Python in the VE

VE Scripts – APS / Vista variables & units

What? This VE Script interrogates an APS file and lists all the variables with display & units data. It provides a utility that will find useful when accessing results from VEScripts . Also provided with this article is a reference sheet that defines the units types that are reported by the script and utilised in the VE.

Why? This VE Script is an example of a number of aspects including using the results interface, getting user inputs from a dialog (GU)I and writer output to a spreadsheet. For the purposes of this example we will concentrate on structure and the APS data rather than the GUI aspect which we will cover in a later article.

Script workflow – start, middle & end:

We are showing the start, middle & end of this script first to explain how it is executed

```python
# VE_Vars_and_Units.py
# Returns a list of the VE APS variables & units contained in the selected APS file inc. both Apache & display names
# Note that APS files contains only variables dictated by sim setup/ options so the list may not be all variables
# Saves the output Xlsx to the project > Content > Python reports folder

import iesve
import tkinter as tk
import xlsxwriter
import os
from datetime import date
```

Import the modules whose methods we intend to use in the script. We import the Tkinter package (a number of modules that provide GUI elements) but rename it to tk for convenience as we will call *tk.module.method* which we do not want to be too long

```python
def generate_window(project, ve_folder, results_reader):

    class Window(tk.Frame):
        # Setup class - inherits from tk frame class
        # https://pythonprogramming.net/python-3-tkinter-basics-tutorial/
```

The middle part of this script is where we have defined a function (a block of code that runs when called), in this case for a tk dialog. A function must de defined ahead where it is called

```python
if __name__ == '__main__':
    project = iesve.VEProject.get_current_project()
    ve_folder = iesve.get_application_folder()
    results_reader = iesve.ResultsReader

    # generate the tkinter GUI
    generate_window(project, ve_folder, results_reader)
```

The end part of this script is where the script starts executing ... *if __name__ == '__main__'* is a guard to prevent accidently invoking the script from another script i.e. you must call it's functions. It is also where you can set parameters to test values that are only used when the script is run directly

First we call some methods to define the parameters we will pass to the function that starts the GUI

We then call the function that starts the GUI

Script GUI – middle:

The middle is shown collapsed so you can see the structure; we will look at tk GUI details more in a later article, but in summary ...

```python
def generate_window(project, ve_folder, results_reader):

    class Window(tk.Frame):
        # Setup class - inherits from tk frame class
        # https://pythonprogramming.net/python-3-tkinter-basics-tutorial/


        def __init__(self, master=None):


        def radio_select(self, pick):


        def save_file(self):


        def init_window(self):


    # Create root widget
    root = tk.Tk()
    # Create instance
    app = Window(root)
    # Show widget & start event loop
    root.mainloop()
```

We define an instance of tk window class

We define a special function called a constructor for the class object (__***__) that initialises the object; it in turn calls init_window

We define functions for the GUI elements that are then called from init_window

This function creates the GUI. Any variables we want to access outside of each function is prefixed with *self* to signify it belongs to the class

We create an instance of the Tk root widget (window, title bar etc.)

We pass the root widget as a parameter to an instance of class window; this adds all the buttons etc

GUI elements are event driven; they require an event loop that waits for a user to press something. This line starts the event loop

Script – save function:

The save function is where we do all the work with the aps file and export it to a formatted spreadsheet ...

```python
        def save_file(self):
            # Saves APS variable names to file

            # Get todays date
            today = date.today()

            # Get selected aps filename
            aps_file_name = self.listbox.get(tk.ACTIVE)
            if not aps_file_name:
                tk.messagebox.showinfo("APS error", "No APS file selected. Please select an APS file.")
                return

            # Open aps file, assert and get APS variable & unit data, then close aps file
            print (aps_file_name)
            results = iesve.ResultsReader.open(aps_file_name)
            assert results is not None, "Error opening results file"
            aps_vars = results.get_variables()
            aps_units = results.get_units()

            results.close()

            # Get entered save filename
            self.save_file_name = self.save_file_entry_box.get()
            print('Save File name = ' + self.save_file_name)

            # Create excel workbook
            file_name = self.save_file_name + '.xlsx'
            report_path = project.path + 'Content\\Python Reports\\'
            if not os.path.exists(report_path):
                os.makedirs(report_path)
            print (report_path)
            workbook = xlsxwriter.Workbook(report_path + file_name)

            # Set formatting options
            bold = workbook.add_format({'bold': True})
            heading = workbook.add_format({'bg_color': '#C8C8C8'})
            right = workbook.add_format({'align': 'right'})

            # Create excel worksheet
            sheet1 = workbook.add_worksheet('APS variables')

            # Insert logo to worksheet, set row height to 50
            sheet1.insert_image('A1', self.ve_folder + 'Templates\\workflow\\IESlogo.png', {'x_scale': 0.9, 'y_scale': 0.9})
            sheet1.set_row(0, 50)

            # Write text to worksheets
            if self.units_system == 'units_metric' or self.units_system == 'units_IP':
                sheet1.write('A3', 'IESVE Variables & Units')
                sheet1.write('J3', 'Source APS file:', right)
                sheet1.write('K3', aps_file_name, right)
                sheet1.write('H3', today.strftime("%d/%m/%Y"), right)
                sheet1.write('A5', 'Source:', heading)
                sheet1.write('B5', 'Apache name:', heading)
                sheet1.write('C5', 'Display name:', heading)
                sheet1.write('D5', 'Units type:', heading)
                sheet1.write('E5', 'Metric units display name:', heading)
                sheet1.write('F5', 'Metric units divisor:', heading)
                sheet1.write('G5', 'Units offset:', heading)
                sheet1.write('H5', 'Model level:', heading)
                sheet1.write('I5', 'Post process spec:', heading)
                sheet1.write('J5', 'Custom type:', heading)
                sheet1.write('K5', 'Sub type:', heading)

            if self.units_system == 'units_both':
                sheet1.write('A3', 'IESVE Variables & Units')
                sheet1.write('J3', 'Source APS file:', right)
                sheet1.write('K3', aps_file_name, right)
                sheet1.write('H3', today.strftime("%d/%m/%Y"), right)
                sheet1.write('A5', 'Source:', heading)
                sheet1.write('B5', 'Apache name:', heading)
                sheet1.write('C5', 'Display name:', heading)
                sheet1.write('D5', 'Units type:', heading)
                sheet1.write('E5', 'Metric units display name:', heading)
                sheet1.write('F5', 'Metric units divisor:', heading)
                sheet1.write('G5', 'Metric Units offset:', heading)
                sheet1.write('H5', 'IP units display name:', heading)
                sheet1.write('I5', 'IP units divisor:', heading)
                sheet1.write('J5', 'IP Units offset:', heading)
                sheet1.write('K5', 'Model level:', heading)
                sheet1.write('L5', 'Post process spec:', heading)
                sheet1.write('M5', 'Custom type:', heading)
                sheet1.write('N5', 'Sub type:', heading)

            # set column widths
            sheet1.set_column('A:A', 25)
            sheet1.set_column('B:C', 35)
            sheet1.set_column('D:D', 25)
            sheet1.set_column('E:N', 20)

            # Write data to excel worksheets
            print('Writing results to Excel Sheet')
            print(self.units_system)
            row = 7
            for var in aps_vars:

                # Find associated units data for var
                units_type = var['units_type']
                units = aps_units[units_type]

                if self.units_system == 'units_metric' or self.units_system == 'units_IP':
                    unit = units[self.units_system]
                    sheet1.write(row, 0, var['source'])
                    sheet1.write(row, 1, var['aps_varname'])
                    sheet1.write(row, 2, var['display_name'])
                    sheet1.write(row, 3, var['units_type'])
                    sheet1.write(row, 4, unit['display_name'])
                    sheet1.write(row, 5, unit['divisor'])
                    sheet1.write(row, 6, unit['offset'])
                    sheet1.write(row, 7, var['model_level'])
                    sheet1.write(row, 8, var['post_process_spec'])
                    sheet1.write(row, 9, var['custom_type'])
                    sheet1.write(row, 10, var['subtype'])
                    row += 1

                if self.units_system == 'units_both':
                    unit_m = units['units_metric']
                    unit_ip = units['units_IP']
                    # Write data to xls row cells
                    sheet1.write(row, 0, var['source'])
                    sheet1.write(row, 1, var['aps_varname'])
                    sheet1.write(row, 2, var['display_name'])
                    sheet1.write(row, 3, var['units_type'])
                    sheet1.write(row, 4, unit_m['display_name'])
                    sheet1.write(row, 5, unit_m['divisor'])
                    sheet1.write(row, 6, unit_m['offset'])
                    sheet1.write(row, 7, unit_ip['display_name'])
                    sheet1.write(row, 8, unit_ip['divisor'])
                    sheet1.write(row, 9, unit_ip['offset'])
                    sheet1.write(row, 10, var['model_level'])
                    sheet1.write(row, 11, var['post_process_spec'])
                    sheet1.write(row, 12, var['custom_type'])
                    sheet1.write(row, 13, var['subtype'])
                    row += 1

            try:
                workbook.close()
            except PermissionError as e:
                print("Couldn't close workbook: ", e)

            project.register_content(report_path, 'Python Reports', file_name, True)
            root.destroy()
```

We call a datetime module method to get todays date so we can write this to the output

We call the GET method for the tkinker listbox object to get it's value

We check if the user has picked a an APS file; if not we pop-up a message

We create an instance of resultsreader using the selected aps file as the parameter & assign it to a variable. We check if the file opens

We call resultsreader methods to get the variables & units

We close the resultsreader object as we now have what we need

We call the GET method for the tkinter entrybox object to get it's value

We make a name for the Excel file by adding the user input to a suffix

We make a name for the file folder by adding the project URL to a fixed string and then check if it exists, else we make it

We make a new Excel file with the Xlswriter.Workbook method with the path and filename as parameters

We define & add formatting options (header, bold, right justify) to the workbook

We add a tab to the workbook and name it

We add a logo the worksheet in cell A1

We set the height of row 1 (index 0) to 50 so the logo fits

We check the units selected, if either we write header info & column headings to the worksheet both both metric & IP

We use the worksheet.write method and pass in the cell reference, column headings strings and the format we want as a parameter

We check the units selected, if both we write header info & column headings strings to the worksheet both both metric & IP and the format we want as a parameter

We use the worksheet.set_column method to set column width each for a range of columns

We initialise a row iterator so that writing starts after the logo and header info

We loop through the list of DICTS of the variables in the aps file

For the var we look up the units type in the DICT; we then use this to look the units of the type in the units DICT

We check the units selected, if either we write the data for the row index each entry incrementing a column position and calling the data in the DICT by key

We increment the row interator

We check the units selected, if both we write the data for the row index each entry incrementing a column position and calling the data in the DICT by key; both metric & IP data is written

We increment the row interator

We close the workbook using a TRY statement to catch if there is an error (such as it is open in Excel!)

We register the new output with VE content manager

We close the GUI

Sample output:

Unit types are detailed on the supplementary spreadsheet

	A	B	C	D	E	F	G	H	I	J	K
1											
2											
3	IESVE Variables & Units							20/07/2021		Source APS file:	december.aps
4											
5	Source:	Apache name:	Display name:	Units type:	Metric units display na	Metric units divisor:	Units offset:	Model level:	Post process spec:	Custom type:	Sub type:
6											
7											
8	0	System elec. CE	System elec. CE	Carbon emission	kgC/h	1	0	c	u	3	0
9	0	System nat. gas CE	System nat. gas CE	Carbon emission	kgC/h	1	0	c	u	3	0
10	0	Total CE ex equip	Total CE ex equip	Carbon emission	kgC/h	1	0	c	u	3	0
11	0	Total system CE	Total system CE	Carbon emission	kgC/h	1	0	c	u	3	0
12	0	Ideal heating CE	CoSP rooms heating CE [obs]	Carbon emission	kgC/h	1	0	c	u	0	0
13	0	Ideal cooling CE	CoSP rooms cooling CE [obs]	Carbon emission	kgC/h	1	0	c	u	0	0
14	0	Ideal humid. CE	CoSP rooms humid. CE [obs]	Carbon emission	kgC/h	1	0	c	u	0	0
15	0	Ideal dehum. CE	CoSP rooms dehum. CE [obs]	Carbon emission	kgC/h	1	0	c	u	0	0
16	0	Ideal mech vent htg CE	CoSP mech vent htg CE [obs]	Carbon emission	kgC/h	1	0	c	u	0	0
17	0	Ideal mech vent clg CE	CoSP mech vent clg CE [obs]	Carbon emission	kgC/h	1	0	c	u	0	0
18	0	Ideal mech vent dehum CE	CoSP mech vent dehum CE [obs]	Carbon emission	kgC/h	1	0	c	u	0	0
19	0	System nat. gas CE	System nat. gas CE	Carbon emission	kgC/h	1	0	c	u	0	0