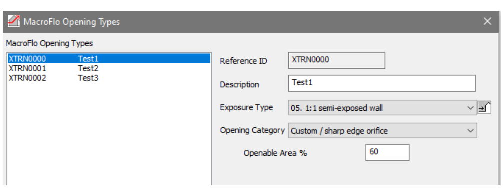
Python in the VE

VE Scripts – working with Macroflo & Model openings

What? We use GET and SET methods with Macroflo opening types. We also use GET and SET methods with model openings data. The GET & SET methods typically use data in DICT form. Using DICTs is also a good way of organising data that we GET from the VE. In this example we use a DICT to match selections and revise data when we iterate to assess the impact of changing Macroflo opening extent on maximum room temperate

Why? In this example we navigate VEMacroflo and VEBody 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

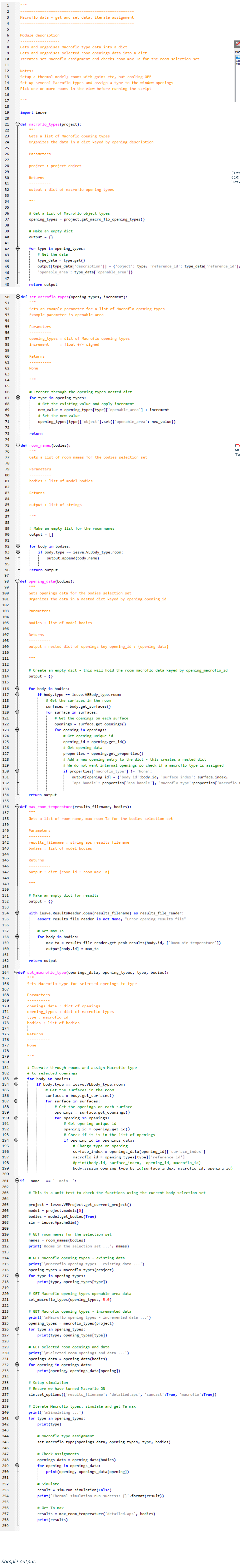
For this example we need to set-up a model with Macroflo opening types created, assigned and make sure that cooling is OFF


MacroFlo Opening Types

MacroFlo Opening Types

XTRN0000 Test1
XTRN0001 Test2
XTRN0002 Test3

Reference ID: XTRN0000
Description: Test1
Exposure Type: 05. 1:1 semi-exposed wall
Opening Category: Custom / sharp edge orifice
Openable Area %: 60



```
"""
====================================================
Macroflo data - get and set data, iterate assignment
====================================================

Module description
------------------
Gets and organises Macroflo type data into a dict
Gets and organises selected room openings data into a dict
Iterates set Macroflo assignment and checks room max Ta for the room selection set

Notes:
Setup a thermal model; rooms with gains etc, but cooling OFF
Set up several Macroflo types and assign a type to the window openings
Pick one or more rooms in the view before running the script

"""

import iesve

def macroflo_types(project):
    """
    Gets a list of Macroflo opening types
    Organizes the data in a dict keyed by opening description

    Parameters
    ----------
    project : project object

    Returns
    ----------
    output : dict of macroflo opening types

    """

    # Get a list of Macroflo object types
    opening_types = project.get_macro_flo_opening_types()

    # Make an empty dict
    output = {}

    for type in opening_types:
        # Get the data
        type_data = type.get()
        output[type_data['description']] = {'object': type, 'reference_id': type_data['reference_id'],
        'openable_area': type_data['openable_area']}

    return output
```

We create a function to GET data for all the Macroflo opening types. We are only interested in some of the opening data so we will assemble the required data and output it in a nested DICT … the format is …

{'**Test1**': {'object': <iesve.VEMacroFlo object at 0x0000027 17C5A2450>, 'reference_id': 'XTRN0000', 'openable_area': 60.0},
'**Test2**': {'object ….. etc }}

We GET the Macroflo opening types

We create an empty DICT to receive the opening data we want

We iterate through the list of opening types & for each we GET it's data; this data is returned as a DICT

We assemble the data we want in a DICT and assign it to a key in the output nested DICT

```
def set_macroflo_types(opening_types, increment):
    """
    Sets an example parameter for a list of Macroflo opening types
    Example parameter is openable area

    Parameters
    ----------
    opening_types : dict of Macroflo opening types
    increment     : float +/- signed

    Returns
    ----------
    None

    """

    # Iterate through the opening types nested dict
    for type in opening_types:
        # Get the existing value and apply increment
        new_value = opening_types[type]['openable_area'] + increment
        # Set the new value
        opening_types[type]['object'].set({'openable_area': new_value})

    return
```

We create a function that will SET the *openable area* value for all Macroflo types. In this example we increment the value with the parameter *increment* (so it will go up or down each time we call the function)

We iterate through the *opening types* nested DICT; as it is a DICT the iterator (*type)* is the DICT keys

We access the *opening types* nested DICT data using the *type* key & the *property* key. We increment the value and assign it to the *new_value* variable

We then SET the new value by passing a DICT of revisions via the SET method on the Macroflo object (the Macroflo object reference is saved in the openings type DICT keyed by *type* & *object*) …

{'Test1': {'object': <iesve.VEMacroFlo object at 0x0000027 17C5A2450>, 'reference_id': 'XTRN0000', 'openable_area': 60.0},
'Test2': {'object ….. etc }}

```
def room_names(bodies):
    """
    Gets a list of room names for the bodies selection set

    Parameters
    ----------
    bodies : list of model bodies

    Returns
    ----------
    output : list of strings

    """

    # Make an empty list for the room names
    output = []

    for body in bodies:
        if body.type == iesve.VEBody_type.room:
            output.append(body.name)

    return output
```

We create a function that will return a LIST of body names for the list of bodies we have passed in as a parameter. We can then use this list to iterate through the geometry. It is good practice to create a function of any code that is used more than once

We create an empty list for the output

We iterate through the body list, check it is a room and then append the body attribute *name* to the output list

```
def opening_data(bodies):
    """
    Gets openings data for the bodies selection set
    Organizes the data in a nested dict keyed by opening opening_id

    Parameters
    ----------
    bodies : list of model bodies

    Returns
    ----------
    output : nested dict of openings key opening_id : {opening data}

    """

    # Create an empty dict - this will hold the room macroflo data keyed by opening_macroflo_id
    output = {}

    for body in bodies:
        if body.type == iesve.VEBody_type.room:
            # Get the surfaces in the room
            surfaces = body.get_surfaces()
            for surface in surfaces:
                # Get the openings on each surface
                openings = surface.get_openings()
                for opening in openings:
                    # Get opening unique id
                    opening_id = opening.get_id()
                    # Get opening data
                    properties = opening.get_properties()
                    # Add a new opening entry to the dict - this creates a nested dict
                    # We do not want internal openings so check if a macroflo type is assigned
                    if properties['macroflo_type'] != 'None':
                        output[opening_id] = {'body_id':body.id, 'surface_index': surface.index,
                        'aps_handle': properties['aps_handle'], 'macroflo_type':properties['macroflo_type']}

    return output
```

We create a function that will return a nested DICT of all openings in the model that have a Macroflo type assigned. The DICT keyed by opening ID will include body id, surface index, aps handle and Macroflo type

We create an empty DICT for the output

We iterate through the list of bodies passed in as a parameter. We then use nested loops to drill down through the room surfaces and the openings

We GET opening data; this returns a DICT

We assemble the data we want in a DICT and assign it to a key using the opening ID in the output nested DICT

```
def max_room_temperature(results_filename, bodies):
    """
    Gets a list of room name, max room Ta for the bodies selection set

    Parameters
    ----------
    results_filename : string aps results filename
    bodies : list of model bodies

    Returns
    ----------
    output : dict {room id : room max Ta}

    """

    # Make an empty dict for results
    output = {}

    with iesve.ResultsReader.open(results_filename) as results_file_reader:
        assert results_file_reader is not None, "Error opening results file"

        # Get max Ta
        for body in bodies:
            max_ta = results_file_reader.get_peak_results(body.id, ['Room air temperature'])
            output[body.id] = max_ta

    return output
```

We create a function that will return a DICT of the selected bodies and the max air temperature the space reaches with results file name that is passed in as a parameter

We create an empty DICT for the output

We open the results file using *with*; this automatically closes the results file outside of the *with* code block. We also use *assert* to check open has returned an object

We iterate through all bodies using the results reader API to get the peak value. We add a key : value pair to the output DICT

```
def set_macroflo_type(openings_data, opening_types, type, bodies):
    """
    Sets Macroflo type for selected openings to type

    Parameters
    ----------
    openings_data : dict of openings
    opening_types : dict of macroflo types
    type : macroflo_id
    bodies : list of bodies

    Returns
    ----------
    None

    """

    # Iterate through rooms and assign Macroflo type
    # to selected openings
    for body in bodies:
        if body.type == iesve.VEBody_type.room:
            # Get the surfaces in the room
            surfaces = body.get_surfaces()
            for surface in surfaces:
                # Get the openings on each surface
                openings = surface.get_openings()
                for opening in openings:
                    # Get opening unique id
                    opening_id = opening.get_id()
                    # Check if it is in the list of openings
                    if opening_id in openings_data:
                        # Change type on opening
                        surface_index = openings_data[opening_id]['surface_index']
                        macroflo_id = opening_types[type]['reference_id']
                        #print(body.id, surface_index,  opening_id, macroflo_id)
                        body.assign_opening_type_by_id(surface_index, macroflo_id, opening_id)
```

We create a function that will assign a Macroflo opening type that we pass in as a parameter to all openings in the bodies list

We iterate through the list of bodies . We then use nested loops to drill down through the room surfaces and the openings

We check that the opening id is in the openings data DICT. If it is we than use the openings data DICT and openings types DICT to look-up parameters we need for the *body.assign_opening_type_by_id()* method to make the assignment

Note that we do not return anything so there is no *return* statement

```
if __name__ == '__main__':

    # This is a unit test to check the functions using the current body selection set

    project = iesve.VEProject.get_current_project()
    model = project.models[0]
    bodies = model.get_bodies(True)
    sim = iesve.ApacheSim()

    # GET room names for the selection set
    names = room_names(bodies)
    print('Rooms in the selection set ...', names)

    # GET Macroflo opening types - existing data
    print('\nMacroflo opening types - existing data ...')
    opening_types = macroflo_types(project)
    for type in opening_types:
        print(type, opening_types[type])

    # SET Macroflo opening types openable area data
    set_macroflo_types(opening_types, 5.0)

    # GET Macroflo opening types - incremented data
    print('\nMacroflo opening types - incremented data ...')
    opening_types = macroflo_types(project)
    for type in opening_types:
        print(type, opening_types[type])

    # GET selected room openings and data
    print('\nSelected room openings and data ...')
    openings_data = opening_data(bodies)
    for opening in openings_data:
        print(opening, openings_data[opening])

    # Setup simulation
    # Ensure we have turned Macroflo ON
    sim.set_options({'results_filename': 'detailed.aps', 'suncast':True, 'macroflo':True})

    # Iterate Macroflo types, simulate and get Ta max
    print('\nSimulating ...')
    for type in opening_types:
        print(type)

        # Macroflo type assignment
        set_macroflo_type(openings_data, opening_types, type, bodies)

        # Check assignments
        openings_data = opening_data(bodies)
        for opening in openings_data:
            print(opening, openings_data[opening])

        # Simulate
        result = sim.run_simulation(False)
        print('Thermal simulation run success: {}'.format(result))

        # Get Ta max
        results = max_room_temperature('detailed.aps', bodies)
        print(results)
```

We use *if __name__ == '__main__':* to create a means to test the functions

We get the current project, then the actual model, then a list of bodies that are currently *selected* by the user in the VE UI by setting the *get_bodies* method parameter to *True*

We create an instance of the ApacheSim class & assign it to a variable

We call the function to get a list of room bodies

We call the function to get a DICT of Macroflo opening types & print

We call the function to SET openable area for all Macroflo opening types

We call the function to get a DICT of Macroflo opening types & print to see the change

We call the function to get a DICT of openings data & print

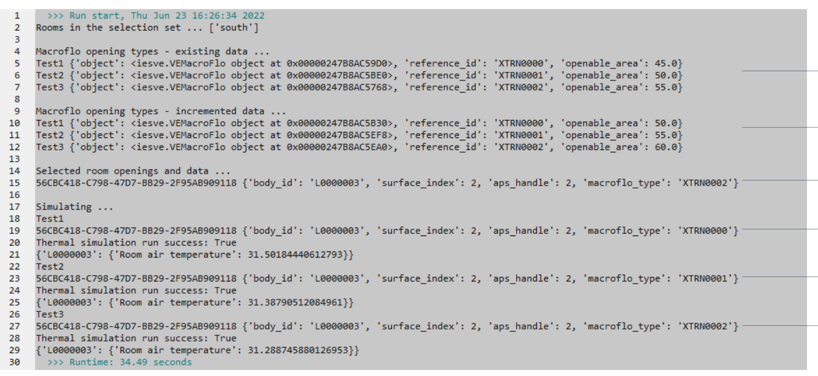
We SET the options for thermal simulation

We use the Macroflo opening type as an iterator; within each iteration we will:

- SET type for all openings using the function we created
- Print out the openings data to see the change
- Simulate the revised model. We print out what the sim object returns i.e. if it successfully ran
- Get a DICT of body : max air temperature using the function we created & print

It would be straightforward to revise the loop to include a test and increment the data so that it iterates to a maxima or minima

Sample output:



```
  >>> Run start, Thu Jun 23 16:26:34 2022
Rooms in the selection set ... ['south']

Macroflo opening types - existing data ...
Test1 {'object': <iesve.VEMacroFlo object at 0x00000247B8AC59D0>, 'reference_id': 'XTRN0000', 'openable_area': 45.0}
Test2 {'object': <iesve.VEMacroFlo object at 0x00000247B8AC5BE0>, 'reference_id': 'XTRN0001', 'openable_area': 50.0}
Test3 {'object': <iesve.VEMacroFlo object at 0x00000247B8AC5768>, 'reference_id': 'XTRN0002', 'openable_area': 55.0}

Macroflo opening types - incremented data ...
Test1 {'object': <iesve.VEMacroFlo object at 0x00000247B8AC5B30>, 'reference_id': 'XTRN0000', 'openable_area': 50.0}
Test2 {'object': <iesve.VEMacroFlo object at 0x00000247B8AC5EF8>, 'reference_id': 'XTRN0001', 'openable_area': 55.0}
Test3 {'object': <iesve.VEMacroFlo object at 0x00000247B8AC5EA0>, 'reference_id': 'XTRN0002', 'openable_area': 60.0}

Selected room openings and data ...
56CBC418-C798-47D7-BB29-2F95AB909118 {'body_id': 'L0000003', 'surface_index': 2, 'aps_handle': 2, 'macroflo_type': 'XTRN0002'}

Simulating ...
Test1
56CBC418-C798-47D7-BB29-2F95AB909118 {'body_id': 'L0000003', 'surface_index': 2, 'aps_handle': 2, 'macroflo_type': 'XTRN0000'}
Thermal simulation run success: True
{'L0000003': {'Room air temperature': 31.50184440612793}}
Test2
56CBC418-C798-47D7-BB29-2F95AB909118 {'body_id': 'L0000003', 'surface_index': 2, 'aps_handle': 2, 'macroflo_type': 'XTRN0001'}
Thermal simulation run success: True
{'L0000003': {'Room air temperature': 31.38790512084961}}
Test3
56CBC418-C798-47D7-BB29-2F95AB909118 {'body_id': 'L0000003', 'surface_index': 2, 'aps_handle': 2, 'macroflo_type': 'XTRN0002'}
Thermal simulation run success: True
{'L0000003': {'Room air temperature': 31.288745880126953}}
  >>> Runtime: 34.49 seconds
```

Macroflo opening types starting data

Macroflo opening types revised data

Selection set Openings data

1st iteration, sim was successful, Ta max

2nd iteration, sim was successful, Ta max

3rd iteration, sim was successful, Ta max