API*) so that the GridAPPS-D platform can create a custom GridLab-D file from the CIM XML model stored in the Blazegraph Database.

The accepted set of key-value pairs for the Simulation API call to start a new simulation is

```
message = {
    "power_system_config": {
        "key1": "value1",
        "key2": "value2"},
    "simulation_config": {
        "key1": "value1",
        "key2": "value2",
        "model_creation_config": {
            "key1": "value1",
            "key2": "value2",
            "model_state": {
                "key1": "value1",
                "key2": "value2"}
            }
         },
    "simulation_output": {
        "key1": "value1",
        "key2": "value2"},
    "application_config": [{
        "key1": "value1",
        "key2": "value2"}],
    "test_config": {
        "events": [{,
            "message": {
                "forward_differences": [{
                    "key1": "value1",
                    "key2": "value2"}]
                "reverse_differences": [{
                    "key1": "value1",
                    "key2": "value2"}],
            "key1": "value1",
            "key2": "value2"}
            }]
    },
    "service_configs": {
        "key1": "value1",
        "key2": "value2"}
}
```

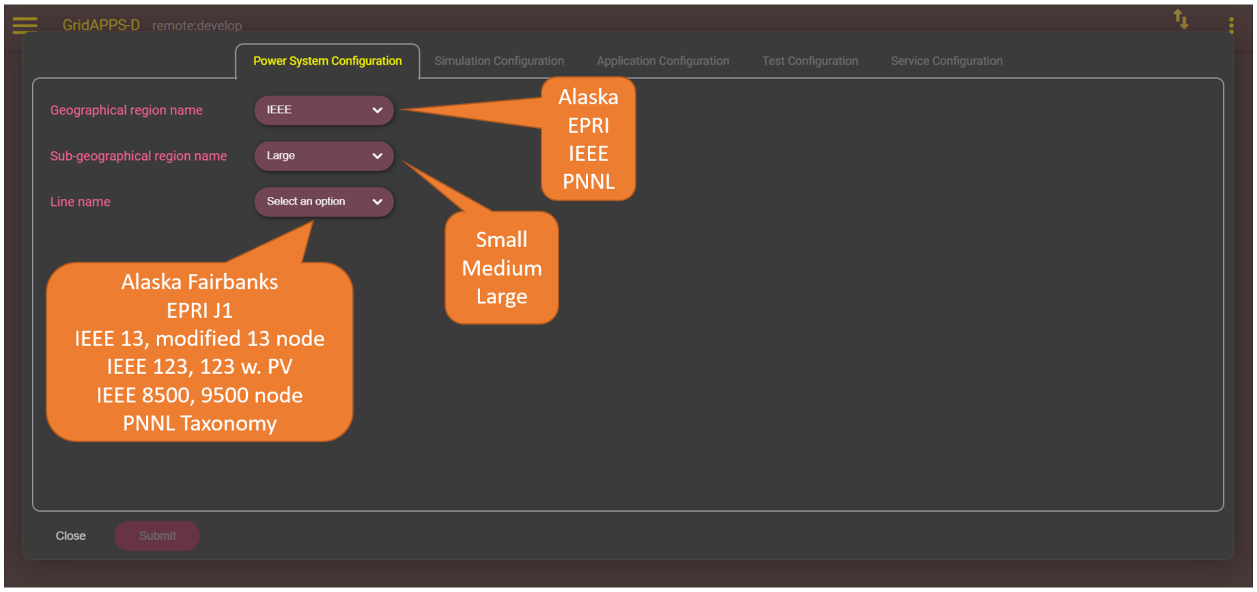
The components of the message are identical to the options in the tabs in the GridAPPS-D. Each key-value pair is explained in detail below.

19.3.3 Power System Configuration

This key is **required** and specifies the CIM XML model to be imported and used for the simulation. Required key-value pairs are

```
"power_system_config": {   REQUIRED KEY                          REQUIRED VALUE
                     "GeographicalRegion_name":          mRID as string ,
                     "SubGeographicalRegion_name":       mRID as string ,
                     "Line_name":                        mRID as string }
```

As can be seen, the required key-value pairs are identical to those that are selected graphically using the GridAPPS-D Viz:


GridAPPS-D remote:develop
Power System Configuration
Simulation Configuration
Application Configuration
Test Configuration
Service Configuration
Close
Submit

Geographical region name: IEEE
Sub-geographical region name: Large
Line name: Select an option

19.3.4 Simulation Configuration

The `"simulation_config":` key is **required** and specifies the runtime parameters, such as simulation duration and time step. Required key-value pairs are

```
"simulation_config": {   REQUIRED KEY                          REQUIRED VALUE
                    "start_time":                    epoch time string ,
                    "duration":                      number (seconds) ,
                    "simulator":                     "GridLAB-D" ,
                    "timestep_frequency":            number (milliseconds) ,
                    "timestep_increment":            number (milliseconds) ,
                    "run_realtime":                  true OR false ,
                    "simulation_name":               string ,
                    "power_flow_solver_method":      "NR" ,
                    "model_creation_config":         key-value pairs }
```

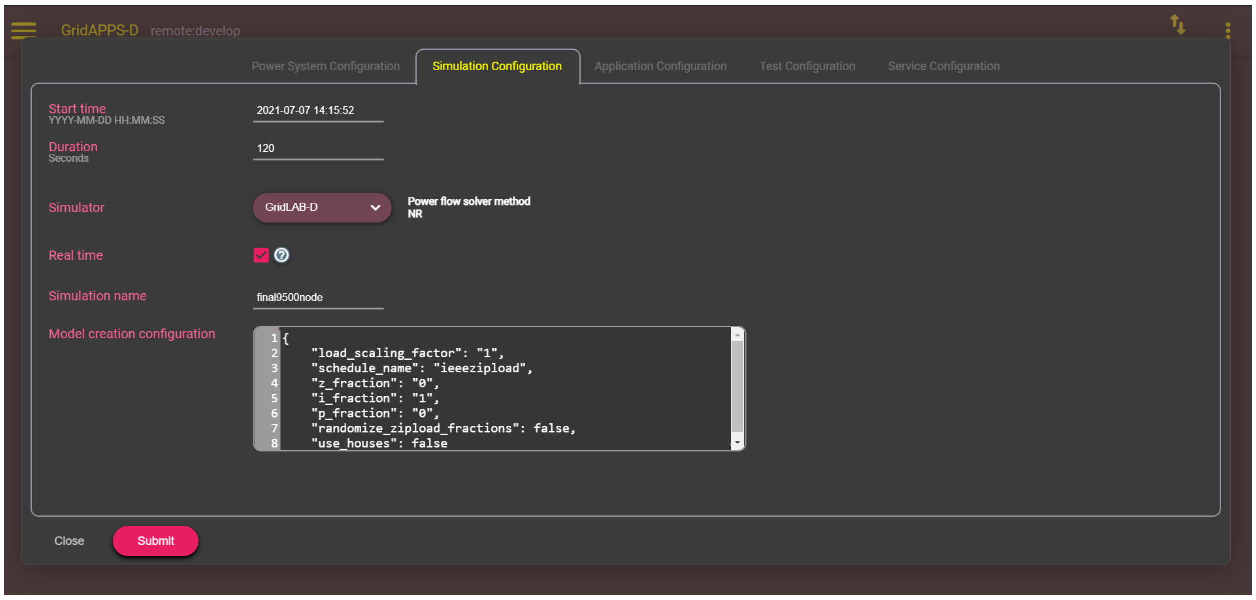
The key `"model_creation_config":` is one of the **optional** key-value pairs that can specified. This specifies the load profile and changes to the base case used for the simulation. Key-value pairs are

```
"model_creation_config": { KEY                            VALUE
                    "load_scaling_factor":                  number ,
                    "schedule_name":                        string ,
                    "z_fraction":                           number ,
                    "i_fraction":                           number ,
                    "p_fraction":                           number ,
                    "randomize_zipload_fractions":          true OR false ,
                    "use_houses":                           true OR false ,
                    "model_state":                          key-value pairs }
```

The key `"model_state:"` is one of the **optional** key-value pairs used to specify changes to switch positions and DER setpoints from the base case stored in the Blazegraph database. Key-value pairs are

```
"model_state": {
                "synchronousmachines":[
                        {"name": "GLM/DSS name","p": number,"q": number},
                        {"name": "GLM/DSS name","p": number,"q": number}],
                "switches":[
                        {"name":"GLM/DSS name", "open": true},
                        {"name":"GLM/DSS name", "open": false}]
    }
```

As can be seen, the required key-value pairs are identical to those that are selected graphically using the GridAPPS-D Viz. By default, only the required key-value pairs are listed in the viz. The simulation launched in the Viz can be configured using any of the `"model_creation_config"` and `"model_state"` key-value pairs.


GridAPPS-D remote:develop
Power System Configuration
Simulation Configuration
Application Configuration
Test Configuration
Service Configuration
Close
Submit

Start time
YYYY-MM-DD HH:MM:SS: 2021-07-07 14:15:52

Duration
Seconds: 120

Simulator: GridLAB-D

Power flow solver method
NR

Real time: ✓

Simulation name: final9500node

Model creation configuration:

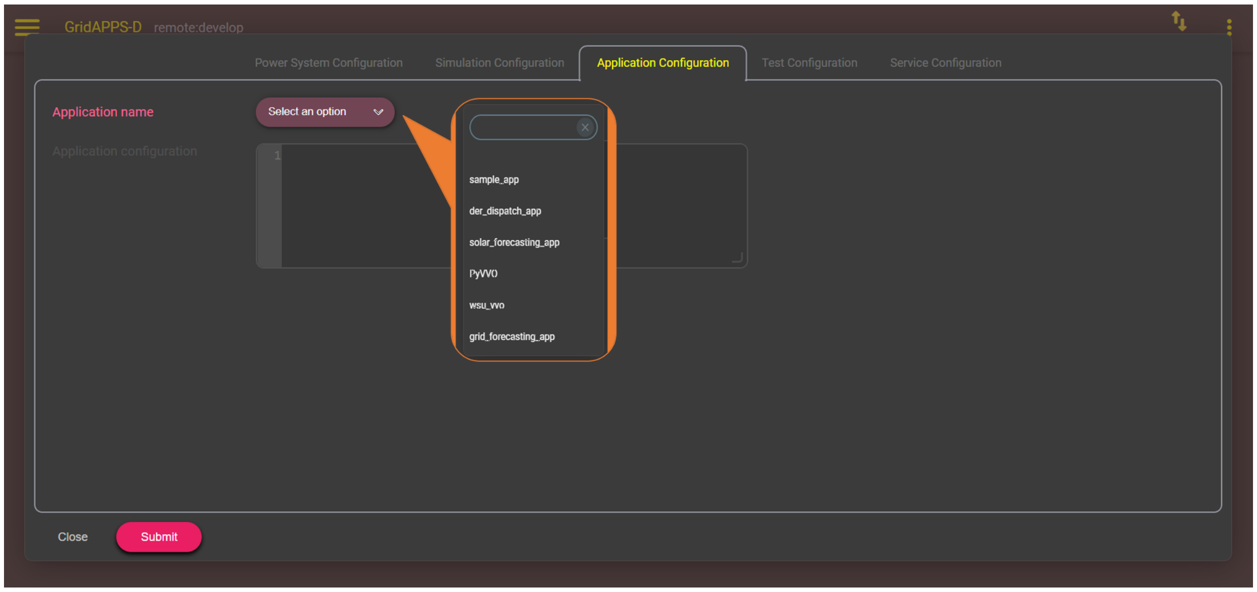
```
{
    "load_scaling_factor": "1",
    "schedule_name": "ieeezipload",
    "z_fraction": "0",
    "i_fraction": "1",
    "p_fraction": "0",
    "randomize_zipload_fractions": false,
    "use_houses": false
```

19.3.5 Application Configuration

The `"application_config":` key is **optional** and specifies which applications will run during the simulation. The associated value is a list that specifies the name and application-specific configuration parameters.

```
"application_config": {
        "applications": [
                {
                "name": "application1",
                "config_string": "application-specific config string"
                },
                {
                "name": "application2",
                "config_string": "application-specific config string"
                }]
}
```

As can be seen, the required key-value pairs are identical to those that are selected graphically using the GridAPPS-D Viz. Note: simulations started using the GridAPPS-D Viz can only run one app at a time. This is due to the GUI design and not a limitation of the Simulation API.

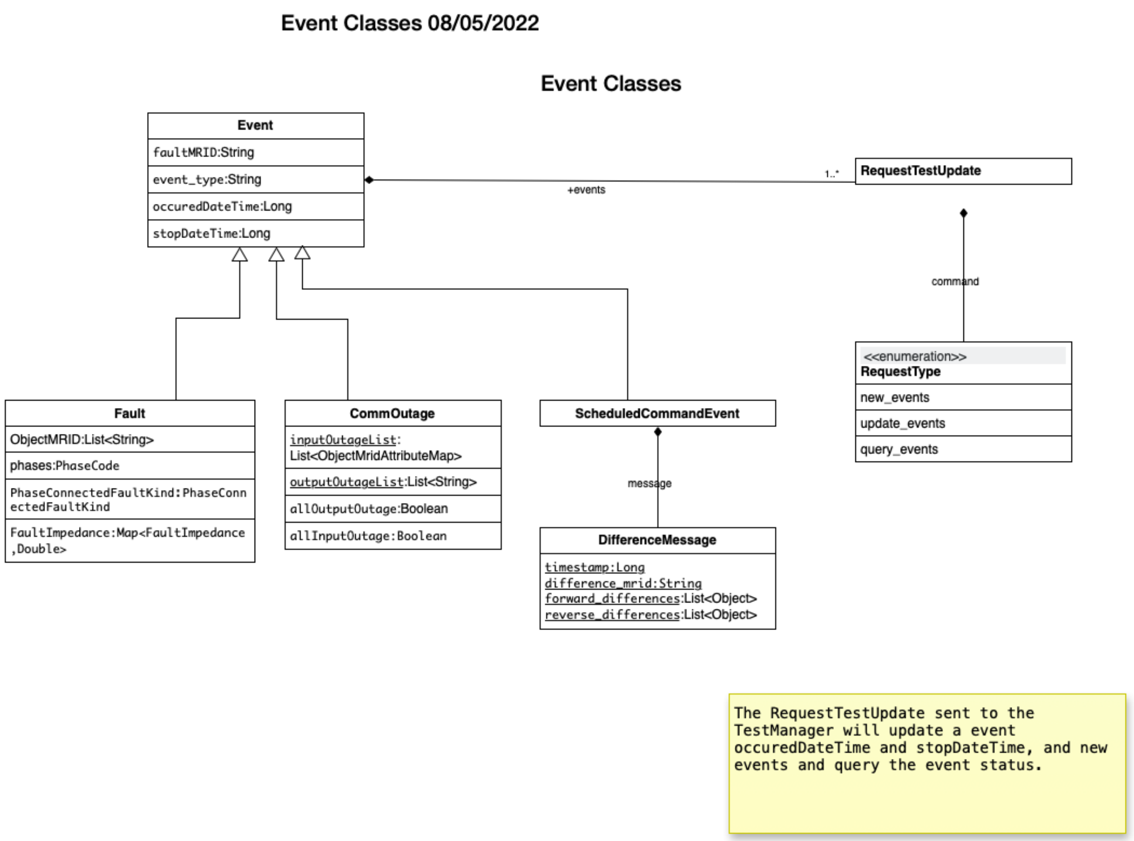


19.3.6 Test Manager Configuration

The TestManager is used to create realistic operational events during the simulation run, including faults, communication outages, and other unexpected equipment actions. These types of events can be used to test robustness of applications in variable conditions similar to those of the real world.

There are three types events supported by the TestManager and the GridAPPS-D Platform:

1. CIM defined fault events, used when a line is down or for taking a piece of equipment out of service.
2. Communication outage events which simulates measurements or control message outages.
3. Scheduled command at specific time which sends commands to a piece of equipment.



When starting a simulation, any combination of event categories can be specified using the `"test_config":` key-value pair structure. This key is **optional** and specifies whether any pre-scripted events should be scheduled to occur during the simulation, such as faults, communication outages, and equipment malfunctions.

The TestManager configuration can be written within the text of the Simulation API call or imported from a JSON file. Each of the event categories are discussed in further detail below.

```
"test_config": {
    "events": [
        {
            "event_type": "CommOutage",
            "tag": "unique_tag",
            "startDateTime": "YYYY-MM-DD HH:MM:SS" OR epoch time number,
            "stopDateTime": "YYY-MM-DD HH:MM:SS" OR epoch time number,
            "allInputOutage": true or false,
            "inputList": [
                {
                    "name": "equipment name",
                    "type": "CIM equipment type",
                    "mRID": "equipment object mRID",
                    "phases": [
                        {
                            "phaseLabel": "A" or "B" or "C",
                            "phaseIndex": number
                        }
                    ],
```

(continues on next page)

(continued from previous page)

```
                "attribute": "CIM Control Attribute"
            }
        ],
        "allOutputOutage": true or false,
        "outputList": [
            {
                "name": "equipment name",
                "type": "CIM equipment type",
                "mRID": "equipment object mRID",
                "phases": [ "A" or "B" or "C"  ],
                "measurementTypes": [ "PNV" or "VA" or "POS" ]
            }
        ]
    },
    {
        "event_type": "Fault",
        "tag": "unique_tag",
        "equipmentType": "CIM object type",
        "equipmentName": "equipment name",
        "mRID": [
            "equipment object mRID",
            "equipment object mRID",
            "equipment objhect mRID"
        ],
        "phases": [
            {
                "phaseLabel": "A" or "B" or "C",
                "phaseIndex": number
            }
        ],
        "faultKind": "lineToGround" or "lineToLine" or "lineToLineToGround",
        "faultImpedance": {
            "rGround": "value",
            "xGround": "value"
        },
        "startDateTime": "YYYY-MM-DD HH:MM:SS" OR epoch time number,
        "stopDateTime": "YYYY-MM-DD HH:MM:SS" OR epoch time number
    },
    {
        "message": {
            "forward_differences": [
                {
                    "object": "control object mRID",
                    "attribute": "CIM Class Attribute",
                    "value": number
                }
            ],
            "reverse_differences": [
                {
                    "object": "control object mRID",
                    "attribute": "CIM Class Attribute",
                    "value": number
                }
            ]
        },
        "event_type": "ScheduledCommandEvent",
        "occuredDateTime": "YYYY-MM-DD HH:MM:SS" OR epoch time number,
```

(continues on next page)

(continued from previous page)

```
            "stopDateTime": "YYYY-MM-DD HH:MM:SS" OR epoch time number
        }
    ]
}
```

Fault Events

Fault Events are defined in a Test Script and define the CIM Fault events that will be intialized and cleared at scheduled times.

The phases string is all the combinations of the 3-phases plus neutral and secondary phases. Some examples are :

1. "A"
2. "AN" would be line to line.
3. "AB" would be line to line.
4. "S12N" both hot wires to ground
5. "S12" both hot wires together.

PhaseConnectedFaultKind is an enumeration with the following values:

1. lineToGround
2. lineToLine
3. lineToLineToGround
4. lineOpen

The key-value pairs for specifying a fault event are:

```
{
    "PhaseConnectedFaultKind": string,
    "FaultImpedance": {
        "rGround": float,
        "xGround": float,
        "rLineToLine":float,
        "xLineToLine":float },
    "ObjectMRID": [string],
    "phases": string,
    "event_type": string,
    "occuredDateTime": long,
    "stopDateTime": long
}
```

Example 1: Fault Event Example in test script

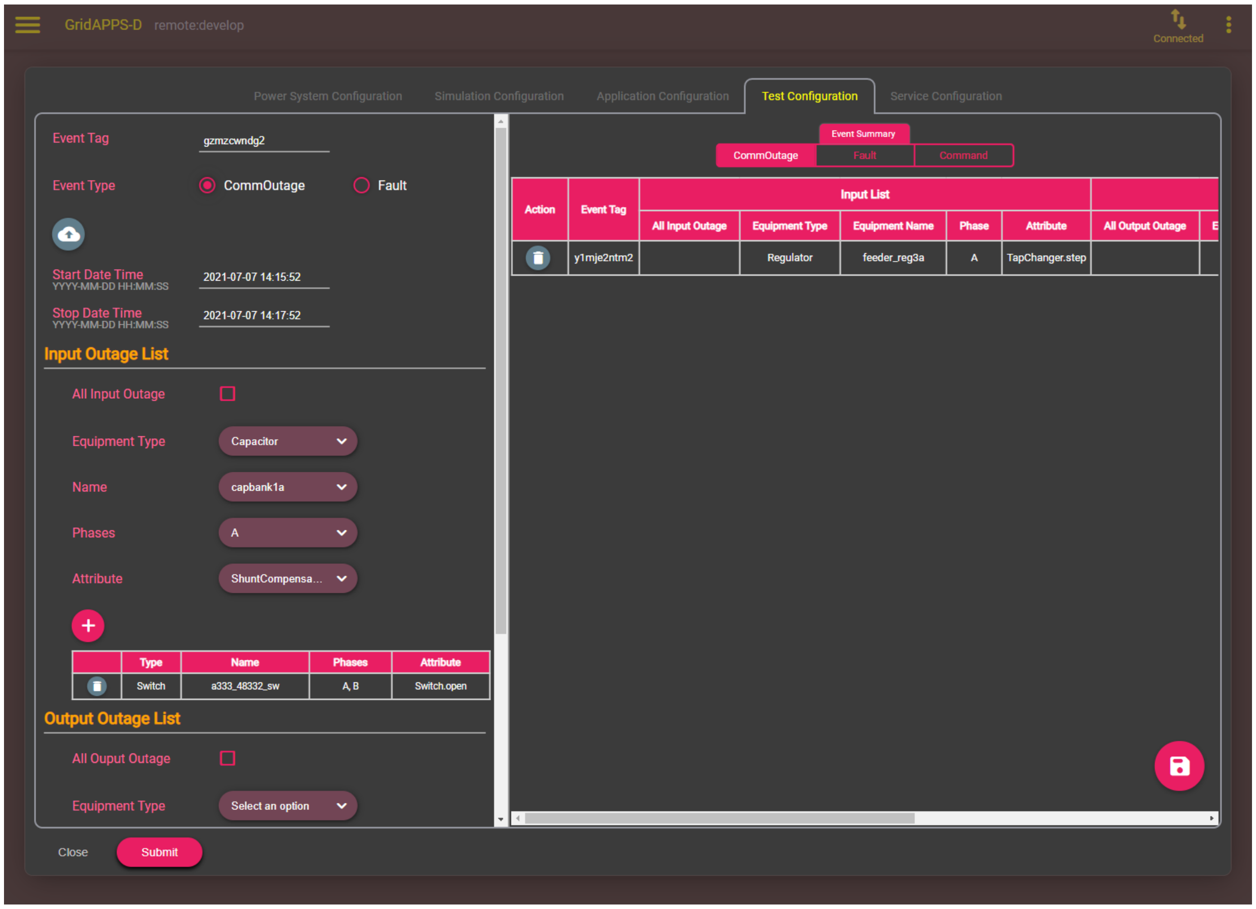
```
{
    "command": "new_events",
    "events" : [{
        "PhaseConnectedFaultKind": "lineToGround",
        "FaultImpedance": {
            "rGround": 0.001,
            "xGround": 0.001 },
        "ObjectMRID": ["_9EF94B67-7279-21F4-5CEE-B2724E3C3FE6"],
        "phases": "ABC",
```

(continues on next page)

(continued from previous page)

```
        "event_type": "Fault",
        "occuredDateTime": 1248130809,
        "stopDateTime": 1248130816}
    ]
}
```

As can be seen, the required key-value pairs are identical to those that are selected graphically using the GridAPPS-D Viz. The Viz offers the equivalent options to create line faults, communications outages, or upload a custom difference message JSON file.



Event Tag: gzmzcwndg2

Event Type: [✓ CommOutage] Fault

Start Date Time (YYYY-MM-DD HH:MM:SS): 2021-07-07 14:15:52

Stop Date Time (YYYY-MM-DD HH:MM:SS): 2021-07-07 14:17:52

Input Outage List

All Input Outage: ☐

Equipment Type: Capacitor

Name: capbank1a

Phases: A

Attribute: ShuntCompensa...

	Type	Name	Phases	Attribute
🗑	Switch	a333_48332_sw	A, B	Switch.open

Output Outage List

All Ouput Outage: ☐

Equipment Type: Select an option

Action	Event Tag	Input List						
		All Input Outage	Equipment Type	Equipment Name	Phase	Attribute	All Output Outage	E
🗑	y1mje2ntm2		Regulator	feeder_reg3a	A	TapChanger.step		

Example 2: Commands sent from the Test Manager to the simulation

```
{
    "command": "update",
    "input": {
        "timestamp": 1553201000414,
        "reverse_differences": [],
        "difference_mrid": "_ee4e4055-222f-4ccf-bed1-93063bd4392c",
        "forward_differences": [
        {
            "ObjectMRID": "12344",
            "FaultImpedance": {
                "xLineToLine": 0.0,
                "rGround": 0.001,
                "rLineToLine": 0.0,
```

(continues on next page)

(continued from previous page)

```
                "xGround": 0.001
                },
            "FaultMRID": "1233",
            "PhaseCode": "AN",
            "PhaseConnectedFaultKind": "lineToGround"
            }
        ]
    }
}
```

Example 3: Clear a Fault Command example

```
{
    "command": "update",
    "input": {
        "timestamp": 1553201003561,
        "reverse_differences": [{
            "ObjectMRID": "12344",
            "FaultImpedance": {
                "xLineToLine": 0.0,
                "rGround": 0.001,
                "rLineToLine": 0.0,
                "xGround": 0.001
            },
            "FaultMRID": "1233",
            "PhaseCode": "AN",
            "PhaseConnectedFaultKind": "lineToGround"
        }
    ],
    "difference_mrid": "_00b4668d-8454-4f1c-aed9-42d1424af149",
    "forward_differences": []
    }
}
```

Communication Outage Events

Communication Outage events are separate from the CIM events but have occuredDateTime and stopDateTime.

1. The inputOutageList is the list of objectMRID and attribute pair. The objectMRID is anything that can be controllable and specific control attribute i.e. “RegulatingControl.mode”.
2. The outputOutageList is the list of MRIDs for the measurement device that is associated to the .
3. If allInputOutage is True the inputOutageList is not needed as all inputs to the simulator are blocked.
4. If allOutputOutage is True the outputOutageList is not needed as all outputs from the simulator are blocked.

The key-value pairs for communication outage events follow the format

```
{
    "allOutputOutage": boolean,
    "allInputOutage": boolean,
    "inputOutageList": [{"objectMRID":string, "attribute":string}],
    "outputOutageList": [string],
    "event_type": string,
```

(continues on next page)

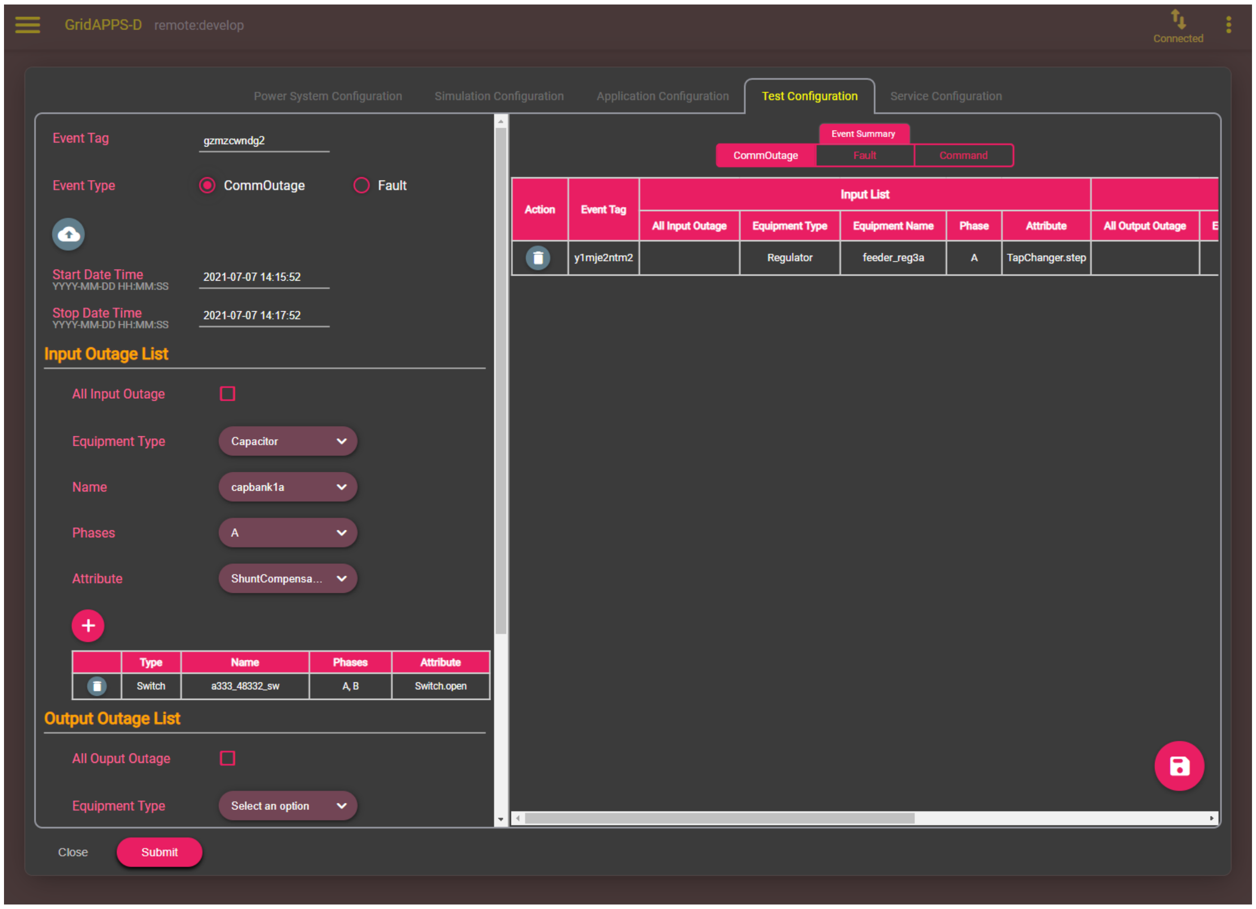
(continued from previous page)

```
    "occuredDateTime": long,
    "stopDateTime": long
}
```

Example 1: JSON Communication Outage command for the TestManager

```
{
    "command": "new_events",
    "events": [{
        "allOutputOutage": false,
        "allInputOutage": false,
        "inputOutageList": [{
            "objectMRID":"_EF2FF8C1-A6A6-4771-ADDD-A371AD929D5B",
            "attribute":"ShuntCompensator.sections" },
            {"objectMRID":"_C0F73227-012B-B70B-0142-55C7C991A343",
            "attribute":"ShuntCompensator.sections"}],
        "outputOutageList": ["_5405BE1A-BC86-5452-CBF2-BD1BA8984093"],
        "event_type": "CommOutage",
        "occuredDateTime": 1248130819,
        "stopDateTime": 1248130824
        }
    ]
}
```

As can be seen, the required key-value pairs are identical to those that are selected graphically using the GridAPPS-D Viz. The Viz offers the equivalent options to create line faults, communications outages, or upload a custom difference message JSON file.



Event Tag gzmzcwndg2

Event Type [✓ CommOutage] Fault

Start Date Time YYYY-MM-DD HH:MM:SS 2021-07-07 14:15:52

Stop Date Time YYYY-MM-DD HH:MM:SS 2021-07-07 14:17:52

Input Outage List

All Input Outage ☐

Equipment Type Capacitor

Name capbank1a

Phases A

Attribute ShuntCompensa...

	Type	Name	Phases	Attribute
	Switch	a333_48332_sw	A, B	Switch.open

Output Outage List

All Ouput Outage ☐

Equipment Type Select an option

Action	Event Tag	Input List						E
		All Input Outage	Equipment Type	Equipment Name	Phase	Attribute	All Output Outage	E
	y1mje2ntm2		Regulator	feeder_reg3a	A	TapChanger.step		

Example 2: Communication Event to the Simulation Bridge

```
{
    "command": "CommOutage",
    "input": {
        "timestamp": 1248130819,
        "forward_differences": [
        {
            "allOutputOutage": false,
            "allInputOutage": false,
            "inputOutageList": [
            {
                "objectMRID": "_EF2FF8C1-A6A6-4771-ADDD-A371AD929D5B",
                "attribute": "ShuntCompensator.sections"
            },
            {
                "objectMRID": "_C0F73227-012B-B70B-0142-55C7C991A343",
                "attribute": "ShuntCompensator.sections"
            }
            ],
            "outputOutageList": [
            "_5405BE1A-BC86-5452-CBF2-BD1BA8984093"
            ],
            "faultMRID": "_ce5ee4c9-9c41-4f5e-8c5c-f19990f9cfba",
            "event_type": "CommOutage",
            "occuredDateTime": 1248130819,
            "stopDateTime": 1248130824
```

(continues on next page)

(continued from previous page)

```
        }
        ],
        "reverse_differences": []
    }
}
```

Scheduled Command Events

Schedule Commands are events that can be scheduled for a specific point in time of the simulation. This can be used to trigger a fault like behavior, such as changing the taps of a regular or mimicking behavior of protective devices.

Faults in topologically meshed networks need to created using a Scheduled Command event to open all switches that would be tripped by protective relaying due to a known bug in the GridLAB-D logical fault processor.

The set of key-value pairs for Scheduled Command events follow the format below. The message parameters follow the format of a difference message, which is explained in detail in *Format of Difference Messages*.

```
{
    "command": "new_events",
    "events":[{
        "message":{
            "forward_differences": [{
                "object": "control object mRID",
                "attribute": "CIM Class Attribute",
                "value": number }],
            "reverse_differences": [{
                "object": "control object mRID",
                "attribute": "CIM Class Attribute",
                "value": number }]
            },
        },
        "occuredDateTime":long,
        "stopDateTime":long,
        }
    ]
}
```

19.3.7 Service Configuration

The `"service_configs":` key is used to start a GridAPPS-D service, such as the Sensor Service, State Estimator, or DNP3 Service. The key-value pairs for starting the service depend on the particular service and will be covered in the documentation for each GridAPPS-D Service. Note that the syntax of this key is a plural noun, unlike the other config keys.

- DNP3 Service
- Sensor Simulator Service
- State Estimator Service

19.3.8 Complete Simulation Start Message

The complete simulation message that is passed to the Simulation API is a Python dictionary / JSON string that specified the `"power_system_conf":`, `"application_config":`, `"simulation_config":`, `"test_config":`, and `"service_configs":` key-value pairs.

A sample simulation start message for the IEEE 123 bus system running the Sensor Simulator service and including one Scheduled Command event is shown below.

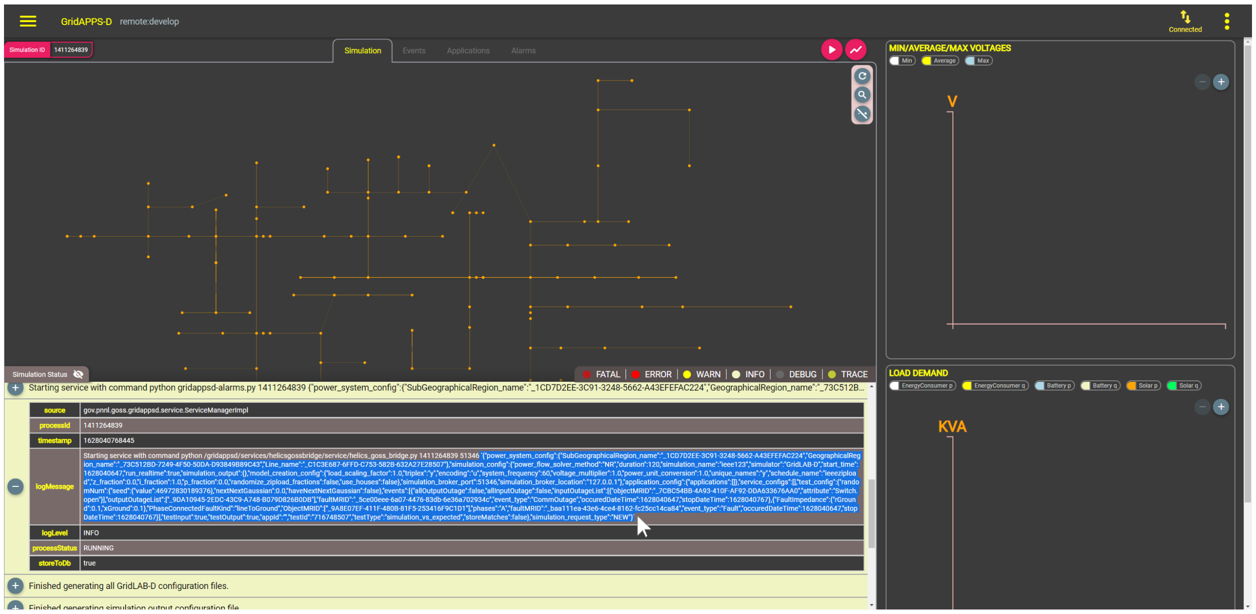
[ ]:
```
run_config_123 = {
    "power_system_config": {
        "GeographicalRegion_name": "_73C512BD-7249-4F50-50DA-D93849B89C43",
        "SubGeographicalRegion_name": "_1CD7D2EE-3C91-3248-5662-A43EFEFAC224",
        "Line_name": "_C1C3E687-6FFD-C753-582B-632A27E28507"
    },
    "application_config": {
        "applications": []
    },
    "simulation_config": {
        "start_time": "1570041113",
        "duration": "120",
        "simulator": "GridLAB-D",
        "timestep_frequency": "1000",
        "timestep_increment": "1000",
        "run_realtime": True,
        "simulation_name": "ieee123",
        "power_flow_solver_method": "NR",
        "model_creation_config": {
            "load_scaling_factor": "1",
            "schedule_name": "ieeezipload",
            "z_fraction": "0",
            "i_fraction": "1",
            "p_fraction": "0",
            "randomize_zipload_fractions": False,
            "use_houses": False
        }
    },
    "test_config": {
        "events": [{
            "message": {
                "forward_differences": [
                    {
                        "object": "_6C1FDA90-1F4E-4716-BC90-1CCB59A6D5A9",
                        "attribute": "Switch.open",
                        "value": 1
                    }
                ],
                "reverse_differences": [
                    {
                        "object": "_6C1FDA90-1F4E-4716-BC90-1CCB59A6D5A9",
                        "attribute": "Switch.open",
                        "value": 0
                    }
                ]
            },
            "event_type": "ScheduledCommandEvent",
            "occuredDateTime": 1570041140,
            "stopDateTime": 1570041200
```

(continues on next page)

(continued from previous page)

```
            }]
        },
         "service_configs": [{
            "id": "gridappsd-sensor-simulator",
            "user_options": {
                "sensors-config": {
                    "_99db0dc7-ccda-4ed5-a772-a7db362e9818": {
                        "nominal-value": 100,
                        "perunit-confidence-band": 0.02,
                        "aggregation-interval": 5,
                        "perunit-drop-rate": 0.01
                    },
                    "_ee65ee31-a900-4f98-bf57-e752be924c4d": {},
                    "_f2673c22-654b-452a-8297-45dae11b1e14": {}
                },
                "random-seed": 0,
                "default-aggregation-interval": 30,
                "passthrough-if-not-specified": False,
                "default-perunit-confidence-band": 0.01,
                "default-perunit-drop-rate": 0.05
            }
        }]
}
```

The simulation start message can be built from within the application or copied from the GridAPPS-D Viz by starting a new simulation and copying the start message from the log file. Pausing the simulation immediately after it starts is extremely helpful in locating the start message and copying it.



```
Starting service with command python gridappsd-alarms.py 1411264839 {"power_system_config":{"SubGeographicalRegion_name":"_1CD7D2EE-3C91-3248-5662-A43EFEFAC224","GeographicalRegion_name":"_73C512B...
```

source	gov.pnnl.goss.gridappsd.service.ServiceManagerImpl
processId	1411264839
timestamp	1628040768445
logMessage	Starting service with command python /gridappsd/services/helicsgossbridge/service/helics_goss_bridge.py 1411264839 51346 '{"power_system_config":{"SubGeographicalRegion_name":"_1CD7D2EE-3C91-3248-5662-A43EFEFAC224","GeographicalRegion_name":"_73C512BD-7249-4F50-50DA-D93849B89C43","Line_name":"_C1C3E687-6FFD-C753-582B-632A27E28507"},"simulation_config":{"power_flow_solver_method":"NR","duration":120,"simulation_name":"ieee123","simulator":"GridLAB-D","start_time":1628040647,"run_realtime":true,"simulation_output":{},"model_creation_config":{"load_scaling_factor":1.0,"triplex":"y","encoding":"u","system_frequency":60,"voltage_multiplier":1.0,"power_unit_conversion":1.0,"unique_names":"y","schedule_name":"ieeezipload","z_fraction":0.0,"i_fraction":1.0,"p_fraction":0.0,"randomize_zipload_fractions":false,"use_houses":false},"simulation_broker_port":51346,"simulation_broker_location":"127.0.0.1"},"application_config":{"applications":[]},"service_configs":[],"test_config":{"randomNum":{"seed":{"value":46972830189376},"nextNextGaussian":0.0,"haveNextNextGaussian":false},"events":[{"allOutputOutage":false,"allInputOutage":false,"inputOutageList":[{"objectMRID":"_7CBC54BB-4A93-410F-AF92-DDA633676AA0","attribute":"Switch.open"}],"outputOutageList":["_9DA10945-2EDC-43C9-A748-B079D826B0DB"],"faultMRID":"_5ce00eee-6a07-4476-83db-6e36a702934c","event_type":"CommOutage","occuredDateTime":1628040647,"stopDateTime":1628040767},{"FaultImpedance":{"rGround":0.1,"xGround":0.1},"PhaseConnectedFaultKind":"lineToGround","ObjectMRID":["_9A8E07EF-411F-480B-81F5-253416F9C1D1"],"phases":"A","faultMRID":"_baa111ea-43e6-4ce4-8162-fc25cc14ca84","event_type":"Fault","occuredDateTime":1628040647,"stopDateTime":1628040767}],"testInput":true,"testOutput":true,"appId":"","testId":"716748507","testType":"simulation_vs_expected","storeMatches":false},"simulation_request_type":"NEW"}'
logLevel	INFO
processStatus	RUNNING
storeToDb	true

```
Finished generating all GridLAB-D configuration files.
Finished generating simulation output configuration file
```

19.3.9 Starting the Simulation

To start the simulation, import the `gridappsd.simulation` library, which provides multiple shortcut functions for running and controlling a simulation.

The simulation start message python dictionary created in the previous section is then passed as an argument to Simulation object.

The `.start_simulation()` method is then used to pass the Simulation API call to the GOSS Message Bus and the GridAPPS-D Platform.

The simulation id can be obtained by invoking the `.simulation_id` method.

```
[ ]: from gridappsd.simulation import Simulation # Import Simulation Library

     simulation_obj = Simulation(gapps_sim, run_config_123) # Create Simulation object
     simulation_obj.start_simulation() # Start Simulation

     simulation_id = simulation_obj.simulation_id # Obtain Simulation ID
     print("Successfully started simulation with simulation_id: ", simulation_id)
```

The simulation start message can also be saved as a JSON file that is loaded using the `json.load("filename.json")` method. The simulation is then started by passing the start message to the `.start_simulation()` method.

```
[ ]: import json
     from gridappsd.simulation import Simulation # Import Simulation Library

     run123_config = json.load(open("Run123NodeFileSimAPI.json")) # Pull simulation start␣
     ↪message from saved file

     simulation_obj = Simulation(gapps, run123_config) # Create Simulation object
     simulation_obj.start_simulation() # Start Simulation

     simulation_id = simulation_obj.simulation_id # Obtain Simulation ID
     print("Successfully started simulation with simulation_id: ", simulation_id)
```

19.4 Pausing, Resuming, or Stopping a Simulation

Whether running a simulation through the GridAPPS-D Viz or a parallel digital twin simulation started from within your application, it is frequently useful to be able to pause, resume, or stop the simulation.

19.4.1 Using the gridappsd.simulation Python Library

For simulations that are started using the `.start_simulation()` method of the `gridappsd.simulation` python library, it is possible to use the associated methods to pause, resume, and stop simulations.

The library provides the following methods:

- `.pause()` – Pause the simulation
- `.resume()` – Resume the simulation
- `.resume_pause_at(pause_time)` – Resume the simulation, and then pause it in so many seconds
- `.stop()` – Stop the simulation

- `.run_loop()` – Loop the entire simulation until interrupted

19.4.2 Using a Topic + Message API Call

If the simulation was started using the GridAPPS-D Viz, then pause, resume, and stop commands need to be issued using an API call to the Simulation API specifying the topic and message.

Specifying the Topic

This is a dynamic `/topic/` communication channel that is best implemented by importing the GriAPPSD-Python library function for generating the correct topic.

```
[ ]: viz_simulation_id = "paste sim_id here"
```

```
[ ]: from gridappsd.topics import simulation_input_topic
     topic = simulation_input_topic(viz_simulation_id)
```

Pause Message

The simulation can be paused using a very simple pause message. Since no response is expected from the Simulation API, we used the `.send(topic, message)` method for the GridAPPS-D connection object.

```
[ ]: message = {"command": "pause"}
     gapps.send(topic, message)
```

Resume Message

The simulation can be resumed using a very simple message. Since no response is expected from the Simulation API, we used the `.send(topic, message)` method for the GridAPPS-D connection object.

```
[ ]: message = {"command": "resume"}
     gapps.send(topic, message)
```

Resume then Pause Message

The simulation can be resumed and then paused using a very simple message. Since no response is expected from the Simulation API, we used the `.send(topic, message)` method for the GridAPPS-D connection object.

```
[ ]: message = {
         "command": "resumePauseAt",
         "input": {"pauseIn": 10}
     }
     gapps.send(topic, message)
```

|GridAPPS-D_narrow.png|

CHAPTER

TWENTY

PUBLISHING AND SUBSCRIBING WITH THE SIMULATION API

20.1 Introduction to the Simulation API

The Simulation API is used for all actions related to a power system simulation. It is used to start, pause, restart, and stop a simulation from the command line or inside an application. It is all used to subscribe to measurements of equipment (such as line flows, loads, and DG setpoints) and the statuses of switches, capacitors, transformer taps, etc. It is also used to publish equipment control and other simulation input commands.

In the Application Components diagram (explained in detail with sample code in *GridAPPS-D Application Structure*), the PowerGrid Models API is used for controlling the simulation, subscribing to measurement data, and controlling equipment.

This section covers only the portion of the API used for subscribing to measurements and publishing equipment control commands. Usage of the API for starting, stopping, and pausing simulations is covered in *Creating and Running Simulations with Simulation API*

20.2 Processing Measurements & App Core Algorithm

The central portion of a GridAPPS-D application is the measurement processing and core algorithm section. This section is built as either a class or function definition with prescribed arguments. Each has its advantages and disadvantages:

- The function-based approach is simpler and easier to implement. However, any parameters obtained from other APIs or methods to be used inside the function currently need to be defined as global variables.
- The class-based approach is more complex, but also more powerful. It provides greater flexibility in creating additional methods, arguments, etc.

20.2.1 App Core Information Flow

This portion of the application does not communicate directly with the GridAPPS-D platform.

Instead, the next part of the GridAPPS-D application (*Subscribing to Simulation Output*) delivers the simulated SCADA measurement data to the core algorithm function / class definition. The core algorithm processes the data to extract the desired measurements and run its optimization / control agorithm.

20.2.2 Structure of Simulation Output Message

The first part of the application core is parsing simulated SCADA and measurement data that is delivered to the application.

The general format of the messages received by the Simulation API is a python dictionary with the following key-value pairs:

```
{
    "simulation_id" :           string,
    "message" : {
        "timestamp" :           epoch time number,
        "measurements" : {
            "meas mrid 1":{
                                "PNV measurement_mrid": "meas mrid 1"
                                "magnitude": number,
                                "angle": number },
            "meas mrid 2":{
                                "VA measurement_mrid": "meas mrid 2"
                                "magnitude": number,
                                "angle": number },
            "meas mrid 3":{
                                "Pos measurement_mrid": "meas mrid 3"
                                "value": number },
                           .
                           .
                           .
            "meas mrid n":{
                                "measurement_mrid": "meas mrid n"
                                "magnitude": number,
                                "angle": number },
    }
}
```

20.2.3 Format of Measurement Values

In the message above, note the difference in the key-value pair structure of different types of measurements:

PNV Voltage Measurements

These are specified as `magnitude` and `angle` key-value pairs.

- Magnitude is the RMS phase-to-neutral voltage.
- Angle is phase angle of the voltage at the particular node.

VA Volt-Ampere Apparent Power Measurements

These are specified as `magnitude` and `angle` key-value pairs. * Magnitude is the apparent power. * Angle is complex power triangle angle (i.e. *acos(power factor)*)

Pos Position Measurements

These are specified as a `value` key-value pair. * Value is the position of the particular measurement * For switch objects: value = 1 means “closed”, value = 0 means “open” * For capacitor objects, values are reversed: value = 1 means “on”, value = 0 means “off” * For regulator objects, value is the tap position, ranging from -16 to 16

20.2.4 Role of Measurement mRIDs

The simulation output message shown above typically contains the measurement mRIDs for all available sensors for all equipment in the power system model. The application needs to filter the simulation output message to just the set of measurements relevant to the particular application (e.g. switch positions for a FLISR app or regulator taps for a VVO app).

The equipment and measurement mRIDs are obtained in the first two sections of the application. See *Query for Power System Model* and *Query for Measurement mRIDs* for examples of how these code sections fit in a sample app.

API syntax details for the query messages to PowerGrid Models API to obtain equipment info and measurement mRIDs are given in *Query for Object Dictionary* and *Query for Measurements*.

These mRIDs will be needed to parse the simulation output message and filter it to just the desired set of measurements.

For the example below, we will be interested in only the measurement associated with switches, so we will use the PowerGrid Models API to query for the set of measurements associated with the CIM Class `LoadBreakSwitch`. We then will filter those values to just the mRIDs associated with each type of measurement.

[ ]:
```python
from gridappsd import topics as t

# Create query message to obtain measurement mRIDs for all switches
message = {
    "modelId": model_mrid,
    "requestType": "QUERY_OBJECT_MEASUREMENTS",
    "resultFormat": "JSON",
    "objectType": "LoadBreakSwitch"
}

# Pass query message to PowerGrid Models API
response_obj = gapps.get_response(t.REQUEST_POWERGRID_DATA, message)
measurements_obj = response_obj["data"]

# Switch position measurements (Pos)
Pos_obj = [k for k in measurements_obj if k['type'] == 'Pos']

# Switch phase-neutral-voltage measurements (PNV)
PNV_obj = [k for k in measurements_obj if k['type'] == 'PNV']

# Switch volt-ampere apparent power measurements (VA)
VA_obj = [k for k in measurements_obj if k['type'] == 'VA']

# Switch current measurements (A)
A_obj = [k for k in measurements_obj if k['type'] == 'A']
```

20.2.5 App Core as a Function Definition

The first approach used to build the application core is to define a function with the correct set of arguments that is then passed to the `.subscribe()` method associated with the `GridAPPPSD()` object.

The function does not require a specific name, and is somewhat easier to define and use. However, the arguments of the function need to be named correctly for the GridAPPSD-Python library to process the simulation output correctly.

The format for the function definition is

```
def mySubscribeFunction(header, message):
    # do something when receive a message
    # parse to get measurments
    # do some calculations
    # publish some equipment commands
    # display some results
```

That function handle is then passed as an argument to the `.subscribe(topic, function_handle)` method when subscribing to the simulation in the next section.

Note that the subscription function definition does not allow any additional parameters to be passed. The only allowed arguments are `header` and `message`.

Any other parameters, such as measurement mRIDs will need to be defined as global variables.

```
[ ]: # Define global python dictionary of position measurements
global Pos_obj
Pos_obj = [k for k in measurements_obj if k['type'] == 'Pos']

# Define global python dictionary of phase-neutral-voltage measurements (PNV)
global PNV_obj
PNV_obj = [k for k in measurements_obj if k['type'] == 'PNV']

# Define global python dictionary of volt-ampere apparent power measurements (VA)
VA_obj = [k for k in measurements_obj if k['type'] == 'VA']

# Current measurements (A)
A_obj = [k for k in measurements_obj if k['type'] == 'A']
```

Below is the sample code for the core section of a basic application that tracks the number of open switches and the number of switches that are outaged.

```
[ ]: # Only allowed arguments are `header` and `message`
# message is simulation output message in format above
def DemoAppCoreFunction(header, message):

    # Extract time and measurement values from message
    timestamp = message["message"]["timestamp"]
    meas_value = message["message"]["measurements"]

    # Obtain list of all mRIDs from message
    meas_mrid = list(meas_value.keys())

    # Example 1: Count the number of open switches
    open_switches = []
    for index in Pos_obj:
        if index["measid"] in meas_value:
            mrid = index["measid"]
            power = meas_value[mrid]
            if power["value"] == 0: # Filter to measurements with value of zero
                open_switches.append(index["eqname"])

    # Print message to command line
    print("............")
    print("Number of open switches at time", timestamp, ' is ', len(set(open_
↪switches)))
```

(continues on next page)

(continued from previous page)

```
    # Example 2: Count the number of outaged switches (voltage = 0)
    dead_switches = []
    for index in PNV_obj:
        if index["measid"] in meas_value:
            mrid = index["measid"]
            voltage = meas_value[mrid]
            if voltage["magnitude"] == 0.0:
                dead_switches.append(index["eqname"])

    # Print message to command line
    print("...........")
    print("Number of outaged switches at time", timestamp, ' is ', len(set(dead_
↪switches)))
```

20.2.6 App Core as a Class Definition

The second approach used to build the app core and process measurements is to define a class containing two methods named `__init__` and `on_message`.

These methods specify 1) how your app would initialize variables and attributes at the start of the simulation and 2) how your app behaves when it receives various messages.

IMPORTANT! The GridAPPS-D Platform uses the exact names and syntax for the methods:

- `__init__(self, simulation_id, gapps_object, optional_objects)` – This method requires the simulation_id and GridAPPS-D connection object. It is also possible add other user-defined arguments, such as measurement mRIDs or other information required by your application.
- `on_message(self, headers, message)` – This method allows the class to subscribe to simulation measurements. It also contains the core behavior of your application and how it responds to each type of message.

It is also possible to use the same class definition to subscribe to other topics, such as Simulation Logs. This is done by creating additional user-defined methods and then passing those methods to the `.subcribe()` method associated with the GridAPPS-D connection object. An example of how this is done is provided for subcribing to simulation logs in Logging with a Class Method.

```
class YourSimulationClassName(object):
    # Your documentation text here on what app does

    def __init__(self, simulation_id, gapps_obj, meas_obj, your_obj):
        # Instantiate class with specific initial state

        # Attributes required by Simulation API
        self._gapps = gapps_obj
        self._simulation_id = simulation_id

        # Attributes to publish difference measurements
        self.diff = DifferenceBuilder(simulation_id)

        # Custom attributes for measurements, custom info
        self.meas_mrid = meas_obj
        self.your_attribute1 = your_obj["key1"]
        self.your_attribute2 = your_obj["key2"]
```

(continues on next page)

(continued from previous page)

```
    def on_message(self, headers, message):
        # What app should do when it receives a subscription message

        variable1 = message["message"]["key1"]
        variable2 = message["message"]["key2"]

        # Insert your custom app behavior here
        if variable1 == foo:
            bar = my_optimization_result

        # Insert your custom equipment commands here
        if variable2 == bar:
            self.diff.add_difference(object_mrid, control_attribute, new_value, old_
↪value)

    def my_custom_method_1(self, headers, message):
        # Use extra methods to subscribe to other topics, such as simulation logs
        variable1 = message["key1"]
        variable2 = message["key2"]

    def my_custom_method_2(self, param1, param2):
        # Use extra methods as desired
        variable1 = foo
        variable2 = bar
```

Below is the sample code for the core section of a basic application that tracks the number of open switches and the number of switches that are outaged.

[ ]:
```
# Application core built as a class definition
class DemoAppCoreClass(object):

    # Subscription callback from GridAPPSD object
    def __init__(self, simulation_id, gapps_obj, meas_obj):
        self._gapps = gapps_obj # GridAPPS-D connection object
        self._simulation_id = simulation_id # Simulation ID
        self.meas_mrid = meas_obj # Dictionary of measurement mRIDs obtained earlier

    def on_message(self, headers, message):

        # Extract time and measurement values from message
        timestamp = message["message"]["timestamp"]
        meas_value = message["message"]["measurements"]

        # Filter measurement mRIDs for position and voltage sensors
        Pos_obj = [k for k in self.meas_mrid if k['type'] == 'Pos']
        PNV_obj = [k for k in self.meas_mrid if k['type'] == 'PNV']

         # Example 1: Count the number of open switches
        open_switches = []
        for index in Pos_obj:
            if index["measid"] in meas_value:
                mrid = index["measid"]
                power = meas_value[mrid]
                if power["value"] == 0: # Filter to measurements with value of zero
                    open_switches.append(index["eqname"])
```

(continues on next page)

(continued from previous page)

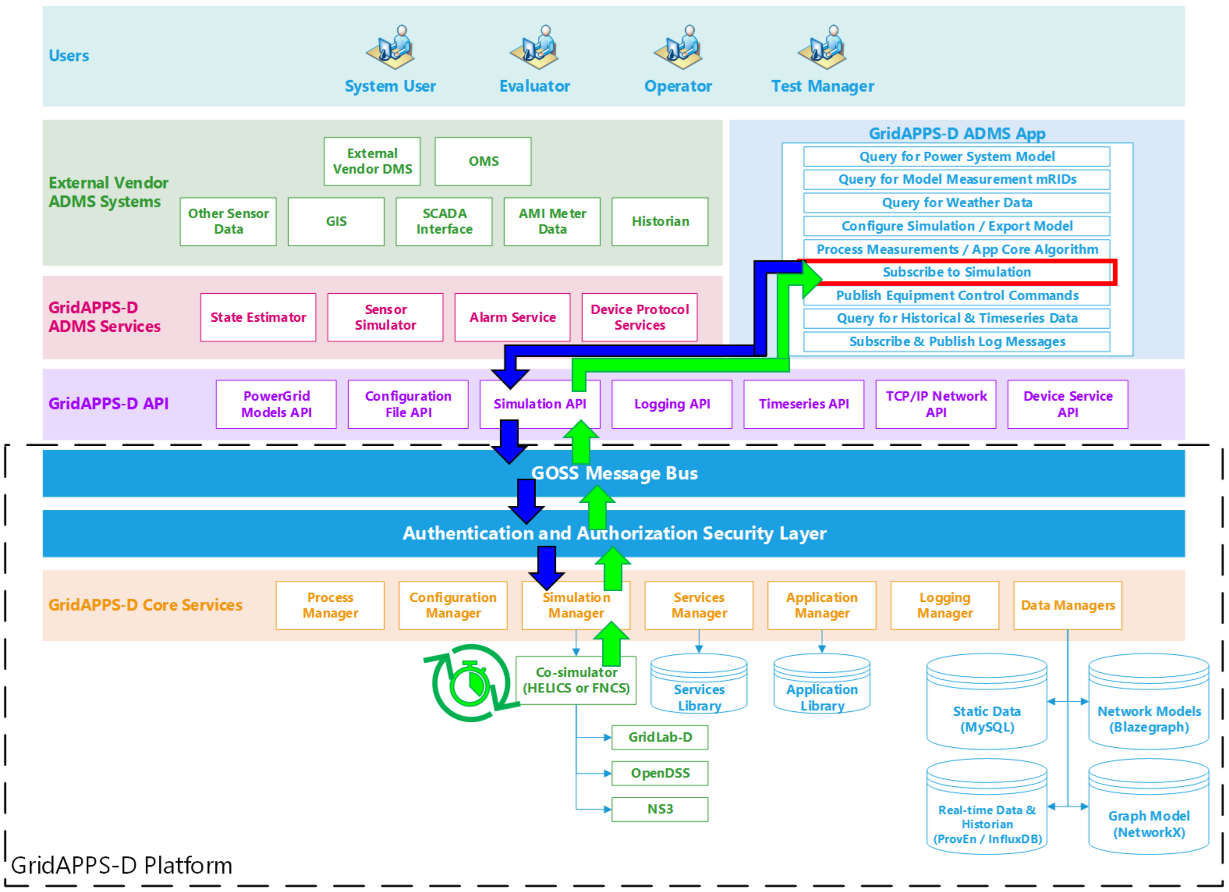
```
        # Print message to command line
        print("...........")
        print("Number of open switches at time", timestamp, ' is ', len(set(open_
↪switches)))


        # Example 2: Count the number of outaged switches (voltage = 0)
        dead_switches = []
        for index in PNV_obj:
            if index["measid"] in meas_value:
                mrid = index["measid"]
                voltage = meas_value[mrid]
                if voltage["magnitude"] == 0.0:
                    dead_switches.append(index["eqname"])

        # Print message to command line
        print("...........")
        print("Number of outaged switches at time", timestamp, ' is ', len(set(dead_
↪switches)))
```

20.3 Subscribing to Simulation Output

20.3.1 Simulation Subscription Information Flow

The figure below shows the information flow involved in subscribing to the simulation output.

The subscription request is sent using `gapps.subscribe(topic, class/function object)` on the specific Simulation topic channel (explained in *API Communication Channels*). No immediate response is expected back from the platform. However, after the next simulation timestep, the Platform will continue to deliver a complete set of measurements back to the application for each timestep until the end of the simulation.



Application passes subscription request to GridAPPS-D Platform

The subscription request is perfromed by passing the app core algorithm function / class definition to the `gapps.subscribe` method. The application then passes the subscription request through the Simulation API to the topic channel for the particular simulation on the GOSS Message Bus. If the application is authorized to access simulation output, the subscription request is delivered to the Simulation Manager.

GridAPPS-D Platform delivers published simulation output to Application

Unlike the previous queries made to the various databases, the GridAPPS-D Platform does not provide any immediate response back to the application. Instead, the Simulation Manager will start delivering measurement data back to the application through the Simulation API at each subsequent timestep until the simulation ends or the application unsubscribes. The measurement data is then passed to the core algorithm class / function, where it is processed and used to run the app's optimization / control algorithms.

20.3.2 Subscription API Communication Channel

This is a dynamic `/topic/` communication channel that is best implemented by importing the GriAPPSD-Python library function for generating the correct topic. This communication channel is used for all simulation subscription API calls.

```
[ ]: from gridappsd.topics import simulation_output_topic

     output_topic = simulation_output_topic(viz_simulation_id)
```

20.3.3 Comparison of Subscription Approaches

Each approach has its advantages and disadvantages.

- The function-based approach is simpler and easier to implement. However, any parameters obtained from other APIs or methods to be used inside the function currently need to be defined as global variables.
- The class-based approach is more complex, but also more powerful. It provides greater flexibility in creating additional methods, arguments, etc.
- The Simulation Library-based approach is easiest, but only works currently for parallel digital twin simulations started using the `simulation_obj.start_simulation()` method.

The choice of which approach is used depends on the personal preferences of the application developer.

20.3.4 Subscription for Function-based App Core

If the application core was created as a function definition as shown in *App Core as Function Definition*, then the function name is passed to the `.subscribe(output_topic, core_function)` method of the GridAPPS-D Connection object.

```
[ ]: conn_id = gapps.subscribe(output_topic, DemoAppCoreFunction)
```

20.3.5 Subscription for Class-Based App Core

If the application core was created as a class definition as shown in *App Core as Class Definition*, then the function name is passed to the `.subscribe(output_topic, object)` method of the GridAPPS-D connection object.

After defining the class for the application core as shown above, we create another object that will be passed to the subscription method. The required parameters for this object are the same as those defined for the `__init__()` method of the app core class, typically the Simulation ID, GridAPPS-D connection object, dictionary of measurements needed by the app core, and any user-defined objects.

`class_obj = AppCoreClass(simulation_id, gapps_obj, meas_obj, your_obj)`

```
[ ]: demo_obj = DemoAppCoreClass(viz_simulation_id, gapps, measurements_obj)

     conn_id = gapps.subscribe(subscribe_topic, demo_obj)
```

If we wish to subscribe to an additional topic (such as the Simulation Logs, a side communication channel between two different applications, or a communication with a particular service), we can define an additional method in the class (such as my_custom_method_1 in the *example class definition* above) and then pass it to to the `.subscribe(topic, object.method)` method associated with the GridAPPS-D connection object:

`gapps.subscribe(other_topic, demo_obj.my_custom_method_1)`

20.4 Subscribing to Parallel Simulations

Parallel simulations started using the Simulation API (as shown in *Starting a Simulation*) and the `Simulation` library in GridAPPSD-Python do not need to use the `gapps.subscribe` method.

Instead, the GridAPPSD-Python library contains several shortcut functions that can be used. These methods currently cannot interact with a simulation started from the Viz. This functionality will be added in a future release.

The code block below shows how a parallel simulation can be started using a simulation start message stored in a JSON file. The simulation is started using the `.start_simulation()` method.

```
[ ]: import json, os
     from gridappsd import GridAPPSD
     from gridappsd.simulation import Simulation

     # Connect to GridAPPS-D Platform
     os.environ['GRIDAPPSD_USER'] = 'tutorial_user'
     os.environ['GRIDAPPSD_PASSWORD'] = '12345!'
     gapps = GridAPPSD()
     assert gapps.connected

     model_mrid =  "_C1C3E687-6FFD-C753-582B-632A27E28507"
     run123_config = json.load(open("Run123NodeFileSimAPI.json")) # Pull simulation config␣
     ↪from saved file
     simulation_obj = Simulation(gapps, run123_config) # Create Simulation object
     simulation_obj.start_simulation() # Start Simulation

     print("Successfully started simulation with simulation_id: ", simulation_obj.
     ↪simulation_id)
```

```
[ ]: simulation_id = simulation_obj.simulation_id
```

The Simulation library provides four methods that can be used to define how the platform interacts with the simulation:

- `.add_ontimestep_callback(myfunction1)` – Run the desired function on each timestep
- `.add_onmesurement_callback(myfunction2)` – Run the desired function when a measurement is received.
- `.add_oncomplete_callback(myfunction3)` – Run the desired function when simulation is finished
- `.add_onstart_callback(myfunction4)` – Run desired function when simulation is started

Note: method name ``.add_onmesurement_callback`` is misspelled in the library definition!!

Note that the measurement callback method returns just the measurements and timestamps without any of the message formatting used in the messages received by using the `gapps.subscribe(output_topic, object)` approach.

The python dictionary returned by the GridAPPS-D Simulation output to the `.add_onmesurement_callback()` method is always named `measurements` and uses the following key-value pairs format:

```
{
    '_pnv_meas_mrid_1': {'angle': number,
                         'magnitude': number,
                         'measurement_mrid': '_pnv_meas_mrid_1'},
    '_va_meas_mrid_2': { 'angle': number,
                         'magnitude': number,
                         'measurement_mrid': '_va_meas_mrid_2'},
    '_pos_meas_mrid_3': {'measurement_mrid': '_pos_meas_mrid_3',
                         'value': 1},
            .
            .
            .
    '_pnv_meas_mrid_n': {'angle': number,
                         'magnitude': number,
                         'measurement_mrid': '_pnv_meas_mrid_1'}
}
```

To use use these methods, we define a set of functions that determine the behavior of the application for each of the four types of callbacks listed above. These functions are similar to those defined for the function-based app core algorithm.

```
def my_onstart_func(sim):
    # Do something when the simulation starts
    # Do something else when the sim starts

simulation_obj.add_onstart_callback(my_onstart_func)
```

```
def my_onmeas_func(sim, timestamp, measurements):
    # Do something when app receives a measurement
    # Insert your custom app behavior here
    if measurements[object_mrid] == foo:
        bar = my_optimization_result

simulation_obj.add_onmesurement_callback(my_onmeas_func)
```

```
def my_oncomplete_func(sim):
    # Do something when simulation is complete
    # example: delete all variables, close files

simulation_obj.add_oncomplete_callback(my_oncomplete_func)
```

The code block below shows how the same app core algorithm can be used for a parallel simulation using the `.add_onmesurement_callback()` method:

```
[ ]: def demo_onmeas_func(sim, timestamp, measurements):

         open_switches = []
         for index in Pos_obj:
             if index["measid"] in measurements:
                 mrid = index["measid"]
                 power = measurements[mrid]
                 if power["value"] == 0:
                     open_switches.append(index["eqname"])

         print("............")
         print("Number of open switches at time", timestamp, ' is ', len(set(open_
↪switches)))
```

[ ]:
```
simulation_obj.add_onmesurement_callback(demo_onmeas_func)
```

20.5 Publishing Commands to Simulation Input

The next portion of a GridAPPS-D App is publishing equipment control commands based on the optimization results or objectives of the app algorithm.

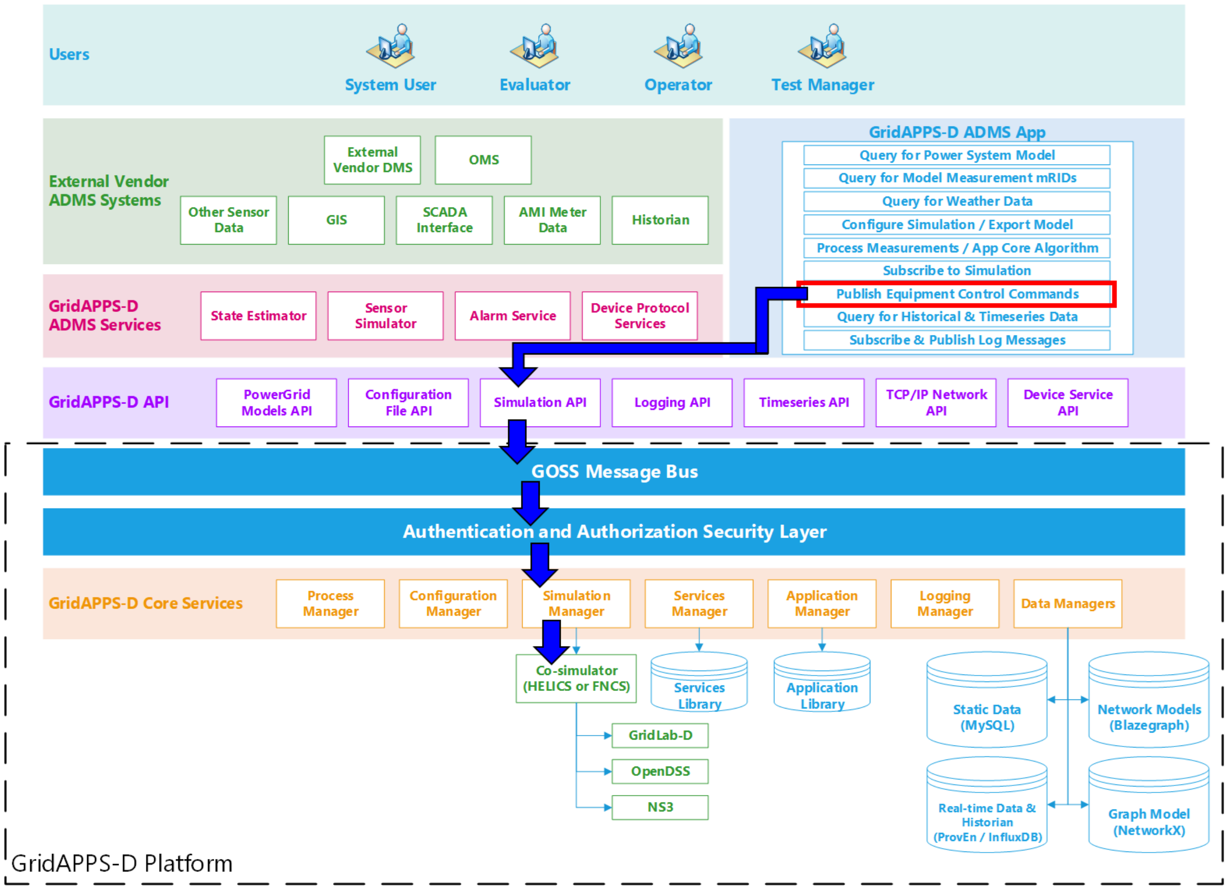
Depending on the preference of the developer, this portion can be a separate function definition, or included as part of the main class definition as part of the *App Core as a Class Definition* described earlier.

20.5.1 Equipment Command Information Flow

The figure below outlines information flow involved in publishing equipment commands to the simulation input.

Unlike the various queries to the databases in the app sections earlier, equipment control commands are passed to the GridAPPS-D API using the `gapps.send(topic, message)` method. No response is expected from the GridAPPS-D platform.

If the application desires to verify that the equipment control command was received and implemented, it needs to do so by 1) checking for changes in the associated measurements at the next timestep and/or 2) querying the Timeseries Database for historical simulation data associated with the equipment control command.



Application sends difference message to GridAPPS-D Platform

First, the application creates a difference message containing the current and desired future control point / state of the particular piece of power system equipment to be controlled. The difference message is a JSON string or equivalant Python dictionary object. The syntax of a difference message is explained in detail below in *Format of Difference Message*.

The application then passes the query through the Simulation API to the GridAPPS-D Platform, which publishes it on the topic channel for the particular simulation on the GOSS Message Bus. If the app is authenticated and authorized to control equipment, the difference message is delivered to the Simulation Manager. The Simulation Manager then passes the command to the simulation through the Co-Simulation Bridge (either FNCS or HELICS).

No response from GridAPPS-D Platform back to Application

The GridAPPS-D Platform does not provide any response back to the application after processing the difference message and implementing the new equipment control setpoint.

20.5.2 Simulation Input API Channel

This is a dynamic `/topic/` communication channel that is best implemented by importing the GriAPPSD-Python library function for generating the correct topic.

- `from gridappsd.topics import simulation_input_topic`
- `input_topic = simulation_input_topic(simulation_id)`

[ ]:
```python
from gridappsd.topics import simulation_input_topic

input_topic = simulation_input_topic(viz_simulation_id)
```

20.5.3 Equipment Control mRIDs

The mRIDs for controlling equipment are generally the same as those obtained using the `QUERY_OBJECT_DICT` key with the PowerGrid Models API, which was covered in *Query for Object Dicionary*.

However, the control attributes for each class of equipment in CIM use a different naming convention than those for the object types. Below is a list of `"objectType"` used to query for mRIDs and the associated control attribute used in a difference message for each category of power system equipment:

- **Switches**
  - CIM Class Key: `"objectType": "LoadBreakSwitch"`
  - Control Attribute: `"attribute": "Switch.open"`
  - Values: `1` is open, `0` is closed
- **Capacitor Banks:**
  - CIM Class Key: `"objectType": "LinearShuntCompensator"`
  - Control Attribute: `"attribute": "ShuntCompensator.sections"`
  - Values: `0` is off/open, `1` is on/closed
- **Inverter-based DERs:**
  - CIM Class Key: `"objectType": "PowerElectronicsConnection"`

    - Control Attribute: `"attribute": "PowerElectronicsConnection.p"`
    - Control Attribute: `"attribute": "PowerElectronicsConnection.q"`
    - Values: number in Watts or VArs (not kW)

- **Synchronous Rotating (diesel) DGs:**
    - CIM Class Key: `"objectType": "SynchronousMachine"`
    - Control Attribute: `"attribute": "RotatingMachine.p"`
    - Control Attribute: `"attribute": "RotatingMachine.q"`
    - Values: number in Watts or VArs (not kW)

- **Regulating Transformer Tap:**
    - CIM Class Key: `"objectType": "RatioTapChanger"`
    - Control Attribute: `"attribute": "TapChanger.step"`
    - Values: integer value for tap step

The query for RatioTapChanger is not supported in the PowerGrid Models API at the current time. A custom SPARQL query needs to be done using the sample query in `CIMHub Sample Queries <https://github.com/GRIDAPPSD/CIMHub/blob/master/queries.txt>`__

The example below shows a query to obtain the correct mRID for switch SW2 in the IEEE 123 node model:

[ ]:
```
from gridappsd import topics as t

message = {
    "modelId": model_mrid,
    "requestType": "QUERY_OBJECT_DICT",
    "resultFormat": "JSON",
    "objectType": "LoadBreakSwitch"
}

response_obj = gapps.get_response(t.REQUEST_POWERGRID_DATA, message)
switch_dict = response_obj["data"]

# Filter to get mRID for switch SW2:
for index in switch_dict:
    if index["IdentifiedObject.name"] == 'sw2':
        sw_mrid = index["IdentifiedObject.mRID"]
```

20.5.4 Format of a Difference Message

The general format for a difference message is a python dictionary or equivalent JSON string that specifies the reverse difference and the forward difference, in compliance with the CIM standard:

The **reverse difference** is the current status / value associated with the control attribute. It is a formatted as a list of dictionary constructs, with each dictionary specifying the equipment mRID associated with the CIM class keys above, the control attribute, and the current value of that control attribute. The list can contain reverse differences for multiple pieces of equipment.

The **forward difference** is the desired new status / value associated with the control attribute. It is a formatted as a list of dictionary constructs, with each dictionary specifying the equipment mRID associated with the CIM class keys above, the control attribute, and the current value of that control attribute. The list can contain foward differences for multiple pieces of equipment.

```
message = {
  "command": "update",
  "input": {
      "simulation_id": "simulation id as string",
      "message": {
          "timestamp": epoch time number,
          "difference_mrid": "optional unique mRID for command logs",
          "reverse_differences": [{

                  "object": "first equipment mRID",
                  "attribute": "control attribute",
                  "value": current value
              },
              {

                  "object": "second equipment mRID",
                  "attribute": "control attribute",
                  "value": current value
              }
          ],
          "forward_differences": [{

                  "object": "first equipment mRID",
                  "attribute": "control attribute",
                  "value": new value
              },
              {

                  "object": "second equipment mRID",
                  "attribute": "control attribute",
                  "value": new value
              }
              ]
              }
      }
}
```

Note: The GridAPPS-D platform does not validate whether `"reverse_differences":` has the correct equipment control values for the current time. It is used just for compliance with the CIM standard.

20.5.5 Using GridAPPSD-Python DifferenceBuilder

The `DifferenceBuilder` class is a GridAPPSD-Python tool that can be used to automatically build the difference message with correct formatting.

First, import DifferenceBuilder from the GridAPPSD-Python Library and create an object that will be used to create the desired difference messages.

[ ]:
```
from gridappsd import DifferenceBuilder

my_diff_build = DifferenceBuilder(viz_simulation_id)
```

We then use two methods associated with the DifferenceBuilder object:

- `.add_difference(self, object_mrid, control_attribute, new_value, old_value)` – Generates a correctly formatted difference message.
- `.get_message()` – Saves the message as a python dictionary that can be published using `gapps.send(topic, message)`

```
[ ]: my_diff_build.add_difference(sw_mrid, "Switch.open", 1, 0) # Open switch given by sw_
     ↪mrid

     message = my_diff_build.get_message()
```

The difference message is then published to the GOSS Message Bus and the Simulation API using the `.send()` method associated with the GridAPPS-D connection object.

```
[ ]: gapps.send(input_topic, message)
```

20.6 Unsubscribing from a Simulation

To unsubscribe from a simulation, pass the connection ID to `.unsubscribe(conn_id)` method of the GridAPPS-D connection object. The connection ID was obtained earlier when subscribing using `conn_id = gapps.subscribe(topic, message)`.

```
[ ]: gapps.unsubscribe(conn_id)
```

|GridAPPS-D_narrow.png|

CHAPTER

TWENTYONE

USING THE TIMESERIES API

21.1 Introduction to the Timeseries API

The Timeseries API is used to query the Influx Database, which stores measurement data from simulations. The API calls can be used to * obtain weather data * obtain measurements from simuation data using measurement mRIDs * obtain equipments commands and other simulation input data * obtain simulated field data from the Sensor Service

21.2 API Syntax Overview

Application passes query to GridAPPS-D Platform

First, the application creates a query message for requesting information about the desired power system components in the format of a JSON string or equivalant Python dictionary object. The syntax of this message is explained in detail below.

The query is sent using the `gapps.get_response(topic, message)` method on the Timeseries queue channel with a response expected back from the GridAPPS-D platform within the specified timeout period.

The application then passes the query through the Timeseries API to the GridAPPS-D Platform, which publishes it to the `goss.gridappsd.process.request.data.timeseries` queue channel on the GOSS Message Bus. If the app is authenticated and authorized to pass queries, the query message is delivered to the Data Managers, which obtain the desired information from the Timeseries Influx Database.

GridAPPS-D Platform responds to Application query

The Data Managers then publish the response from the Timeseries Influx Database to the appropriate queue channel. The Timeseries API then returns the desired information back to the application as a JSON message or equivalant Python dictionary object.

21.2.1 API Communication Channel

API calls to the timeseries databases use a static `/queue/` channel, which was covered in Lesson 1.4. The topic can be specified as a text string or by importing the topics library:

Text String: The topic can be specified as a static string:

- `topic = "goss.gridappsd.process.request.data.timeseries"`
- `gapps.get_response(topic, message)`

GridAPPSD-Python Library Method: The correct topic can also be imported from the GridAPPSD-Python topics library:

- `from gridappsd import topics as t`
- `gapps.get_response(t.TIMESERIES, message)`

21.2.2 Structure of a Query Message

Queries passed to Timeseries API are formatted as python dictionaries or equivalent JSON scripts wrapped as a python string.

The accepted set of key-value pairs for the Timeseries API query message is

```
message = """
{
    "queryMeasurement": "INSERT QUERY HERE",
    "queryFilter": {"key1": "value1"
                   "key2": "value2"},
    "responseFormat": "JSON"
}
```

The components of the message are as follows:

- `"queryMeasurement":` – Specifies the type of measurement being requested. Allowed queryMeasurement values are listed in the next section.
- `"queryFilter":` – Filters the measurements to just the set values desired by the application. The set of allowed values depends on the value associated with queryMeasurement.
- `"responseFormat":` – Specifies the format of the response, can be `"JSON"`, `"CSV"`, or `"XML"`. (CAUTION: the Timeseries API uses the key *reponseFormat*, while the PowerGridModel API uses the key *resultFormat*. Using the wrong key for either API will result in a java.lang error.)

The usage of each of these message components are explained in detail with code block examples below.

Important: Be sure to pay attention to placement of commas (**,**) at the end of each line. Commas are placed at the end of each line *except* the last line. Incorrect comma placement will result in a JsonSyntaxException.

All of the queries are passed to the Timeseries API using the `.get_response(topic, message)` method for the GridAPPS-D platform connection variable.

21.2.3 Specifying the queryMeasurement value

Below are the allowable values associated with the `queryMeasurement` key, which are used to specify the type of data requested by each query. Executable code block examples are provided for each of the requests in the subsections below.

- `"queryMeasurement": "weather"` – *Query for weather data*
- `"queryMeasurement": "simulation"` – *Query for simulation output data* and *query for simulation intput data*
- `"queryMeasurement": "gridappsd-sensor-simulator"` – *Query for sensor service data*

21.3 Querying for Timeseries Data

This section outlines the details of key-value pairs for the possible queries associated with each value of the `queryMeasurement` key listed above.

21.3.1 Querying for Weather Data

Interpreting GridAPPS-D 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 during data 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
```

Querying Weather Data

GridAPPS-D contains one year of weather data (details given in next subsection), which can be obtained with the following query

Measurement request key-value pair:

- `"queryMeasurement": "weather"`

Allowed key-value pairs for `queryFilter`:

```
"queryFilter": {    KEY                      VALUE
                "startTime":             epoch time number,
                "endTime":               epoch time number,
                "AvgWindDirection":      number,
                "AvgWindSpeed":          number,
```

(continues on next page)

(continued from previous page)

```
                "Diffuse":                number,
                "DirectCH1":              number,
                "GlobalCM22":             number,
                "MST":                    number,
                "TowerDryBulbTemp":       number,
                "TowerRH":                number,
                "lat":                    string,
                "long":                   string,
                "place":                  string }
```

Not all of the `queryFilter` key-value pairs need to be used. These are filters that can be used to restrict the returned data to particular values.

Note: The Timeseries API currently does not support querying for a range of values (e.g. all data for temperature betweeen 25 and 30C). This functionality will be added in a future release. Queries are currently limited to a single value.

Example 1: query for just the data between a given ``"startTime"`` and ``"endTime"``:

```
[ ]: topic = "goss.gridappsd.process.request.data.timeseries" # Specify Timeseries API␣
     ↪GridAPPS-D topic

     # Use queryFilter of "startTime" and "endTime"
     message = {
         "queryMeasurement":"weather",
         "queryFilter":{"startTime":"1357048800000000",
                        "endTime":"1357048860000000"},
         "responseFormat":"JSON"
     }

     gapps.get_response(topic, message) # Pass API call
```

Example 2: Query for weather data of particular values:

```
[ ]: topic = "goss.gridappsd.process.request.data.timeseries" # Specify Timeseries API␣
     ↪GridAPPS-D topic

     # Use queryFilter of "TowerDryBulbTemp" and "AvgWindSpeed"
     message = {
         "queryMeasurement":"weather",
         "queryFilter":{"TowerDryBulbTemp": 30.0326,
                        "AvgWindSpeed": 7.4624},
         "responseFormat":"JSON"
     }

     gapps.get_response(topic, message) # Pass API call
```

21.3.2 Query for Simulation Output Data

All of the measurement output from a simulation is stored in the Influx Database and can be queried with the Timeseries API

Measurement request key-value pair:

- `"queryMeasurement": "simulation"`

Allowed key-value pairs for `queryFilter` for output data

```
"queryFilter": {    KEY                             VALUE
                "startTime":                    epoch time number ,
                "endTime":                      epoch time number ,
                "measurement_mrid":             string OR [array of string values] ,
                "simulation_id":                numeric string ,
                "hasSimulationMessageType":     "OUTPUT" OR "INPUT" ,
                "angle":                        number ,
                "magnitude":                    number }
```

In the Getting Started section of this tutorial, we started a demo simulation of the IEEE 123 node model. That simulation has now completed, and we can pass queries to the Timeseries API to retrieve measurements from the completed simulation.

Example 1: Query for all measurements between a ``"startTime"`` and ``"endTime"``

```
[ ]: topic = "goss.gridappsd.process.request.data.timeseries" # Specify Timeseries API␣
     ↪GridAPPS-D topic

     # Use queryFilter of "startTime" and "endTime"
     message = {
         "queryMeasurement": "simulation",
         "queryFilter": {
             "simulation_id": simulation_id,
             "startTime": "1570041130",
             "endTime": "1570041140"},
         "responseFormat": "JSON"
     }

     gapps.get_response(topic, message) # Pass API call
```

Example 2: Query for all measurements associated with list of measurement mRIDs

```
[ ]: topic = "goss.gridappsd.process.request.data.timeseries" # Specify Timeseries API␣
     ↪GridAPPS-D topic

     # Query for a particular set of measurments
     message = {
         "queryMeasurement":"simulation",
         "queryFilter":{"simulation_id": simulation_id,
                        "measurement_mrid":["_5efa022e-da12-4c33-b127-10186624a8f7","_
     ↪4c515f50-51df-494a-a860-653c0c874fe1"]},
         "responseFormat":"JSON"
     }

     gapps.get_response(topic, message) # Pass API call
```

```
[ ]: topic = "goss.gridappsd.process.request.data.timeseries" # Specify Timeseries API␣
     ↪GridAPPS-D topic

     # Query for a particular set of measurments
     message = {
         "queryMeasurement":"simulation",
         "queryFilter":{"simulation_id": "1313108762",
                         "measurement_mrid":"_63CDE28F-FFC9-4869-BC12-EC3244A056CA"},
         "responseFormat":"JSON"
     }

     gapps.get_response(topic, message) # Pass API call
```

21.3.3 Query for Simulation Input Data

All of the equipment control commands and other input data from a simulation is stored in the Influx Database and can be queried with the Timeseries API

Measurement request key-value pair:

- `"queryMeasurement": "simulation"`

Allowed key-value pairs for `queryFilter` for output data

```
"queryFilter": {    KEY                             VALUE
                "startTime":                        epoch time number ,
                "endTime":                          epoch time number ,
                "measurement_mrid":                 mRID string OR [array of string␣
↪values] ,
                "simulation_id":                    numeric string ,
                "hasSimulationMessageType":         "OUTPUT" OR "INPUT" ,
                hasMeasurementDifference            "FORWARD" OR "REVERSE" ,
                attribute                           string ,
                difference_mrid                     mRID string ,
                object                              string ,
                value                               number  }
```

Known issue in GridAPPS-D releases_2019.09.0: Events passed from the Test Manager or from a Configuration File are not registering correctly in the Influx database. Simulation Input from operator control actions in the viz are being stored correctly. For this example, run a simulation of the 123 Node Model in the Viz and open any switch. Then copy the simulation_id and paste it into the first code block below.

Example 1: Query for all simulation input commands

```
[ ]: viz_simulation_id = "602611015"
```

```
[ ]: topic = "goss.gridappsd.process.request.data.timeseries" # Specify Timeseries API␣
     ↪GridAPPS-D topic

     # Query for all equipment command inputs passed to simulation
     message = {
         "queryMeasurement": "simulation",
         "queryFilter": {
             "simulation_id": viz_simulation_id,
```

(continues on next page)

(continued from previous page)

```
        "hasSimulationMessageType": "INPUT"},
    "responseFormat": "JSON"
}

gapps.get_response(topic, message) # Pass API call
```

21.3.4 Query for Sensor Service Data

Using the Sensor Service

The GridAPPS-D Sensor Service simulates the noise and packet losses of real field device measurements based upon the magnitude of "prestine" simulated values from the GridLab-D simulation. This service has been specifically designed to work within the gridappsd platform container. The GridAPPS-D platform will start the service when it is specified as a dependency of an application or when a service configuration is specified within the GridAPPS-D Visualization.

Python Application Usage: The python application using this service should require gridappsd-sensor-simulator as a requirement. In addition, the following python code shows how to get the correct topic for the service.

Service Configuration: The sensor-config in the above image shows an example of how to configure a portion of the system to have sensor output. Each mrid (such as `_99db0dc7-ccda-4ed5-a772-a7db362e9818`) will be monitored by this service and either use the default values or use the specified values during the service runtime.

The general format for the Sensor Service configuration message is

```
{
   "_99db0dc7-ccda-4ed5-a772-a7db362e9818": {
      "nominal-value": 100,
      "perunit-confidence-band": 0.01,
      "aggregation-interval": 30,
      "perunit-drop-rate": 0.01
   },
   "_ee65ee31-a900-4f98-bf57-e752be924c4d":{},
   "_f2673c22-654b-452a-8297-45dae11b1e14": {}
}
```

The other options for the service are:

- `"default-perunit-confidence-band"`
- `"default-aggregation-interval"`
- `"default-perunit-drop-rate"`
- `"passthrough-if-not-specified"`

These options will be used when not specified within the sensor-config block.

Note: Currently the nominal-value is not looked up from the database. At this time services aren't able to tell the platform when they are "ready". This will be implemented in the near future and then all of the nominal-values will be queried from the database.

A complete example of Sensor Service configuration is available in the sample python script Run123NodeSensorServiceDemo.py for three measurements in the IEEE 123 Node Model generated over a two minute simulation.

Querying Sensor Service Data

All of the simulated field measurements created by the Sensor Service are stored in the Influx Database and can be queried using the Timeseries API:

Measurement request key-value pair:

- `"queryMeasurement": "gridappsd-sensor-simulator"`

Allowed key-value pairs for `queryFilter` for output data

```
"queryFilter": {    KEY                             VALUE
                "startTime":                    number ,
                "endTime":                      number ,
                "measurement_mrid":             string OR [array of string values] ,
                "simulation_id":                string ,
                "angle":                        number ,
                "magnitude":                    number ,
                "value"                         number }
```

Example 1: Query for all Sensor Service Data between a given ``"startTime"`` and ``"endTime"``:

[ ]:
```
topic = "goss.gridappsd.process.request.data.timeseries" # Specify Timeseries API␣
↪GridAPPS-D topic

# Query for all Sensor Service Data in time range:
message = {
    "queryMeasurement": "gridappsd-sensor-simulator",
    "queryFilter": {"simulation_id": simulation_id,
                    "startTime": "1570041130",
                    "endTime": "1570041140"},
    "responseFormat": "JSON"
}

gapps.get_response(topic, message) # Pass API call
```

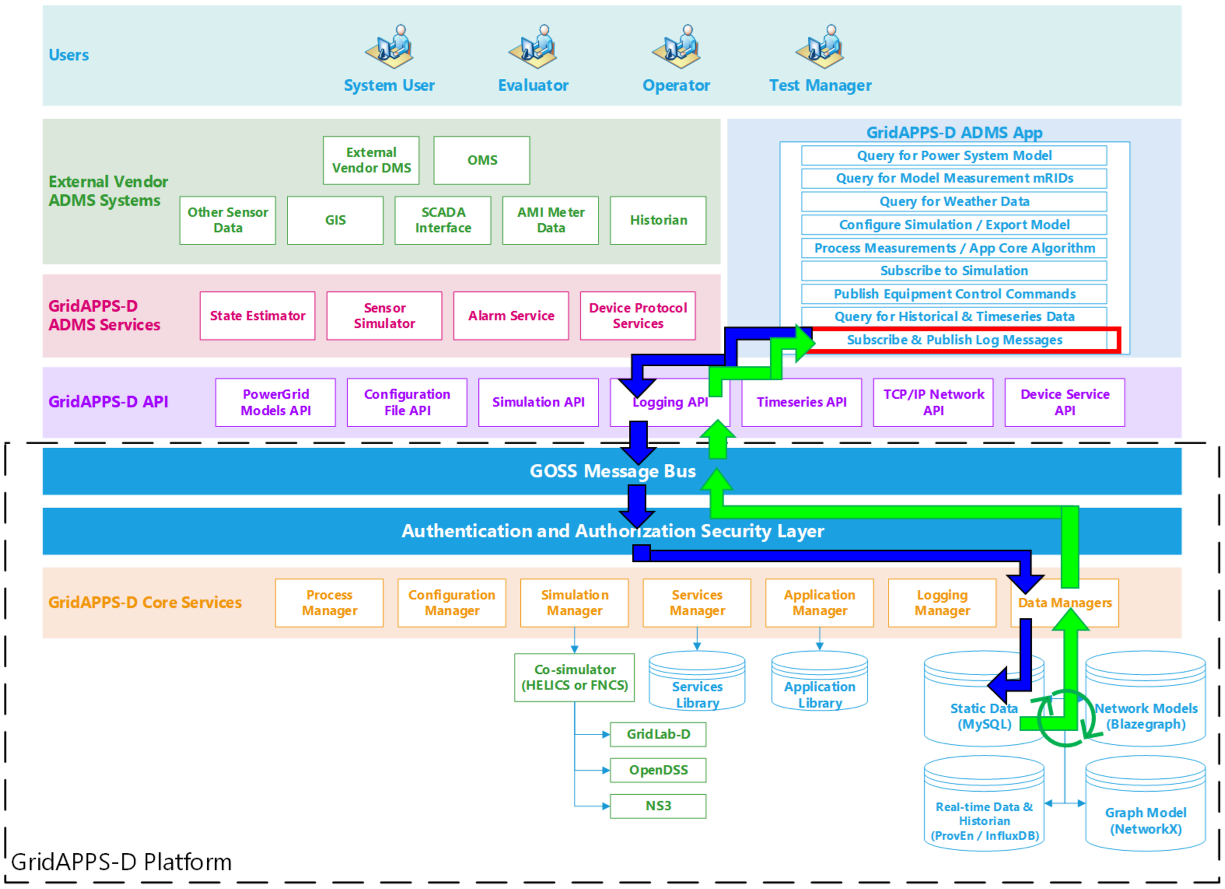
CHAPTER

TWENTYTWO

USING THE LOGGING API

22.1 Introduction to the Logging API

The Logging API enables applications to subscribe to real-time log messages from a simulation, query previously logged messages from the MySQL database, and publish messages to their either own log or their GridAPPS-D logs.



22.1.1 API Communication Channel

As with the Simulation API, the logging API uses both static `/queue/` and dynamic `/topic/` communication channel names depending on whether the API is being used for real-time simulation logs or historic logs that have already been saved in the database.

For a review of GridAPPS-D topics, see Lesson 1.4.

The correct topic for each Logging API call will be provided in the corresponding section for each API task below.

22.1.2 Message Structure

Logging messages in the GridAPPS-D environment follow the format of a python dictionary or equivalent JSON string with the format below.

```
{       KEY                 VALUE
    "source":                 filename,
    "processId":              simulation_id,
    "timestamp":              epoch time number,
    "processStatus":          "STARTED" or "STOPPED" or "RUNNING" or "ERROR" or "PASSED
↪" or "FAILED",
    "logMessage":             string,
    "logLevel":               "INFO" or "DEBUG" or "ERROR",
    "storeToDb":              true or false
}
```

All of the messages from a single instantiation will have the same format, with the only difference being the logMessage. As a result, it is possible to use the shortcuts available from the GridAPPSD-Python library to build out the repetitive portions of the message and pass just the logMessage string.

22.2 GridAPPSD-Python Logging API Extensions

The GridAPPSD-Python library uses several extensions to the standard Python logging library that enable applications to easily create log messages using the same syntax. These extensions support the additional log message formatting required by GridAPPS-D, such as simulation_id, log source, and process status.

The following code block enables the

```
[ ]: import logging
     import os

     os.environ['GRIDAPPSD_APPLICATION_ID'] = 'gridappsd-sensor-simulator'
     os.environ['GRIDAPPSD_APPLICATION_STATUS'] = 'STARTED'
     os.environ['GRIDAPPSD_SIMULATION_ID'] = opts.simulation_id
```

Important! Run this import command ***BEFORE*** creating the GridAPPS-D connection object `gapps = GridAPPSD(...)`.

Note: If your application is containerized in Docker and registered with the GridAPPS-D platform using the docker-compose file, these extensions will be imported automatically.

22.3 Subscribing to Simulation Logs

Similar to the two approaches used to subscribe to simulation measurements discussed in *Comparison of Subscription Approaches*, it is possible to use either a function or a class definition to subscribe to the simulation logs.

22.3.1 Subscription API Communication Channel

This is a dynamic `/topic/` communication channel that is best implemented by importing the GriAPPSD-Python library function for generating the correct topic.

- `from gridappsd.topics import simulation_log_topic`
- `log_topic = simulation_log_topic(simulation_id)`

Note on Jupyter Notebook environment: In the examples below, the Jupyter Notebook environment does not update definitions of the subscription object or function definitions. As a result, it is necessary to restart the notebook kernel. The gapps connection object definition is included again for convenience in executing the notebook code blocks

[3]:
```
viz_simulation_id = "paste sim id here"

# Establish connection to GridAPPS-D Platform:
from gridappsd import GridAPPSD

# Set environment variables - when developing, put environment variable in ~/.bashrc␣
↪file or export in command line
# export GRIDAPPSD_USER=system
# export GRIDAPPSD_PASSWORD=manager

import os # Set username and password
os.environ['GRIDAPPSD_USER'] = 'tutorial_user'
os.environ['GRIDAPPSD_PASSWORD'] = '12345!'

# Connect to GridAPPS-D Platform
gapps = GridAPPSD(viz_simulation_id)
assert gapps.connected
```

[ ]:
```
from gridappsd.topics import simulation_log_topic
log_topic = simulation_log_topic(viz_simulation_id)
```

Note on Jupyter Notebook environment: In the examples below, the Jupyter Notebook environment does not update definitions of the subscription object or function definitions. As a result, it is necessary to restart the notebook kernel. The gapps connection object definition is included again for convenience in executing the notebook code blocks

---

22.3.2 Subscribing using a Function Definition

The first approach used to subscribe to measurements is to define a function with the correct set of arguments that is then passed to the `.subscribe()` method associated with the `GridAPPPSD()` object.

The function does not require a specific name, and is somewhat easier to define and use. However, the arguments of the function need to be named correctly for the GridAPPSD-Python library to process the simulation output correctly.

The format for the function definition is

```
def myLogFunction(header, message):
    # do something when receive a log message
    # do something else
```

That function handle is then passed as an argument to the `.subscribe(topic, function_handle)` method:

```
[ ]: def demoLogFunction(header, message):
         timestamp = message["timestamp"]
         log_message = message["logMessage"]

         print("Log message received at timestamp ", timestamp, "which reads:")
         print(log_message)
         print("........................")
```

```
[ ]: gapps.subscribe(log_topic, demoLogFunction)
```

22.3.3 Subscribinb using a Class Definition

The second approach used to subscribe to simulation logs is to define add a custom method to the same class with `__init__` and `on_message` methods that was created to subscribe to measurements.

Unlike the Simulation API, the Logging API does not require a specific name for the method used to subscribe to log messages.

It is possible to create additional methods in the subscription class definition to enable the app to subscribe to additional topics, such as the simulation log topic, as shown in the example below.

```
[ ]: class PlatformDemo(object):
         # A simple class for interacting with simulation

         def __init__(self, simulation_id, gapps_obj):
             # Initialize variables and attributes
             self._gapps = gapps_obj
             self._simulation_id = simulation_id
             # self.foo = bar

         def on_message(self, headers, message):
             # Do things with measurements
             meas_value = message["message"]["measurements"]
             # Do more stuff with measurements

         def my_logging_method(self, headers, message):
             timestamp = message["timestamp"]
             log_message = message["logMessage"]

             print("Log message received at timestamp ", timestamp, "which reads:")
             print(log_message)
             print("........................")
```

```
[ ]: # Create subscription object
     demo_obj = PlatformDemo(viz_simulation_id, gapps)

     # Subscribe to logs using method
     gapps.subscribe(log_topic, demo_obj.my_custom_method)
```

22.4 Publishing to Simulation Logs

The GridAPPSD-Python library enables use of the standard Python logging syntax to create logs, publish them to the GOSS Message Bus, and store them in the MySQL database.

Documentation of the standard Python logging library is available on Python Docs.

It is possible to publish to either local app logs (which are more useful for debugging) or the GridAPPS-D logs (which can be accessed by other applications and should be used for completed applications).

22.4.1 Publishing to Local App Logs

The first approach is to use the default Python logger to write to local app logs by importing the `logging` library and then use the `.getLogger()` method from the Python library.

[6]:
```
import logging

python_log = logging.getLogger(__name__)
```

Log messages can then be published by invoking the methods

- `python_log.debug("log message")`
- `python_log.info("log message")`
- `python_log.warning("log message")`
- `python_log.error("log message")`
- `python_log.fatal("log message")`

22.4.2 Publishing to GridAPPS-D Logs

The second approach is to use the GridAPPS-D logs. Importing the python logging library is not necessary. Instead initialize a logging object using the `.get_logger()` method associated with the GridAPPS-D connection object. Note the difference in spelling of the GridAPPS-D Library and default Python Library methods.

[5]:
```
gapps_log = gapps.get_logger()
```

Log messages can then be published by invoking the methods

- `gapps_log.debug("log message")`
- `gapps_log.info("log message")`
- `gapps_log.warning("log message")`
- `gapps_log.error("log message")`
- `gapps_log.fatal("log message")`

22.5 Querying Saved Logs

Log messages published using the Logging API and the GOSS Message Bus are saved to the MySQL database. These log messages can be accessed with a Logging API query.

22.5.1 Log Query API Communication Channel

The query for logs uses a static `/queue/` channel that is imported from the GridAPPS-D Topics library.

This topic is used with the `.get_response(topic, message)` method associated with the GridAPPS-D connection object.

- `from gridappsd import topics as t`
- `gapps.get_response(t.LOGS, message)`

22.5.2 Structure of the Log Query Message

The first approach to querying with the Logging API is to use a python dictionary or equivalent JSON string that follows formatting similar to the query messages used by all the other GridAPPS-D APIs:

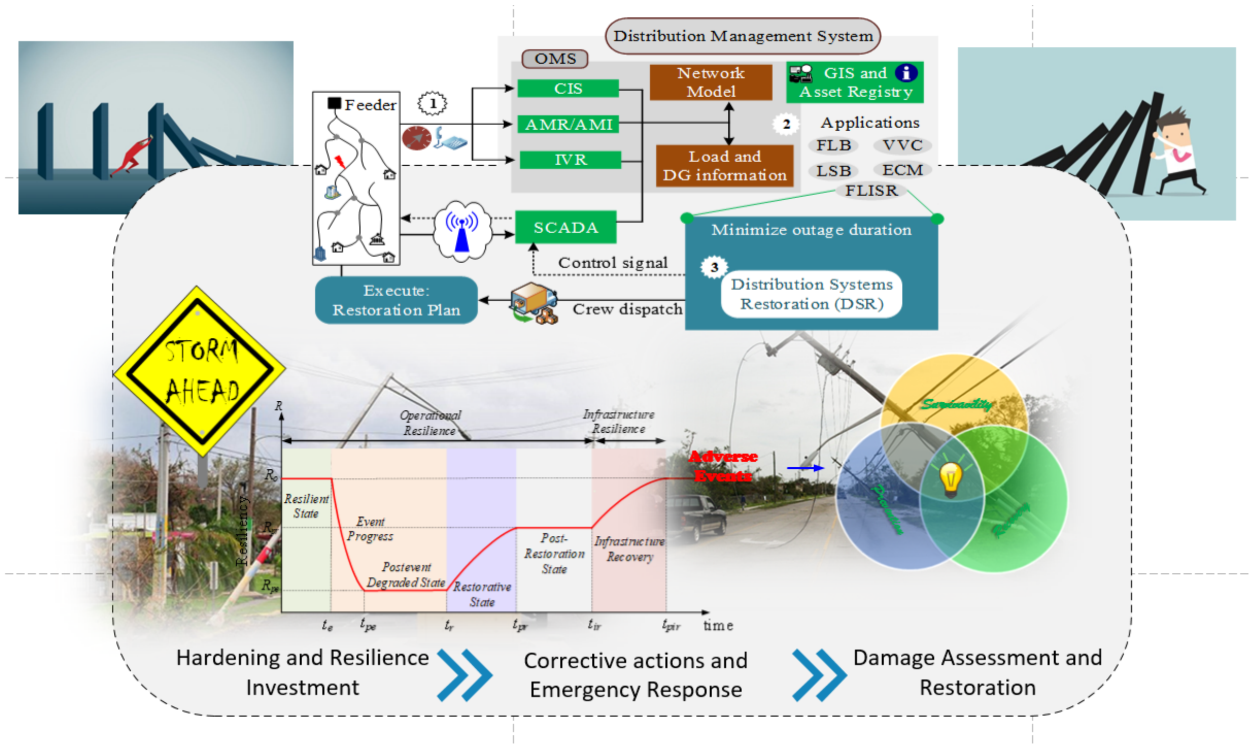
[ ]:
```
from gridappsd import topics as t

message = {
    "source": "ProcessEvent",
    "processId": viz_simulation_id,
    "processStatus": "INFO",
    "logLevel": "INFO"
}

gapps.get_reponse(t.LOGS, message)
```

|GridAPPS-D_narrow.png|

CHAPTER

TWENTYTHREE

RESILIENT RESTORATION (WSU)



An advanced distribution management system (ADMS) supports grid management and decision support applications to address the growing operational challenges faced by the modern electric power distribution systems while ensuring reliable and resilient operations. In this document, we describe the development of the proposed fault location, isolation, and restoration (FLISR) and it's integration with an open-source standards-based platform for ADMS application development viz. GridAPPS-D, developed by Pacific Northwest National Laboratory (PNNL).

Essentially, an ADMS allows for applications that can readily access information from various systems including, but not limited to Distributed Energy Resource Management System (DERMS), Supervisory Control and Data Acquisition (SCADA), Outage Management Systems (OMS), Graphical Information Management Systems (GIS), and Advanced Metering Infrastructure (AMI). Recently, researchers at PNNL developed an open-source ADMS application development platform, GridAPPS-D, that provides an open-source, extensible application development environment. These new developments call for research not only on advanced applications for the distribution systems that leverage the interoperability of the ADMS platform but also on providing a proof-of-concept for integrating such advanced applications into the future ADMS.

23.1 Overview

The electric power distribution system is facing a magnitude of challenges due to an increase in severe weather events leading to widespread network outages, coupled with the growing regulatory requirements for increased reliability and resilience, reduced carbon emissions, and growing distributed energy resources (DER) penetrations. The Fault Location Isolation and Service Restoration (FLISR) is one of the most critical applications currently being adopted by the majority of distribution companies and made available by most ADMS vendors to manage system outages [CIT1].

This report documents the restoration application developed by Washington State University (WSU) and demonstrate its functionality by integrating it with a standards-based open-source platform – GridAPPS-D, developed by the Pacific Northwest National Laboratory (PNNL).

23.1.1 Application Architecture

The proposed restoration application can be implemented in an advanced distribution management system (ADMS) as shown in Figure 1. Figure shows the overall architecture of a modern DMS with integration of several subsystems such as customer information system (CIS), geographical information system (GIS), interactive voice response (IVR), advanced metering infrastructure (AMI), SCADA and flowchart of the proposed DSR framework. Once a power outage happens, three specific tasks are performed in the DMS to restore the power to out-of-service area:

1. **Information Collection**: To monitor the power distribution system condition and gather resource information.
2. **Information Processing**: For system model identification and fault location.
3. **Service Restoration**: To find the candidate switch and generate DER control signals for circuit reconfiguration.

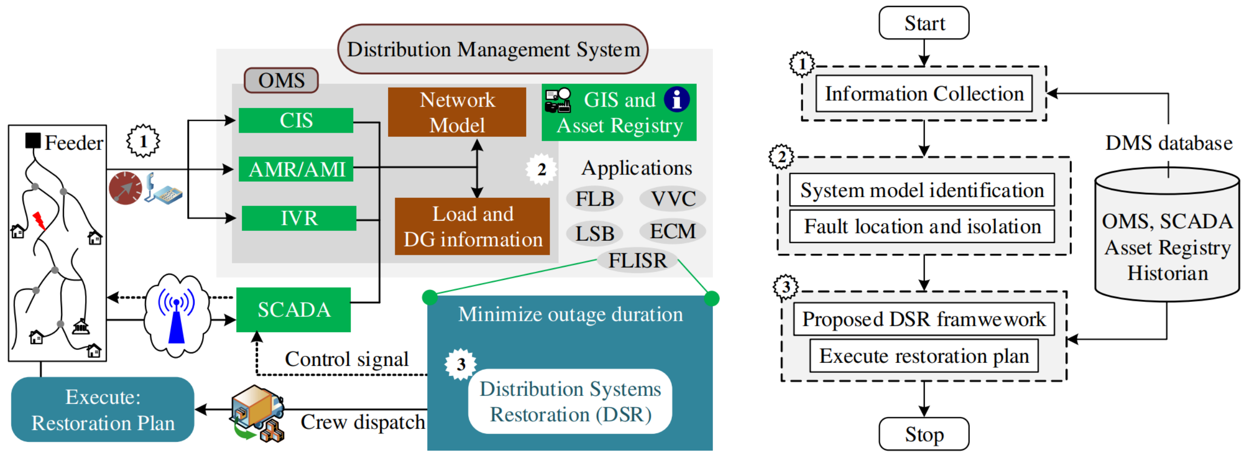


Figure 1: Architecture of a modern distribution management system and flowchart of the proposed restoration application.

23.1.2 Leveraging the GridAPPS-D Platform

The realization of an autonomous restoration application requires a measurement and control environment that provides post-fault situational awareness and the ability to remotely deploy the decisions for restoration. GridAPPS-D is an open-source, standards-based platform designed to support the development of advanced, data-driven distribution system operation and/or planning applications that take advantage of the data-rich environment expected in modernized electric power distribution systems with smart grid technologies.

Figure 2 shows a schematic for the interaction and communication among the distribution system operational sub-system for the proposed restoration application. The platform is typically integrated with other related data and decision-support systems/subsystems such as DERMS, SCADA, OMS, GIS, AMI, CIS to: a) monitor the distribution system conditions, b) obtain the DER availability and operating conditions, and c) for load estimation and control.

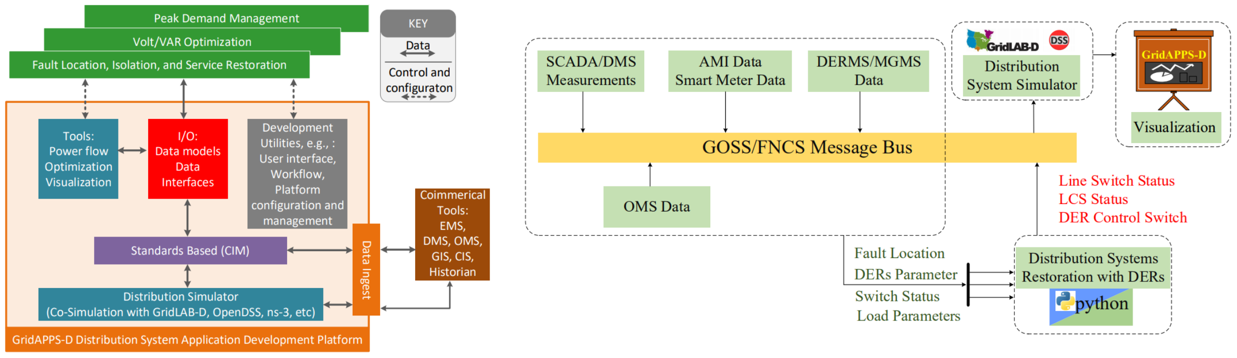

Peak Demand Management
Volt/VAR Optimization
Fault Location, Isolation, and Service Restoration
KEY
Data
Control and configuraton
Tools: Power flow Optimization Visualization
I/O: Data models Data Interfaces
Development Utilities, e.g., : User interface, Workflow, Platform configuration and management
Standards Based (CIM)
Data Ingest
Coimmerical Tools: EMS, DMS, OMS, GIS, CIS, Historian
Distribution Simulator (Co-Simulation with GridLAB-D, OpenDSS, ns-3, etc)
GridAPPS-D Distribution System Application Development Platform
SCADA/DMS Measurements
AMI Data Smart Meter Data
DERMS/MGMS Data
Distribution System Simulator
GridAPPS-D
Visualization
GOSS/FNCS Message Bus
OMS Data
Line Switch Status
LCS Status
DER Control Switch
Fault Location
DERs Parameter
Switch Status
Load Parameters
Distribution Systems Restoration with DERs
python

Figure 2: Integration of proposed application to the GridAPPS-D platform. GOSS/FNCS is the PNNL's platform for data exchange among subsystems. GOSS: GridOPTICS Software System; FNCS: Framework for network simulation.

23.1.3 Definition of Terms

Fault - Opening of normally-closed switch in response to any abnormal operating condition on its downstream.

Platform - Refers to GridAPPS-D platform.

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

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

CPLEX - A commercial optimization software package for solving the large-scale optimization problem

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

Power System Model - A modified IEEE 8500-node feeder is used as the test case

23.1.4 References

23.1.5 Contact Us

WSU team can be reached at shiva.poudel@wsu.edu or anamika.dubey@wsu.edu.

For more information about the lab, Click Here

23.2 Installing GridAPPS-D

GridAPPS-D is available using docker containers

23.2.1 Requirements

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

23.2.2 Install Docker on Ubuntu

- Clone or download the repository

```
gridappsd@gridappsd-VirtualBox:~$ git clone https://github.com/GRIDAPPSD/gridappsd-
↪docker
gridappsd@gridappsd-VirtualBox:~$ cd gridappsd-docker
```

- run the docker-ce installation script

```
gridappsd@gridappsd-VirtualBox:~/gridappsd-docker$ ./docker_install_ubuntu.sh
```

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

23.3 Application Setup

23.3.1 Download the application

- Clone or download the repository. The updated code is in the develop branch.

```
gridappsd@gridappsd-VirtualBox:~$ git clone https://github.com/shpoudel/WSU-
↪Restoration -b develop
gridappsd@gridappsd-VirtualBox:~$ cd WSU-Restoration
```

23.3.2 Creating the application container

- From the command line execute the following commands to build the wsu-restoration container. Note that there is a dot at end of command.

```
gridappsd@gridappsd-VirtualBox:~/WSU-Restoration$ docker build --network=host -t wsu-
↪restoration-app .
```

23.3.3 Mount the application

- Add following to the docker-compose.yml file if CPLEX is available

```
wsu_res_app:
image: wsu-restoration-app
volumes:
  - /opt/ibm/ILOG/CPLEX_Studio129/:/opt/ibm/ILOG/CPLEX_Studio129
environment:
  GRIDAPPSD_URI: tcp://gridappsd:61613
```

(continues on next page)

(continued from previous page)

```
depends_on:
  - gridappsd
```

- Add following to the docker-compose.yml file if CPLEX is not available. In addition, replace prob.solve(CPLEX(msg=1)) with prob.solve() in restoration_WSU.py

```
wsu_res_app:
image: wsu-restoration-app
environment:
  GRIDAPPSD_URI: tcp://gridappsd:61613
depends_on:
  - gridappsd
```

23.4 Starting Service

23.4.1 Start the docker container services

- Note that this documentation is based on develop tag

```
gridappsd@gridappsd-VirtualBox:~/gridappsd-docker$ ./run.sh -t develop
```

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

The message in the container looks something like this:

```
Starting gridappsddocker_redis_1 ...
Starting gridappsddocker_proven_1 ...
Starting gridappsddocker_blazegraph_1 ...
Starting gridappsddocker_influxdb_1 ...
Starting gridappsddocker_mysql_1 ... done
Starting gridappsddocker_gridappsd_1 ... done
Starting gridappsddocker_wsu_res_app_1 ...
Starting gridappsddocker_wsu_res_app_1 ... done

Getting blazegraph status

Checking blazegraph data

Blazegrpah data available (1954268)

Getting viz status

Containers are running

Connecting to the gridappsd container
```

(continues on next page)

(continued from previous page)

```
docker exec -it gridappsddocker_gridappsd_1 /bin/bash

gridappsd@78a3d22dd2b9:/gridappsd$
```

23.4.2 Restoration application container

```
gridappsd@gridappsd-VirtualBox:~/WSU-Restoration$ docker exec -it gridappsddocker_wsu_
↪res_app_1 bash
```

- This will take you inside the application container.

```
root@1b762c641f24:/usr/src/gridappsd-restoration#
```

At this point, we should have two terminal open with gridappsd-docker container and restoration application terminal.

```
root@1b762c641f24:/usr/src/gridappsd-restoration#
gridappsd@78a3d22dd2b9:/gridappsd$
```

- Installing CPLEX in container
- Note that the installation command can be written inside the Dockerfile beforehand. However, we do the installation here manually before starting the platform.

```
root@1b762c641f24:/usr/src/gridappsd-restoration# cd /opt/ibm/ILOG/CPLEX_Studio129/
↪cplex/python/3.6/x86-64_linux/
root@1b762c641f24:/opt/ibm/ILOG/CPLEX_Studio129/cplex/python/3.6/x86-64_linux# python␣
↪setup.py install
```

- ATTENTION: It is required that the application container has the python version compatible with the CPLEX. For example, CPLEX_STUDIO129 requires python 3.6. Thus, Python3.6 should be made available in the application container.

23.4.3 Executing the application container

- Now, get back to the path where application is mounted.

```
root@1b762c641f24:/opt/ibm/ILOG/CPLEX_Studio129/cplex/python/3.6/x86-64_linux# cd /
↪usr/src/gridappsd-restoration
```

- The following runs the application from terminal

```
root@1b762c641f24:/usr/src/gridappsd-restoration# cd Restoration
root@1b762c641f24:/usr/src/gridappsd-restoration/Restoration# python main.py␣
↪[simulation_ID] '{"power_system_config":  {"Line_name":"_AAE94E4A-2465-6F5E-37B1-
↪3E72183A4E44"}}'
```

- Running application from the terminal requires Simulation_ID. To get the correct Simulatio_ID, we need to start the platform through the browser. This will be explained in detail in the next section (Visualization).

23.4.4 Starting GridAPPS-D Platform

- Start the platform from the gridappsd-docker container

```
gridappsd@78a3d22dd2b9:/gridappsd$ ./run-gridappsd.sh
```

- Following message can be seen at the end of running terminal. This confirms, the platform is running and we can start the application from the browser.

```
Registering user roles: application2 --  application
Registering user roles: application1 --  application
Registering user roles: operator3 --  operator
Registering user roles: operator2 --  operator
Registering user roles: evaluator2 --  evaluator,operator
Registering user roles: operator1 --  operator
Registering user roles: evaluator1 --  evaluator,operator
Registering user roles: testmanager2 --  testmanager
Registering user roles: testmanager1 --  testmanager
Registering user roles: service2 --  service
Registering user roles: service.pid --  pnnl.goss.gridappsd.security.rolefile
Registering user roles: service1 --  service
CREATING LOG DATA MGR MYSQL
{"id":"WSU_restoration","description":"Resilient Restoration Application","creator":
↪"WSU","inputs":[],"outputs":[],"options":["(simulationId)","\u0027(request)\u0027"],
↪"execution_path":"python /usr/src/gridappsd-restoration/Restoration/main.py","type":
↪"REMOTE","launch_on_startup":false,"prereqs":["gridappsd-voltage-violation",
↪"gridappsd-alarms"],"multiple_instances":true}
{"heartbeatTopic":"/queue/goss.gridappsd.remoteapp.heartbeat.WSU_restoration",
↪"startControlTopic":"/topic/goss.gridappsd.remoteapp.start.WSU_restoration",
↪"stopControlTopic":"/topic/goss.gridappsd.remoteapp.stop.WSU_restoration",
↪"errorTopic":"Error","applicationId":"WSU_restoration"}
```

23.5 Data Model

23.5.1 IEEE 8500-Node Test Feeder

An IEEE Working Group specified a set of distribution test circuits [CIT9] and we have chosen the largest one of these as a sample circuit for GridAPPS-D which is IEEE 8500-Node model [CIT10]. The original 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 6 voltage regulator banks, making it a reasonable test for solving voltage problems such as VVO. The circuit is splitted into three substation and several new distributed energy resources (DERs) are added in the test case (See Fig. 3). In addition, 7 tie switches are added such that each substation can share the load of another substation during fault to restore the outage customers. A new set of loads is added near S2 and the region is referred to as a new neighborhood.

- The detailed model of this feeder can be found at:

```
https://github.com/GRIDAPPSD/Powergrid-Models/tree/develop/blazegraph/test/dss/WSU
```

- The OpenDSS data can be extracted here:

```
https://github.com/shpoudel/D-Net
```

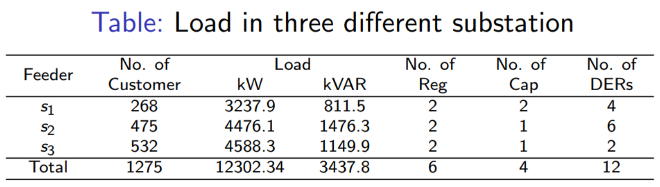


Table: Load in three different substation

Feeder	No. of Customer	Load kW	Load kVAR	No. of Reg	No. of Cap	No. of DERs
s_1	268	3237.9	811.5	2	2	4
s_2	475	4476.1	1476.3	2	1	6
s_3	532	4588.3	1149.9	2	1	2
Total	1275	12302.34	3437.8	6	4	12

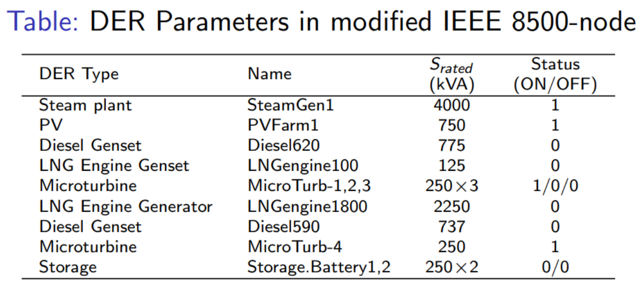


Table: DER Parameters in modified IEEE 8500-node

DER Type	Name	S_{rated} (kVA)	Status (ON/OFF)
Steam plant	SteamGen1	4000	1
PV	PVFarm1	750	1
Diesel Genset	Diesel620	775	0
LNG Engine Genset	LNGengine100	125	0
Microturbine	MicroTurb-1,2,3	250×3	1/0/0
LNG Engine Generator	LNGengine1800	2250	0
Diesel Genset	Diesel590	737	0
Microturbine	MicroTurb-4	250	1
Storage	Storage.Battery1,2	250×2	0/0

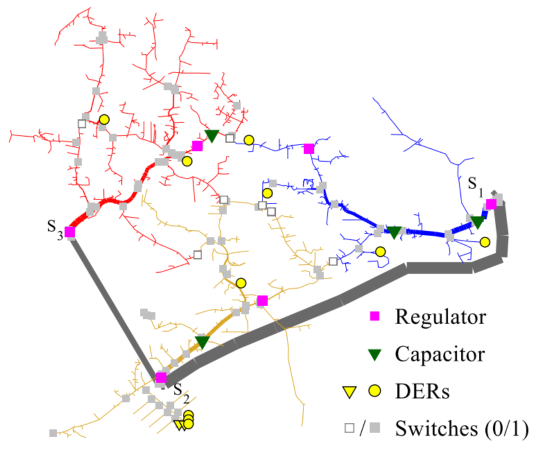


Figure 3: Modified IEEE 85000-node feeder.

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.

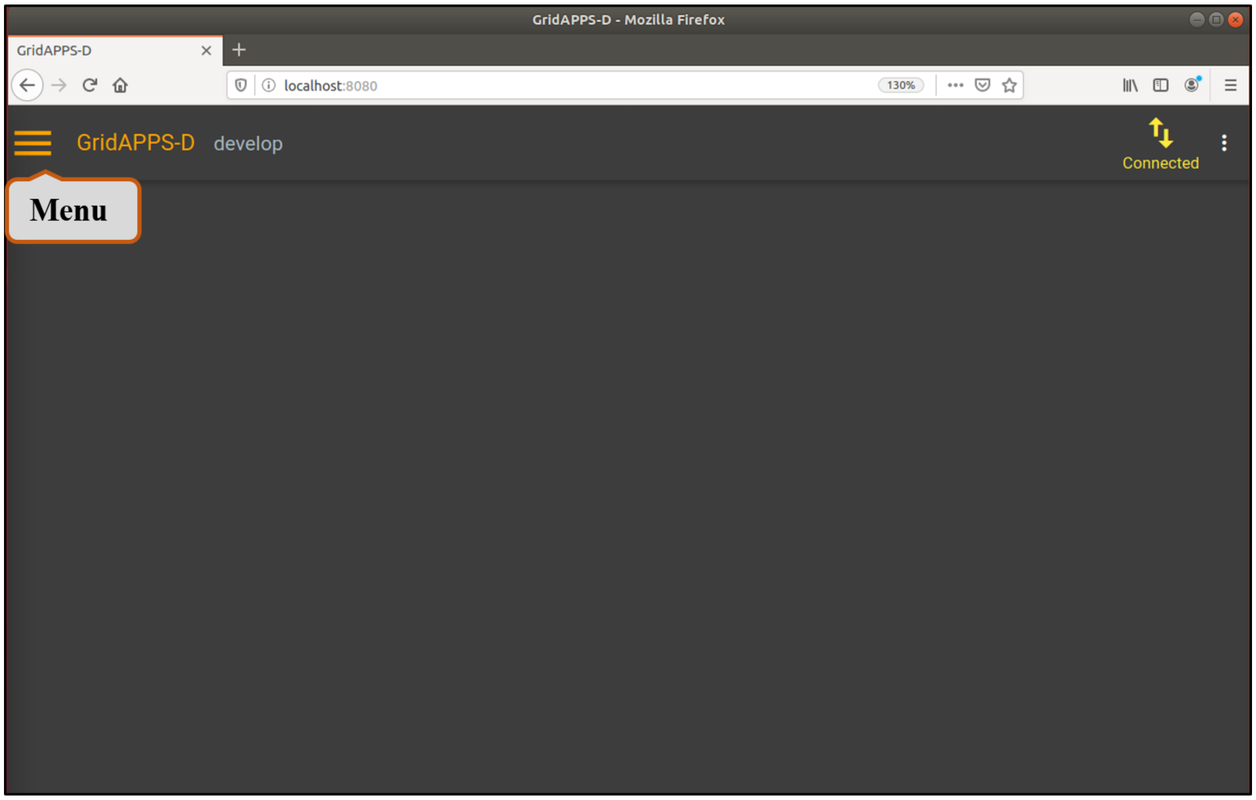
3. **Substitute a variable called VSOURCE for the SWING bus nominal voltage**. This needs to be set at 1.05 per-unit of nominal on the 115-kV system (i.e. 69715.065) in order to solve at peak load. Other conditions may require different source voltage values.

4. **Use a schedule for the loads** so they can vary with time during GridAPPS-D simulation. The file should be named *zipload_schedule.player*.

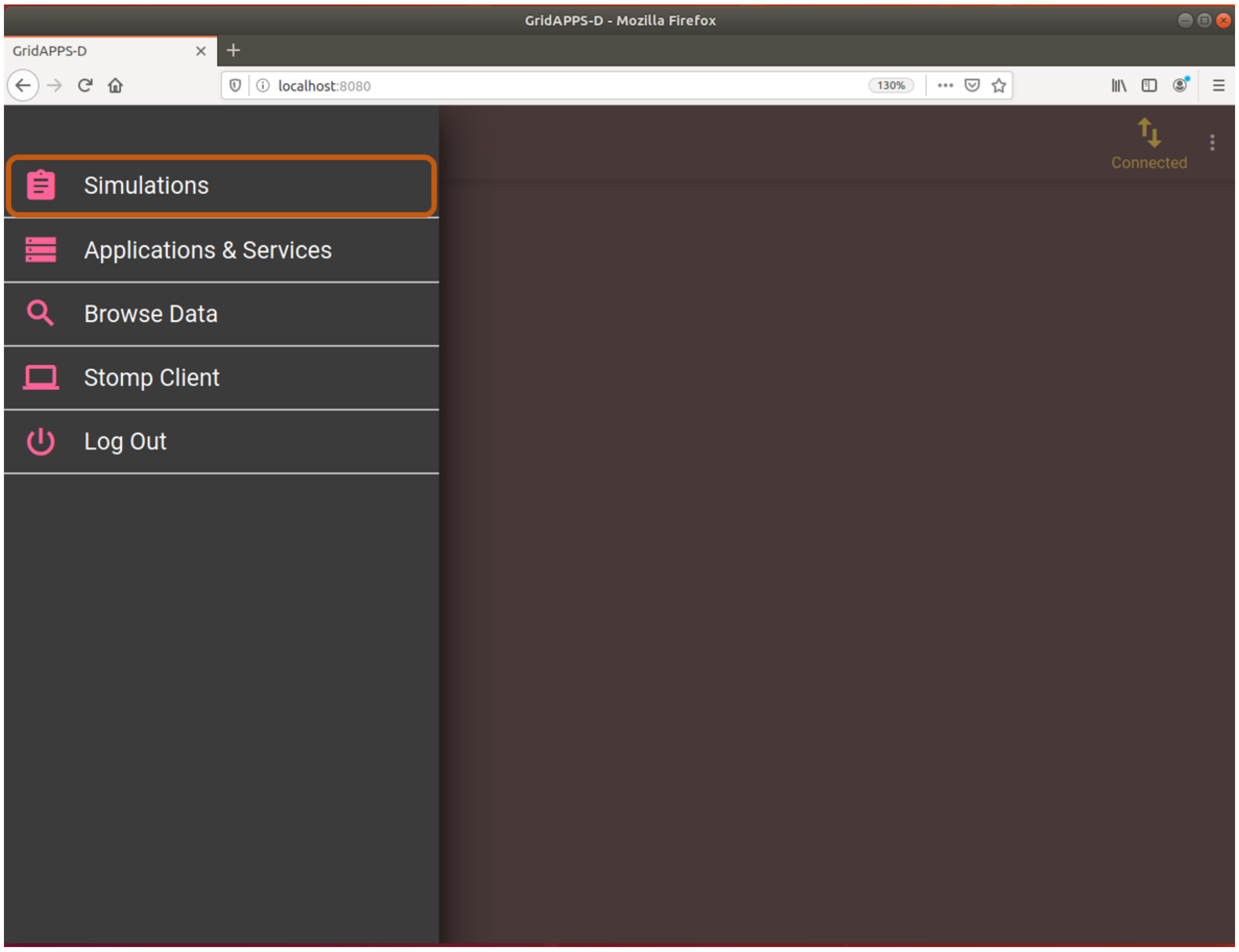
23.6 Visualization

23.6.1 Start a Simulation

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

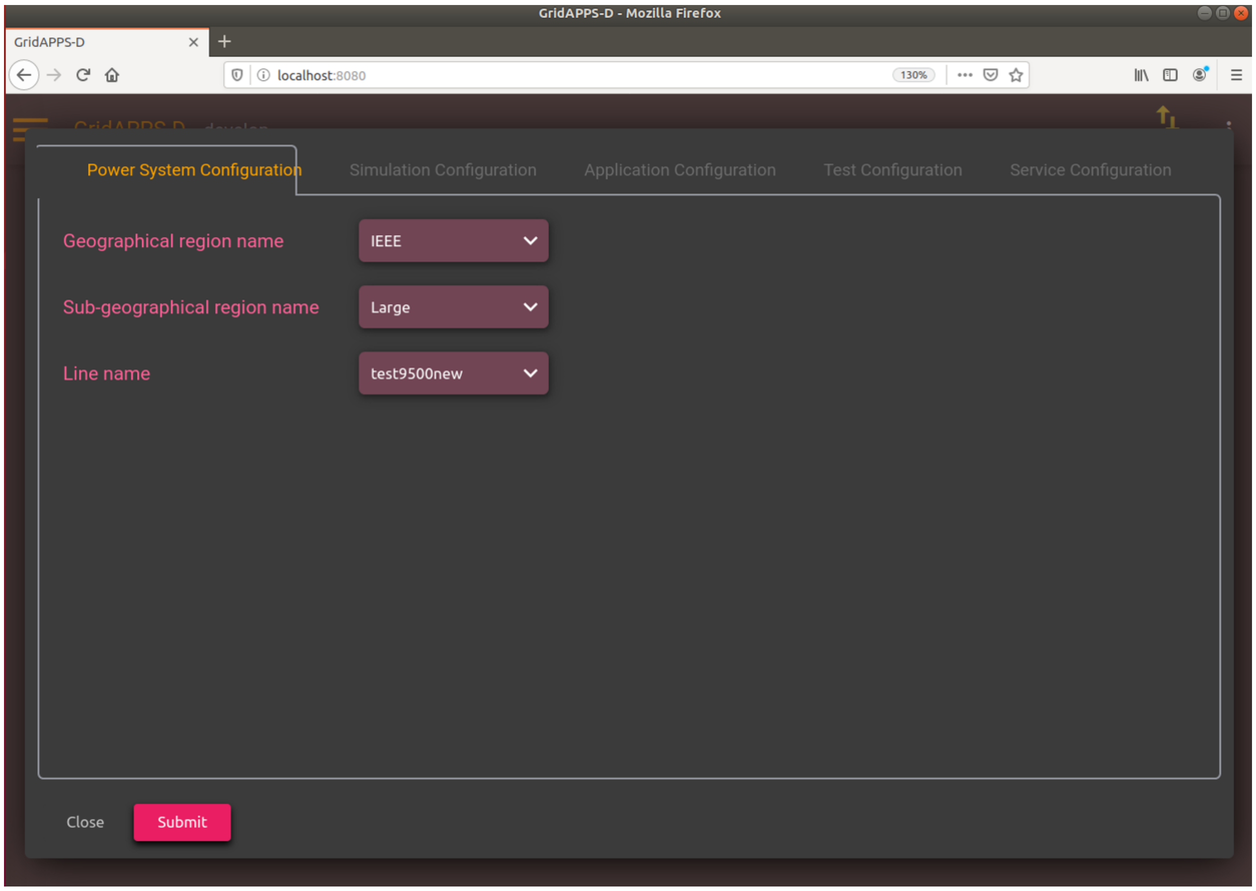


- Choose Simulations from the menu.

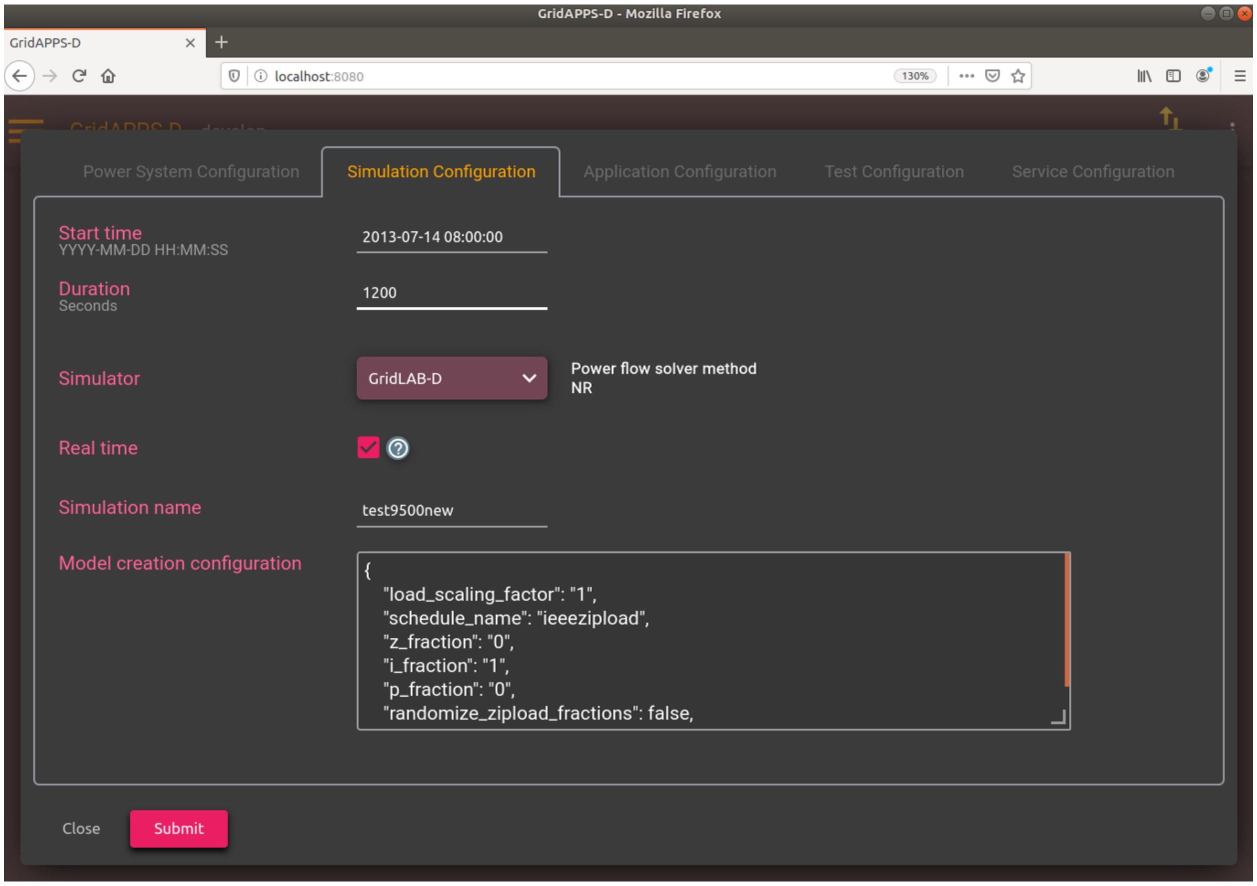


23.6.2 Change Configurations

- Change the Power System Configuration, Simulation Configuration, Application Configuration, and Test Configuration as follows:



Geographical region name: IEEE
Sub-geographical region name: Large
Line name: test9500new



Start time
YYYY-MM-DD HH:MM:SS
2013-07-14 08:00:00

Duration
Seconds
1200

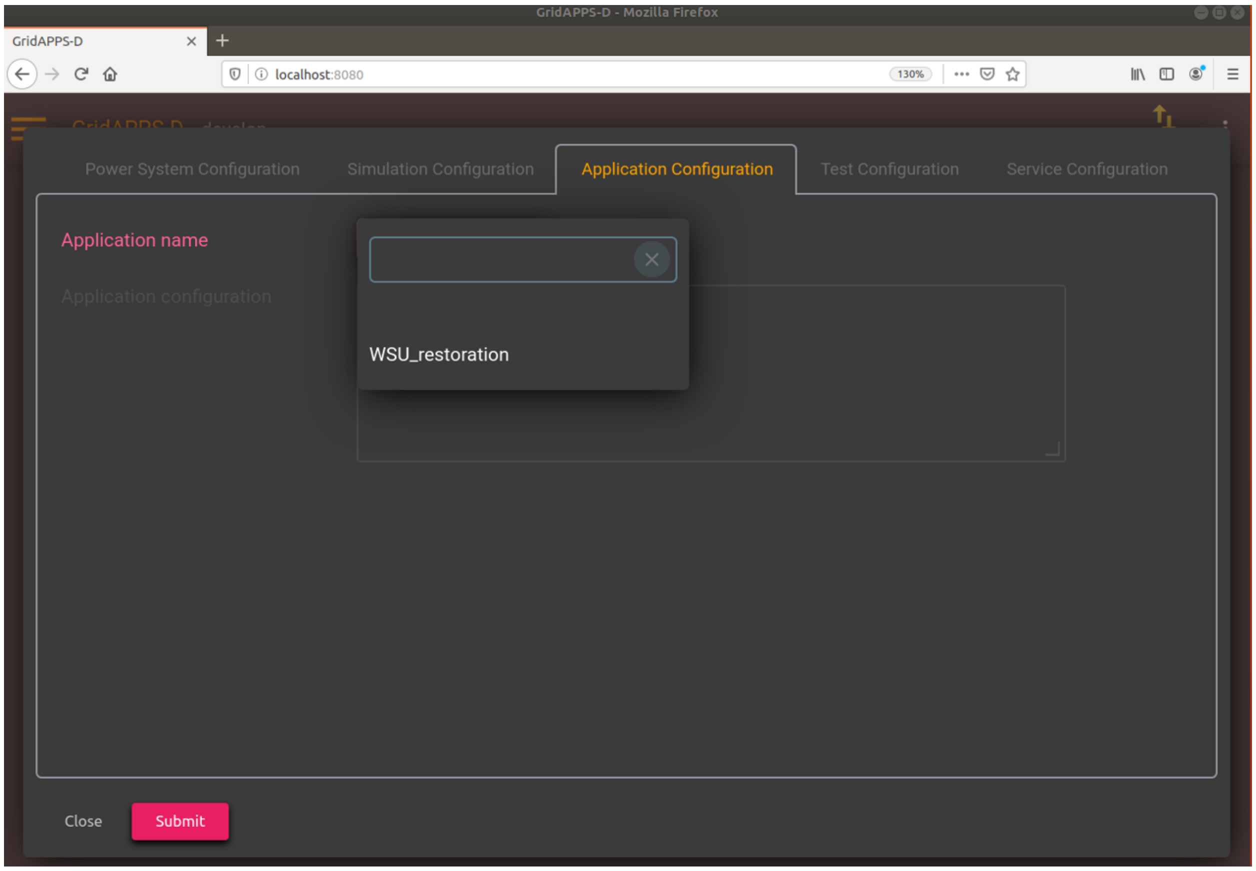
Simulator
GridLAB-D
Power flow solver method
NR

Real time ✓

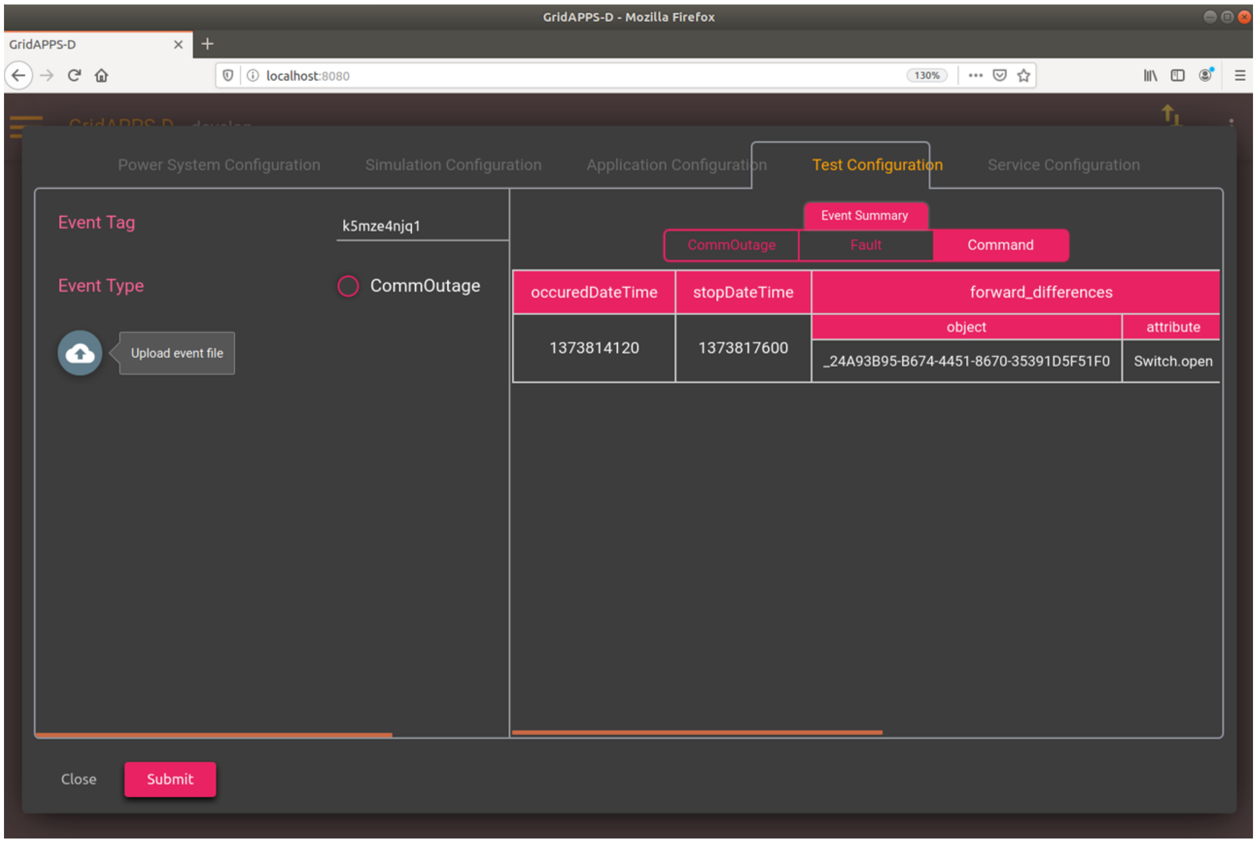
Simulation name
test9500new

Model creation configuration

```
{
    "load_scaling_factor": "1",
    "schedule_name": "ieeezipload",
    "z_fraction": "0",
    "i_fraction": "1",
    "p_fraction": "0",
    "randomize_zipload_fractions": false,
```



- Communication outage and fault events can be added using the Test Configuration page as shown below. Event can be added using radio button or upload option.



Event Tag: k5mze4njq1

Event Type: CommOutage

occuredDateTime	stopDateTime	forward_differences	
		object	attribute
1373814120	1373817600	_24A93B95-B674-4451-8670-35391D5F51F0	Switch.open

- Click the submit button to save all the configurations.

23.6.3 Adding an event

- Since we are running the restoration application, a fault event is required to trigger the application. Before proceeding, an event file is required. There are several ways to add event in the test feeder. Here, we use the upload option and use the following JSON file.

```
{
  "commandEvents": [
    {
      "message": {
        "forward_differences": [
          {
            "object": "_24A93B95-B674-4451-8670-35391D5F51F0",
            "attribute": "Switch.open",
            "value": 1
          }
        ],
        "reverse_differences": [
          {
            "object": "_24A93B95-B674-4451-8670-35391D5F51F0",
            "attribute": "Switch.open",
            "value": 0
          }
        ]
      },
```

(continues on next page)

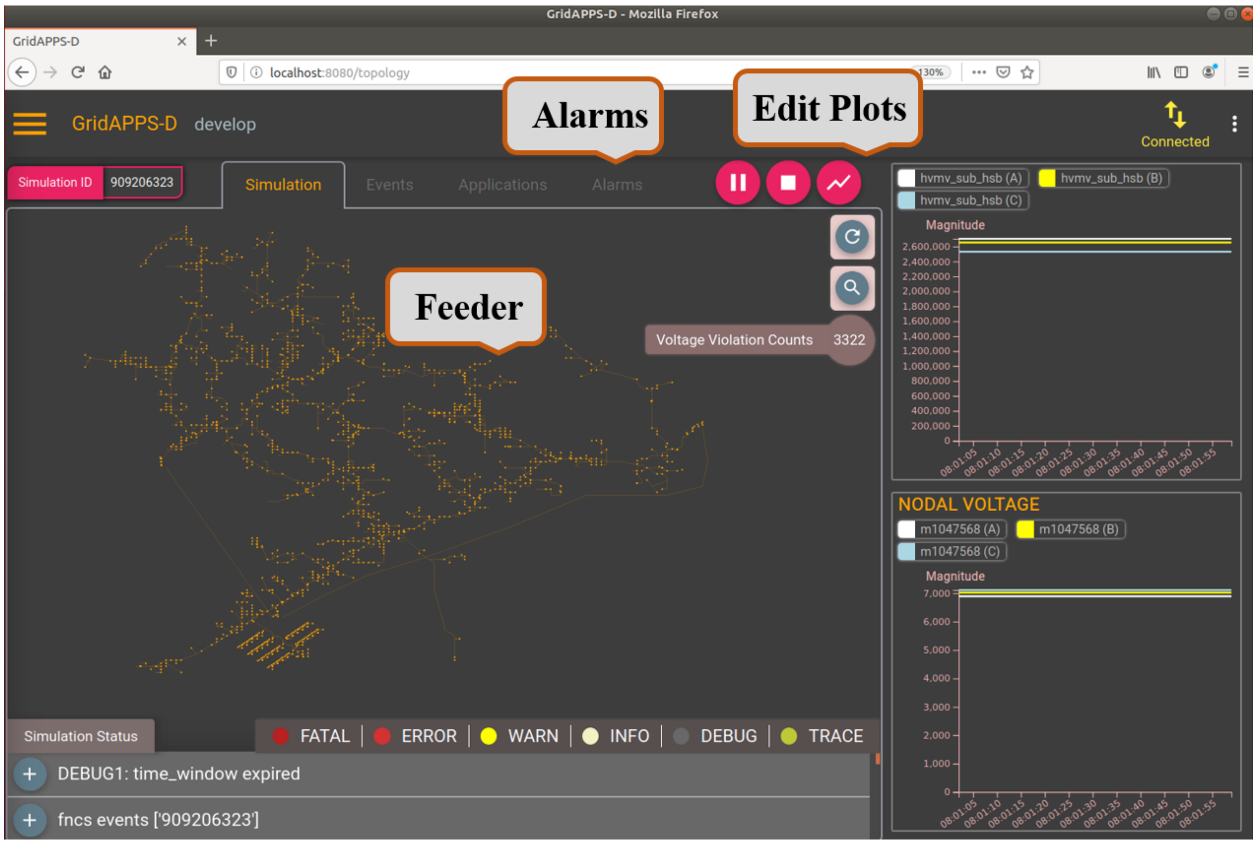
(continued from previous page)

```
        "event_type": "ScheduledCommandEvent",
        "occuredDateTime": "2013-07-14 08:02:00",
        "stopDateTime": "2013-07-14 09:00:00"
      }
    ]
}
```

- This event opens a switch (LN0895780_SW) in the feeder at time 2013-07-14 08:02:00 and remains open until 09:00:00. Please adjust the *Start time* in Simulation Configuration based on *occuredDateTime* of the event such that event occurs after the simulation has started.

23.6.4 Running the platform

- After few seconds, the test-feeder will load. In the meantime, use "Edit plots" to add power, voltage and tap of different components to visualize as the simulation progress.
- Click on the triangle to start the simulation. Once you get the Simulation_ID, use it to run the application in terminal. See the python command here.
- Note that the WSU-Restoration application gets triggered only when the fault event is added in the test case. During normal operation, application stays quite. The occurrence of the event can be verified on the Alarm tab. Once the event has occured, the applicatin starts and runs the optimization problem to generate the candidate switches for optimal circuit reconfiguration. The candidate switches are toggled and the system is restored.



DEBUG1: time_window expired

fncs events ['909206323']

23.7 License

CHAPTER TWENTYFOUR

VOLT-VAR OPTIMIZATION (WSU)

This repository contains the restoration application for IEEE 9500-node model (https://github.com/GRIDAPPSD/Powergrid-Models/tree/develop/blazegraph/test/dss/WSU). The application is hosted on GridAPPS-D platform.

24.1 Layout

Please 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)

24.2 Creating the restoration application container

1. From the command line execute the following commands to build the wsu-restoration container

```
osboxes@osboxes> cd WSU-VVO-app
osboxes@osboxes> docker build --network=host -t wsu-vvo-app .
```

2. Add the following to the gridappsd-docker/docker-compose.yml file

```
wsu_vvo:
  image: wsu-vvo-app
  depends_on:
    gridappsd
```

3. Run the docker application

```
osboxes@osboxes> cd gridappsd-docker
osboxes@osboxes> ./run.sh

# you will now be inside the container, the following starts gridappsd

gridappsd@f4ede7dacb7d:/gridappsd$ ./run-gridappsd.sh
```

4. Run the application container

```
osboxes@osboxes> cd WSU-VVO-app
osboxes@osboxes> docker exec -it gridappsddocker_wsu_vvo_1 bash

# you will now be inside the application container, the following runs the␣
↪application from terminal
```

(continues on next page)

(continued from previous page)

```
root@1b762c641f24:/usr/src/gridappsd-vvo# cd wsu_vvo
root@1b762c641f24:/usr/src/gridappsd-vvo/wsu_vvo# python main.py 1234 '{"power_
→system_config":  {"Line_name":"_AAE94E4A-2465-6F5E-37B1-3E72183A4E44"}}'
```

Next to start the application through the viz follow the directions here: https://gridappsd.readthedocs.io/en/latest/using_gridappsd/index.html#start-gridapps-d-platform

24.3 Forming optimization problem

In the optimization problem, test feeders are modeled as graph; G (V, E), where V is set of nodes and E is set of edges. The graph and line parameters of the test feeder can be extracted from (https://github.com/shpoudel/D-Net).

24.4 Get real-time topology and load data of the test case

First, the real time topology of the test case is extracted from the operating feeder in platform (top_identify.py). Then, load data for each node is obtained (get_Load.py) and stored in json format to be used in linear power flow while writing constraints for optimization problem.

CHAPTER

TWENTYFIVE

DER DISPATCH (NREL)

25.1 Purpose

The purpose of this repository is to document the chosen way of registering and running applications within a GridAPPS-D deployment.

25.2 Requirements

1. Docker ce version 17.12 or better. You can install this via the docker_install_ubuntu.sh script. (note for mint you will need to modify the file to work with xenial rather than ubuntu generically)

25.3 Quick Start

1. Please 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)

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

ls -l

drwxrwxr-x  7 osboxes osboxes 4096 Sep  4 14:56 gridappsd-docker
drwxrwxr-x  5 osboxes osboxes 4096 Sep  4 19:06 DER-Dispatch-app
```

25.4 Creating the der-dispatch-app application container

1. From the command line execute the following commands to build the der-dispatch-app container

```
osboxes@osboxes> cd DER-Dispatch-app
osboxes@osboxes> docker build --network=host -t der-dispatch-app .
```

2. Add the following to the gridappsd-docker/docker-compose.yml file

```
  derdispatch:
    image: der-dispatch-app
    ports:
```

(continues on next page)

(continued from previous page)

```
    - 9001:9001
  environment:
    GRIDAPPSD_URI: tcp://gridappsd:61613
  depends_on:
    - gridappsd
  volumes:
    - $HOME/git/DER-Dispatch-app-Public:/usr/src/gridappsd-der-dispatch
```

3. Run the docker application

```
osboxes@osboxes> cd gridappsd-docker
osboxes@osboxes> ./run.sh

# you will now be inside the container, the following starts gridappsd

gridappsd@f4ede7dacb7d:/gridappsd$ ./run-gridappsd.sh
```

4. Test opt_function.pyc file.

```
docker exec -it gridappsd-docker_derdispatch_1 bash

cd der_dispatch_app
## This function is slow but it should not hang
python test_opt_function.py
```

Example output

```
> python test_opt_function.py
('gridappsd', 61613)
Mon Feb 24 18:20:24 2020
(9480, 9480) (9480, 10)
Mon Feb 24 18:21:22 2020
Mon Feb 24 18:21:29 2020
```

25.5 Application Configuration

Set start time to 2013-07-22 10:30:00

Options: Run with OPF = 0 and a specific time and duration first to get the baseline for comparison. Then Run with OPF = 1 and the same time and duration evaluate the applications performance against the baseline. run_freq - Number of seconds between appliction runs. run_on_host - True then run on the host machine. False then run in the container.

The baseline will be in this folder in the container: /usr/src/gridappsd-der-dispatch/der_dispatch_app/adms_result_ieee123pv_house_156389

25.5.1 Baseline run

```
    {
      "OPF": 0,
      "run_freq": 15,
      "run_on_host": false,
      "run_realtime": true,
      "stepsize_xp": 0.2,
      "stepsize_xq": 2,
      "coeff_p": 0.005,
      "coeff_q": 0.0005,
      "Vupper": 1.025,
      "Vlower": 0.95,
      "stepsize_mu": 50000,
      "optimizer_num_iterations": 10
    }
```

25.5.2 Optimal Powerflow on Run

Set OPF to 1 and run again.

```
    {
      "OPF": 1,
      "run_freq": 15,
      "run_on_host": false,
      "run_realtime": true,
      "stepsize_xp": 0.2,
      "stepsize_xq": 2,
      "coeff_p": 0.005,
      "coeff_q": 0.0005,
      "Vupper": 1.025,
      "Vlower": 0.95,
      "stepsize_mu": 50000,
      "optimizer_num_iterations": 10
    }
```

1. docker copy command

```
docker cp 422635e932bb:/usr/src/gridappsd-der-dispatch/der_dispatch_app/adms_result_
↪test9500new_1374510720_OPF_1_stepsize_xp_0_2_stepsize_xq_2_coeff_p_0_1_coeff_q_5e_
↪neg_05_stepsize_mu_500/ $HOME/
```

Next to start the application through the viz follow the directions here: https://gridappsd.readthedocs.io/en/latest/using_gridappsd/index.html#start-gridapps-d-platform

```
python /usr/src/gridappsd-der-dispatch/der_dispatch_app/main_app_new.py 952325492 '{
↪"power_system_config":{"SubGeographicalRegion_name":"_1CD7D2EE-3C91-3248-5662-
↪A43EFEFAC224","GeographicalRegion_name":"_24809814-4EC6-29D2-B509-7F8BFB646437",
↪"Line_name":"_EBDB5A4A-543C-9025-243E-8CAD24307380"},"simulation_config":{"power_
↪flow_solver_method":"NR","duration":600,"simulation_name":"ieee123","simulator":
↪"GridLAB-D","start_time":1374510720,"run_realtime":false,"simulation_output":{},
↪"model_creation_config":{"load_scaling_factor":1.0,"triplex":"y","encoding":"u",
↪"system_frequency":60,"voltage_multiplier":1.0,"power_unit_conversion":1.0,"unique_
↪names":"y","schedule_name":"ieeezipload","z_fraction":0.0,"i_fraction":1.0,"p_
↪fraction":0.0,"randomize_zipload_fractions":false,"use_houses":false},"simulation_
↪broker_port":59469,"simulation_broker_location":"127.0.0.1"},"application_config":{
↪"applications":[{"name":"der_dispatch_app","config_string":"{\"OPF\": 0, \"run_freq\
↪": 60, \"run_on_host\": false, \"run_realtime\": false, \"stepsize_xp\": 0.2, \
↪"stepsize_xq\": 2, \"coeff_p\": 0.1, \"coeff_q\": 5e-05, \"stepsize_mu\": 500}"}]},
↪"simulation_request_type":"NEW"}' '{OPF:0,run_freq:60,run_on_host:false,run_
↪realtime:false,stepsize_xp:0.2,stepsize_xq:2,coeff_p:0.1,coeff_q:5e-05,stepsize_mu:
↪500}'
```

(continues on next page)

(continued from previous page)

```
python /usr/src/gridappsd-der-dispatch/der_dispatch_app/main_app_new.py 1522305637 '{
↪"power_system_config":{"SubGeographicalRegion_name":"_1CD7D2EE-3C91-3248-5662-
↪A43EFEFAC224","GeographicalRegion_name":"_24809814-4EC6-29D2-B509-7F8BFB646437",
↪"Line_name":"_E407CBB6-8C8D-9BC9-589C-AB83FBF0826D"},"simulation_config":{"power_
↪flow_solver_method":"NR","duration":600,"simulation_name":"ieee123","simulator":
↪"GridLAB-D","start_time":1374510720,"run_realtime":false,"simulation_output":{},
↪"model_creation_config":{"load_scaling_factor":1.0,"triplex":"y","encoding":"u",
↪"system_frequency":60,"voltage_multiplier":1.0,"power_unit_conversion":1.0,"unique_
↪names":"y","schedule_name":"ieeezipload","z_fraction":0.0,"i_fraction":1.0,"p_
↪fraction":0.0,"randomize_zipload_fractions":false,"use_houses":false},"simulation_
↪broker_port":59469,"simulation_broker_location":"127.0.0.1"},"application_config":{
↪"applications":[{"name":"der_dispatch_app","config_string":"{\"OPF\": 0, \"run_freq\
↪": 60, \"run_on_host\": false, \"run_realtime\": false, \"stepsize_xp\": 0.2, \
↪"stepsize_xq\": 2, \"coeff_p\": 0.1, \"coeff_q\": 5e-05, \"stepsize_mu\": 500}"}]},
↪"simulation_request_type":"NEW"}' '{OPF:0,run_freq:60,run_on_host:false,run_
↪realtime:false,stepsize_xp:0.2,stepsize_xq:2,coeff_p:0.1,coeff_q:5e-05,stepsize_mu:
↪500}'
```

```
{"OPF": 0, "run_freq": 60, "run_on_host": false, "run_realtime": true, "stepsize_xp":␣
↪0.2, "stepsize_xq": 2, "coeff_p": 0.1, "coeff_q": 5e-05, "stepsize_mu": 500}
```

CHAPTER

TWENTYSIX

SOLAR FORECASTING (NREL)

Solar-Forecast GridAPPS-D Application

26.1 Purpose

The application to forecast GHI data

26.2 Requirements

1. Docker ce version 17.12 or better. You can install this via the docker_install_ubuntu.sh script. (note for mint you will need to modify the file to work with xenial rather than ubuntu generically)

26.3 Quick Start

1. Please 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)

```
git clone https://github.com/GRIDAPPSD/gridappsd-docker
git clone https://github.nrel.gov/PSEC/Solar-Forecasting

ls -l

drwxrwxr-x  7 osboxes osboxes 4096 Sep  4 14:56 gridappsd-docker
drwxrwxr-x  5 osboxes osboxes 4096 Sep  4 19:06 Solar-Forecasting
```

26.4 Creating the sample-app application container

1. From the command line execute the following commands to build the sample-app container

```
osboxes@osboxes> cd Solar-Forecasting
osboxes@osboxes> docker build --network=host -t solar-forecasting-app .
```

2. Add the following to the gridappsd-docker/docker-compose.yml file

```
  solar_forecast:
    image: solar-forecast-app
    ports:
      - 9002:9002
    environment:
      GRIDAPPSD_URI: tcp://gridappsd:61613
    depends_on:
      - gridappsd
```

3. Run the docker application

```
osboxes@osboxes> cd gridappsd-docker
osboxes@osboxes> ./run.sh -t develop

# you will now be inside the container, the following starts gridappsd

gridappsd@f4ede7dacb7d:/gridappsd$ ./run-gridappsd.sh
```

4. Run bokeh on port 9002

```
python -m bokeh serve --show bokeh_two_plot/ --allow-websocket-origin=*:5006 --args --
↪port 9002
```

Next to start the application through the viz follow the directions here: https://gridappsd.readthedocs.io/en/latest/using_gridappsd/index.html#start-gridapps-d-platform

If you want to run the application WITHOUT the viz, open another terminal window and move to the solar-forecasting GitHub directory. Perform the following commands:

```
user@local> export PYTHONPATH=/Users/Solar-Forecasting/
user@local> python solar_forecasting/util/post_goss_ghi.py --start_time 1357140600 --
↪duration 720 --interval 10 -t .1
user@local> python solar_forecasting/util/post_goss_ghi.py --start_time 1370260800 --
↪duration 43200 --interval 60 -t .1

# The above options can be changed to a desired output
```

Docker

Two notes to use inside a docker container:

1. Add 9002 to the ports in the gridappsd-docker/docker-compose.yml like this:

```
  solar_forecasting:
    image: solar-forecasting-app
    ports:
      - 9002:9002
    environment:
      GRIDAPPSD_URI: tcp://gridappsd:61613
    depends_on:
      - gridappsd
```

```
>python /usr/src/gridappsd-solar-forecasting/solar_forecasting/app/main.py 1112306683
↪'{"power_system_config": {"SubGeographicalRegion_name": "_1CD7D2EE-3C91-3248-5662-
↪A43EFEFAC224", "GeographicalRegion_name": "_24809814-4EC6-29D2-B509-7F8BFB646437",
↪"Line_name": "_EBDB5A4A-543C-9025-243E-8CAD24307380"}, "simulation_config": {"power_
↪flow_solver_method": "NR", "duration": 48, "simulation_name": "ieee123", "simulator
↪": "GridLAB-D", "start_time": 1538484951, "run_realtime": true, "simulation_output":
↪ {}, "model_creation_config": {"load_scaling_factor": 1.0, "triplex": "y", "encoding
↪": "u", "system_frequency": 60, "voltage_multiplier": 1.0, "power_unit_conversion":␣
↪1.0, "unique_names": "y", "schedule_name": "ieeezipload", "z_fraction": 0.0, "i_
↪fraction": 1.0, "p_fraction": 0.0, "randomize_zipload": false, "use_houses
↪": false}, "simulation_broker_port": 52798, "simulation_broker_location": "127.0.0.1
↪"}, "application_config": {"applications": [{"name": "der_dispatch_app", "config_
↪string": "{\"OPF\": 1, \"run_freq\": 60, \"run_on_host\": false, \"run_realtime\":␣
↪true, \"stepsize_xp\": 0.2, \"stepsize_xq\": 2, \"coeff_p\": 0.1, \"coeff_q\": 5e-
```

(continues on next page)

(continued from previous page)

```

```

CHAPTER

TWENTYSEVEN

GRID FORECASTING (NREL)

Gird-Forecasting GridAPPS-D application

27.1 Purpose

The application to forecast GHI data

27.2 Requirements

1. Docker ce version 17.12 or better. You can install this via the docker_install_ubuntu.sh script. (note for mint you will need to modify the file to work with xenial rather than ubuntu generically)

27.3 Quick Start

1. Please 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)

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

ls -l

drwxrwxr-x  7 osboxes osboxes 4096 Sep  4 14:56 gridappsd-docker
drwxrwxr-x  5 osboxes osboxes 4096 Sep  4 19:06 Grid-Forecasting
```

27.4 Creating the sample-app application container

1. From the command line execute the following commands to build the sample-app container

```
osboxes@osboxes> cd Grid-Forecasting
osboxes@osboxes> docker build --network=host -t grid-forecasting-app .
```

2. Add the following to the gridappsd-docker/docker-compose.yml file Add 9003 to the ports in the docker-compose.yml allows for data to be sent to the bokeh streaming tool.

```
grid_forecasting:
  image: grid-forecasting-app
  ports:
    - 9003:9003
  environment:
    GRIDAPPSD_URI: tcp://gridappsd:61613
  depends_on:
    - gridappsd
  volumes:
    - $HOME/git/adms/Grid-Forecasting-Public:/usr/src/gridappsd-grid-forecasting
```

1. Run the docker application

```
osboxes@osboxes> cd gridappsd-docker
osboxes@osboxes> ./run.sh

# you will now be inside the container, the following starts gridappsd

gridappsd@f4ede7dacb7d:/gridappsd$ ./run-gridappsd.sh
```

Next to start the application through the viz follow the directions here: https://gridappsd.readthedocs.io/en/latest/using_gridappsd/index.html#start-gridapps-d-platform

CHAPTER

TWENTYEIGHT

GRIDAPPS-D DNP3 SERVICE

GridAPPS-D DNP3 service is an application service to integrate GridAPPS-D and a Distrbuted Newtork Protocol(DNP3)[1] based commercial product that allows operation, monitoring, analysis, restoration, and optimization of network operations to enable data exchange bewteen the applications. The DNP3 data exchange interface translates Common Information Model (CIM) standards data within GridAPPS-D to DNP3 packets and vice-versa to communicate with the DNP3 based software.Likewise, the service uses the existing GridAPPS-D platform to translate DNP3 data received from SurvalentONE. The use of GridAPPS-D made the development easier because of its standardized programming interface that unifies data integration.

Integration Architecture

This application is implemented as an application service consisting of python modules. DNP3 is a robust, efficient and interoperable protocol. Its compatibility with other protocols has made this integration easier. The integration is shown as a Master and Outstation link communication as shown in the figure below.

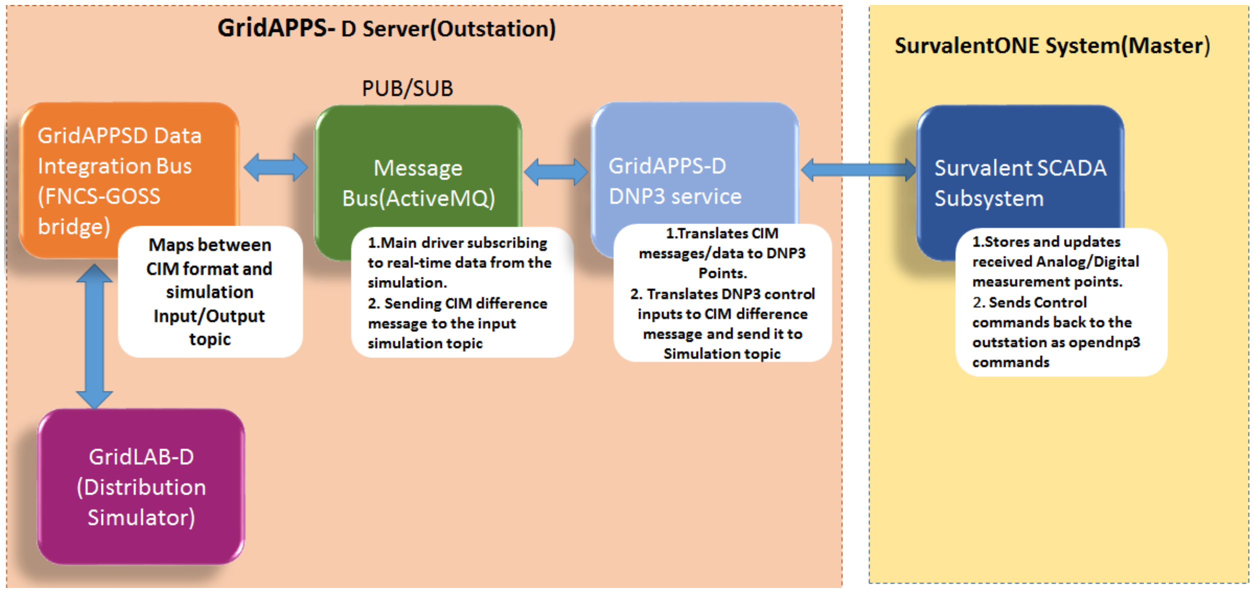

GridAPPS- D Server(Outstation)
SurvalentONE System(Master)
PUB/SUB
GridAPPSD Data Integration Bus (FNCS-GOSS bridge)
Message Bus(ActiveMQ)
GridAPPS-D DNP3 service
Survalent SCADA Subsystem
Maps between CIM format and simulation Input/Output topic
1.Main driver subscribing to real-time data from the simulation.
2. Sending CIM difference message to the input simulation topic
1.Translates CIM messages/data to DNP3 Points.
2. Translates DNP3 control inputs to CIM difference message and send it to Simulation topic
1.Stores and updates received Analog/Digital measurement points.
2. Sends Control commands back to the outstation as opendnp3 commands
GridLAB-D (Distribution Simulator)

Figure 1: Integration architecture

- The commercial tool used here is SurvalentONE[2], a DNP3 based SCADA system. The SurvalentONE SCADA system was configured as per the manuals provided by the Survalent Company. The Master server configuration includes creating a Station, Communication Link, Intelligent Electronic Device(IED) . The name and description of CIM measurement points of an IEEE model is added to a template (an excel sheet) containing all the DNP3 related metadata such as predefined datatypes, groups, variation. The template is created using a template maker provided by Survalent.Then the template is loaded in the Master IED of the Master server.
- Once this configuration is done and the DNP3 application service is created,start the service by creating a simulation

request. Since the service is added to gridapps container, it gets started as soon as the gridappsd service is started. A model dictionary file(CIM measurements) is generated for each simulation ID when a simulation request for a specific model is sent to the simulator(GridLAB-D). The simulation output from the simulator running under GridAPPS-D is received on the simulation output topic by the Message Bus.

- The DNP3 service at the Outstation and also the Communication Line at the Master Server on startup initiate a connection request from the Master, to the IP address of the Outstation at port 20000. In our case, the Outstation listens on port 20000. The DNP3 points created from the CIM model dictionary file are loaded by the DNP3 service in the Outstation database with the datatype, attribute,

description etc. fields. The Master polls the Outstation every 60 seconds to update the realtime data. The service translates model dictionary data into DNP3 points as JSON structures.An example of a JSON packet loaded in the Outstation database and sent to the Master is shown below.

```
{
        "attribute": "Switch.open",
        "data_type": "DO",
        "description":"Name:671692,ConductingEquipment_type:LoadBreakSwitchPhase:B,
↪controlAttribute:Switch.open",
        "group": 12,
        "index": 14,
        "measurement_id": "_56495819-9E14-DFEC-CC01-67526BBC9987",
        "name": "564958199E14DFECCC0167526BBC9987",
        "value": "0",
        "variation": 1
}
```

- The datatype, group and variation of the points are shown in the Table below.

```
Datatype                Group   Variation       Examples                              ␣
↪   Control/Status                  Sent From
--------------          ------  ----------      ----------------------                ␣
↪   -----------------               ------------
Analog  Output          42      3               Solar Panels, Capacitors, Regulators  ␣
↪   Control(Update set points)      Master to Outstation
Analog Input            30      1               AC Line segment                       ␣
↪   Measurement values               Outstation to Master
Digital Output          12      1               Switches,reclosers                    ␣
↪   Control, Enable, Disable        Master to Outstation
Digital Input           1       2               Breakers                              ␣
↪   Status                          Outstation to Master
```

- The on_message function in the service code keeps on updating the real-time

values for all the DNP3 points in Master server. The status and amgnitude values updates can be seen in the User Interfaces(Point viewers) provided by Survalent.

- SurvalentONE SCADA has the capability of controlling the points/measurements.

The output commands are generated using the point viewers' user interface. For example, a switch can be turned ON/OFF by selecting the control option manually. Similarly, analog values the AC Line segment can also be modified. The control points can be chosen from the options in the telemetry of the points. For a binary point five different inputs can be sent. The control inputs are sent in the form of OpenDNP3 Control relay Output Block commands to the Outstation and translated to CIM difference messages, sent back to FNCS bridge and then to the simulator. OpenDNP3 is an implementation of the DNP3 communication in the form of a library which provides C++ programming APIs[3]. The analog points can also be modified in a similar way. The values can be set and sent as an OpenDNP3 AnalogOutputInt16 command. The command has the value and status as input to the Outstation.

Results

The integration was tested using a 13-bus system. The JSON stored in Oustation database as shown in the example above is created for all the CIM measurement values in the dictionary file. An example of logs from Outstation at service start-up is shown below.

```
Adding Point: PointDefinition 00730ee504e148bf917fe47d5be16e6f (30.1, index=0,
↪type=Analog Input)
Recording DNP3 Analog measurement, index=88,value=564.143691452854
Recording DNP3 Binary measurement, index=10,value=True
Recording DNP3 Binary measurement, index=13,value=True
```

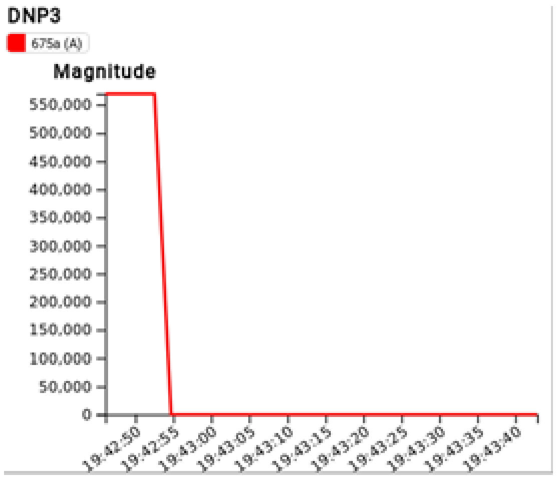
The control input from Master by sending a PULSE_OFF command for a switch which is an OpenDNP3 command sent from Master to Outstation as shown below.

```
cmdtype=Operate,command=<pydnp3.opendnp3.ControlRelayOutputBlock object at
0x7fd1f69527d8>,index=4,optype=OperateType.SELECT_BEFORE_OPERATE

Received DNP3 Point value ControlCode.PULSE_OFF (6ed7855451e84bc79db2c810d07509dd, 12.
↪1.4, Operate)
command_code=CommandStatus.SUCCESS,command_code=ControlCode.PULSE_OFF,command_ontime=0
```

The above command was sent back by the GridAPPS-D service in the form of a CIM difference message to the simulation input topic as shown below.The forward_differences value is the current value/status and reverse_differences value contains the older value/status of the switch.

```
{ "command":"update",
        "input":{
                "simulation_id":"854814554",
                "message":{ "timestamp":1568058139,
                "difference_mrid":"8be2c7a6-7cef-4e0e-af91-4f5eff37cf90",
                "reverse_differences":[
                        {
                        "object":"_56495819-9E14-DFEC-CC01-67526BBC9987",
                        "attribute":"Switch.open",
                        "value":1
                        }],
                "forward_differences":[
                        {
                        "object":"_56495819-9E14-DFEC-CC01-67526BBC9987",
                        "attribute":"Switch.open",
                        "value":0 } ]
}}}
```

This received CIM message was processed and the below plot shows the state of the switch at Viz.



28.1 References

[1] G. R. Clarke, D. Reynders, and E. Wright, Practical modern SCADA protocols: DNP3, 60870.5 and related systems. London: Elsevier, 2008.

[2] Survalent Technology Corporation. "Survalent.". https://www.survalent.com (accessed: Aug 2019).

[3] Automatak. "opendnp3". https://dnp3.github.io/ (accessed: June 2019).

CHAPTER

TWENTYNINE

GRIDAPPS-D SENSOR SIMULATOR SERVICE

The *GridAPPSD's Sensor Simulator* simulates real devices based upon the magnitude of "prestine" simulated values. This service has been specifically designed to work within the gridappsd platform container. The *GridAPPSD* platform will start the service when it is specified as a dependency of an application or when a service configuration is specified within the GridAPPSD Visualization. The image below shows a portion of the configuration options available through the service configuration panel.

_static/sensor-simulator-service-configuration.png

29.1 Python Application Usage

The python application using this service should require *gridappsd-sensor-simulator* as a requirement. In addition, the following python code shows how to get the correct topic for the service.

29.2 Service Configuration

The sensor-config in the above image shows an example of how to configure a portion of the system to have sensor output. Each mrid (*_99db0dc7-ccda-4ed5-a772-a7db362e9818*) will be monitored by this service and either use the default values or use the specified values during the service runtime.

```
{
   "_99db0dc7-ccda-4ed5-a772-a7db362e9818": {
      "nominal-value": 100,
      "perunit-confidence-band": 0.01,
      "aggregation-interval": 30,
      "perunit-drop-rate": 0.01
   },
   "_ee65ee31-a900-4f98-bf57-e752be924c4d":{},
   "_f2673c22-654b-452a-8297-45dae11b1e14": {}
}
```

The other options for the service are:

- default-perunit-confidence-band
- default-aggregation-interval

- default-perunit-drop-rate
- passthrough-if-not-specified

These options will be used when not specified within the sensor-config block.

Note: Currently the nominal-value is not looked up from the database. At this time services aren't able to tell the platform when they are "ready". This will be implemented in the near future and then all of the nominal-values will be queried from the database.

29.3 Request Example

The following is a full request example for use within the context of the main GridAPPSD api. This example uses the 123 node system with 3 sensors simulated. Also for this example those are the only measurements that will be published to the output sensor output topic.

```
{
    "power_system_config": {
        "GeographicalRegion_name": "_73C512BD-7249-4F50-50DA-D93849B89C43",
        "SubGeographicalRegion_name": "_1CD7D2EE-3C91-3248-5662-A43EFEFAC224",
        "Line_name": "_C1C3E687-6FFD-C753-582B-632A27E28507"
    },
    "application_config": {
        "applications": []
    },
    "simulation_config": {
        "start_time": "1570041113",
        "duration": "120",
        "simulator": "GridLAB-D",
        "timestep_frequency": "1000",
        "timestep_increment": "1000",
        "run_realtime": false,
        "simulation_name": "ieee123",
        "power_flow_solver_method": "NR",
        "model_creation_config": {
            "load_scaling_factor": "1",
            "schedule_name": "ieeezipload",
            "z_fraction": "0",
            "i_fraction": "1",
            "p_fraction": "0",
            "randomize_zipload_fractions": false,
            "use_houses": false
        }
    },
    "test_config": {
        "events": [],
        "appId": ""
    },
    "service_configs": [{
        "id": "gridappsd-sensor-simulator",
        "user_options": {
            "sensors-config": {
                "_99db0dc7-ccda-4ed5-a772-a7db362e9818": {
                    "nominal-value": 100,
```

(continues on next page)

(continued from previous page)

```
                        "perunit-confidence-band": 0.02,
                        "aggregation-interval": 5,
                        "perunit-drop-rate": 0.01
                    },
                    "_ee65ee31-a900-4f98-bf57-e752be924c4d": {},
                    "_f2673c22-654b-452a-8297-45dae11b1e14": {}
                },
                "random-seed": 0,
                "default-aggregation-interval": 30,
                "passthrough-if-not-specified": false,
                "default-perunit-confidence-band": 0.01,
                "default-perunit-drop-rate": 0.05
            }
        }]
}
```

Further information about the GridAPPSD platform can be found at https://gridappsd.readthedocs.org.

CHAPTER

THIRTY

GRIDAPPS-D VOLTAGE VIOLATION SERVICE

30.1 Purpose

This is a GridAPPS-D service that calculates and publishes voltage voilations during a simulation. Voltage violations are published every 15 seconds by default.

30.2 Topics

- Subscribes to simulation output topic /topic/goss.gridappsd.simulation.output.[simulation_id]
- Publishes on topic /topic/goss.gridappsd.simulation.voltage_violation.[simulation_id].output

30.3 Message Structure

- Simulation output message structure is available here: https://gridappsd.readthedocs.io/en/master/using_gridappsd/index.html#subscribe-to-simulation-output
- Voltage violation service publishes a JSON message with measurement MRIDs and their per unit voltage value. For example,

```
{
        "_00730ee5-04e1-48bf-917f-e47d5be16e6f": 1.07,
        "_078cfc25-8a2e-4f34-8631-0346abe2214d": 1.06,
        "_40df103f-b458-4c15-a2eb-ffe7d029ef65": 1.06,
        "_51fe8d20-a89e-497f-8c4b-d5bd654449bf": 1.08
}
```

CHAPTER

THIRTYONE

GRIDAPPS-D STATE ESTIMATOR SERVICE

31.1 Purpose

The state estimator service will produce and output the best available system state from measurements for use by other applications.

31.2 State estimator service layout

The following is the structure of the state estimator:

```
.
├── README.md
├── LICENSE
└── state-estimator
    ├── include
    ├── src
    ├── bin
    ├── obj
    ├── Makefile
    └── state-estimator.config
└───(Prerequisite libraries--SuiteSparse, ActiveMQ-CPP, Json)
```

31.3 Prerequisites

1. Docker ce version 17.12 or newer is required. You can install this via the docker_install_ubuntu.sh script in the gridappsd-docker repository described in the next step. (note for mint you will need to modify the file to work with xenial rather than ubuntu generically)
2. Clone the repository https://github.com/GRIDAPPSD/gridappsd-docker (referred to as gridappsd-docker repository).

```
~/git
└── gridappsd-docker
```

1. To run the gridappsd-docker platform, follow the instructions provided at https://github.com/GRIDAPPSD/gridappsd-docker/README.md.
2. To configure and run a simulation under the platform, follow the instructions provided at https://gridappsd.readthedocs.io/en/master/using_gridappsd.

3. The state estimator is distributed pre-built under the gridappsd-docker repository, but you may instead build the state estimator from source code from its own repository if you wish to modify it, run a different branch than master, or otherwise run it outside the gridappsd-docker container.
4. If you wish to run the state estimator provided with gridappsd-docker, follow the instructions in the following section, *Running state estimator from the gridappsd-docker container*.
5. Alternatively, to build the state estimator from source code and then run that version from the command line, skip to the section *Building state estimator* below.

31.4 Running state estimator from the gridappsd-docker container

1. Configure the simulation from the GRIDAPPSD platform web browser visualization.
2. Click on the "Service Configuration" tab and then click on the checkbox under the gridappsd-state-estimator section of the configuration user interface to specify that state estimator should be started with the simulation.
3. Click on the "Submit" button and then the play button to start the simulation.
4. The state estimator will process running simulation measurements producing state estimate messages for other applications.
5. The gridappsd-state-plotter application can be used to plot state estimator output as described at https://github.com/GRIDAPPSD/gridappsd-state-plotter.

The remainder of these instructions apply only when building the state estimator from source code and running that build from the command line.

31.5 Building state estimator

1. Clone the repository https://github.com/GRIDAPPSD/gridappsd-state-estimator next to the gridappsd-docker repository (they should both have the same parent folder, assumed to be ~/git in docker-compose.yml)

```
~/git
├── gridappsd-docker
└── gridappsd-state-estimator
```

1. Then the following two repositories should be cloned under the top-level gridappsd-state-estimator directory of the repository cloned above

```
        - https://github.com/GRIDAPPSD/SuiteSparse
        - https://github.com/GRIDAPPSD/json
```

1. The ActiveMQ C++ client library, ActiveMQ-CPP, should be downloaded from the URL below as a Unix source code distrubtion. Both the 3.9.4 and 3.9.5 releases have been successfully used with state estimator. The tar.gz or tar.bz2 distribution should be extracted under the gridappsd-state-estimator directory, the same location as the SuiteSparse and json repositories.

```
    - https://activemq.apache.org/components/cms/download
```

1. Building prerequisite libraries requires some other packages to be installed first. The following apt-get install commands should install those packages if they are not already installed:

```
sudo apt-get install cmake
sudo apt-get install m4
sudo apt-get install liblapack-dev libblas-dev
sudo apt-get install libapr1 libapr1-dev
sudo apt-get install libssl-dev
```

1. From the gridappsd-state-estimator directory, run the following commands to build the prerequisite libraries and then the state-estimator executable:

```
cd activemq-cpp-library-*
./configure
make
sudo make install

cd ../SuiteSparse
make LAPACK=-llapack BLAS=-lblas

cd ../state-estimator
make
```

1. The executable application will be placed in bin/state-estimator. The Json distribution consists entirely of include files and therefore is not compiled separately from the application using it.

31.6 Running state estimator from the command line

1. Edit the run-se.sh script in the state-estimator subdirectory of the top-level gridappsd-state-estimator repository, uncomment the appropriate SIMREQ variable for the model being run based on the comment at the end of each SIMREQ line denoting the corresponding model, and save changes.
2. Configure and start the simulation from the GRIDAPPSD platform web browser visualization, click on the "Simulation ID" value in the upper left corner of the simulation diagram to copy the value to the clipboard.
3. Invoke the script "./run-se.sh" from the command line with the Simulation ID value pasted from the clipboard as the command line argument to the script.
4. The state estimator will process running simulation measurements producing state estimate messages for other applications along with diagnostic log output to the terminal.
5. The gridappsd-state-plotter application can be used to plot state estimator output as described at https://github.com/GRIDAPPSD/gridappsd-state-plotter.

404: Not Found

CHAPTER

THIRTYTWO

GRIDAPPS-D ALARM SERVICE

This service publishes alarms.

CHAPTER

THIRTYTHREE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[CIT1] A. Dubey, A. Bose, M. Liu and L. N. Ochoa, "Paving the Way for Advanced Distribution Management Systems Applications: Making the Most of Models and Data," in *IEEE Power and Energy Magazine*, vol. 18, no. 1, pp. 63-75, Jan.-Feb. 2020

[CIT2] S. Poudel, A. Dubey and K. P. Schneider, "A Generalized Framework for Service Restoration in a Resilient Power Distribution System," *Submitted to IEEE Systems Journal (Under Revision)*

[CIT3] S. Poudel and A. Dubey, "Critical Load Restoration Using Distributed Energy Resources for Resilient Power Distribution System," *in IEEE Transactions on Power Systems*, vol. 34, no. 1, pp. 52-63, Jan. 2019.

[CIT4] S. Poudel, A. Dubey and A. Bose, "Risk-Based Probabilistic Quantification of Power Distribution System Operational Resilience," *in IEEE Systems Journal*, doi: 10.1109/JSYST.2019.2940939.

[CIT5] S. Poudel and A. Dubey, "A Graph-theoretic Framework for Electric Power Distribution System Service Restoration," *2018 IEEE Power & Energy Society General Meeting (PESGM)*, Portland, OR, 2018, pp. 1-5.

[CIT6] S. Poudel, A. Dubey, P. Sharma, and K. P. Schneider, "A Standalone FLISR Application Using GridAPPS-D – Standards-Based Open-Source ADMS Platform," *in prep for IEEE Power and Energy Technology Systems Journal*

[CIT7] S. Poudel, A. Dubey, and A. Bose, "Probabilistic Quantification of Power Distribution System Operational Resilience," *2019 IEEE Power & Energy Society General Meeting (PESGM)*, Atlanta, GA, 2019, pp. 1-5

[CIT8] S. Poudel, M. Mukherjee and A. Dubey, "Optimal Positioning of Mobile Emergency Resources for Resilient Restoration," *2018 North American Power Symposium (NAPS)*, Fargo, ND, 2018, pp. 1-6.

[CIT9] 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.

[CIT10] R. F. Arritt and R. C. Dugan, "The IEEE 8500-node test feeder," in *IEEE PES T&D 2010*, pp. 1-6.