SEUP Docs Documentation

Release 0.0.1

Ashray Manur

Apr 11, 2019

Contents

Simple Electric Utility Platform (SEUP) is an end-to-end platform for running and managing microgrids.

CHAPTER 1

Contents:

1.1 Contact

Question? Please contact Ashray Manur at manur@wisc.edu

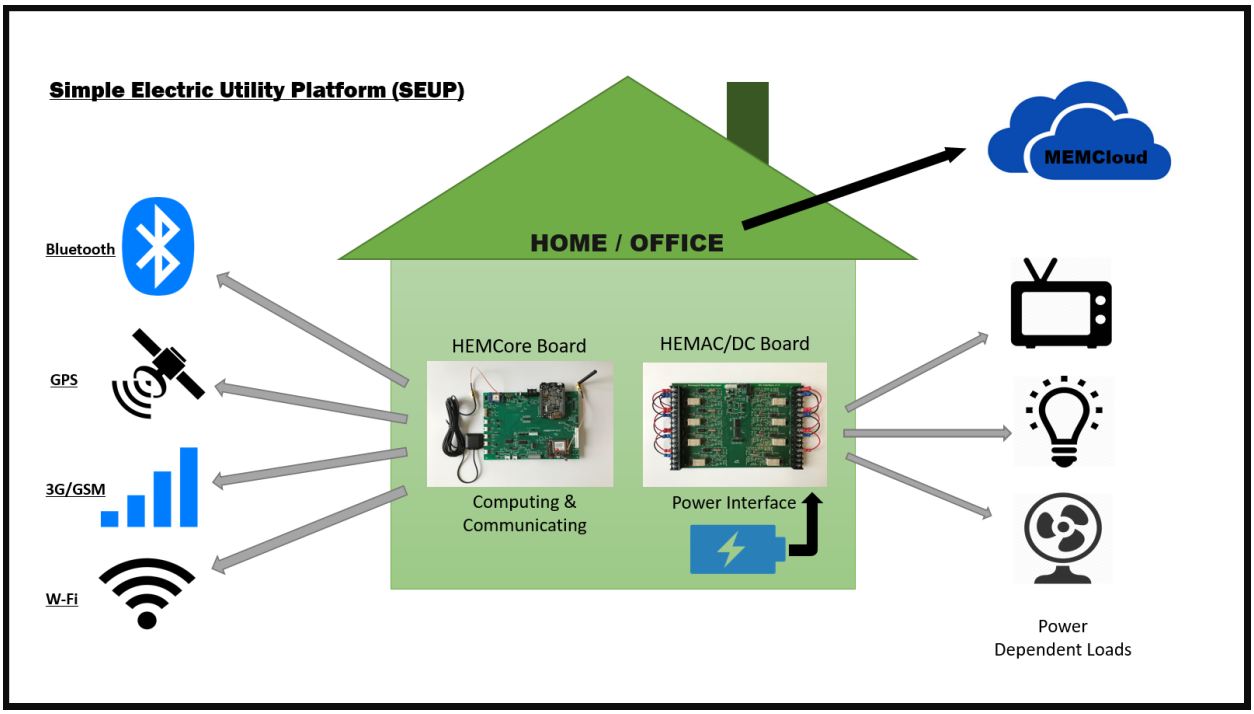
1.2 Nomenclature

- **SEUP** - Simple Electric Utility Platform. This is an end-to-end microgrid management solution designed and developed at WEMPEC, UW-Madison. It has various components listed below.
- **HEM** - Homegrid Energy Manager. Each home, building etc. becomes an autonomous entity called a homegrid which is managed by a HEM.
- **MGO** - Microgrid Operator. All the HEMs of a microgrid are managed by the microgrid operator.
- **MEM** - Microgrid Energy Manager - The entire infrastructure system to host multiple MGOs together is termed as microgrid energy manager
- **HEMCore** - hardware module used for communication and computing in a homegrid.
- **HEMAC/DC** - hardware modules used as power interface for power system.
- **HEMApp** - core software running HEM
- **MEMCloud** - cloud platform for microgrid management
- **MEMWeb** - web application for microgrid monitoring, control and management.

1.3 Introduction to SEUP

The physical system at it's most basic level is composed of two boards: the HEMCore board and the HEMAC/DC Board. These boards have an array of functionalities used by the microgrid to maximize performance for the user. The HemCore board, along with a processor mounted on it, is the brains of the device. It controls and maintains all of the

services the microgrid has to offer. The HEMAC/DC board controls the microgrid's power distribution to the various loads that are connected via nodes. Alongside the physical components of the system, there are several digital features that enable more robust control and monitoring of the system. These features include a cloud platform (MEMCloud) and web application (MEMWeb) that allows users to control their grid with ease and monitor their energy usage over time. Users simply need a stable Internet connection. The system also includes text-based control and monitoring (MEMText) of the grid through a variety of sms commands which are detailed later on.



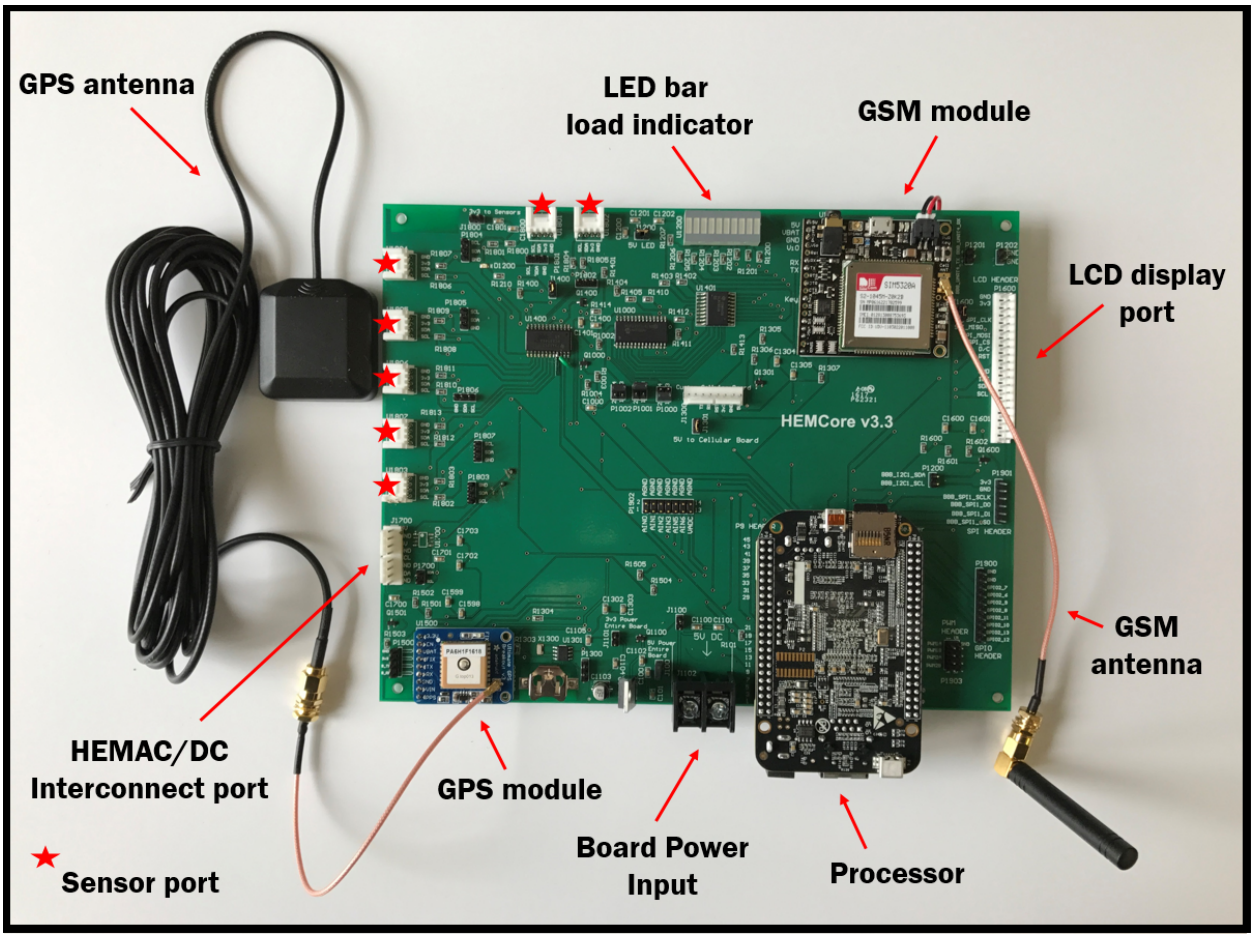
The system comes with a variety of communication features including: cellular, Wireless/Wired and Bluetooth. This not only allows for more flexibility in how one can control their grid but it also provides the user with a backup in case one feature cannot be accessed at a given time. The grid uses the cloud for encrypted data transmission/reception meaning your information is protected. GPS functionality allows users to geolocate their grid to get accurate scheduling time

1.3.1 Where can SEUP be used?

- Microgrids/Smartgrids
- Home Automation
- Building Energy Management
- Campus Grid Management
- Industry Power Management
- Teaching/Research

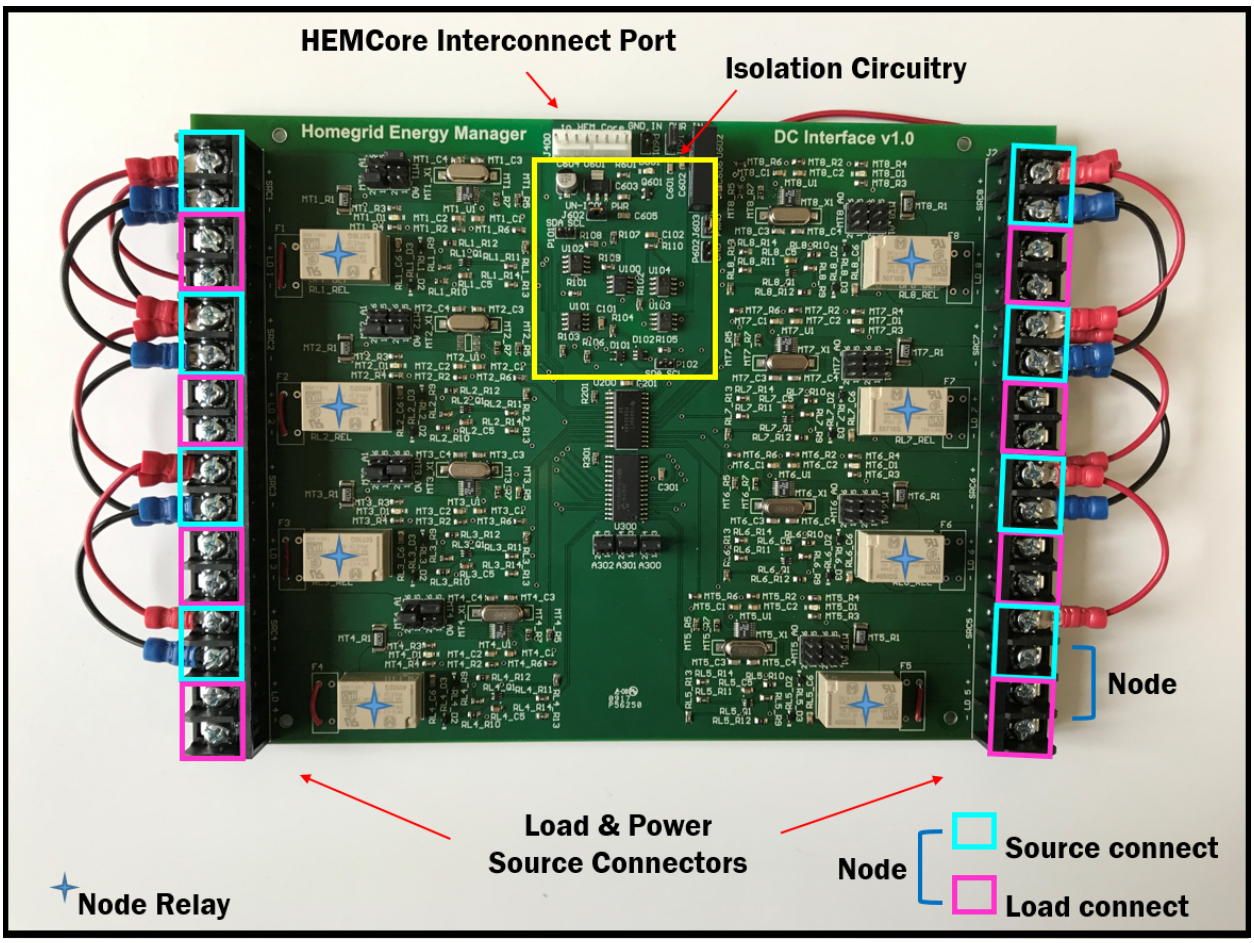
1.4 SEUP System Overview

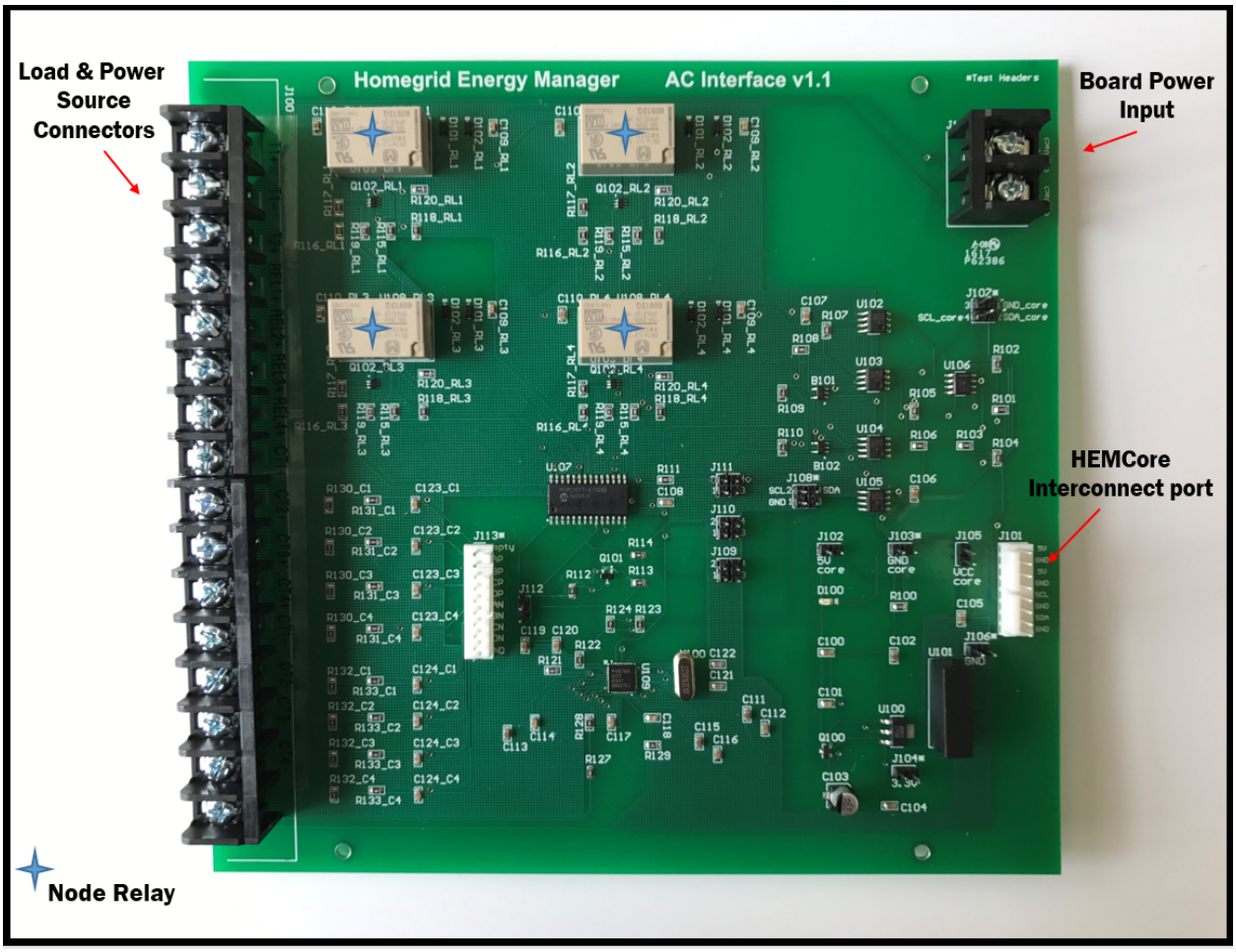
1.4.1 HEMCore Board Overview



The HEMCore board is the hub for all microgrid processes and communications made to the user/cloud. It has a processor mounted onboard that controls these processes which can be connected to the internet wirelessly via a USB wireless adapter or through Ethernet. The processor also controls the functionality on the microgrid's maintenance LCD screen. The board uses a GPS module to obtain the microgrids location and acquire the local time. The LED bar indicates the status (on/off) of each load connected to the board. The attached GSM module allows the device to send and receive texts. The board is powered by a 5V source via the 2 screw terminal block. On top of everything else, the processor has a microSD card slot which allows power usage data to be stored locally on an SD card in case the connection to the cloud is interrupted

1.4.2 HEMCore AC/DC Board Overview





The HEMAC and HEMDC boards are used for power facilitation between the HEMCore Board and the various loads connected to the microgrid system. These boards can open up/shut off power to these loads and also give feedback to the processor on power consumption and other useful data. Loads such as lights and fans are connected to boards through the terminal blocks marked with the text "LD X". Each load section has text indicating how to properly hook up the loads. The "+" and "-" symbols designate where to screw in the phase (power) and earth (ground) wires respectively. The DC and AC boards incorporate isolation circuitry to reduce the risk of failure and increase the safety of people working on the device

Note: The term "Node" will be used throughout to describe aspects of the device. for all intensive purposes, a node is what a load (fan, light, etc.) is hooked up to on the board. The node channel can either be opened (powered) or closed (un-powered). The channel is controlled by the Node relays connected to each node. There are 8 nodes on the DC board and 6 nodes on the AC board (pending). When sending commands via sms, we use node NOT load

1.5 Developing Energy Apps

You can develop and deploy energy apps on HEM or on MEMCloud. For latency sensitive applications, apps can be deployed on HEM. Microgrid-level and computationally intensive applications can be deployed on MEMCloud

1.6 Developing Apps with MEMCloud

Stay tuned! Coming soon!

1.7 Developing Apps with HEM

This section deals with developing and deploying apps locally on HEM.

1.7.1 Before you start

Make sure you are authorized to access the HEM boards

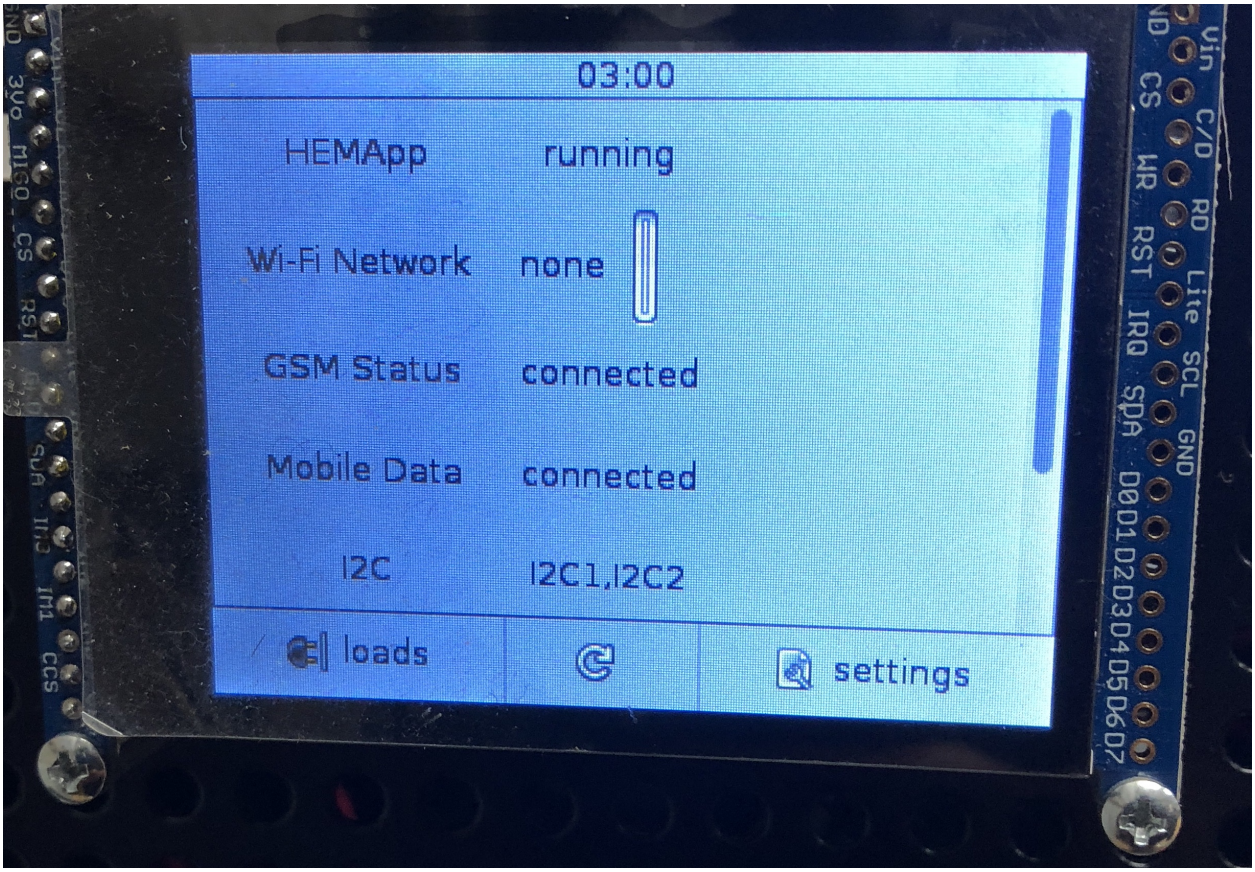
If you're working with HEM systems connected to live microgrid, make sure you are authorized to access it. HEMs control critical infrastructure and any unverified access and modifications might cause power system failure. You should get necessary permissions and training before you proceed.

Accessing HEM boards

HEM boards can be accessed through SSH (wired or wireless). The boards are password protected. Talk to the microgrid admin/manufacturer for these details.

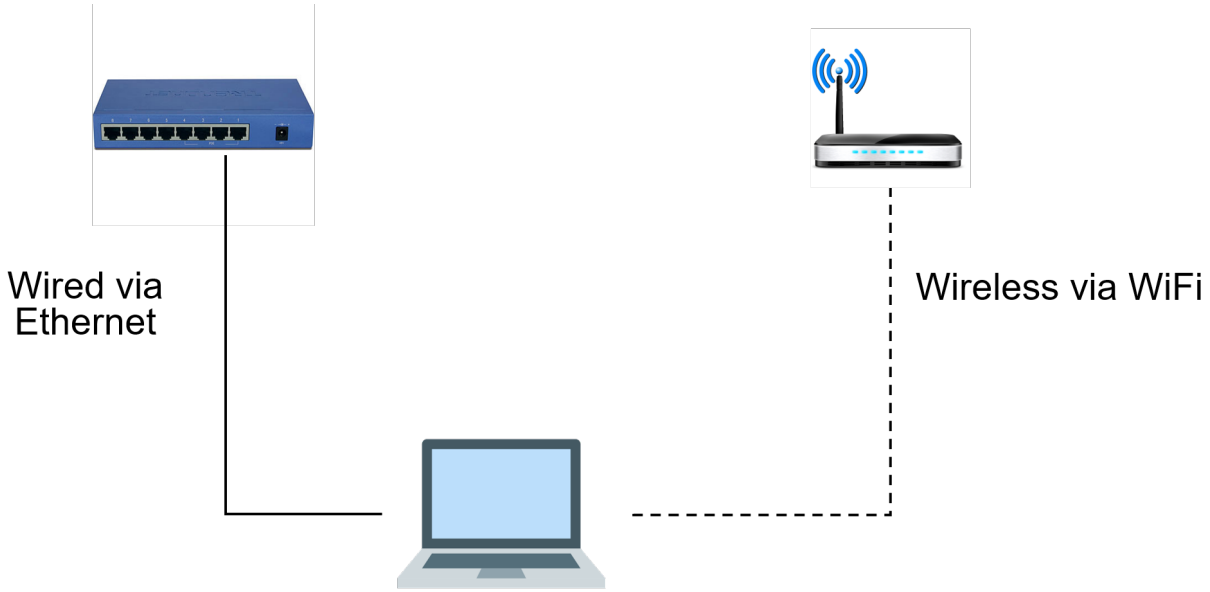
Making sure HEMApp is running

HEMApp is the core software on HEM which ensures these boards are up and running. Check if this is running. The easiest way is to check the touch-display and see if it displays the default screen as shown below.



1.7.2 Connect to SEUP Network

The SEUP network is a local network of all HEMs in the microgrid (wired or wireless). Once you connect your computer to the SEUP network, you can access information from all HEMs. If you're using a wired connection, please plug in your computer to the router or switch. If you're using a wireless connection, please connect your computer to the router. THe login details can be obtained from the microgrid admin.



1.7.3 Login

Each HEM board will have a static ip. This list is available from the microgrid admin.

Example only. Actual IP may differ

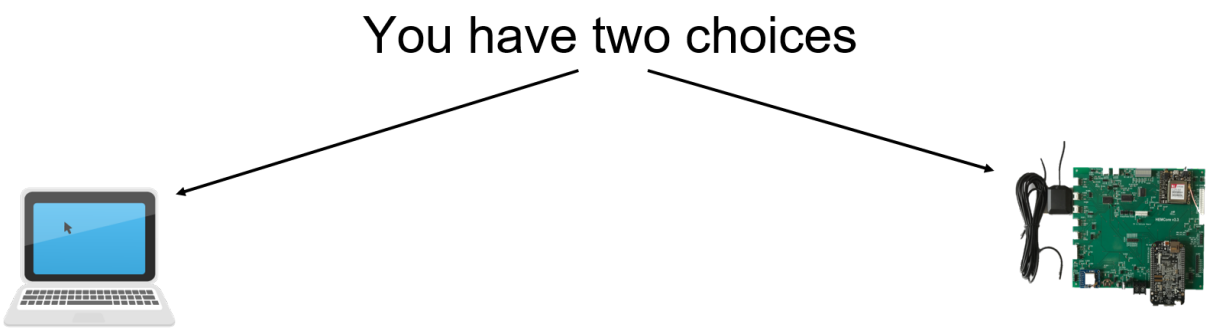
To login

```
ssh hem@192.168.1.28
```

1.7.4 What is an app?

When people hear the word 'app', most think of a sophisticated iOS/Android application for phones. We are here to change that mindset. For us, an app is a piece of software than can perform a task(s). It can be something as simple as a Python script with few lines of code which sends you an alert when your energy consumption crosses a threshold. While this might seem simple (and it is), its a valuable feature to have when you're battery is running low or your solar panel is not producing enough energy.

1.7.5 Where should you run your app?



- You can deploy your app on a computer connected to the SEUP network
- You can deploy your app on one of the HEMs in the SEUP network

When should you deploy your app on the computer?

- Your app does offline analysis
- It is computationally intensive
- Your app is UI (user interface) based and needs a large monitor, keyboard etc.
- Note: You can also run online apps if you have a dedicated computer on the SEUP network

When should you run your app on HEM?

- App has to be running continuously (24*7)
- It it computationally less intensive
- App is latency sensitive

1.7.6 Downloading the HEM Energy App Development Tool Kit

Create a new GitLab account

https://about.gitlab.com/

Upload your keys to GitLab

Here is the guide from GitLab

- https://docs.gitlab.com/ee/gitlab-basics/create-your-ssh-keys.html
- https://about.gitlab.com/2014/03/04/add-ssh-key-screencast/

SEUP HEM Energy App Dev. Repository

Go to https://gitlab.com/microgridenergymanager/SeupAppKit

Fork the repository

Here is the guide from GitLab - https://docs.gitlab.com/ce/gitlab-basics/fork-project.html

Clone the fork you created

```
git clone PASTE_YOUR_SSH_OR_HTTPS_PROJECT_NAME_HERE
```

Navigate to the development tool kit directory

```
$ cd SeupAppKit
```

`hem` directory contains all files necessary for developing and deploying apps on HEM

Understanding development tool kit directory structure

In `SeupAppKit`

- `mainHem_` - any file starting with this name is a sample app which is used to demo a particular feature
- `module` - directory which contains all the custom hem modules necessary to create an app
- `data` - this contains sample data from HEM for testing your apps
- `3rdPartyApps` - this contains apps developed for HEM deployment

Creating your first app

To create your first app, copy the folder `module` and the file `mainHem_bare.py` to the desired location. Both of these entities should be in the same directory.

`mainHem_bare.py` is a skeleton app. You can add your logic to this .

Sample apps included

There are a number of sample apps include to help you speed up your development. They are in `microgridbootcamp/hem/code`

Here is a list

- `mainHem_pullData.py` - Pull node(all) data from HEMApp Server through API
- `mainHem_pullData_single.py` - Pull single node data from HEMApp Server through API
- `mainHem_actuate.py` - Turn on/off nodes through API
- `mainHem_demandManage.py` - Perform load management when power consumption crosses threshold
- `mainHem_email.py` - Send an email from your app
- `mainHem_demandManage_email.py` - Send an email from your app when you do load management
- `mainHem_file.py` - Extract necessary information from data dump
- `mainHem_parsing.py`- Parsing data from data dump
- `mainHem_powerAggregate.py`- aggregate power data based on load/sources/charger for plotting and analytics
- `mainHem_report.py` - create a pdf report

Running a sample app

Let's run an app which turns on/off a particular node on HEM

Copy `mainHem_actuate.py` and `module`.

Check server address

If you're running this app on HEM, you can keep the `SERVER_NAME` in `mainHem_actuate.py` as `localhost`. If you're running this app on your computer on a different HEM (which is different from where you're pulling the data), change the `SERVER_NAME` to the local ip address.

If your app has to run in the background, use `nohup`

```
nohup python mainHem_actuate.py &
```

Stopping your app

Get the process id (pid) of your app process

```
ps aux | grep mainHem_actuate
```

The output of this looks something like this

```
user    70413  0.0  0.1 110404  9804 pts/8    Sl   13:33   0:00 python mainHem_
↪actuate.py
```

The pid in this case is *70413*

pid is the number listed in the PID column for your app

Now kill that process

```
pkill -9 <pid>
```

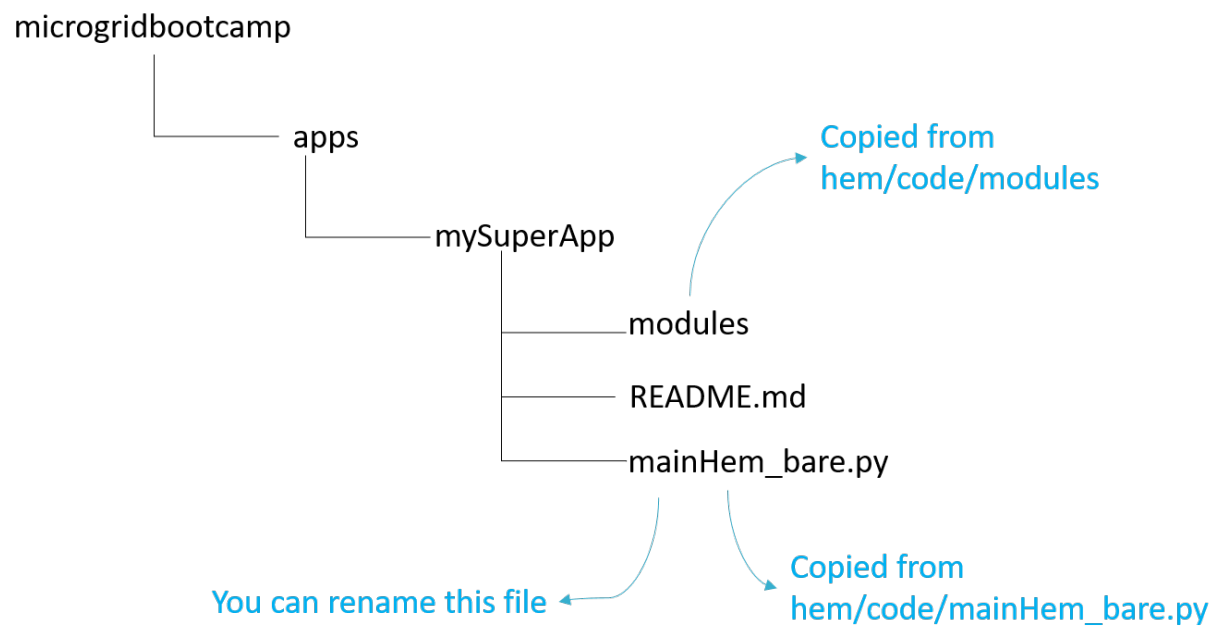
In this case, it would be something like

```
pkill -9 70413
```

After you're done developing and testing your app

After you've completed developing and testing your app you can add that as a folder to `SeupAppKit/3rdPartyApps` directory in your local repository

File/directory structure for your app



Documentation for your app

Add a README to your app

What should your README contain? (adapted from https://guides.github.com/features/wikis/)

- Project Name - Your project's name is the first thing people will see upon scrolling down to your README, and is included upon creation of your README file
- Description - A description of your project follows. A good description is clear, short, and to the point. Describe the importance of your project, and what it does.
- Installation - Installation is the next section in an effective README. Tell other users how to install your project locally. Optionally, include a gif to make the process even more clear for other people.
- Usage - The next section is usage, in which you instruct other people on how to use your project after they've installed it. This would also be a good place to include screenshots of your project in action.
- Bugs – Any bugs or quirks that exists in your app that developer or user should know

Make sure you comment your code so that it can be maintained and further developed

Here's a simple guide to commenting - https://www.cs.utah.edu/~germain/PPS/Topics/commenting.html

Creating a pull request

Now you want to push back your app to the main microgrid repository. You can do this through a pull request.

Here are two tutorials

- https://docs.gitlab.com/ee/gitlab-basics/add-merge-request.html
- https://www.digitalocean.com/community/tutorials/how-to-create-a-pull-request-on-github - this is for GitHub but can be applied to GitLab

1.7.7 Skeleton app

Let's look at a skeleton app that can be used as a starting point for app development

Open `mainHem_bare.py` in `microgridbootcamp/hem/code` to follow along with the rest of the section

```
#
# mainHem_bare.py
# Skeleton app that can be used for app deployments

# Author: Ashray Manur

import datetime
import threading
import time
import module.hemParsingData
from module.hemSuperClient import HemSuperClient

#Create a new HEM Client. This is used to send and receive data from your HEM
# It takes two arguments - IP/hostname and port
# If you're deploying your app on the same HEM you're getting data from use localhost.
↪Otherwise give IP address
#Port is usually 9931
hemSuperClient = HemSuperClient("localhost", 9931)

#This is function which gets triggered whenever you get data from HEM
#You can add more logic here for post processing
def update(message, address):
        print 'Rec:', message, address
        #{'NODE': 'ALL', 'TYPE': 'DCPOWER', 'VALUE': [0.185, 5.9, 85.6, 10.4, 0, 0, 0,
↪ 12.5]} ('192.168.1.236', 9931)
        #message is a list which gives you the type of response and the corresponding␣
↪nodes
        #address is a tuple giving you the server address and the port

#Subscribe to data from HEMs
# The argument to this is the name of the function you want triggered when you get␣
↪data
hemSuperClient.subscribe(update)

def main():

        while(1):

                #This sends a request to HEM every 5 seconds
                #Argument is the APIs
                hemSuperClient.sendRequest("api/getdcpower/all")
                time.sleep(5)

if __name__ == "__main__":
    main()
```

The sections below are explaining the various components of the app framework using `mainHem_bare.py` as an example

Importing Modules

The code sections below should be there in all your apps

Import your standard python modules

```
import datetime
import threading
import time
```

Import hem modules

```
from module.hemSuperClient import HemSuperClient
import module.hemParsingData
import module.hemEmail
```

Note: there might be other standard or custom modules that you might need to import based on what you're doing

Defining a trigger function

The code sections below should be there in all your apps

```
def update(message, address):
        print 'Rec:', message, address
```

This functions gets triggered every time your app gets a message from the HEM module. It takes two arguments `message` and `address`. When it is triggered, the actual data gets passed to `message` and the address of the server responding to your request gets passed to `response`

Creating a new HEM Client

This client is used to communicate with the HEM module. The HEM module runs a server to which you can send requests and receive responses This client abstracts the communication to make data communication between your app and the server easier.

The code sections below should be there in all your apps

```
#Creates a new HEM client to talk to the server
hemSuperClient = HemSuperClient("localhost", 9931)
```

If you're pulling data from the HEM where your app is deployed, keep it as `localhost`

If you're pulling data from other HEMs or your computer, change `localhost` to server ip like `192.168.1.28`

The port is usually `9931`

Defining multiple HEM Clients

Not included in `mainHem_bare.py`

In case your app is talking to many HEMs

```
hemSuperClient1 = HemSuperClient('localhost', 9931)
hemSuperClient2 = HemSuperClient('192.168.1.28', 9931)
hemSuperClient3 = HemSuperClient('192.168.1.29, 9931)
```

You can request data from all these servers as along as they are in the same network

Subscribing to updates

After you define your trigger function and the HEM client, you need to subscribe to updates. This ensures that whenever you get a response from HEM. This can be done by

The code sections below should be there in all your apps

```
hemSuperClient.subscribe(update)
```

The argument to this function is the name of the trigger function.

Understanding response data from server

Whenever data is sent from the server, the trigger function gets called and it gives you two values `message` and `address`.

`message` contains the response to the request you made `address` contains the details of the server making that response

`message` is a dictionary (key:value) and `address` is a tuple

```
print 'Rec:', message, address
```

A sample `message` is as follows. This response if for a request of power data for all nodes in HEM

```
{'NODE': 'ALL', 'TYPE': 'DCPOWER', 'VALUE': [0.185, 5.9, 85.6, 10.4, 0, 0, 0, 12.5]}
```

To extract the value:

```
message['VALUE']
```

The output would be

```
[0.185, 5.9, 85.6, 10.4, 0, 0, 0, 12.5]
```

To extract the node corresponding to this value

```
message['NODE']
```

The output would be

```
ALL
```

There is usually one key:value per request

1.7.8 API

API general format

```
/api/<action>/<node-number>
```

Actions can be

- Turn on/off
- Get electrical data

- Send messages

Node numbers usually go from 0-7

How to use the API

```
hemSuperClient.sendRequest(apiRequest)
```

Sample request

```
hemSuperClient.sendRequest("api/getdcpower/0")
```

Sample response

```
#message
{'NODE': '1', 'TYPE': 'DCPOWER', 'VALUE': [45.7]}
```

SEUP DC Systems

This section is for SEUP DC.

Turn off nodes

Format

```
/api/turnoff/<node-number>
```

Sample

```
/api/turnoff/3
```

Sample response

```
#message
{'NODE': '3', 'TYPE': 'TURNON', 'VALUE': [0]}
```

For 'VALUE' 1 means ON and 0 means OFF

Turn on nodes

Format

```
/api/turnon/<node-number>
```

Sample

```
/api/turnon/1
```

Sample response

```
#message
{'NODE': '1', 'TYPE': 'TURNON', 'VALUE': [1]}
```

For 'VALUE' 1 means ON and 0 means OFF

Status of single node

Format

```
/api/getnodestatus/<node-number>
```

Sample

```
/api/getnodestatus/1
```

Sample response

```
#message
{'NODE': '1', 'TYPE': 'STATUS', 'VALUE': [1]}
```

For 'VALUE' 1 means ON and 0 means OFF

Status for all nodes

Format

```
/api/getnodestatus/all
```

Sample response

```
#responseData
{'NODE': 'ALL', 'TYPE': 'STATUS', 'VALUE': [0,0,0,1,1,1,0,1]}
```

0 is off and 1 is on

DC voltage for single node

Format

```
/api/getdcvoltage/<node-number>
```

Sample

```
/api/getdcvoltage/3
```

Sample response

```
#message
{'NODE': '3', 'TYPE': 'DCVOLT', 'VALUE': [14.2]}
```

The value is in Volts

DC voltage values for all nodes

Format

```
/api/getdcvoltage/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'DCVOLT', 'VALUE': [13.625, 13.6, 13.625, 13.4, 13.7, 13.8,␣
↪13.6, 13.6]}
```

The value is in Volts

DC current for single node

Format

```
/api/getdccurrent/<node-number>
```

Sample

```
/api/getdccurrent/4
```

Sample response

```
#message
{'NODE': '4', 'TYPE': 'DCCURRENT', 'VALUE': [1.88375]}
```

The value is in Amps

DC current values for all nodes

Format

```
/api/getdccurrent/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'DCCURRENT', 'VALUE': [1.6, 1.2, 0, 0.4, 3.7, 1.8, 1.3, 3.6]}
```

The value is in Amps

DC power for single node

Format

```
/api/getdcpower/<node-number>
```

Sample

```
/api/getdccpower/5
```

Sample response

```
#message
{'NODE': '5', 'TYPE': 'DCPOWER', 'VALUE': [26.74925]}
```

The value is in Watts

DC power values for all nodes

Format

```
/api/getdcpower/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'DCPOWER', 'VALUE': [0.185, 0.185, 5.3,0,0,0,0,0]}
```

The value is in Watts

DC energy for single node

Format

```
/api/getdcenergy/<node-number>
```

Sample

```
/api/getdcenergy/1
```

Sample response

```
#message
{'NODE': '1', 'TYPE': 'DCENERGY', 'VALUE': [2252428.391018]}
```

The value is in Joules. You can convert it to Wh

DC energy values for all nodes

Format

```
/api/getdcenergy/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'DCENERGY', 'VALUE': [29.21, 16.688, 13.5,3.52,0,0,0.3358,0]}
```

The value is in Joules. You can convert that to Wh

DC charge for single node

Format

```
/api/getdccharge/<node-number>
```

Sample

```
/api/getdccharge/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'DCCHARGE', 'VALUE': [170782.33881]}
```

The value is in Coulombs

DC charge values for all nodes

Format

```
/api/getdccharge/all
```

Sample response

```
#responseData
{'NODE': 'ALL', 'TYPE': 'DCCHARGE', 'VALUE': [8779.87, 5014.19, 4066.334, 1040.689,0.
↪000328,0.000328,15.080517,0]}
```

The value is in Coulombs

SEUP AC Systems

This section is for SEUP AC.

Turn off nodes

Format

```
/api/turnoff/<node-number>
```

Sample

```
/api/turnoff/3
```

Sample response

```
#message
{'NODE': '3', 'TYPE': 'TURNON', 'VALUE': [0]}
```

For 'VALUE' 1 means ON and 0 means OFF

Turn on nodes

Format

```
/api/turnon/<node-number>
```

Sample

```
/api/turnon/1
```

Sample response

```
#message
{'NODE': '1', 'TYPE': 'TURNON', 'VALUE': [1]}
```

For 'VALUE' 1 means ON and 0 means OFF

Status of single node

Format

```
/api/getnodestatus/<node-number>
```

Sample

```
/api/getnodestatus/1
```

Sample response

```
#message
{'NODE': '1', 'TYPE': 'STATUS', 'VALUE': [1]}
```

For 'VALUE' 1 means ON and 0 means OFF

Status for all nodes

Format

```
/api/getnodestatus/all
```

Sample response

```
#responseData
{'NODE': 'ALL', 'TYPE': 'STATUS', 'VALUE': [0,0,0,1,1,1,0,1]}
```

For 'VALUE' 1 means ON and 0 means OFF

AC voltage for single node

Format

```
api/getacvoltage/<node-number>
```

Sample

```
/api/getacvoltage/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACVOLT', 'VALUE': [115.717569]}
```

This is the RMS voltage and is in Volts

AC voltage for all nodes

Format

```
/api/getacvoltage/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'ACVOLT', 'VALUE': [115.82487, 115.82487, 115.82487, 115.
↪82487, 115.82487, 115.82487]}
```

This is the RMS voltage and is in Volts

AC current for single node

Format

```
/api/getdcenergy/<node-number>
```

Sample

```
/api/getaccurrent/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACCURRENT', 'VALUE': [0.339167]}
```

This is the RMS current and is in Amps

AC current for all nodes

Format

```
/api/getaccurrent/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'ACCURRENT', 'VALUE': [0.339141, 0.304396, 0.009357, 0.005452,
↪ 0.003518, 0.003526]}
```

This is the RMS current and is in Amps

AC active power for single node

Format

```
api/getacpoweractive/<node-number>
```

Sample

```
api/getacpoweractive/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACPOWERACTIVE', 'VALUE': [39.164083]}
```

AC active power for all nodes

Format

```
/api/getacpoweractive/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'ACPOWERACTIVE', 'VALUE': [39.127099, 34.572928, 0, 0, 0, 0]}
```

AC reactive power for single node

Format

```
/api/getacpowerreactive/<node-number>
```

Sample

```
api/getacpowerreactive/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACPOWERREACTIVE', 'VALUE': [-0.74302]}
```

AC reactive power for all nodes

Format

```
/api/getacpowerreactive/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'ACPOWERREACTIVE', 'VALUE': [-0.744076, -5.766048, 0, 0, 0,␣
↪0]}}
```

AC active energy for single node

Format

```
/api/getacenergyactive/all
```

Sample

```
api/getacenergyactive/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACENERGYACTIVE', 'VALUE': [0.034566]}
```

AC active energy for all nodes

Format

```
/api/getdcenergy/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'ACENERGYACTIVE', 'VALUE': [0.034648, 0.030616, 0, 0, 0, 0]}
```

AC reactive energy for single node

Format

```
api/getacenergyreactive/<node-number>
```

Sample

```
api/getacenergyreactive/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACENERGYREACTIVE', 'VALUE': [-0.000741]}
```

AC reactive energy for all nodes

Format

```
api/getacenergyreactive/all
```

Sample response

```
#message
{u'NODE': u'ALL', u'TYPE': u'ACENERGYREACTIVE', u'VALUE': [-0.000658, -0.005185, 0, 0,
↪ 0, 0]}
```

AC frequency for single node

Format

```
api/getacfrequency/<node-number>
```

Sample

```
api/getacfrequency/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACFREQUENCY', 'VALUE': [59.967205]}
```

AC frequency for all nodes

Format

```
api/getacfrequency/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'ACFREQUENCY', 'VALUE': [59.967205, 59.967205, 59.967205, 59.
↪967205, 59.967205, 59.967205, 0, 0]}
```

AC angle for single node

Format

```
"api/getacangle/<node-number>"
```

Sample

```
api/getacangle/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACANGLE', 'VALUE': [-0.815625]}
```

AC angle for all nodes

Format

```
api/getacangle/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'ACANGLE', 'VALUE': [-0.73125, -9.50625, -21.65625, -27.5625,␣
↪68.428125, -11.025]}
```

AC power factor for single node

Format

```
/api/getacangle/<node-number>
```

Sample

```
api/getacangle/0
```

Sample response

```
#message
{'NODE': '0', 'TYPE': 'ACPF', 'VALUE': [0.999797]}
```

AC power factor for all nodes

Format

```
api/getacangle/all
```

Sample response

```
#message
{'NODE': 'ALL', 'TYPE': 'ACPF', 'VALUE': [0.999797, 0.985777, 0.924442, 0.27144, 0.
↪364928, 0.012272]}
```

1.7.9 HEM Modules

This section deals with the important modules that you can use to build your app. These can be found in `SeupAppKit/module`

hemSuperClient.py

This module is used to communicate with the server

You can import it with

```
from module.hemSuperClient import HemSuperClient
```

Check out `mainHem_pullData.py` to learn how to use it.

hemEmail.py

This module is used to send email with the server

You can import it with

```
import module.hemEmail
```

Check out `mainHem_email.py` to learn how to use it

hemParsingData.py

This module parses data from HEM data dump to extract electrical and node data

You can import with

```
import module.hemParsingData
```

Check out `mainHem_parsing.py` to learn how to use it

hemPowerAggregator.py

This module parses data and aggregates power data based on node types (sources, loads, charger)

You can import with

```
from module.hemPowerAggregator import HemPowerAggregator
```

Check out `mainHem_powerAggregate.py` to learn how to use it

1.7.10 Tutorials

Understanding the HEM data dump

HEMApp (the core software which runs HEM) dumps all the electrical data (voltage, current, power etc.) on the microSD card on the processor. The file is `data.csv` and is located at `/media/card`

Here are few lines from `data.csv`

```
VOLT:13.65,NODE:0,DATE:12/8/17-2:28:40
VOLT:13.62,NODE:1,DATE:12/8/17-2:28:40
VOLT:13.65,NODE:2,DATE:12/8/17-2:28:40
VOLT:13.62,NODE:3,DATE:12/8/17-2:28:40
VOLT:13.65,NODE:4,DATE:12/8/17-2:28:40
VOLT:13.65,NODE:5,DATE:12/8/17-2:28:40
VOLT:13.62,NODE:6,DATE:12/8/17-2:28:40
VOLT:13.62,NODE:7,DATE:12/8/17-2:28:40
AMP:0.00,NODE:0,DATE:12/8/17-2:28:40
AMP:0.00,NODE:1,DATE:12/8/17-2:28:40
AMP:0.00,NODE:2,DATE:12/8/17-2:28:40
AMP:0.00,NODE:3,DATE:12/8/17-2:28:40
AMP:0.00,NODE:4,DATE:12/8/17-2:28:40
AMP:0.00,NODE:5,DATE:12/8/17-2:28:40
AMP:0.00,NODE:6,DATE:12/8/17-2:28:40
AMP:0.00,NODE:7,DATE:12/8/17-2:28:40
POW:1.00,NODE:0,DATE:12/8/17-2:28:40
POW:2.00,NODE:1,DATE:12/8/17-2:28:40
POW:3.00,NODE:2,DATE:12/8/17-2:28:40
POW:4.00,NODE:3,DATE:12/8/17-2:28:40
POW:5.00,NODE:4,DATE:12/8/17-2:28:40
POW:6.00,NODE:5,DATE:12/8/17-2:28:40
POW:7.00,NODE:6,DATE:12/8/17-2:28:40
POW:8.00,NODE:7,DATE:12/8/17-2:28:40
```

Here the labels mean the following

- `VOLT:13.65` - `VOLT` is dc voltage and the value followed by `:` is the voltage in volts. Similarly, `AMP`, `POW`, `ENERGY`, `CHARGE` are for dc current, power, energy and charge respectively.
- `NODE:0` - indicates the node number. This can go from `0-7` for dc systems and `0-5` for ac systems.
- `DATE:12/8/17-2:28:40` - indicates the date and time.

Extracting/copying data from HEM data dump

HEM Data Dump - HEM stores all the electrical data in a file `data.csv` in `/media/card`. This file is usually hundreds of megabytes. Copying the entire file for a small amount of specific data is unreasonable.

This tutorial shows how you can copy specific information from the data dump for your app. Three examples will be demonstrated.

- Only voltage data of all nodes for 2 days
- All electrical data of all nodes for a week
- Only power data of all nodes for 1 day

Open `mainHem_file.py` to follow along

To copy one day's worth of data from `sample.csv` (you can replace this with `/media/card/data.csv`) into a new file `oneDay.csv`:

```
cmd = "grep -e '10/1/17' ../data/sample.csv > ../data/oneDay.csv"
```

- `'10/1/17'` - the date of interest
- `/data/sample.csv` - this is the source file (for this demo). If you want the actual data dump, replace this with `/media/catd/data.csv`
- `/data/oneDay.csv` - this is the destination file

Execute the command

```
result = subprocess.check_output(cmd, shell=True)
```

Check if your new file was created

```
fileExists = Path('../oneDay.csv')
if(fileExists.is_file()):
        print 'file exists'
```

Copy one day's worth of voltage data

The command

```
cmd = "grep -e 'VOLT.*10/1/17' ../data/sample.csv > ../data/oneDayVolt.csv"
```

The search should satisfy both date and data type. Here the electrical data type is *VOLT*

Execute the command

```
result = subprocess.check_output(cmd, shell=True)
```

Check if the file exists

```
fileExists = Path('../data/oneDayVolt.csv')
        if(fileExists.is_file()):
                print 'file exists'
```

Copying one week worth of all data

The command

```
cmd = "grep -e '10/[1-8]/17' ../data/sample.csv > ../data/oneWeek.csv"
```

Execute the command

result = subprocess.check_output(cmd, shell=True)

Check if the file exists

```
#Check if the file exists. This is to make sure the file was created
fileExists = Path('../data/oneWeek.csv')
if(fileExists.is_file()):
        print 'file exists'
```

Parsing data from HEM data dump

Now that you've understood the format of the HEM data dump and know how to extract specific info let's see how we can get the electrical values from it

Open `mainHem_parsing.py` to follow along

Make sure you import the module

```
import module.hemParsingData
```

Open the data file and put it into read mode. In this example it is called `data.csv`

```
fh = open('../data/data.csv', 'r')
```

Iterate through each line of the file

```
for i, line in enumerate(fh):
```

Now pass this line into the module

```
parsedObj = module.hemParsingData.lineParser(line)
```

If you want to checkout `parsedObj`

```
print parsedObj['type'] #VOLT
print parsedObj['value'] #13.65
print parsedObj['node'] #0
print parsedObj['date'] # 2017-12-08 02:28:40
```

You can add your logic to use this data.

Developing a SCADA app for Monitoring and Control of HEMs on a local network

This tutorial deals with building a browser-based app for control and monitoring of all HEMs in a local network (or VPN).The HEM/computer running this SCADA app should also be connected to the same SEUP network either locally or through VPN.

A skeleton app already exists and you can use this to add more features such as live graphing, pricing, revenue management, remote on/off etc.

All the files necessary for this tutorial are available in `microgridbootcamp/hemDashboard`

Prerequisites

You should have experience and basic understanding of some web technologies and frameworks such as HTML, CSS, JavaScript, Bootstrap and jQuery.

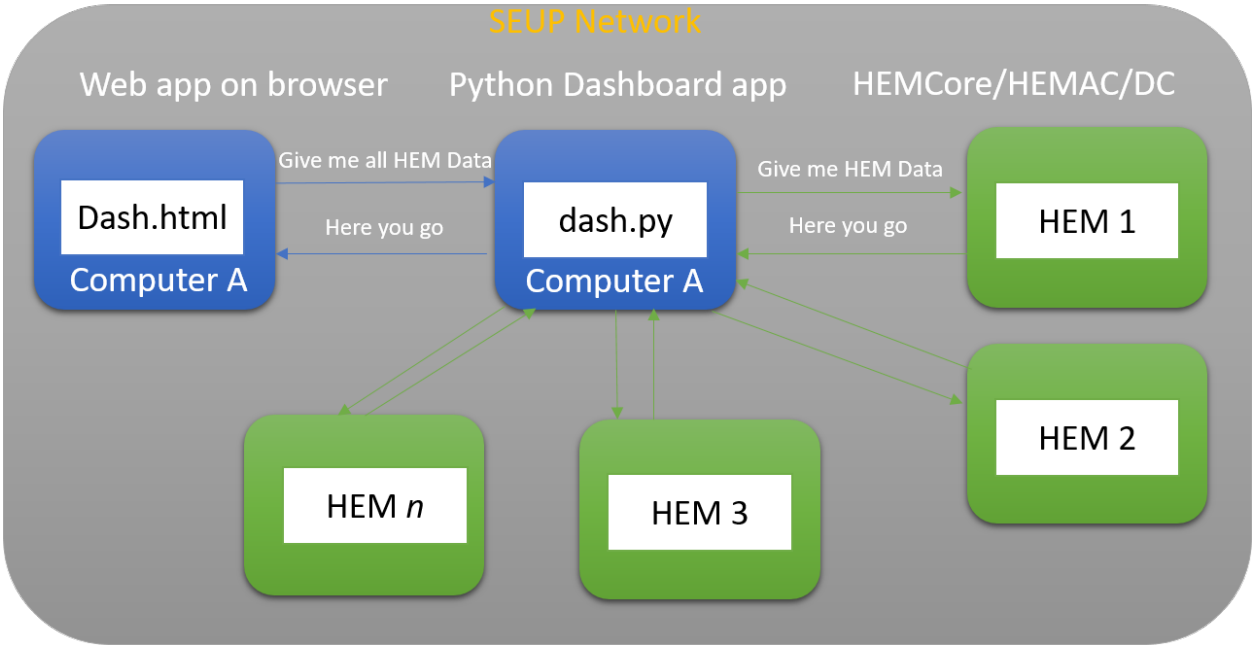
Understanding the directory structure

- `module` - this contains the custom HEM modules
- `templates` - this contains the HTML files
- `static` - contains all the design files and headers - CSS/JS/Bootstrap, images, videos etc.
- `dash.py` - is the skeleton app. You can use this as a starting point
- `dash.html` - is the skeleton html file. This is what is rendered by the browser. You can edit this file to change the way your app is rendered on the browser.

Where can I run this app

- You can run this app on Linux-based OS/MacOS/Windows computer as long as you have/can install Python.
- You could also run this on HEM boards connected to a monitor via HDMI. However, this would require a special version of HEMApp. The first option is recommended.

Understanding the basic architecture



The app `dash.py` runs on a computer which is on the same network (local or through VPN) as HEM. It pulls data from all these HEMs and also has the capability to send actuation signals to all the HEMs. Usually, the HEMs are all assigned static IPs and these are pre-defined and known to all applications.

The web app can be accessed on the same computer as the app or on a different computer. The web app can be accessed on the browser through an URL. The developer can modify the web dashboard and the Python app to suit the needs of the application.

How does it work

- The app `dash.py` requests data from all the HEMs and stores it locally. Accessing HEM data from Python app can be done through standard APIs defined earlier. See *API*
- The web app `dash.html` and associated web files then request data from `dash.py` to render it on the browser. The browser app also requests data from `dash.py` through a standard WEB API defined later.

Starting the dashboard app

For testing you can run

```
python dash.py
```

For running it continuously in the background

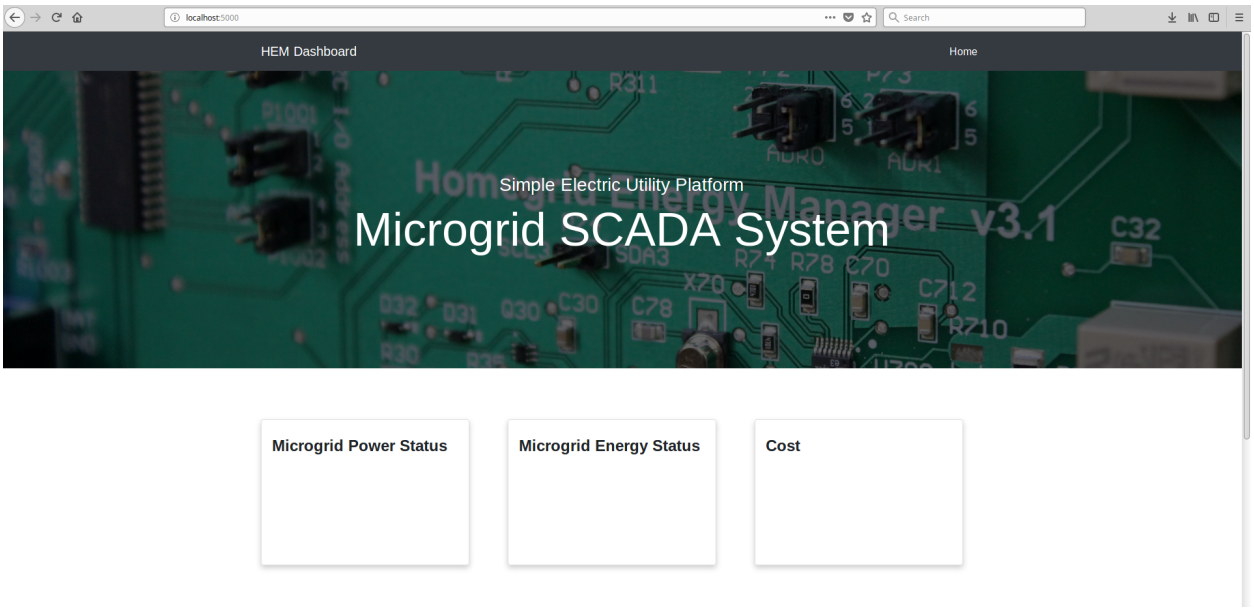
```
nohup python dash.py &
```

As soon as you start running the app, you should see this output

```
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 318-503-355
```

From this we know the URL of the web app is `http://127.0.0.1:5000/`

Open a browser and type in this URL. You should see something like below



Features of the skeleton app

The skeleton already has a couple of basic features

Pulls data from all HEMs

It automatically pulls `power`, `voltage`, `current`, `energy` and `charge` data from all HEMs you define. You can see `hemList = ['192.168.1.28']` lists only one HEM. You can add more HEMs to the list. For example if there was another HEM with the ip address `192.168.1.29``your new list looks like this - ``hemList = ['192.168.1.28', '192.168.1.29']`.

How to pull additional data from HEM

If you want more data from your HEM apart from the standard electrical data, you can pull data from HEM through the standard APIs defined. See *API*.

This section in `dash.py` is where the data is being requested from HEMs. You add your request to this section.

```
hemClient.sendRequest('/api/getdcpower/all', server_address)
time.sleep(0.1)
hemClient.sendRequest('/api/getdcvoltage/all', server_address)
time.sleep(0.1)
hemClient.sendRequest('/api/getdccurrent/all', server_address)
time.sleep(0.1)
hemClient.sendRequest('/api/getdcenergy/all', server_address)
time.sleep(0.1)
hemClient.sendRequest('/api/getdccharge/all', server_address)
time.sleep(0.1)
```

Maintains a global object with information of all HEMs

The app maintains `hemData` which is a dictionary of all HEMs and their data.

Whenever you request data from HEMs, it automatically sorts it and adds it to the HEM

Understanding the global object

`hemData` is a nested dictionary. The key is the ip address of the HEM. In this case it is *192.168.1.28*. The value associated with is another dictionary. This nested dictionary has keys `DCCURRENT`, `DCVOLT`, `DCENERGY`, `DCCHARGE` etc.

For example,

For example for a two HEM network (192.168.1.28 and 192.168.1.29), `hemData` looks like this

```
{'192.168.1.28': {u'DCVOLT': [13.625, 13.6, 13.625, 13.625, 13.625, 13.625, 13.6, 13.
↪6], u'DCENERGY': [54.698562, 42.207546, 55.773058, 3.760736, 0, 0, 0.033578, 0], u
↪'DCPOWER': [0.204375, 0.204, 0.204375, 0, 0, 0, 0, 0], u'DCCHARGE': [16438.2083,␣
↪12712.912803, 16767.096363, 1147.163972, 0.000328, 0.000328, 15.080517, 0], u
↪'DCCURRENT': [0.015, 0.015, 0.015, 0, 0, 0, 0, 0]},
'192.168.1.29': {u'DCVOLT': [13.625, 13.6, 13.625, 13.625, 13.625, 13.625, 13.6, 13.
↪6], u'DCENERGY': [54.698562, 42.207546, 55.773058, 3.760736, 0, 0, 0.033578, 0], u
↪'DCPOWER': [0.204375, 0.204, 0.204375, 0, 0, 0, 0, 0], u'DCCHARGE': [16438.2083,␣
↪12712.912803, 16767.096363, 1147.163972, 0.000328, 0.000328, 15.080517, 0], u
↪'DCCURRENT': [0.015, 0.015, 0.015, 0, 0, 0, 0, 0]}}
```

Now if we want to extract info for ip `192.168.1.28`, we can do `hemSubData = hemData['192.168.1.28']`

`print hemSubData` would look something like this

```
{u'DCVOLT': [13.625, 13.6, 13.625, 13.625, 13.625, 13.625, 13.6, 13.6], u'DCENERGY':␣
↪[54.698562, 42.207546, 55.773058, 3.760736, 0, 0, 0.033578, 0], u'DCPOWER': [0.
↪204375, 0.204, 0.204375, 0, 0, 0, 0, 0], u'DCCHARGE': [16438.2083, 12712.912803,␣
↪16767.096363, 1147.163972, 0.000328, 0.000328, 15.080517, 0], u'DCCURRENT': [0.015,␣
↪0.015, 0.015, 0, 0, 0, 0, 0]}
```

Now if we want to extract info of voltage, power etc. you can access them through individual keys

`print hemSubData['DCVOLT']` would give you

```
[13.625, 13.6, 13.625, 13.625, 13.625, 13.625, 13.6, 13.6]
```

`print hemSubData['DCCURRENT']` would give you

```
[0.015, 0.015, 0.015, 0, 0, 0, 0, 0]
```

`print hemSubData['DCCHARGE']` would give you

```
[16438.2083, 12712.912803, 16767.096363, 1147.163972, 0.000328, 0.000328, 15.080517,␣
↪0]
```

They each return a list of values for all nodes. The order is node 0 to 7. This is consistent across the code base.

Adding logic/intelligence

Let's say you want to do something more than just displaying data - add logic/intelligence.

You can do this in two ways

- Add a function in `dash.py` - this works best when you want your logic/intelligence to run at certain instants in time (or called from other functions/threads)
- Add a thread in `dash.py` - this works best when you want your logic to run continuously/periodically.

How to add a thread? (assuming you know how to add/define functions)

`app.py` has two default threads

```
thread.start_new_thread(sendToServer, ())
thread.start_new_thread(receiveFromServer, ())
```

You can add your thread in this section of the code

```
thread.start_new_thread(sendToServer, ())
thread.start_new_thread(receiveFromServer, ())
thread.start_new_thread(myManagementAlgorithm, ())
```

Then define your `myManagementAlgorithm` thread

```
def myManagementAlgorithm():

        # run this continuously
        while True:

                #add your logic here

                #loop (or execute your logic) every 2 seconds
                time.sleep(2)
```

Adding a route

Adding routes means defining a new URL so that your web application can access certain data using this URL. URl which stands for uniform resource locator is a way to access resources from your web application.

`dash.py` already defines many routes. Here's a list of the default routes

- `/` . Whenever you type in `http://localhost:5000` this route gets called.
- `/getdcpower/<hemip>` - this is used to get the dc power values for the hem with a specific ip.
- `/getdcvoltage/<hemip>` - this is used to get the dc voltage values for the hem with a specific ip.
- `/getdccurrent/<hemip>` - this is used to get the dc current values for the hem with a specific ip.
- `/getdcenergy/<hemip>` - this is used to get the dc energy values for the hem with a specific ip.
- `/getdccharge/<hemip>` - this is used to get the dc charge values for the hem with a specific ip.

How do you add a new route?

Add your code just below the default routes

Start with this line

```
@app.route("/getsomething")
```

When the browser accesses this route `http://localhost:5000/getsomething`, it is handled by the above line of code.

Now you want to respond to the request at this route. So you define a function to respond

```
@app.route("/getsomething")
def getsomething():
        return 'Hello, World'
```

Note: your function name should match the route. You can notice that both the route and function have a common name - `getsomething`

`return 'Hello, World'` sends a response to the request. You can substitute this with any response object.

For more on routing, see this

1.7.11 Sample Apps for Deployment on HEM

Sample apps can be found in

```
microgridbootcamp/hem/code
```

You can copy code snippets from these apps into your app

HEM Data Acquisition App

```
mainHem_pullData_single.py
```

This app demos how to request data from server on a per node basis

HEM Data Acquisition App 2

```
mainHem_pullData.py
```

This app demos how to request data from server for all nodes

HEM Node Actuation App

```
mainHem_actuate.py
```

This app shows you how to turn on/off nodes

HEM Online Demand Management App

```
mainHem_demandManage.py
```

This app checks cumulative power consumption of all loads and shuts off the non-critical loads if the threshold is crossed

HEM Email Reporting App

```
mainHem_email.py
```

Simple app to show you how to send an email

HEM Demand Management + Email Reporting App

```
mainHem_demandManage_email.py
```

This app shows you how to do demand management and send an email notification/report

HEM Data Extraction App

```
mainHem_file.py
```

Shows you how to copy the necessary information from the HEM data dump

1.8 Troubleshooting

1.8.1 HEM related questions

Coming Soon!

1.8.2 MEMCloud related questions

Coming Soon!

1.9 Student Contributors

A number of students have worked on the SEUP project in varying capacity and time period (few weeks to 1-2 years)

1.9.1 University of Wisconsin-Madison

(In no particular order)

- Zeng Fan
- Matthew Wawiorka

- Lee Shaver
- Travis Leanna
- Ben Chylla
- Sam Singer
- Alec Sivit
- Morgan Zhang
- Ashray Manur
- David Sehloff
- Adria Brooks
- Andrew Gilles
- Raul Martins
- Maitreyee Marathe

1.9.2 National Institute of Engineering

(In no particular order)

- Abhishek R
- Sindhu Dixith
- Vinay Joshi
- Anup Athreya
- Ansu Alex
- Akshay VJ
- Deepak G
- Dhananjaya KN
- NIE Microgid Phase 1-4 student teams

1.9.3 Others

(In no particular order)

- Anna Lunes
- Nicole Bugay