Step 7: Serve data from OpenAgua into WEAP using WaMDaM

By Adel M. Abdallah, Jan 2022

Execute the following cells by pressing `Shift-Enter`, or by pressing the play button on the toolbar above.

Steps

1. Import python libraries
2. Import the published SQLite file for the WEAP model from HydroShare.
3. Prepare to connect to the WEAP API
4. Connect to WEAP API to programmatically populate WEAP with data, run it, get back results Create a copy of the original WEAP Area to use while keeping the original as-as for any later use 5.3 Export the unmet demand percent into Excel to load them into WaMDaM

1. Import python libraries

```
# 1. Import python libraries
### set the notebook mode to embed the figures within the cell
import numpy
import sqlite3
import numpy as np
import pandas as pd
import getpass
from hs_restclient import HydroShare, HydroShareAuthBasic
import os

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

init_notebook_mode(connected=True)         # initiate notebook for offline plot

import os
import csv
from collections import OrderedDict
import sqlite3
import pandas as pd
import numpy as np
from IPython.display import display, Image, SVG, Math, YouTubeVideo
import urllib
import calendar

print 'The needed Python libraries have been imported'
```

2. Connect to the WaMDaM SQLite on HydroSahre

Provide the HydroShare ID for your resource

Example
https://www.hydroshare.org/resource/af71ef99a95e47a89101983f5ec6ad8b/

resource_id='85e9fe85b08244198995558fe7d0e294'

```
# enter your HydroShare username and password here between the quotes
username = ''
password = ''

auth = HydroShareAuthBasic(username=username, password=password)

hs = HydroShare(auth=auth)

print 'Connected to HydroShare'

# Then we can run queries against it within this notebook :)
resource_url='https://www.hydroshare.org/resource/af71ef99a95e47a89101983f5ec6ad8b/'


resource_id= resource_url.split("https://www.hydroshare.org/resource/",1)[1]
resource_id=resource_id.replace('/','')

print resource_id

resource_md = hs.getSystemMetadata(resource_id)
# print resource_md
print 'Resource title'
print(resource_md['resource_title'])
print '----------------------------'

resources=hs.resource(resource_id).files.all()

file = ""

for f in hs.resource(resource_id).files.all():

    file += f.decode('utf8')

import json

file_json = json.loads(file)

for f in file_json["results"]:

    FileURL= f["url"]

    SQLiteFileName=FileURL.split("contents/",1)[1]

cwd = os.getcwd()
print cwd
fpath = hs.getResourceFile(resource_id, SQLiteFileName, destination=cwd)
conn = sqlite3.connect(SQLiteFileName,timeout=10)

print 'Connected to the SQLite file= '+ SQLiteFileName
print 'done'
```

2. Prepare to the Connect to the WEAP API

You need to have WEAP already installed on your machine

First make sure to have a copy of the Water Evaluation And Planning" system (WEAP) installed on your local machine (Windows). If you don't have it installed, download and install the WEAP software which allows you to run the Bear River WEAP model and its scenarios for Use Case 5. https://www.weap21.org/. You need to have a WEAP License. See here (https://www.weap21.org/index.asp?action=217). If you're interested to learning about WEAP API, check it out here: http://www.weap21.org/WebHelp/API.htm

Install dependency and register WEAP

2.1. Install pywin32 extensions which provide access to many of the Windows APIs from Python.

Choose on option

- a. Install using an executable based on your python version. Use version for Python 2.7 https://github.com/mhammond/pywin32/releases

OR

- b. Install it using Anaconda terminal @ https://anaconda.org/anaconda/pywin32

Type this command in the Anaconda terminal as Administrator

```
conda install -c anaconda pywin32
```

OR

- c. Install from source code (for advanced users) https://github.com/mhammond/pywin32

2.2. Register WEAP with Windows

This use case only works on a local Jupyter Notebook server installed on your machine along with WEAP. So it does not work on the online Notebooks in Step 2.1. You need to install Jupyter Server in Step 2.2 then proceed here.

- **Register WEAP with Windows to allow the WEAP API to be accessed**
  Use Windows "Command Prompt". Right click and then **run as Administrator**, navigate to the WEAP installation directory such as and then hit enter

```
cd C:\Program Files (x86)\WEAP
```

Then type the following command in the command prompt and hit enter

```
WEAP /regserver
```

Figure 1: Register WEAP API with windows using the Command Prompt (Run as Administrator)

3. Connect Jupyter Notebook to WEAP API

Clone or download all this GitHub repo
https://github.com/WamdamProject/WaMDaM_UseCases

In your local repo folder, go to the

```
C:\Users\Adel\Documents\GitHub\WaMDaM_UseCases/UseCases_files/1Original_Datasets_preperation_files/WEAP/Bear_Ri
```

Copy this folder **Bear_River_WEAP_Model_2017** and paste it into **WEAP Areas** folder on your local machine. For example, it is at

```
C:\Users\Adel\Documents\WEAP Areas
```

```
# this library is needed to connect to the WEAP API
import win32com.client

# this command will open the WEAP software (if closed) and get the last active model
# you could change the active area to another one inside WEAP or by passing it to the command here
#WEAP.ActiveArea = "BearRiverFeb2017_V10.9"


WEAP=win32com.client.Dispatch("WEAP.WEAPApplication")

# WEAP.Visible = 'FALSE'


print WEAP.ActiveArea.Name
WEAP.ActiveArea = "Bear_River_WEAP_Model_2017_Original"
print WEAP.ActiveArea.Name

WEAP.Areas("Bear_River_WEAP_Model_2017_Original").Open
WEAP.ActiveArea = "Bear_River_WEAP_Model_2017_Original"
print WEAP.ActiveArea.Name


print 'Connected to WEAP API and the '+ WEAP.ActiveArea.Name + ' Area'
print '-------------'
if not WEAP.Registered:
    print "Because WEAP is not registered, you cannot use the API"

# get the active WEAP Area (model) to serve data into it

# ActiveArea=WEAP.ActiveArea.Name


# get the active WEAP scenario to serve data into it
print '-------------'

ActiveScenario= WEAP.ActiveScenario.Name
print '\n ActiveScenario= '+ActiveScenario
print '-------------'

WEAP_Area_dir=WEAP.AreasDirectory
print WEAP_Area_dir


print "\n \n You're connected to the WEAP API"
```

4 Create a copy of the original WEAP Area to use while keeping the orignial as-as for any later use

Add a new CacheCountyUrbanWaterUse scenario from the Reference original WEAP Area:

You can always use this orignal one and delete any new copies you make afterwards.

```
# Create a copy of the WEAP AREA to serve the updated Hyrym Reservoir to it


# Delete the Area if it exists and then add it. Start from fresh
Area="Bear_River_WEAP_Model_2017_Conservation"

if not WEAP.Areas.Exists(Area):
    WEAP.SaveAreaAs(Area)


WEAP.ActiveArea.Save
WEAP.ActiveArea = "Bear_River_WEAP_Model_2017_Conservation"
print  'ActiveArea= '+WEAP.ActiveArea.Name

#  Add new Scenario
#  Add(NewScenarioName, ParentScenarioName or Index):
#  Create a new scenario as a child of the parent scenario specified.
#  The new scenario will become the selected scenario in the Data View.



WEAP=win32com.client.Dispatch("WEAP.WEAPApplication")
# WEAP.Visible = FALSE


WEAP.ActiveArea = "Bear_River_WEAP_Model_2017_Conservation"

print  'ActiveArea= '+ WEAP.ActiveArea.Name

Scenarios=[]
Scenarios=['Cons25PercCacheUrbWaterUse','Incr25PercCacheUrbWaterUse']

# Delete the scenario if it exists and then add it. Start from fresh
for Scenario in Scenarios:
    if WEAP.Scenarios.Exists(Scenario):
        # delete it
        WEAP.Scenarios(Scenario).Delete(True)
        # add it back as a fresh copy
        WEAP.Scenarios.Add(Scenario,'Reference')
    else:
        WEAP.Scenarios.Add(Scenario,'Reference')

WEAP.ActiveArea.Save
WEAP.SaveArea

WEAP.Quit

# or add the scenarios one by one using this command

# Make a copy from the reference (base) scenario
# WEAP.Scenarios.Add('UpdateCacheDemand','Reference')
print '---------------------- \n'
print 'Scenarios added to the original WEAP area'


WEAP.Quit

print 'Connection with WEAP API is disconnected'
```

4.A Query Cache County seasonal "Monthly Demand" for the three sites: Logan Potable, North Cache Potable, South Cache Potable

The data comes from OpenAgua

```
# Use Case 3.1Identify_aggregate_TimeSeriesValues.csv
# plot aggregated to monthly and converted to acre-feet time series data of multiple sources


# Logan Potable
# North Cache Potable
# South Cache Potable

# 2.2Identify_aggregate_TimeSeriesValues.csv
Query_UseCase_URL="""
https://raw.githubusercontent.com/WamdamProject/WaMDaM_JupyterNotebooks/master/3_VisualizePublish/SQL_queries/
"""

# Read the query text inside the URL
Query_UseCase_text = urllib.urlopen(Query_UseCase_URL).read()


# return query result in a pandas data frame
result_df_UseCase= pd.read_sql_query(Query_UseCase_text, conn)

# uncomment the below line to see the list of attributes
# display (result_df_UseCase)
seasons_dict = dict()
seasons_dict2=dict()
Scenarios=['Cons25PercCacheUrbWaterUse','Incr25PercCacheUrbWaterUse']

subsets = result_df_UseCase.groupby(['ScenarioName','InstanceName'])
for subset in subsets.groups.keys():
    if subset[0] in Scenarios:
        df_Seasonal = subsets.get_group(name=subset)
        df_Seasonal=df_Seasonal.reset_index()

        SeasonalParam = ''
        for i in range(len(df_Seasonal['SeasonName'])):
            m_data = df_Seasonal['SeasonName'][i]
            n_data = float(df_Seasonal['SeasonNumericValue'][i])
            SeasonalParam += '{},{}'.format(m_data, n_data)
            if i != len(df_Seasonal['SeasonName']) - 1:
                SeasonalParam += ','

        Seasonal_value="MonthlyValues("+SeasonalParam+")"
        seasons_dict[subset]=(Seasonal_value)
# seasons_dict2[subset[0]]=seasons_dict
# print seasons_dict2

print '-----------------'
# print seasons_dict


# seasons_dict2.get("Cons25PercCacheUrbWaterUse", {}).get("Logan Potable") # 1

print 'Query and data preperation are done'
```

4.B Load the seasonal demand data with conservation into WEAP

```
# 9. Load the seasonal data into WEAP
#WEAP=win32com.client.Dispatch("WEAP.WEAPApplication")
# WEAP.Visible = FALSE


print WEAP.ActiveArea.Name
Scenarios=['Cons25PercCacheUrbWaterUse','Incr25PercCacheUrbWaterUse']
DemandSites=['Logan Potable','North Cache Potable','South Cache Potable']

AttributeName='Monthly Demand'

for scenario in Scenarios:
    WEAP.ActiveScenario = scenario
    print WEAP.ActiveScenario.Name

    for Branch in WEAP.Branches:
        for InstanceName in DemandSites:
            if Branch.Name == InstanceName:
                GetInstanceFullBranch = Branch.FullName
                val=seasons_dict[(scenario,InstanceName)]
                WEAP.Branch(GetInstanceFullBranch).Variable(AttributeName).Expression =val
#                 print val
                print "loaded " + InstanceName
                WEAP.SaveArea

print '\n The data have been sucsesfully loaded into WEAP'

WEAP.SaveArea

print '\n \n The updated data have been saved'
```

5. Run WEAP

Please wait, it will take ~1-3 minutes to finish calcualting the two WEAP Areas with their many scenarios

```
# Run WEAP

WEAP.Areas("Bear_River_WEAP_Model_2017_Conservation").Open
print WEAP.ActiveArea.Name

WEAP.ActiveArea = "Bear_River_WEAP_Model_2017_Conservation"
print WEAP.ActiveArea.Name

print 'Please wait 1-3 min for the calculation to finish'
WEAP.Calculate(2006,10,True)
WEAP.SaveArea

print '\n \n The calculation has been done and saved'
print WEAP.CalculationTime

print '\n \n Done'
```

5.1 Get the unmet demand or Cache County sites in both the reference and the conservation scenarios

```
Scenarios=['Reference','Cons25PercCacheUrbWaterUse','Incr25PercCacheUrbWaterUse']
DemandSites=['Logan Potable','North Cache Potable','South Cache Potable']

UnmetDemandEstimate_Ref = pd.DataFrame(columns = DemandSites)
UnmetDemandEstimate_Cons25 = pd.DataFrame(columns = DemandSites)
UnmetDemandEstimate_Incr25 = pd.DataFrame(columns = DemandSites)

UnmetDemandEstimate= pd.DataFrame(columns = Scenarios)

for scen in Scenarios:
        if scen=='Reference':
            for site in DemandSites:
                param="\Demand Sites\%s: Unmet Demand[Acre-Foot]"%(site)
#                 print param
                for year in range (1966,2006):
                    value=WEAP.ResultValue(param, year, 1, scen, year, WEAP.NumTimeSteps)
                    UnmetDemandEstimate_Ref.loc[year, [site]]=value
        elif scen=='Cons25PercCacheUrbWaterUse':
            for site in DemandSites:
                param="\Demand Sites\%s: Unmet Demand[Acre-Foot]"%(site)
#                 print param
                for year in range (1966,2006):
                    value=WEAP.ResultValue(param, year, 1, scen, year, WEAP.NumTimeSteps)
                    UnmetDemandEstimate_Cons25.loc[year, [site]]=value

        elif scen=='Incr25PercCacheUrbWaterUse':
            for site in DemandSites:
                param="\Demand Sites\%s: Unmet Demand[Acre-Foot]"%(site)
#                 print param
                for year in range (1966,2006):
                    value=WEAP.ResultValue(param, year, 1, scen, year, WEAP.NumTimeSteps)
                    UnmetDemandEstimate_Incr25.loc[year, [site]]=value


UnmetDemandEstimate_Ref['Cache Total']=UnmetDemandEstimate_Ref[DemandSites].sum(axis=1)

UnmetDemandEstimate_Cons25['Cache Total']=UnmetDemandEstimate_Cons25[DemandSites].sum(axis=1)

UnmetDemandEstimate_Incr25['Cache Total']=UnmetDemandEstimate_Incr25[DemandSites].sum(axis=1)

UnmetDemandEstimate['Reference']=UnmetDemandEstimate_Ref['Cache Total']
UnmetDemandEstimate['Cons25PercCacheUrbWaterUse']=UnmetDemandEstimate_Cons25['Cache Total']
UnmetDemandEstimate['Incr25PercCacheUrbWaterUse']=UnmetDemandEstimate_Incr25['Cache Total']

UnmetDemandEstimate=UnmetDemandEstimate.rename_axis('Year',axis="columns")

print 'Done estimating the unment demnd pecentage for each scenario'
# display(UnmetDemandEstimate)
```

5.2 Get the unmet demand as a percentage for the scenarios

```
######################################################################
# estimate the total reference demand for Cahce county to calcualte the percentage
result_df_UseCase= pd.read_sql_query(Query_UseCase_text, conn)

subsets = result_df_UseCase.groupby(['ScenarioName'])
for subset in subsets.groups.keys():
    if subset=='Bear River WEAP Model 2017': # reference
        df_Seasonal = subsets.get_group(name=subset)
        df_Seasonal=df_Seasonal.reset_index()
# display (df_Seasonal)

Tot=df_Seasonal["SeasonNumericValue"].tolist()

float_lst = [float(x) for x in Tot]

Annual_Demand=sum(float_lst)
print Annual_Demand

######################################################################



years =UnmetDemandEstimate.index.values

Reference_vals =UnmetDemandEstimate['Reference'].tolist()
Reference_vals_perc =((numpy.array([Reference_vals]))/Annual_Demand)*100


Cons25PercCacheUrbWaterUse_vals =UnmetDemandEstimate['Cons25PercCacheUrbWaterUse'].tolist()
Cons25PercCacheUrbWaterUse_vals_perc =((numpy.array([Cons25PercCacheUrbWaterUse_vals]))/Annual_Demand)*100

Incr25PercCacheUrbWaterUse_vals =UnmetDemandEstimate['Incr25PercCacheUrbWaterUse'].tolist()
Incr25PercCacheUrbWaterUse_vals_perc =((numpy.array([Incr25PercCacheUrbWaterUse_vals]))/Annual_Demand)*100

print 'done estimating unmet demnd the percentages'
```

5.3 Export the unmet demand percent into Excel to load them into WaMDaM

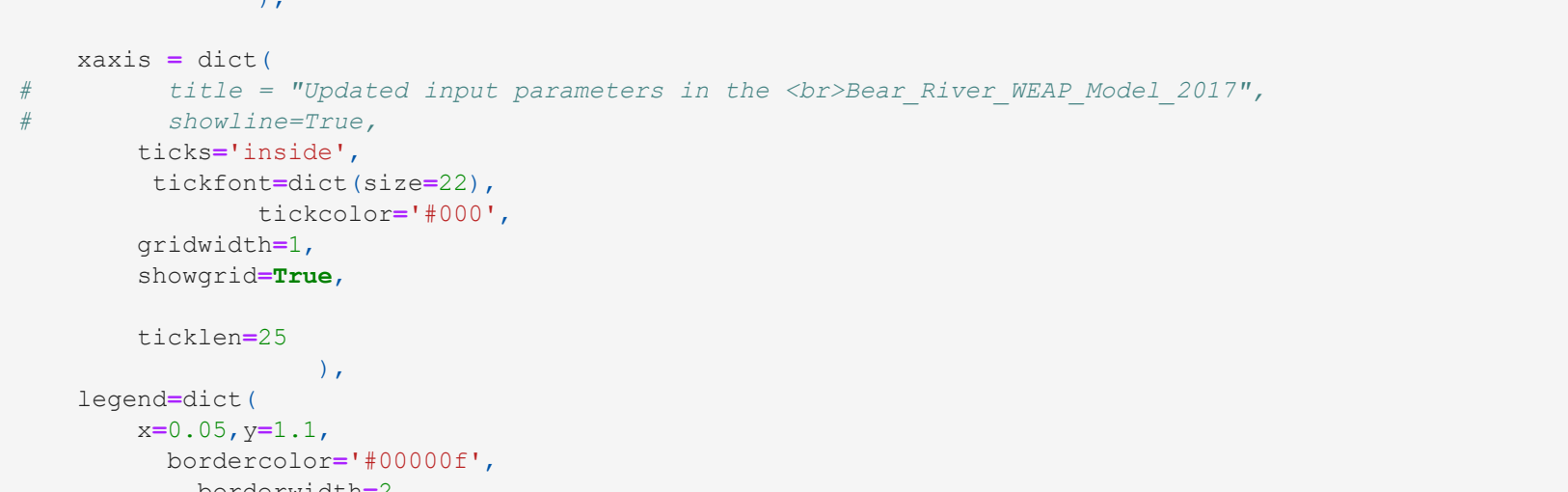
```
# display(UnmetDemandEstimate)
import xlsxwriter
from collections import OrderedDict

UnmetDemandEstimate.to_csv('UnmetDemandEstimate.csv')


ExcelFileName='Test.xlsx'

years =UnmetDemandEstimate.index.values
#print years

Columns=['ObjectType','InstanceName','ScenarioName','AttributeName','DateTimeStamp','Value']

# these three columns have fixed values for all the rows
ObjectType='Demand Site'
InstanceName='Cache County Urban'
AttributeName='UnmetDemand'

# this dict contains the keysL (scenario name) and the values are in a list
# years exist in UnmetDemandEstimate. We then need to add day and month to the year date
# like this format: # DateTimeStamp= 1/1/1993

Scenarios = OrderedDict()

Scenarios['Bear River WEAP Model 2017_result'] = Reference_vals_perc
Scenarios['Incr25PercCacheUrbWaterUse_result'] = Incr25PercCacheUrbWaterUse_vals_perc
Scenarios['Cons25PercCacheUrbWaterUse_result'] = Cons25PercCacheUrbWaterUse_vals_perc
#print Incr25PercCacheUrbWaterUse_vals_perc


workbook = xlsxwriter.Workbook(ExcelFileName)
sheet = workbook.add_worksheet('sheet')


# write headers
for i, header_name in enumerate(Columns):
    sheet.write(0, i, header_name)
row = 1
col = 0


for scenario_name in Scenarios.keys():
    for val_list in Scenarios[scenario_name]:
#         print val_list
        for i, val in enumerate(val_list):
#             print years[i]
            date_timestamp = '1/1/{}'.format(years[i])

            sheet.write(row, 0, ObjectType)
            sheet.write(row, 1, InstanceName)
            sheet.write(row, 2, scenario_name)
            sheet.write(row, 3, AttributeName)
            sheet.write(row, 4, date_timestamp)
            sheet.write(row, 5, val)

            row += 1

workbook.close()


print 'done writing to Excel'

print 'Next, copy the exported data into a WaMDaM workbook template for the WEAP model'
```

6. Plot the unmet demad for all the scenarios and years

```
trace2 = go.Scatter(
    x=years,
    y=Reference_vals_perc[0],
    name = 'Reference demand',
        mode = 'lines+markers',

    marker = dict(
        color = '#264DFF',

))


trace3 = go.Scatter(
    x=years,
    y=Cons25PercCacheUrbWaterUse_vals_perc[0],
    name = 'Conserve demand by 25%',
    mode = 'lines+markers',

    marker = dict(
        color = '#3FA0FF'
))

trace1 = go.Scatter(
    x=years,
    y=Incr25PercCacheUrbWaterUse_vals_perc[0],
    name = 'Increase demand by 25%',
    mode = 'lines+markers',

    marker = dict(
        color = '#290AD8'
))

layout = dict(
    #title = "Use Case 3.3",
    yaxis = dict(
        title = "Annual unmet demand (%)",
        tickformat= ',',
        showline=True,
        dtick='5',
        ticks='outside',

        ticklen=10,
        tickcolor='#000',
        gridwidth=1,
        showgrid=True,

                ),

    xaxis = dict(
#         title = "Updated input parameters in the <br>Bear_River_WEAP_Model_2017",
#         showline=True,
        ticks='inside',
         tickfont=dict(size=22),
                tickcolor='#000',
        gridwidth=1,
        showgrid=True,

        ticklen=25
                    ),
    legend=dict(
        x=0.05,y=1.1,
          bordercolor='#00000f',
            borderwidth=2

                ),
    width=1100,
    height=700,
    #paper_bgcolor='rgb(233,233,233)',
    #plot_bgcolor='rgb(233,233,233)',
    margin=go.Margin(l=130,b=200),
    font=dict(size=25,family='arial',color='#00000f'),
    showlegend=True
)
data = [trace1, trace2,trace3]


# create a figure object
fig = dict(data=data, layout=layout)
#py.iplot(fig, filename = "2.3Identify_SeasonalValues")


## it can be run from the local machine on Pycharm like this like below
## It would also work here offline but in a seperate window
offline.iplot(fig,filename = 'jupyter/UnmentDemand@BirdRefuge' )

print "Figure x is replicated!!"
```

7. Upload the new result scenarios to OpenAgua to visulize them there

You already uploaded the results form WaMDaM SQLite earlier at the begnining of these Jupyter Notebooks. So all you need is to select to display the result in OpenAgua. Finally, click, load data. It should replicate the same figure above and Figure 6 in the paper



8. Close the SQLite and WEAP API connections

```
# 9. Close the SQLite and WEAP API connections
conn.close()

print 'connection disconnected'

# Uncomment
WEAP.SaveArea



# this command will close WEAP
WEAP.Quit

print 'Connection with WEAP API is disconnected'
```

The End :) Congratulations!