

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 (GUI) 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:

```

1  # VE_Vars_and_Units.py
2  # Returns a list of the VE APS variables & units contained in the selected APS file inc. both Apache & display names
3  # Note that APS files contains only variables dictated by sim setup/ options so the list may not be all variables
4  # Saves the output Xlsx to the project > Content > Python reports folder
5
6  import iesve
7  import tkinter as tk
8  import xlswriter
9  import os
10 from datetime import date

```

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

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

```

12 def generate_window(project, ve_folder, results_reader):
13
14     class Window(tk.Frame):
15         # Setup class - inherits from tk frame class
16         # 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 be defined ahead where it is called

```

259 if __name__ == '__main__':
260     project = iesve.VEProject.get_current_project()
261     ve_folder = iesve.get_application_folder()
262     results_reader = iesve.ResultsReader
263
264     # generate the tkinter GUI
265     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 accidentally 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:

```

12 def generate_window(project, ve_folder, results_reader):
13
14     class Window(tk.Frame):
15         # Setup class - inherits from tk frame class
16         # https://pythonprogramming.net/python-3-tkinter-basics-tutorial/
17
18
19         def __init__(self, master=None):
20
21
22         def radio_select(self, pick):
23
24
25         def save_file(self):
26
27
28         def init_window(self):
29
30
31         # Create root widget
32         root = tk.Tk()
33
34         # Create instance
35         app = Window(root)
36         # Show widget & start event loop
37         root.mainloop()

```

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 ...

We define an instance of tk window class

We define a special function called a constructor for the class object (`__init__`) 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:

```

50 def save_file(self):
51     # Saves APS variable names to file
52
53     # Get todays date
54     today = date.today()
55
56     # Get selected aps filename
57     aps_file_name = self.listbox.get(tk.ACTIVE)
58     if not aps_file_name:
59         tk.messagebox.showinfo("APS error", "No APS file selected. Please select an APS file.")
60         return
61
62     # Open aps file, assert and get APS variable & unit data, then close aps file
63     print(aps_file_name)
64     results = iesve.ResultsReader.open(aps_file_name)
65     assert results is not None, "Error opening results file"
66     aps_vars = results.get_variables()
67     aps_units = results.get_units()
68
69     results.close()
70
71     # Get entered save filename
72     self.save_file_name = self.save_file_entry_box.get()
73     print('Save File name = ' + self.save_file_name)
74
75     # Create excel workbook
76     file_name = self.save_file_name + '.xlsx'
77     report_path = project.path + 'Content\\Python Reports\\'
78     if not os.path.exists(report_path):
79         os.makedirs(report_path)
80     print(report_path)
81     workbook = xlswriter.Workbook(report_path + file_name)
82
83     # Set formatting options
84     bold = workbook.add_format({'bold': True})
85     heading = workbook.add_format({'bg_color': '#CBC8C8'})
86     right = workbook.add_format({'align': 'right'})
87
88     # Create excel worksheet
89     sheet1 = workbook.add_worksheet('APS variables')
90
91     # Insert logo to worksheet, set row height to 50
92     sheet1.insert_image('A1', self.ve_folder + 'Templates\\workFlow\\IESlogo.png', {'x_scale': 0.9, 'y_scale': 0.9})
93     sheet1.set_row(0, 50)
94
95     # Write text to worksheets
96     if self.units_system == 'units_metric' or self.units_system == 'units_IP':
97         sheet1.write('A3', 'IESVE Variables & Units')
98         sheet1.write('B3', 'Source APS file:', right)
99         sheet1.write('K3', aps_file_name, right)
100        sheet1.write('H3', today.strftime("%d/%m/%Y"), right)
101        sheet1.write('A5', 'Source:', heading)
102        sheet1.write('B5', 'Apache name:', heading)
103        sheet1.write('C5', 'Display name:', heading)
104        sheet1.write('D5', 'Units type:', heading)
105        sheet1.write('E5', 'Metric units display name:', heading)
106        sheet1.write('F5', 'Metric units divisor:', heading)
107        sheet1.write('G5', 'Units offset:', heading)
108        sheet1.write('H5', 'Model level:', heading)
109        sheet1.write('I5', 'Post process spec:', heading)
110        sheet1.write('J5', 'Custom type:', heading)
111        sheet1.write('K5', 'Sub type:', heading)
112
113        if self.units_system == 'units_both':
114            sheet1.write('A3', 'IESVE Variables & Units')
115            sheet1.write('B3', 'Source APS file:', right)
116            sheet1.write('K3', aps_file_name, right)
117            sheet1.write('H3', today.strftime("%d/%m/%Y"), right)
118            sheet1.write('A5', 'Source:', heading)
119            sheet1.write('B5', 'Apache name:', heading)
120            sheet1.write('C5', 'Display name:', heading)
121            sheet1.write('D5', 'Units type:', heading)
122            sheet1.write('E5', 'Metric units display name:', heading)
123            sheet1.write('F5', 'Metric units divisor:', heading)
124            sheet1.write('G5', 'Metric units offset:', heading)
125            sheet1.write('H5', 'IP units display name:', heading)
126            sheet1.write('I5', 'IP units divisor:', heading)
127            sheet1.write('J5', 'IP Units offset:', heading)
128            sheet1.write('K5', 'Model level:', heading)
129            sheet1.write('L5', 'Post process spec:', heading)
130            sheet1.write('M5', 'Custom type:', heading)
131            sheet1.write('N5', 'Sub type:', heading)
132
133        # set column widths
134        sheet1.set_column('A:A', 25)
135        sheet1.set_column('B:C', 35)
136        sheet1.set_column('D:D', 25)
137        sheet1.set_column('E:N', 20)
138
139        # Write data to excel worksheets
140        print('Writing results to Excel Sheet')
141        print(self.units_system)
142        row = 7
143        for var in aps_vars:
144
145            # Find associated units data for var
146            units_type = var['units_type']
147            units = aps_units[units_type]
148
149            if self.units_system == 'units_metric' or self.units_system == 'units_IP':
150                unit = units[self.units_system]
151                sheet1.write(row, 0, var['source'])
152                sheet1.write(row, 1, var['aps_varname'])
153                sheet1.write(row, 2, var['display_name'])
154                sheet1.write(row, 3, var['units_type'])
155                sheet1.write(row, 4, unit['display_name'])
156                sheet1.write(row, 5, unit['divisor'])
157                sheet1.write(row, 6, unit['offset'])
158                sheet1.write(row, 7, var['model_level'])
159                sheet1.write(row, 8, var['post_process_spec'])
160                sheet1.write(row, 9, var['custom_type'])
161                sheet1.write(row, 10, var['subtype'])
162                row += 1
163
164            if self.units_system == 'units_both':
165                unit_m = units['units_metric']
166                unit_ip = units['units_IP']
167                # Write data to xls row cells
168                sheet1.write(row, 0, var['source'])
169                sheet1.write(row, 1, var['aps_varname'])
170                sheet1.write(row, 2, var['display_name'])
171                sheet1.write(row, 3, var['units_type'])
172                sheet1.write(row, 4, unit_m['display_name'])
173                sheet1.write(row, 5, unit_m['divisor'])
174                sheet1.write(row, 6, unit_m['offset'])
175                sheet1.write(row, 7, unit_ip['display_name'])
176                sheet1.write(row, 8, unit_ip['divisor'])
177                sheet1.write(row, 9, unit_ip['offset'])
178                sheet1.write(row, 10, var['model_level'])
179                sheet1.write(row, 11, var['post_process_spec'])
180                sheet1.write(row, 12, var['custom_type'])
181                sheet1.write(row, 13, var['subtype'])
182                row += 1
183
184            try:
185                workbook.close()
186            except PermissionError as e:
187                print("Couldn't close workbook: ", e)
188
189        project.register_content(report_path, 'Python Reports', file_name, True)
190        root.destroy()

```

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

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 tkinter 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 iterator

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 iterator

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

Source	Apache name	Display name	Units type	Metric units display name	Metric units divisor	Units offset	Model level	Post process spec	Custom type	Sub type
0	System elec. CE	System elec. CE	Carbon emission	kgC/h	1	0	c	u		
0	System nat. gas CE	System nat. gas CE	Carbon emission	kgC/h	1	0	c	u		
0	Total CE ex equip	Total CE ex equip	Carbon emission	kgC/h	1	0	c	u		
0	Total system CE	Total system CE	Carbon emission	kgC/h	1	0	c	u		
0	Ideal heating CE	COSP rooms heating CE (jobs)	Carbon emission	kgC/h	1	0	c	u		
0	Ideal cooling CE	COSP rooms cooling CE (jobs)	Carbon emission	kgC/h	1	0	c	u		
0	Ideal humid. CE	COSP rooms humid. CE (jobs)	Carbon emission	kgC/h	1	0	c	u		
0	Ideal dehum. CE	COSP rooms dehum. CE (jobs)	Carbon emission	kgC/h	1	0	c	u		
0	Ideal mech vent htg CE	COSP mech vent htg CE (jobs)	Carbon emission	kgC/h	1	0	c	u		
0	Ideal mech vent cldg CE	COSP mech vent cldg CE (jobs)	Carbon emission	kgC/h	1	0	c	u		
0	Ideal mech vent dehum CE	COSP mech vent dehum CE (jobs)	Carbon emission	kgC/h	1	0	c	u		
0	System nat. gas CE	System nat. gas CE	Carbon emission	kgC/h	1	0	c	u		