controller_usage_demonstration

May 13, 2017

This Jupyter Notebook demonstrates how to enable the GridBallast controller for a load (a water heater or a zip load) in GridLAB-D by feeding the simulator a stored grid frequency time series contained in an external file.

To run this notebook, please make sure you are in a UNIX based environment and have all the necessary python packages installed (plotly, matplotlib, numpy, pandas).

```
In [1]: !ls
```

```
controller_usage_demonstration.ipynb smSingle.glm
controller_usage_demonstration.pdf   smSingle_base.glm
correct_path.sh                      smSingle_lenient_freq.glm
fan1.csv                             smSingle_strict_freq.glm
fan2.csv                             smSingle_strict_freq_jitter300.glm
fan2_base.csv                        smSingle_strict_freq_jitter60.glm
fan2_lenient_freq.csv                smSingle_strict_freq_jitter600.glm
fan2_strict_freq.csv                 wh1_base.csv
fan2_strict_freq_jitter60.csv        wh1_lenient_freq.csv
fan2_strict_freq_jitter600.csv       wh1_strict_freq.csv
frequency.PLAYER                     wh1_strict_freq_jitter60.csv
hot_water_demand.glm                 wh1_strict_freq_jitter600.csv
local_gd
```

The gridlab-d binary file is stored within **local_gd** directory along with libraries. We need to re-configure the path for GridLAB-D such that the binary can locate the path of the library. If you are using macOS, please make sure you have installed the GNU version of sed, e.g.,

```
brew install gnu-sed
```

```
In [2]: %%bash
        oldpath='/tmp/temp'
        newpath=`pwd`'/local_gd'
        gsed -i "s#$oldpath#$newpath#" local_gd/bin/gridlabd
        gsed -i "s#$oldpath#$newpath#" local_gd/lib/gridlabd/glxengine.la
```

Or you can run **./correct_path.sh** script to correct the path.

```
In [3]: !local_gd/bin/gridlabd --version
```

```
GridLAB-D 4.0.0-17299 (feature/730:17299) 64-bit MACOSX RELEASE
```

The above listed **local_gd/bin/gridlabd** is the binary version of the gridlab-d software with controlling functionality. In addition to that, we have **.glm** files and generated **.csv** files. We also have a **frequency.PLAYER** containing the 1-second resolution frequency information.

The version of the gridlab-d binary file and the content of the frequency.PLAYER can be seen below.

If the version of the gridlab-d does not work, we can disable the comments below and run the command to compile the source and install the gridlab-d to the machine.

```
In [4]: # %%bash
        # cd ~
        # git clone -b feature/730 https://github.com/jingkungao/gridlab-d.git
        # cd gridlab-d
        # cd third_party
        # chmod +x install_xercesc
        # . install_xercesc
        # tar -xvf cppunit-1.12.0.tar.gz
        # cd cppunit-1.12.0
        # ./configure LDFLAGS="-ldl"
        # make
        # sudo make install
        # cd ../..
        # autoreconf -isf
        # ./configure
        # make
        # sudo make install
```

```
In [5]: !head -5 frequency.PLAYER
```

```
2012-01-01 00:00:00 EST,59.9769
2012-01-01 00:00:01 EST,59.9763
2012-01-01 00:00:02 EST,59.9715
2012-01-01 00:00:03 EST,59.9714
2012-01-01 00:00:04 EST,59.972
```

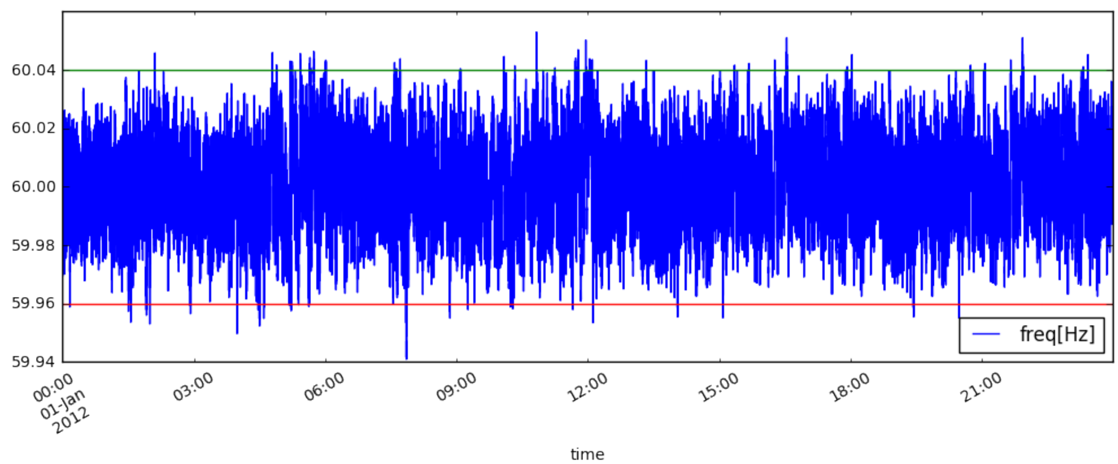
We can further plot the frequency data to get a better sense of it.

```
In [6]: # install necessary packages
        # uncomment the lines below if the system does not have them
        # !pip3 install numpy
        # !pip3 install pandas
        # !pip3 install plotly
```

```
In [7]: %matplotlib inline

        import numpy as np
```

```
import pandas as pd

from plotly.offline import download_plotlyjs, init_notebook_mode,\
                            plot, iplot
import plotly.graph_objs as go
init_notebook_mode(connected=True)

raw_freq = pd.read_csv('frequency.PLAYER',index_col=0,\
                       names=['time','freq[Hz]'],
                       parse_dates=True,\
                       infer_datetime_format=True)

freq_low = 59.96
freq_high = 60.04

ax = raw_freq.plot(figsize=(12,4),rot=30)
ax.axhline(y=freq_low, c='red')
ax.axhline(y=freq_high, c='green')
```

```
Out[7]: <matplotlib.lines.Line2D at 0x112c3f128>
```



Next, we will run **local_gd/bin/gridlabd** on different **.glm** files and plot the outputs showing the difference with and without controllers.

We start with running **smSingle_base.glm**, which is almost same as the original **smSingle.glm** provided by NRECA to us with the main difference being that we changed the simulation clock and added a recoreder for waterheater1 at the end.

1 Base case

We begin with the same circuit provided by NRECA (smSingle.glm), and modify it slightly as follows:

- We change the simulation time to match the time of **frequency.PLAYER** and add a recorder to record the waterheater measurements and the ZIP load measurements (in this case, a fan). Note that we record data for waterheater1 as an example but it could be used for any waterheater.
- We also set the timestep to 1 second instead of 60 seconds.
- For a more realistic water draw schedule, we include a **hot_water_demand.glm** which exhibits typical the weekday and weekend water demand usage patterns.

Below we illustrate some of those changes made to the glm file:

```
In [8]: # from 2012-01-01 to 2012-01-02
        !head -9 smSingle_base.glm
```

```
clock {
        timezone PST+8PDT;
        starttime '2012-01-01 00:00:00';
        stoptime '2012-01-02 00:00:00';
};

#include "hot_water_demand.glm";

#set minimum_timestep=1;
```

```
In [9]: # record data for waterheater1 and fan2(zipload) at 1s resolution
        !tail -14 smSingle_base.glm
```

```
object recorder {
        interval 1;
        property base_power;
        file fan2_base.csv;
        parent fan2;
};

object recorder {
        interval 1;
        property measured_frequency,temperature,actual_load,is_waterheater_on,water
                // current_tank_status,waterheater_model,heatgain,power_state;
        file wh1_base.csv;
        parent waterheater1;
};
```

We are now ready to run a simulation with the base case (no control).

```
In [10]: # run the gridlabd.bin to start the simulation
         !local_gd/bin/gridlabd smSingle_base.glm
```

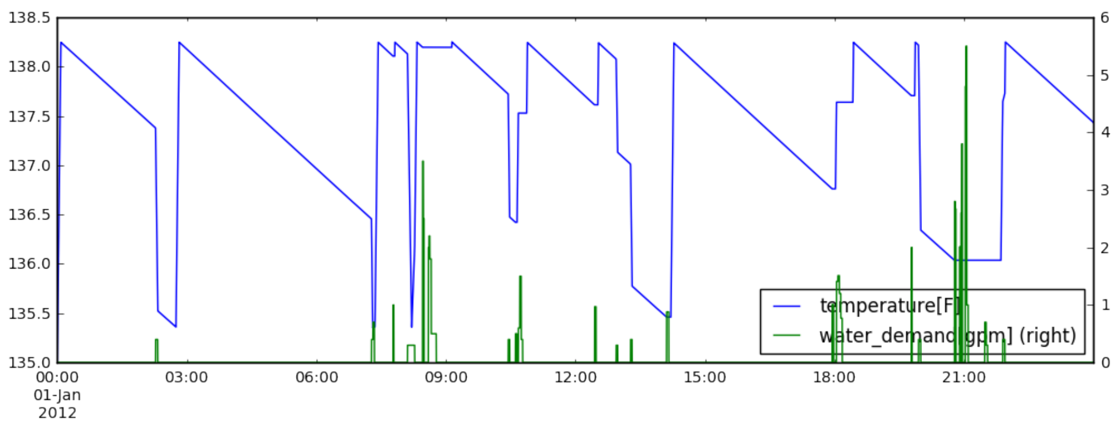
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, def
```

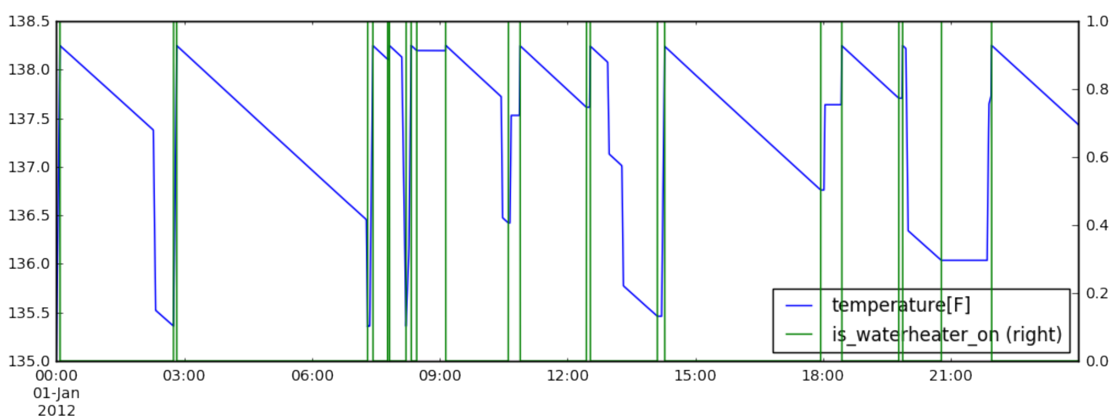
```
Core profiler results
======================

Total objects                 35 objects
Parallelism                    1 thread
Total time                  20.0 seconds
  Core time                  2.7 seconds (13.5%)
    Compiler                 1.0 seconds (4.8%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.2%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (0.9%)
  Model time                17.3 seconds/thread (86.5%)
Simulation time                1 days
Simulation speed              42 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate             4320 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               10.830    62.6%   5415.0
recorder            1.018     5.9%    339.3
triplex_meter       1.009     5.8%    336.3
house               0.828     4.8%    414.0
ZIPload             0.775     4.5%     96.9
waterheater         0.708     4.1%    354.0
transformer         0.600     3.5%    300.0
triplex_line        0.575     3.3%    287.5
regulator           0.359     2.1%    359.0
triplex_node        0.336     1.9%    336.0
auction             0.183     1.1%    183.0
climate             0.084     0.5%     84.0
================ ======== ======== ========
Total              17.305   100.0%    494.4

WARNING  [2012-01-02 00:00:00 PST] : last warning message was repeated 1 times
```

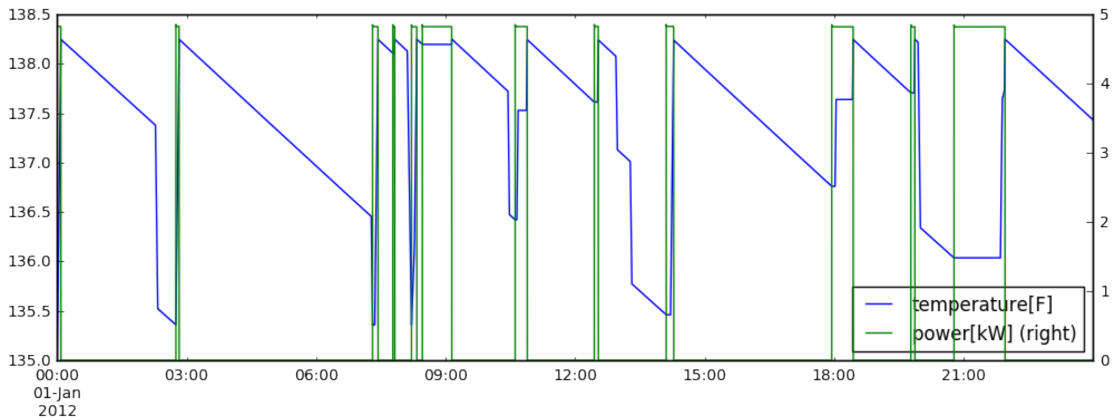
Now, we plot the generated waterheater data stored in **wh1_base.csv** and **fan2_base.csv** from the simulation.

```
In [11]: df_base = pd.read_csv('wh1_base.csv',sep=',',header=8,
                           index_col=0,parse_dates=True,infer_datetime_format=True,
                           names=['freq[Hz]','temperature[F]','power[kW]',\
                                  'is_waterheater_on','water_demand[gpm]'])

         df_base[['temperature[F]',
                  'water_demand[gpm]']].plot(figsize=(12,4),
                                              secondary_y='water_demand[gpm]')

         df_base[['temperature[F]',
                  'is_waterheater_on']].plot(figsize=(12,4),
                                              secondary_y='is_waterheater_on')

         df_base[['temperature[F]',
                  'power[kW]']].plot(figsize=(12,4),
                                      secondary_y='power[kW]')
```

```
Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x11c287048>
```





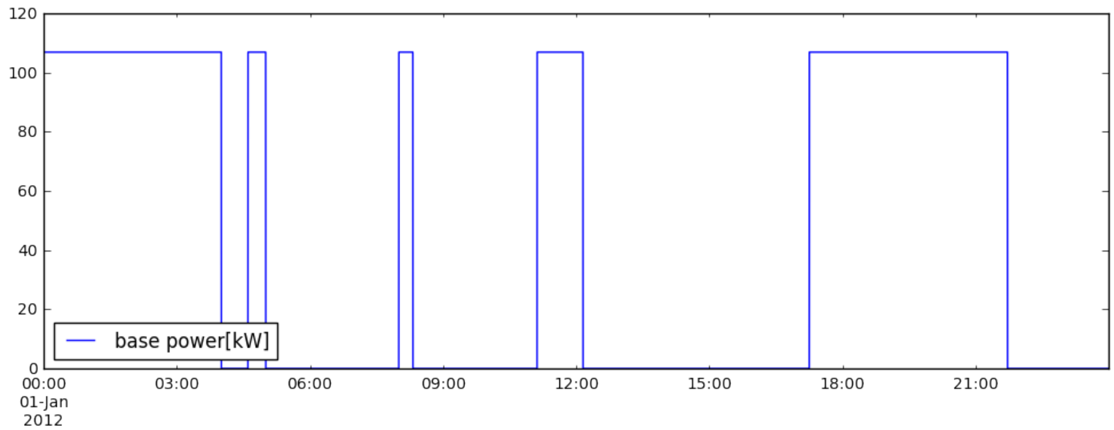



```
In [12]: # We can also plot the interactive version of the plot
         # during certain period
         def plotly_plotdf(df,title='Interactive plot of column variables'):
             if len(df)>20000:
                 print('Too many points, please reduce number of points!')
                 return
             data = []
             for i in df.columns:
                 trace = go.Scatter(
                     name = i,
                     x = df.index,
                     y = df[i]
                 )
                 data.append(trace)
             fig = go.Figure(
                 data = data,
                 layout = go.Layout(showlegend=True,
                                    title=title)
             )
             iplot(fig)
```

```
In [13]: # we can toggle the varialbe to visualize each of them
         # uncomment when you are running IPython notebook
         # plotly_plotdf(df_base.resample('1min').mean())
```

```
In [14]: df_base_fan = pd.read_csv('fan2_base.csv',sep=',',header=8,
                              index_col=0,parse_dates=True,infer_datetime_format=True,
                              names=['base power[kW]'])
         df_base_fan = df_base_fan*1000
         df_base_fan.plot(figsize=(12,4))
```

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x11e4aa358>



2 Lenient Frequency Control

To configure the GridBallast controller, we set specific properties of the waterheater object in the glm file. The properties corresponding to the controller include:

- enable_freq_control [boolean]
- freq_lowlimit [float]
- freq_uplimit [float]
- enable_jitter [boolean]
- average_delay_time [integer]

For this test we modify waterheater 1 and fan 2 to enable the frequency control and set a wide frequency dead-band (59.9Hz - 60.1Hz). We expect the GridBallast controller to be rarely triggered.

In [15]: !head -611 smSingle_lenient_freq.glm|tail -21

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
```

```
                file frequency.PLAYER;
                property measured_frequency;
    };
        enable_freq_control true;
        freq_lowlimit 59.9;
        freq_uplimit 60.1;
        heat_mode ELECTRIC;
};
```

In [16]:
```
!head -756 smSingle_lenient_freq.glm|tail -19
```

```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.9;
        freq_uplimit 60.1;
        groupid fan;
};
```

In [17]:
```
# run the gridlabd.bin to start the simulation
!local_gd/bin/gridlabd smSingle_lenient_freq.glm
```

```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, def

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  20.0 seconds
  Core time                  2.4 seconds (11.8%)
    Compiler                 0.9 seconds (4.3%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
```

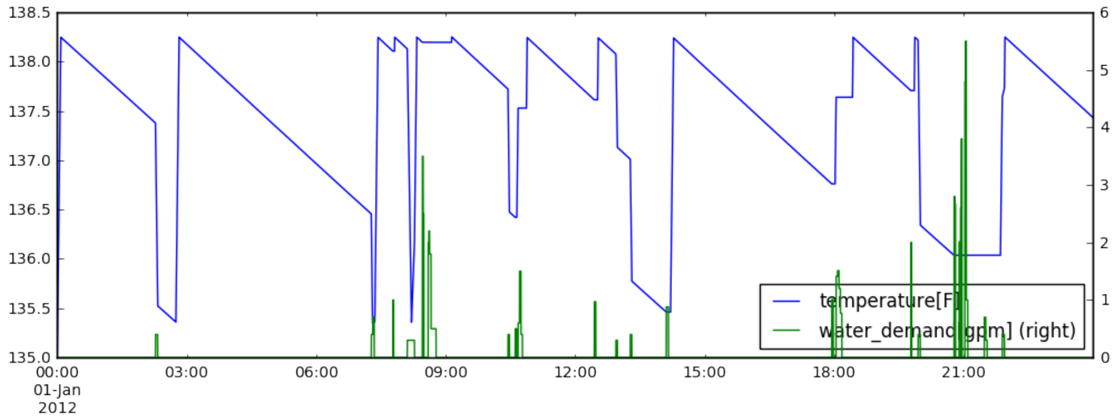
```
    Loadshapes                 0.0 seconds (0.2%)
    Enduses                    0.0 seconds (0.1%)
    Transforms                 0.2 seconds (0.8%)
  Model time                  17.6 seconds/thread (88.2%)
Simulation time                  1 days
Simulation speed                44 object.hours/second
Passes completed             86401 passes
Time steps completed         86401 timesteps
Convergence efficiency        1.00 passes/timestep
Read lock contention          0.0%
Write lock contention         0.0%
Average timestep                 1 seconds/timestep
Simulation rate               4320 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               10.623    60.3%   5311.5
triplex_meter       1.043     5.9%    347.7
recorder            0.950     5.4%    316.7
ZIPload             0.814     4.6%    101.8
house               0.775     4.4%    387.5
player              0.692     3.9%    346.0
waterheater         0.676     3.8%    338.0
transformer         0.598     3.4%    299.0
triplex_line        0.571     3.2%    285.5
regulator           0.354     2.0%    354.0
triplex_node        0.273     1.5%    273.0
auction             0.172     1.0%    172.0
climate             0.089     0.5%     89.0
================ ======== ======== ========
Total              17.630   100.0%    476.5

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

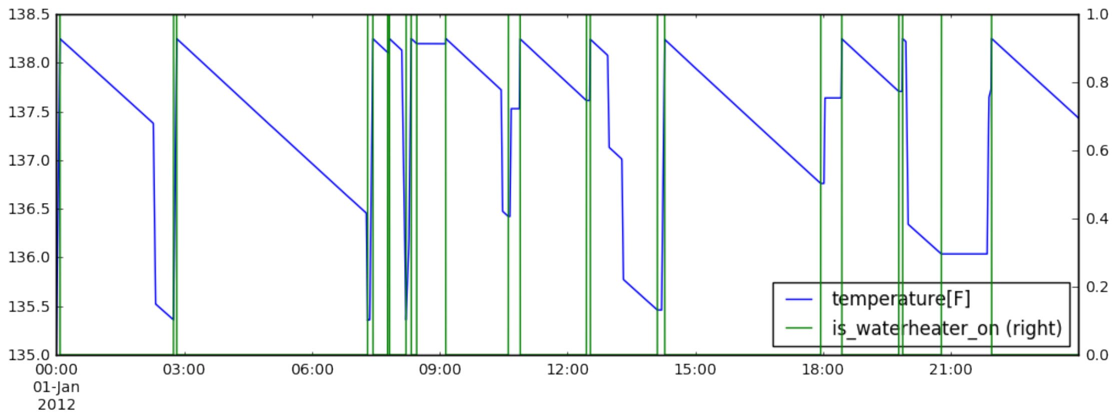
Now, we plot the generated waterheater data stored in **wh1_lenient_freq.csv** and **fan2_lenient_freq.csv** from the simulation.

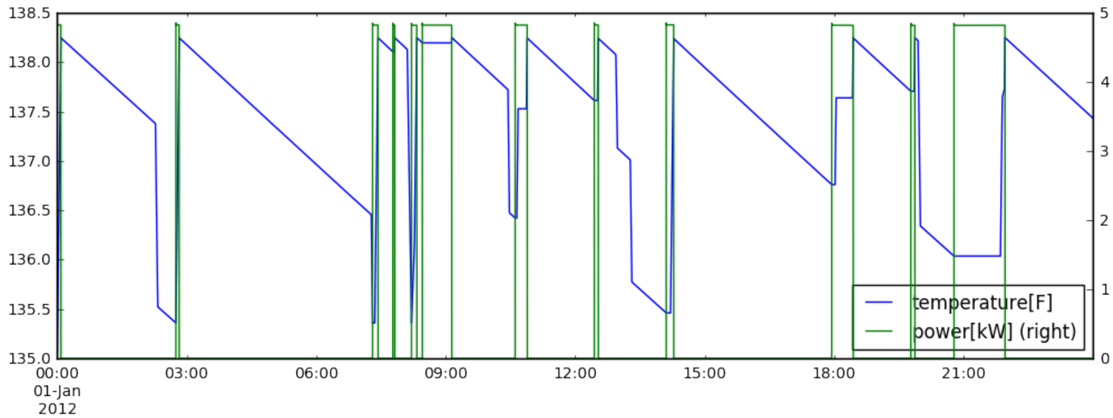
```
In [18]: # We save data to wh1_lenient_freq.csv and plot the results
         df_lenient_freq = pd.read_csv('wh1_lenient_freq.csv',sep=',',header=8,
                               index_col=0,parse_dates=True,infer_datetime_format=True,
                               names=['freq[Hz]','temperature[F]','power[kW]',
                                      'is_waterheater_on','water_demand[gpm]'])
```

```
df_lenient_freq[['temperature[F]',
                 'water_demand[gpm]']].plot(figsize=(12,4),
                                            secondary_y='water_demand[gpm]

df_lenient_freq[['temperature[F]',
                 'is_waterheater_on']].plot(figsize=(12,4),
                                            secondary_y='is_waterheat

df_lenient_freq[['temperature[F]',
                 'power[kW]']].plot(figsize=(12,4),
                                            secondary_y='power[kW]')
```

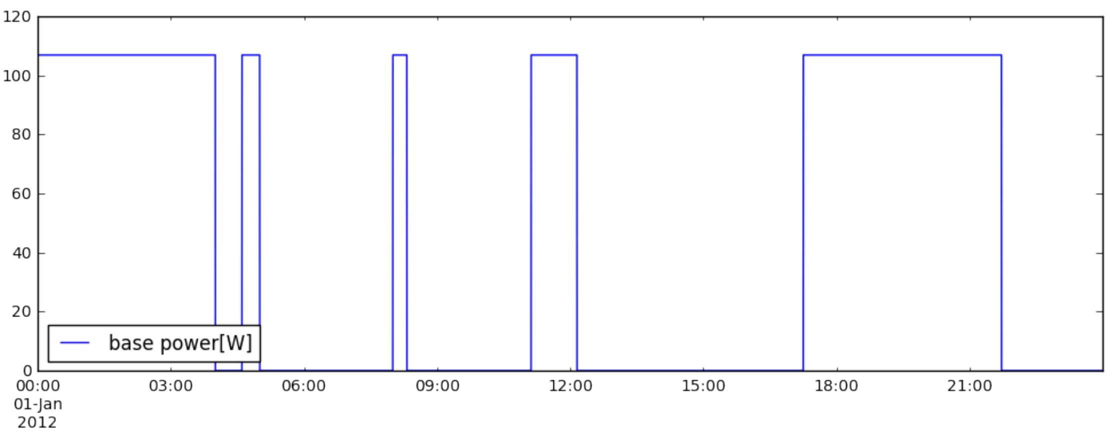
Out[18]: <matplotlib.axes._subplots.AxesSubplot at 0x1203f2908>








```
In [19]: df_lenient_fan = pd.read_csv('fan2_lenient_freq.csv',sep=',',header=8,
                                      index_col=0,parse_dates=True,infer_datetime_format=True,
                                      names=['base power[W]'])
         df_lenient_fan = df_lenient_fan*1000
         df_lenient_fan.plot(figsize=(12,4))

Out[19]: <matplotlib.axes._subplots.AxesSubplot at 0x11e4b4f98>
```



3 Strict Frequency Control

We modify waterheater 1 and fan 2 to enable the frequency control, but we impose a tighter frequency deadband (59.97Hz - 60.03Hz). In other words, the gridballast controller should be triggered very often.

```
In [20]: !head -611 smSingle_strict_freq.glm|tail -21
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        heat_mode ELECTRIC;
};
```

```
In [21]: !head -756 smSingle_strict_freq.glm|tail -19
```

```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        groupid fan;
};
```

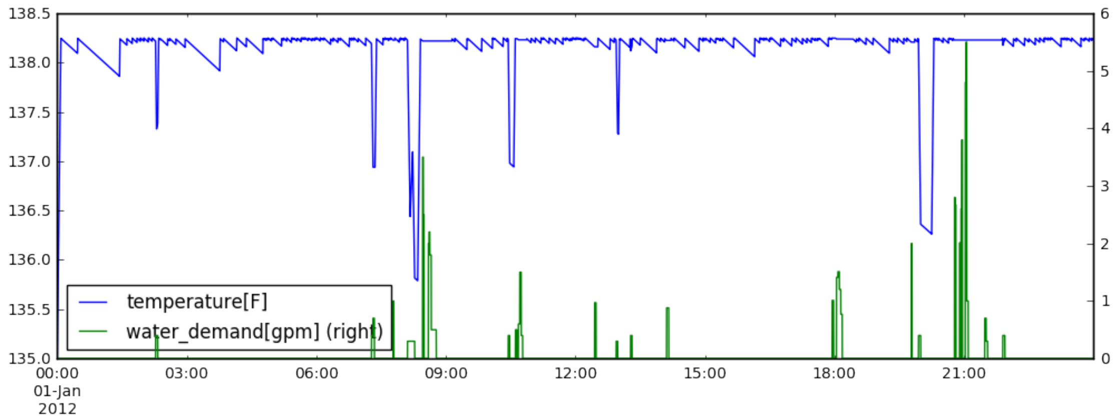
```
In [22]: # run the gridlabd.bin to start the simulation
         !local_gd/bin/gridlabd smSingle_strict_freq.glm
```

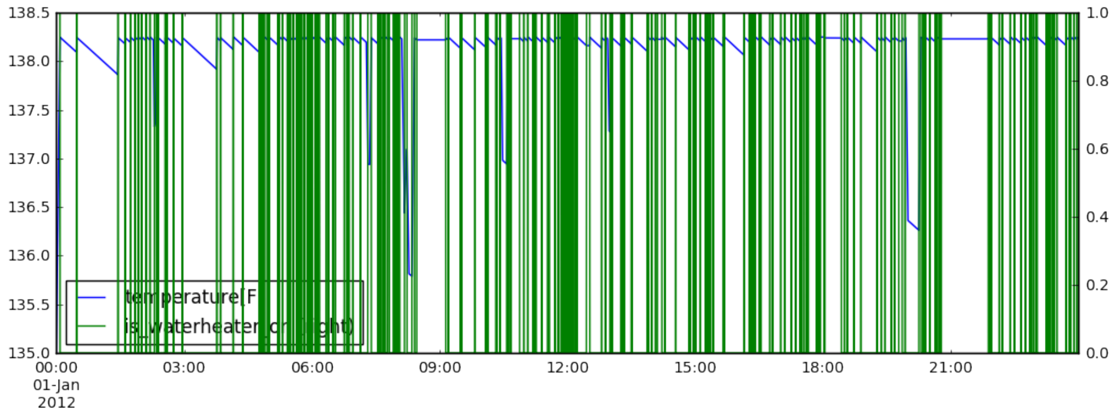
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, def

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  21.0 seconds
  Core time                  3.3 seconds (15.5%)
    Compiler                 1.0 seconds (4.9%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.2%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (0.9%)
  Model time                17.7 seconds/thread (84.5%)
Simulation time                1 days
Simulation speed              42 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate             4114 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node              10.695     60.3%   5347.5
triplex_meter      0.977      5.5%    325.7
recorder           0.938      5.3%    312.7
ZIPload            0.836      4.7%    104.5
house              0.804      4.5%    402.0
waterheater        0.687      3.9%    343.5
player             0.643      3.6%    321.5
transformer        0.622      3.5%    311.0
triplex_line       0.616      3.5%    308.0
regulator          0.384      2.2%    384.0
triplex_node       0.284      1.6%    284.0
auction            0.166      0.9%    166.0
climate            0.095      0.5%     95.0
================ ======== ======== ========
Total             17.747    100.0%    479.6
```

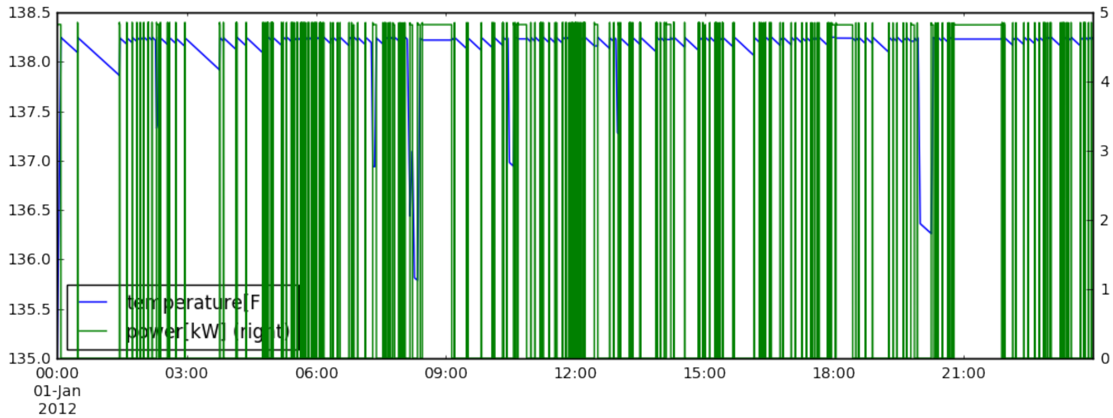
```
WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

Now, we plot the generated waterheater data stored in **wh1_strict_freq.csv** and **fan2_strict_freq.csv** from the simulation.

```
In [23]: # We save data to wh1_lenient_freq.csv and plot the results
         df_strict_freq = pd.read_csv('wh1_strict_freq.csv',sep=',',header=8,
                                  index_col=0,parse_dates=True,infer_datetime_format=True,
                                  names=['freq[Hz]','temperature[F]','power[kW]',
                                         'is_waterheater_on','water_demand[gpm]'])

         df_strict_freq[['temperature[F]',
                         'water_demand[gpm]']].plot(figsize=(12,4),
                                                    secondary_y='water_demar

         df_strict_freq[['temperature[F]',
                         'is_waterheater_on']].plot(figsize=(12,4),
                                                    secondary_y='is_waterheat

         df_strict_freq[['temperature[F]',
                         'power[kW]']].plot(figsize=(12,4),
                                                    secondary_y='power[kW]')

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x11ea424e0>
```

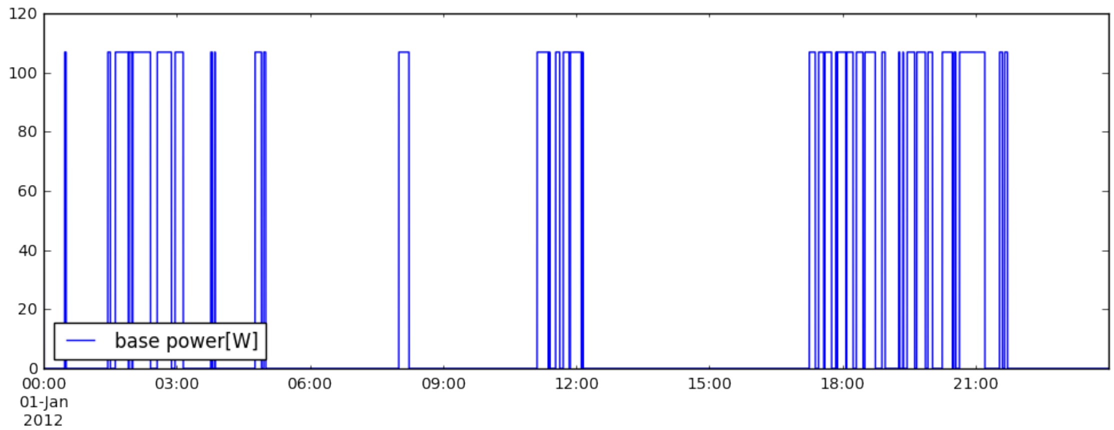




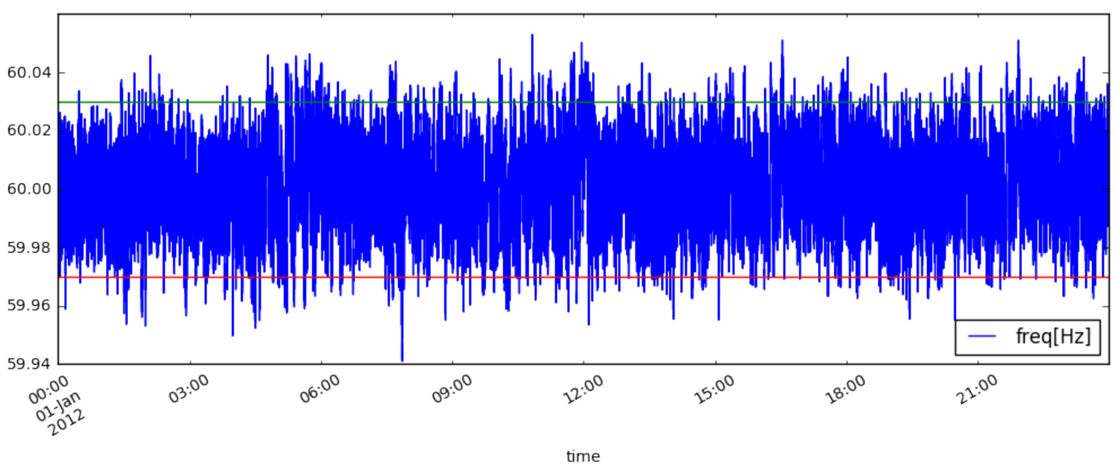


In [24]:
```
df_strict_fan = pd.read_csv('fan2_strict_freq.csv',sep=',',header=8,
                            index_col=0,parse_dates=True,infer_datetime_format=True,
                            names=['base power[W]'])
df_strict_fan = df_strict_fan*1000
df_strict_fan.plot(figsize=(12,4))
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1173c4630>




```
In [25]: # we can plot the frequency and the lower/upper limit again
         freq_low = 59.97
         freq_high = 60.03

         ax = raw_freq.plot(figsize=(12,4),rot=30)
         ax.axhline(y=freq_low, c='red')
         ax.axhline(y=freq_high, c='green')

Out[25]: <matplotlib.lines.Line2D at 0x1173e8898>
```



4 Strict Frequency Control with Jitter (1 min)

We now modify the previous case (with a tight frequency deadband) and add a jitter to the response of the waterheater and fan, such that the start of GridBallast event will delay randomly with an expected value of 60 seconds (1 min). Internally, the controller delay follows a uniform distribution over the interval [1,2*average_delay_time].

We use 60 seconds to clearly illustrate the difference in the power consumption patterns of the water heater previously illustrated and this one with jitter control enabled. Needless to say, users can set these values differently depending on how many water heaters are connected to the network or other considerations.

```
In [26]: !head -613 smSingle_strict_freq_jitter60.glm|tail -23
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        heat_mode ELECTRIC;
        enable_jitter true;
        average_delay_time 60;
};
```

```
In [27]: !head -760 smSingle_strict_freq_jitter60.glm|tail -21
```

```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
```

```
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        enable_jitter true;
        average_delay_time 60;
        groupid fan;
};
```

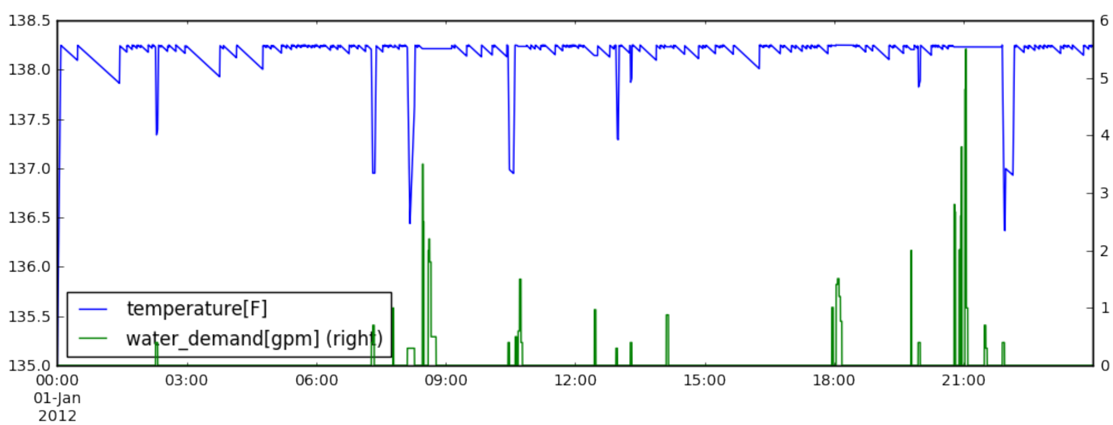
In [28]:
```
# run the gridlabd.bin to start the simulation
!local_gd/bin/gridlabd smSingle_strict_freq_jitter60.glm
```

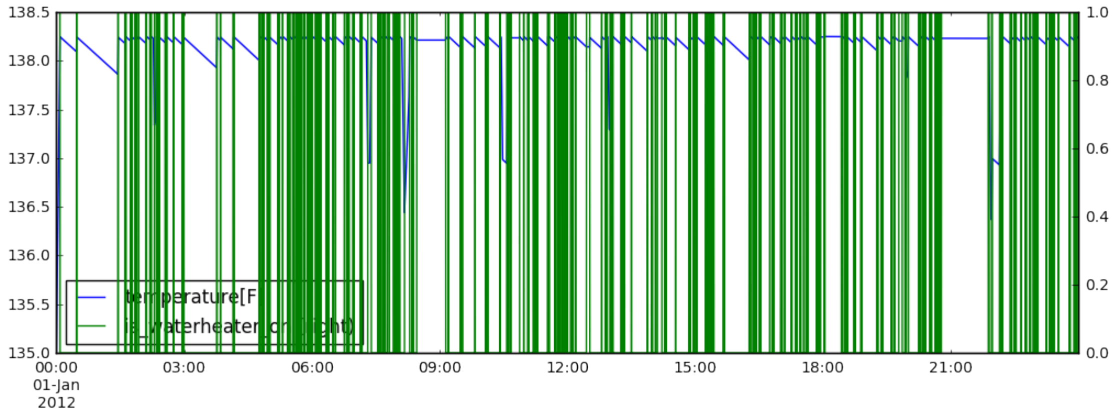
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, def

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  22.0 seconds
  Core time                  2.5 seconds (11.4%)
    Compiler                 1.0 seconds (4.4%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.2%)
    Enduses                  0.0 seconds (0.1%)
    Transforms               0.2 seconds (1.0%)
  Model time                19.5 seconds/thread (88.6%)
Simulation time                1 days
Simulation speed              40 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep               1 seconds/timestep
Simulation rate             3927 x realtime


Model profiler results
======================

Class            Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node               12.040    61.7%   6020.0
recorder            1.076     5.5%    358.7
```

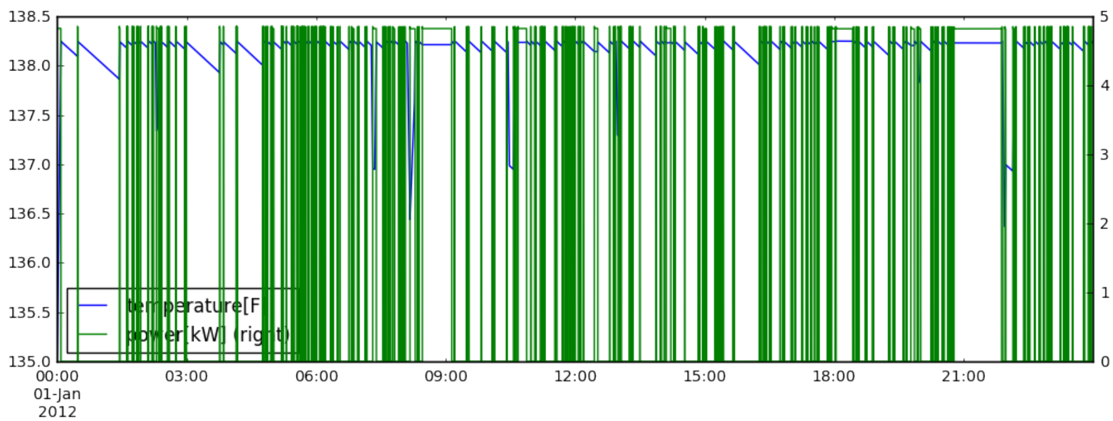
```
triplex_meter      1.049      5.4%    349.7
ZIPload            0.880      4.5%    110.0
house              0.865      4.4%    432.5
player             0.723      3.7%    361.5
waterheater        0.686      3.5%    343.0
transformer        0.603      3.1%    301.5
triplex_line       0.579      3.0%    289.5
regulator          0.374      1.9%    374.0
triplex_node       0.334      1.7%    334.0
auction            0.189      1.0%    189.0
climate            0.104      0.5%    104.0
================ ======== ======== ========
Total             19.502    100.0%    527.1

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

In [29]:
```
# We save data to wh1_strict_freq_jitter60.csv and plot the results
df_wh_jitter60 = pd.read_csv('wh1_strict_freq_jitter60.csv',sep=',',header
                  index_col=0,parse_dates=True,infer_datetime_format=True,
                  names=['freq[Hz]','temperature[F]','power[kW]',
                         'is_waterheater_on','water_demand[gpm]'])

df_wh_jitter60[['temperature[F]',
                'water_demand[gpm]']].plot(figsize=(12,4),
                                           secondary_y='water_demar

df_wh_jitter60[['temperature[F]',
                'is_waterheater_on']].plot(figsize=(12,4),
                                          secondary_y='is_waterheat

df_wh_jitter60[['temperature[F]',
                'power[kW]']].plot(figsize=(12,4),
                              secondary_y='power[kW]')
```

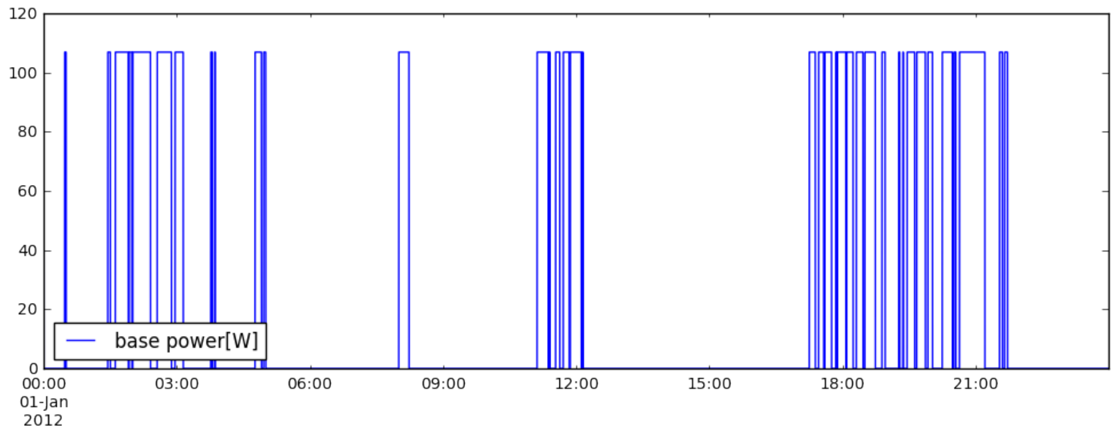
Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x124cf8e48>







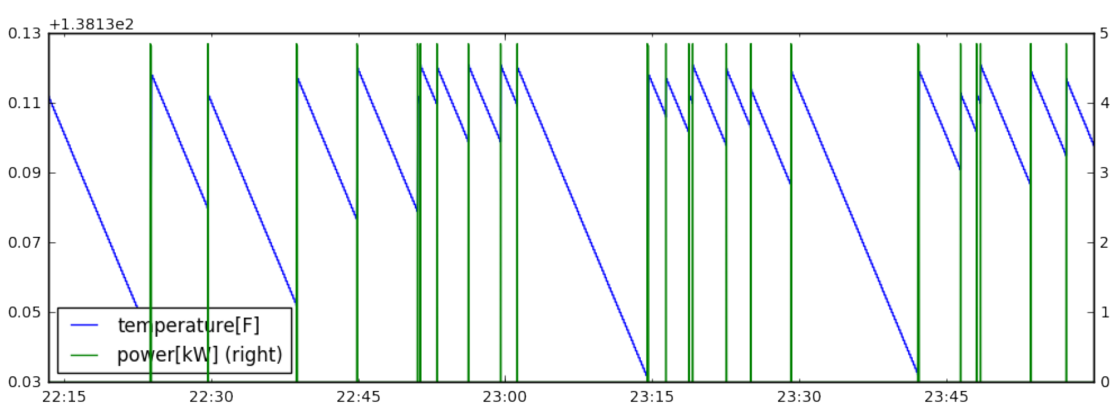

```
In [30]: df_fan_jitter60 = pd.read_csv('fan2_strict_freq_jitter60.csv',sep=',',
                                        header=8,index_col=0,parse_dates=True,
                                        infer_datetime_format=True,
                              names=['base power[W]'])
         df_fan_jitter60 = df_fan_jitter60*1000
         df_fan_jitter60.plot(figsize=(12,4))

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x12a3f3c18>
```

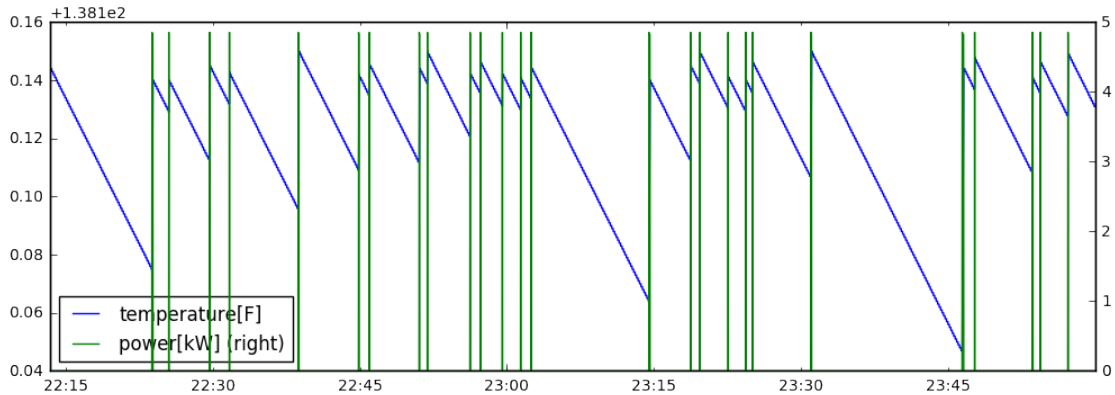


As we can see, after applying the jitter, the water heater should be engaged less often. However, since the jitter time is too short, we can barely see the difference unless we zoom in. Let's try the jitter with longer duration.

```
In [31]: # we look at jitter in shorter duration
         # As we can see, they behave slightly different
         df_strict_freq.iloc[80000:100000][['temperature[F]',
                                            'power[kW]']].plot(figsize=(12,4),
                                                                secondary_y='power[kW]')
         df_wh_jitter60.iloc[80000:100000][['temperature[F]',
                                            'power[kW]']].plot(figsize=(12,4),
                                                                secondary_y='power[kW]')
```

```
Out[31]: <matplotlib.axes._subplots.AxesSubplot at 0x12c30b2b0>
```





5 Strict Frequency Control with Jitter (10 mins)

We now modify the jitter such that the start of GridBallast event will delay randomly with an expected value of 600 seconds (10 mins) so that we can clearly see the jitter effects.

```
In [32]: !head -613 smSingle_strict_freq_jitter600.glm|tail -23
```

```
object waterheater {
        schedule_skew -810;
        water_demand weekday_hotwater*1;
        name waterheater1;
        parent house1;
        heating_element_capacity 4.8 kW;
        thermostat_deadband 2.9;
        location INSIDE;
        tank_volume 50;
        tank_setpoint 136.8;
        tank_UA 2.4;
        temperature 135;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        heat_mode ELECTRIC;
        enable_jitter true;
        average_delay_time 600;
};
```

```
In [33]: !head -760 smSingle_strict_freq_jitter600.glm|tail -21
```

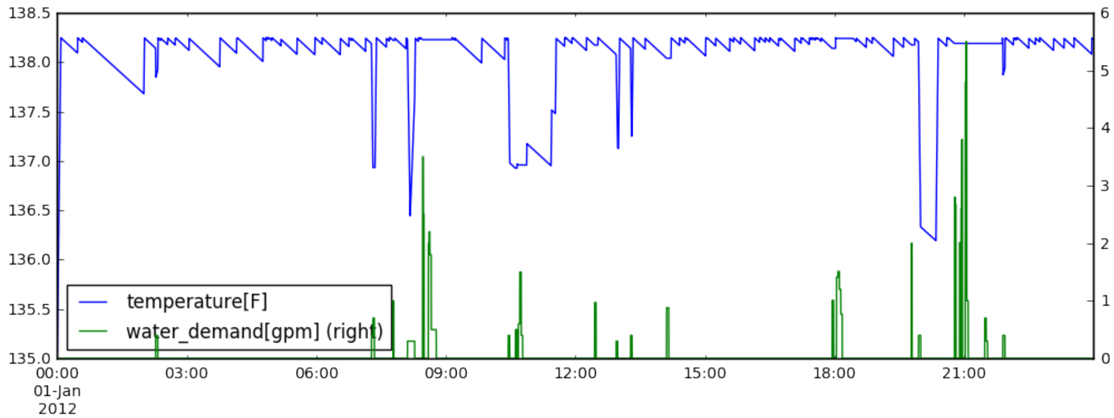
```
object ZIPload {
        name fan2;
        parent house2;
        power_fraction 0.013500;
        current_fraction 0.253400;
        base_power fan1*0.106899;
        impedance_pf 0.970000;
        current_pf 0.950000;
        power_pf -1.000000;
        impedance_fraction 0.733200;
        object player {
                file frequency.PLAYER;
                property measured_frequency;
        };
        enable_freq_control true;
        freq_lowlimit 59.97;
        freq_uplimit 60.03;
        enable_jitter true;
        average_delay_time 600;
        groupid fan;
};
```

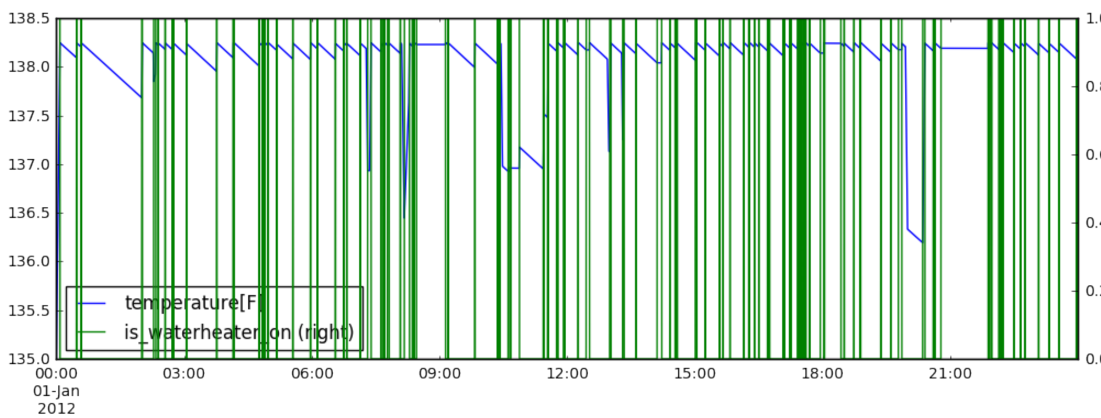
```
In [34]: # run the gridlabd.bin to start the simulation
         !local_gd/bin/gridlabd smSingle_strict_freq_jitter600.glm
```

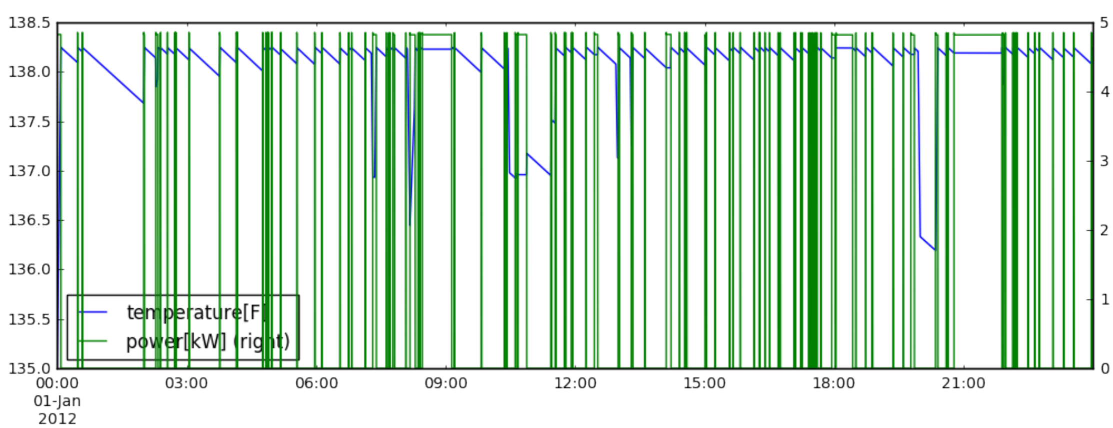
```
WARNING  [INIT] : waterheater::init() : height and diameter were not specified, def

Core profiler results
======================

Total objects                 37 objects
Parallelism                    1 thread
Total time                  19.0 seconds
  Core time                  2.5 seconds (13.4%)
    Compiler                 0.9 seconds (4.8%)
    Instances                0.0 seconds (0.0%)
    Random variables         0.0 seconds (0.0%)
    Schedules                0.0 seconds (0.0%)
    Loadshapes               0.0 seconds (0.1%)
    Enduses                  0.0 seconds (0.2%)
    Transforms               0.1 seconds (0.7%)
  Model time                16.5 seconds/thread (86.6%)
Simulation time                1 days
Simulation speed              47 object.hours/second
Passes completed           86401 passes
Time steps completed       86401 timesteps
Convergence efficiency      1.00 passes/timestep
```

```
Read lock contention        0.0%
Write lock contention       0.0%
Average timestep              1 seconds/timestep
Simulation rate            4547 x realtime


Model profiler results
======================

Class             Time (s) Time (%) msec/obj
---------------- -------- -------- --------
node                9.908    60.2%   4954.0
recorder            0.929     5.6%    309.7
triplex_meter       0.919     5.6%    306.3
house               0.777     4.7%    388.5
ZIPload             0.768     4.7%     96.0
waterheater         0.604     3.7%    302.0
player              0.595     3.6%    297.5
transformer         0.564     3.4%    282.0
triplex_line        0.531     3.2%    265.5
regulator           0.292     1.8%    292.0
triplex_node        0.282     1.7%    282.0
auction             0.203     1.2%    203.0
climate             0.087     0.5%     87.0
================ ======== ======== ========
Total              16.459   100.0%    444.8

WARNING  [2012-01-02 00:00:00 EST] : last warning message was repeated 1 times
```

```
In [35]: # We save data to wh1_strict_freq_jitter600.csv and plot the results
         df_wh_jitter600 = pd.read_csv('wh1_strict_freq_jitter600.csv',sep=',',heac
                          index_col=0,parse_dates=True,infer_datetime_format=True,
                          names=['freq[Hz]','temperature[F]','power[kW]',
                                 'is_waterheater_on','water_demand[gpm]'])

         df_wh_jitter600[['temperature[F]',
                          'water_demand[gpm]']].plot(figsize=(12,4),
                                                     secondary_y='water_demar

         df_wh_jitter600[['temperature[F]',
                          'is_waterheater_on']].plot(figsize=(12,4),
                                                    secondary_y='is_waterheat

         df_wh_jitter600[['temperature[F]',
                          'power[kW]']].plot(figsize=(12,4),
                                                    secondary_y='power[kW]')

Out[35]: <matplotlib.axes._subplots.AxesSubplot at 0x123c59748>
```

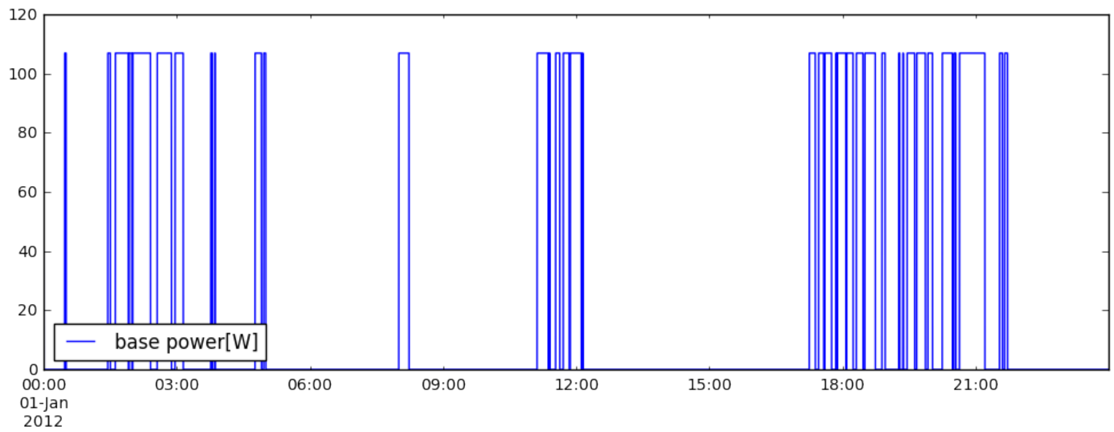

temperature[F]
water_demand[gpm] (right)
138.5
138.0
137.5
137.0
136.5
136.0
135.5
135.0
6
5
4
3
2
1
0
00:00
01-Jan
2012
03:00
06:00
09:00
12:00
15:00
18:00
21:00


temperature[F]
is_waterheater_on (right)
138.5
138.0
137.5
137.0
136.5
136.0
135.5
135.0
1.0
0.8
0.6
0.4
0.2
0.0
00:00
01-Jan
2012
03:00
06:00
09:00
12:00
15:00
18:00
21:00


temperature[F]
power[kW] (right)
138.5
138.0
137.5
137.0
136.5
136.0
135.5
135.0
5
4
3
2
1
0
00:00
01-Jan
2012
03:00
06:00
09:00
12:00
15:00
18:00
21:00

```
In [36]: df_fan_jitter600 = pd.read_csv('fan2_strict_freq_jitter600.csv',sep=',',
                                         header=8,index_col=0,parse_dates=True,
                                         infer_datetime_format=True,
                           names=['base power[W]'])
         df_fan_jitter600 = df_fan_jitter600*1000
         df_fan_jitter600.plot(figsize=(12,4))
```

```
Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x12138ebe0>
```

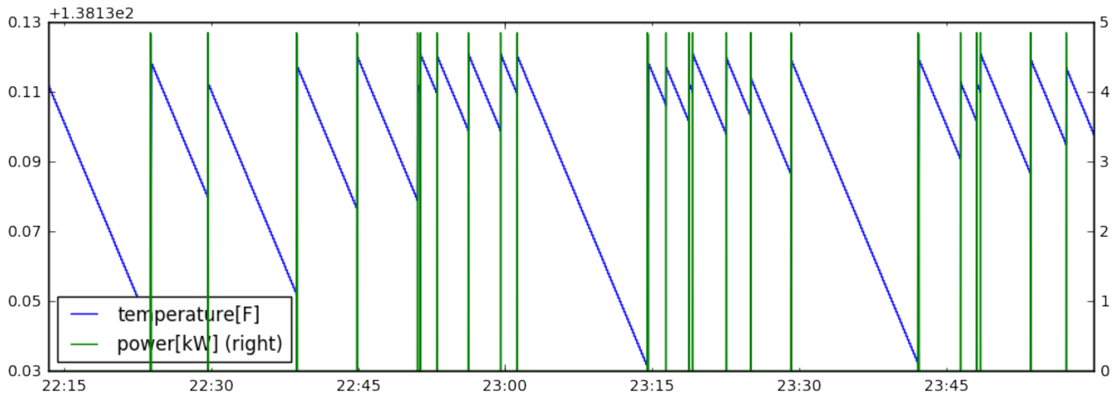


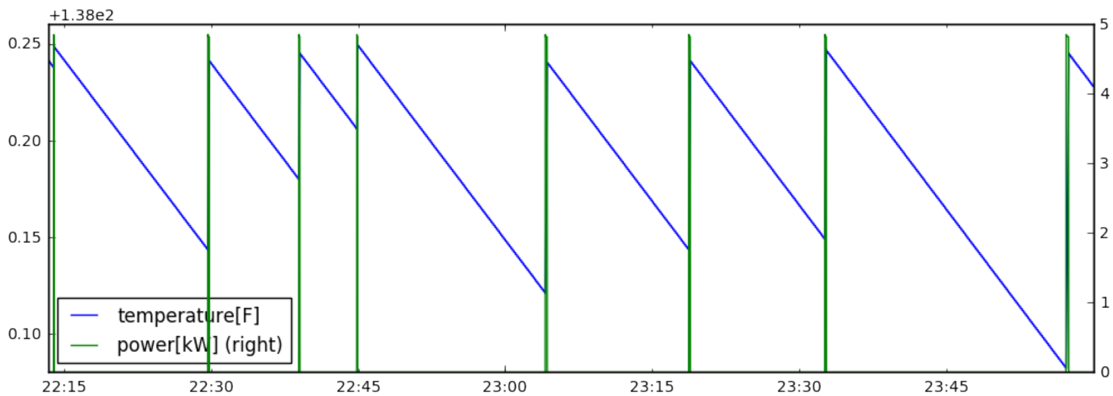
As we can see, after applying the 10 min jitter, now the water heater is engaged less often than in the previous experiment without jitter.

As we did in previous examples, we now look into a shorter duration to better understand the effect of the jitter.

```
In [37]: # we look at jitter in shorter duration
         # As we can see, they behave quite differently

         df_strict_freq.iloc[80000:100000][['temperature[F]',
                                           'power[kW]']].plot(figsize=(12,4),
                                                              secondary_y='power[kW]')
         df_wh_jitter600.iloc[80000:100000][['temperature[F]',
                                            'power[kW]']].plot(figsize=(12,4),
                                                              secondary_y='power[kW]')
```

```
Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x12c9e3240>
```


+1.3813e2
0.13
0.11
0.09
0.07
0.05
0.03
5
4
3
2
1
0
22:15
22:30
22:45
23:00
23:15
23:30
23:45
temperature[F]
power[kW] (right)


+1.38e2
0.25
0.20
0.15
0.10
5
4
3
2
1
0
22:15
22:30
22:45
23:00
23:15
23:30
23:45
temperature[F]
power[kW] (right)