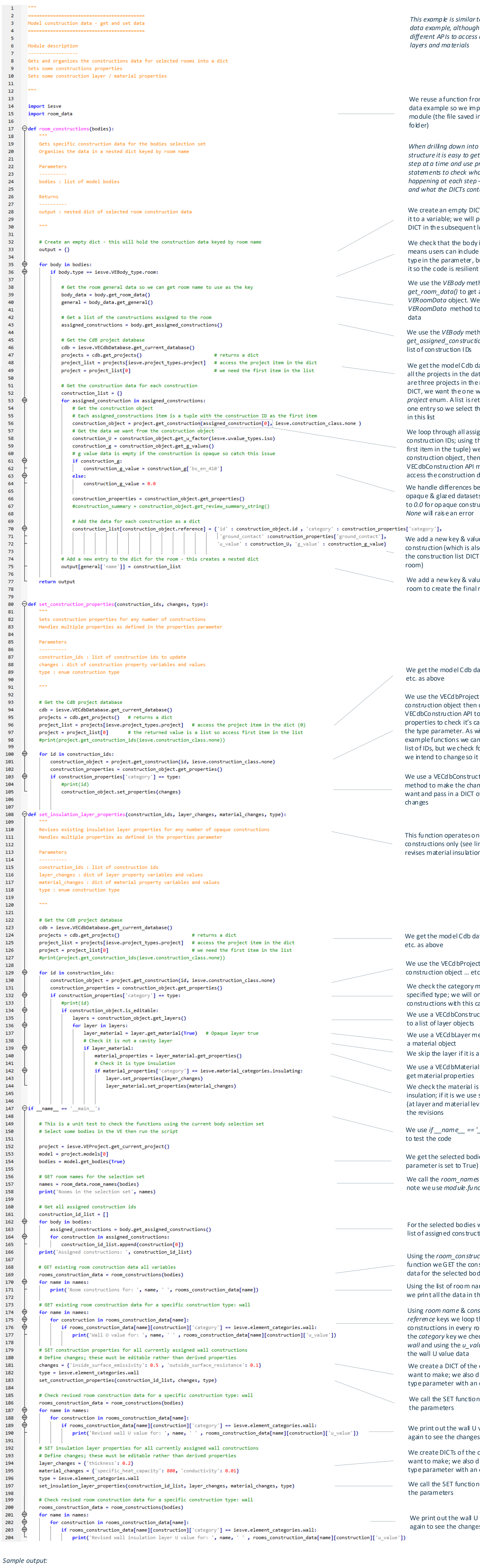
Python in the VE

VE Scripts – working with constructions

What? We use GET and SET methods to access VE construction data. To use them you need to navigate the API hierarchy. The GET methods typically return data in DICT form, often nested, which we need to be able to access. Using DICTs is also a good way of organising data that we GET from the VE.

Why? In this example we navigate VECdb, VECdbConstruction, VECdbLayer and VECdbMaterial and get construction data via methods from these APIs. We use a DICT to organise the data we extract because DICT keys make sure we store what the values mean and give us an elegant way to access specific data. We also utilise DICTs to specify changes we make using SET methods.



```python
1   """
2   ==========================================
3   Model construction data - get and set data
4   ==========================================
5
6   Module description
7   ------------------
8   Gets and organizes the constructions data for selected rooms into a dict
9   Sets some constructions properties
10  Sets some construction layer / material properties
11
12  """
13
14  import iesve
15  import room_data
16
17  def room_constructions(bodies):
18      """
19      Gets specific construction data for the bodies selection set
20      Organizes the data in a nested dict keyed by room name
21
22      Parameters
23      ----------
24      bodies : list of model bodies
25
26      Returns
27      ----------
28      output : nested dict of selected room construction data
29
30      """
31
32      # Create an empty dict - this will hold the construction data keyed by room name
33      output = {}
34
35      for body in bodies:
36          if body.type == iesve.VEBody_type.room:
37
38              # Get the room general data so we can get room name to use as the key
39              body_data = body.get_room_data()
40              general = body_data.get_general()
41
42              # Get a list of the constructions assigned to the room
43              assigned_constructions = body.get_assigned_constructions()
44
45              # Get the CdB project database
46              cdb = iesve.VECdbDatabase.get_current_database()
47              projects = cdb.get_projects()                         # returns a dict
48              project_list = projects[iesve.project_types.project]  # access the project item in the dict
49              project = project_list[0]                             # we need the first item in the list
50
51              # Get the construction data for each construction
52              construction_list = {}
53              for assigned_construction in assigned_constructions:
54                  # Get the construction object
55                  # Each assigned_constructions item is a tuple with the construction ID as the first item
56                  construction_object = project.get_construction(assigned_construction[0], iesve.construction_class.none )
57                  # Get the data we want from the construction object
58                  construction_U = construction_object.get_u_factor(iesve.uvalue_types.iso)
59                  construction_g = construction_object.get_g_values()
60                  # g value data is empty if the construction is opaque so catch this issue
61                  if construction_g:
62                      construction_g_value = construction_g['bs_en_410']
63                  else:
64                      construction_g_value = 0.0
65
66                  construction_properties = construction_object.get_properties()
67                  #construction_summary = construction_object.get_review_summary_string()
68
69                  # Add the data for each construction as a dict
70                  construction_list[construction_object.reference] = {'id' : construction_object.id , 'category' : construction_properties['category'],
71                                                                      'ground_contact' :construction_properties['ground_contact'],
72                                                                      'u_value' : construction_U, 'g_value' : construction_g_value}
73
74              # Add a new entry to the dict for the room - this creates a nested dict
75              output[general['name']] = construction_list
76
77      return output
78
79
80  def set_construction_properties(construction_ids, changes, type):
81      """
82      Sets construction properties for any number of constructions
83      Handles multiple properties as defined in the properties parameter
84
85      Parameters
86      ----------
87      construction_ids : list of construction ids to update
88      changes : dict of construction property variables and values
89      type : enum construction type
90
91      """
92
93      # Get the CdB project database
94      cdb = iesve.VECdbDatabase.get_current_database()
95      projects = cdb.get_projects()   # returns a dict
96      project_list = projects[iesve.project_types.project]   # access the project item in the dict (0)
97      project = project_list[0]       # the returned value is a list so access first item in the list
98      #print(project.get_construction_ids(iesve.construction_class.none))
99
100     for id in construction_ids:
101         construction_object = project.get_construction(id, iesve.construction_class.none)
102         construction_properties = construction_object.get_properties()
103         if construction_properties['category'] == type:
104             #print(id)
105             construction_object.set_properties(changes)
106
107
108 def set_insulation_layer_properties(construction_ids, layer_changes, material_changes, type):
109     """
110     Revises existing insulation layer properties for any number of opaque constructions
111     Handles multiple properties as defined in the properties parameter
112
113     Parameters
114     ----------
115     construction_ids : list of construction ids
116     layer_changes : dict of layer property variables and values
117     material_changes : dict of material property variables and values
118     type : enum construction type
119
120     """
121
122     # Get the CdB project database
123     cdb = iesve.VECdbDatabase.get_current_database()
124     projects = cdb.get_projects()                         # returns a dict
125     project_list = projects[iesve.project_types.project]  # access the project item in the dict
126     project = project_list[0]                             # we need the first item in the list
127     #print(project.get_construction_ids(iesve.construction_class.none))
128
129     for id in construction_ids:
130         construction_object = project.get_construction(id, iesve.construction_class.none)
131         construction_properties = construction_object.get_properties()
132         if construction_properties['category'] == type:
133             #print(id)
134             if construction_object.is_editable:
135                 layers = construction_object.get_layers()
136                 for layer in layers:
137                     layer_material = layer.get_material(True)   # Opaque layer true
138                     # Check it is not a cavity layer
139                     if layer_material:
140                         material_properties = layer_material.get_properties()
141                         # Check it is type insulation
142                         if material_properties['category'] == iesve.material_categories.insulating:
143                             layer.set_properties(layer_changes)
144                             layer_material.set_properties(material_changes)
145
146
147 if __name__ == '__main__':
148
149     # This is a unit test to check the functions using the current body selection set
150     # Select some bodies in the VE then run the script
151
152     project = iesve.VEProject.get_current_project()
153     model = project.models[0]
154     bodies = model.get_bodies(True)
155
156     # GET room names for the selection set
157     names = room_data.room_names(bodies)
158     print('Rooms in the selection set', names)
159
160     # Get all assigned construction ids
161     construction_id_list = []
162     for body in bodies:
163         assigned_constructions = body.get_assigned_constructions()
164         for construction in assigned_constructions:
165             construction_id_list.append(construction[0])
166     print('Assigned constructions: ', construction_id_list)
167
168     # GET existing room construction data all variables
169     rooms_construction_data = room_constructions(bodies)
170     for name in names:
171         print('Room constructions for: ', name, ' ', rooms_construction_data[name])
172
173     # GET existing room construction data for a specific construction type: wall
174     for name in names:
175         for construction in rooms_construction_data[name]:
176             if rooms_construction_data[name][construction]['category'] == iesve.element_categories.wall:
177                 print('Wall U value for: ', name, ' ' , rooms_construction_data[name][construction]['u_value'])
178
179     # SET construction properties for all currently assigned wall constructions
180     # Define changes; these must be editable rather than derived properties
181     changes = {'inside_surface_emissivity': 0.5 , 'outside_surface_resistance': 0.1}
182     type = iesve.element_categories.wall
183     set_construction_properties(construction_id_list, changes, type)
184
185     # Check revised room construction data for a specific construction type: wall
186     rooms_construction_data = room_constructions(bodies)
187     for name in names:
188         for construction in rooms_construction_data[name]:
189             if rooms_construction_data[name][construction]['category'] == iesve.element_categories.wall:
190                 print('Revised wall U value for: ', name, ' ' , rooms_construction_data[name][construction]['u_value'])
191
192     # SET insulation layer properties for all currently assigned wall constructions
193     # Define changes; these must be editable rather than derived properties
194     layer_changes = {'thickness': 0.2}
195     material_changes = {'specific_heat_capacity': 800, 'conductivity': 0.01}
196     type = iesve.element_categories.wall
197     set_insulation_layer_properties(construction_id_list, layer_changes, material_changes, type)
198
199     # Check revised room construction data for a specific construction type: wall
200     rooms_construction_data = room_constructions(bodies)
201     for name in names:
202         for construction in rooms_construction_data[name]:
203             if rooms_construction_data[name][construction]['category'] == iesve.element_categories.wall:
204                 print('Revised wall insulation layer U value for: ', name, ' ' , rooms_construction_data[name][construction]['u_value'])
```

This example is similar to the room data example, although we are using different APIs to access constructions, layers and materials

We reuse a function from the room data example so we import the module (the file saved in the same folder)

When drilling down into the data structure it is easy to get lost; do it a step at a time and use print statements to check what is happening at each step – the objects and what the DICTs contain

We create an empty DICT and assign it to a variable; we will populate the DICT in the subsequent loop

We check that the body is a room; it means users can include any body type in the parameter, but we handle it so the code is resilient

We use the *VEBody* method *get_room_data()* to get a *VERoomData* object. We then call a *VERoomData* method to get *general* data

We use the *VEBody* method *get_assigned_constructions()* to get a list of construction IDs

We get the model Cdb database, then all the projects in the database. There are three projects in the returned DICT, we want the one with the key *project* enum. A list is returned with one entry so we select the first item in this list

We loop through all assigned construction IDs; using the ID (the first item in the tuple) we get the construction object, then using VECdbConstruction API methods we access the construction data we want

We handle differences between opaque & glazed datasets. We set g to *0.0* for opaque constructions as *None* will raise an error

We add a new key & value for each construction (which is also a DICT) to the construction list DICT (for one room)

We add a new key & value for each room to create the final nested DICT

We get the model Cdb database … etc. as above

We use the VECdbProject API to get a construction object then use the VECdbConstruction API to access it's properties to check it's category with the type parameter. As with other example functions we can pass in any list of IDs, but we check for the type we intend to change so it is resilient

We use a VECdbConstruction API method to make the changes we want and pass in a DICT of those changes

This function operates on opaque constructions only (see line 137) as it revises material insulation layers

We get the model Cdb database … etc. as above

We use the VECdbProject API to get a construction object … etc. as above

We check the category matches the specified type; we will only change constructions with this category

We use a VECdbConstruction method to a list of layer objects

We use a VECdbLayer method to get a material object

We skip the layer if it is a cavity

We use a VECdbMaterial method to get material properties

We check the material is of category insulation; if it is we use set methods (at layer and material levels) to make the revisions

We use *if __name__ == '__main__':* to test the code

We get the selected bodies (parameter is set to True)

We call the *room_names* function; note we use *module.function_name*

For the selected bodies we compile a list of assigned construction IDs

Using the *room_constructions* function we GET the constructions data for the selected bodies

Using the list of room names as a key we print all the data in the nested DICT

Using *room name* & *construction reference* keys we loop through the constructions in every room; using the *category* key we check if it is a *wall* and using the *u_value* key print the wall U value data

We create a DICT of the changes we want to make; we also define the type parameter with an enum

We call the SET function and pass in the parameters

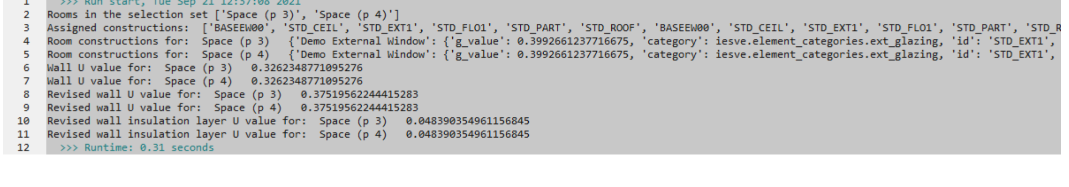
We print out the wall U value data again to see the changes

We create DICTs of the changes we want to make; we also define the type parameter with an enum

We call the SET function and pass in the parameters

We print out the wall U value data again to see the changes

Sample output:



```
  >>> Run start, Tue Sep 21 12:37:08 2021
Rooms in the selection set ['Space (p 3)', 'Space (p 4)']
Assigned constructions:  ['BASEEW00', 'STD_CEIL', 'STD_EXT1', 'STD_FLO1', 'STD_PART', 'STD_ROOF', 'BASEEW00', 'STD_CEIL', 'STD_EXT1', 'STD_FLO1', 'STD_PART', 'STD_R
Room constructions for:  Space (p 3)   {'Demo External Window': {'g_value': 0.3992661237716675, 'category': iesve.element_categories.ext_glazing, 'id': 'STD_EXT1',
Room constructions for:  Space (p 4)   {'Demo External Window': {'g_value': 0.3992661237716675, 'category': iesve.element_categories.ext_glazing, 'id': 'STD_EXT1',
Wall U value for:  Space (p 3)   0.3262348771095276
Wall U value for:  Space (p 4)   0.3262348771095276
Revised wall U value for:  Space (p 3)   0.37519562244415283
Revised wall U value for:  Space (p 4)   0.37519562244415283
Revised wall insulation layer U value for:  Space (p 3)   0.048390354961156845
Revised wall insulation layer U value for:  Space (p 4)   0.048390354961156845
  >>> Runtime: 0.31 seconds
```