EpJSON ExpandObjects

GARD Analytics and NREL for US DOE

Sep 22, 2021

OVERVIEW:

CHAPTER

ONE

DOCUMENTATION

1.1 Code Structure Overview

1.1.1 Overview

The python code provides a framework to support the creation of EnergyPlus objects. The instructions for these actions are stored in a YAML file which is structured to eliminate as much redundancy as possible, and provide a clear outline that explains how template options affect the final output.

1.1.2 Process Flow

Outlined below are the general steps that the program takes to perform its operation.

1. A schema file is read and validated using the jsonschema package Versioned Validator class. In this case, the Draft4Validator is used. If the *–no-schema* option has been passed to the program, then no validation will occur at any point in the process.
2. The epJSON file is read, validated, and loaded as a dictionary object.
3. All HVACTemplate objects are loaded into HVACTemplate classes for processing.
4. The template objects are processed to create regular objects for the specific system, zone, plant, or loop.

i. An *OptionTree* is loaded from the yaml file, which specifies build instructions for a given template.

ii. EnergyPlus objects are created according to the *BuildPath* provided within the *OptionTree*. A *BuildPath* is an ordered list of objects that follow an air loop path. These *BuildPaths* are constructed based on HVACTemplate field inputs.

iii. Additional objects outside of the build path, but specific to the template, are created from instructions provided in a *TemplateObjects* branch. These are objects such as Controllers, NodeLists, SetpointManagers, etc.

iv. Objects that connect equipment to each other are created (e.g. Zone:Splitters/Mixers, SupplyPath, Connectors, BranchLists, etc.) through automated functions.

5. The epJSON objects created within each HVACTemplate class are merged together into a single document. 8. The epJSON file is validated against the schema. 9. A dictionary output is provided to the downstream process. At a minimum, this object will have a key called 'outputPreProcessMessage' which will contain all warnings and errors generated from this process. On a successful template expansion, a valid epJSON object will also be produced with the file name <original-file-name>_expanded.epJSON 10. If the *–backup-files* option has been specified, the optional "<original-file-name>_hvac_templates.epJSON" and "<original-file-name>_base.epJSON" files will be output.

1.1.3 Yaml Description

This program makes use of a YAML file to hold structured objects as well as instructions for alternative object creation based on template inputs. This Template Expansion Structure YAML file makes use of anchors and aliases to reduce the amount of redundant code, ensure object consistency between definitions, and provide a more reliable framework for future development and maintenance. With these tools, a structure has been created to define EnergyPlus objects and reference node locations without the need to statically type text values for every case. Values can be directly inserted, or references to template inputs can be made. All references made are scoped within the HVACTemplate object, and references to other HVACTemplate objects cannot be made.

Object Structure

To hold the necessary information for construction, the Yaml objects which represent EnergyPlus objects in a *BuildPath* have additional higher-level information, and are termed 'super' objects. These standard format for these objects is as follows:

```
EnergyPlusObject:
  Fields:
    name: object_name
    field_name: field_value
  Connectors:
    ConnectorPathType:
      Inlet: inlet_node_name
      Outlet: outlet_node_name
```

- EnergyPlusObject: This field is a valid EnergyPlus object type.
- Fields: This is hierarchical container which holds valid EnergyPlus object field names. Additionally, a *name* field and value must be provided to give the object a unique identifier for that object type.
- field_name: The exact EnergyPlus field name must be provided.
- field_value: A *complex reference* may be provided. Please see below for further details
- Connectors: This is a hierarchical container which holds instructions on how an object connects to it's neighbors in a *BuildPath*.
- ConnectorPathType: Must be a selection of AirLoop (only valid option at this time)
- inlet_node_name, outlet_node_name: Must be a field_name provided from Fields

BuildPath Objects

This is a list of EnergyPlus 'super' objects which are in order of their fluid flow path. Their field names mirror those of EnergyPlus objects while also holding extra information necessary to connect their input/output nodes in an overall path.

- Fields - key/value pairs of EnergyPlus object fields and values.
  - Key: EnergyPlus field name
  - Value: Value to be inserted as default text. References to template inputs can be made through string formatting, which is the use of brackets '{template_input}'. A blank set of brackets '{}' will insert the template's unique name identifier.
- Connectors - object for identifying how the fluid flow loop passes through the object. This is used to connect nodes between objects in a *BuildPath*.
  - Loop Type [AirLoop]
    - Inlet - EnergyPlus inlet node name for the loop type.
    - Outlet - EnergyPlus outlet node name for the loop type.

Example:

```
OutdoorAir:Mixer:
  Fields:
    name: '{} OA Mixing Box'
    mixed_air_node_name: '{} Mixed Air Outlet'
    outdoor_air_stream_node_name: '{} Outside Air Inlet'
    relief_air_stream_node_name: '{} Relief Air Outlet'
    return_air_stream_node_name: '{} Return Air Loop Inlet'
  Connectors:
    Air:
      Inlet: outdoor_air_stream_node_name
      Outlet: mixed_air_node_name
```

Note, some build path objects are created with null connectors (i.e. *{ Connectors: { Air: None }}*). These objects are created this way such that they can be specifically manipulated in *expand_objects._modify_build_path_for_equipment()*. The most common reason for this modification is that a build path exists, but it is not directly reflected in the AirLoop system BranchList.

OptionTree Structure

```
OptionTree:
  HVACTemplate:
    Zone: Zone template instructions
    System: System template instructions
    Plant: Plant template instructions
```

OptionTree Template Structure

This section provides a set of instructions for the expansion process.

- BuildPath - Ordered list of objects to create along the fluid flow path.
  - BaseObjects - *BuildPath* Objects that are inserted in all cases
  - Actions - List of actions that are conditionally applied base on template values
    - ObjectReference - Reference to an existing object in the *BuildPath*
    - Occurrence - The number of times to match the ObjectReference in the *BuildPath* before applying the action.
    - Location - Instruction on where to insert the new object(s) in reference to the ObjectReference. 'Before' or 'After' will insert the object to either side. An integer input will specify the global *BuildPath* location.
    - ActionType - Instruction whether to insert a new object, remove the existing object, or replace the existing object.
- BaseObjects - Objects that are created in all cases
- TemplateObjects - Objects that are conditionally created based on template values

```
OptionTree:
  HVACTemplate:
    System:
      BuildPath:
        BaseObjects:
        Actions:
          ObjectReference:
          Location:
```

(continues on next page)

(continued from previous page)

```
        Objects:
    BaseObjects:
    TemplateObjects:
```

OptionTree Object Structure

Once a set of instructions has been selected, the specific information to create the object(s) is provided.

- Objects - 'Super' objects to be inserted in the *BuildPath* or regular objects to be inserted into the epJSON document.
- Transitions - Transfer of template inputs to object values (e.g. supply_fan_total_efficiency -> Fan Object -> Field [fan efficiency])
- Mappings - Mappings of template field values that trigger one or more values to be updated in other template fields. For example, the selection of 'InletVaneDampers' for 'supply_fan_part_load_power_coefficients' creates multiple field name/value pairs to express the fan PLR curve.
- Transitions

  A field value in an object can be created, or overridden. This set of instructions transfers the value provided in an HVACTemplate field to an EnergyPlus object field.

```
- Object Type Reference (can be a regular expression):
    hvac_template_field: object_field

- Fan:.*:
    supply_fan_delta_pressure: design_pressure_rise
```

- Transitions With String Reformatting

  A string reformat may be specified to mutate the input value. For example, if the template value provided for *chilled_water_design_setpoint* is 12.8, then The following code will yield a string value in the schedule_name field of 'HVACTemplate-Always12.8'. When a string is formatted to 'HVACTemplate-Always[numeric]', a Schedule:Compact object is automateically created.

```
Fields: &SetpointManagerScheduledChilledWater
  <<: *SetpointManagerScheduled
  schedule_name: 'HVACTemplate-Always{chilled_water_design_setpoint}'
```

- Transition Using Different Template Field

  In addition to string reformatting, a separate template field may be updated by specifying the value as another dictionary object. In this example, if a value is given for dehumidification_setpoint then dehumidifying_relative_humidity_setpoint_schedule_name is updated.

```
- ZoneControl:Humidistat:
    dehumidification_setpoint:
      dehumidifying_relative_humidity_setpoint_schedule_name: 'HVACTemplate-Always
↪{dehumidification_setpoint}'
```

- Transition Including Numerical Operations

  Numerical and other mathematical operations that can performed using the eval() function in Python can be used. In this example, a maximum value is returned between a static number and a template field.

```
- SetpointManager:Warmest:
    maximum_setpoint_temperature: 'max(18, {cooling_coil_design_setpoint}+5.2)'
```

- Mappings

  A set of field objects can be created or overridden base on one template input. The mapped values can be statically typed. In this example 'None' means that no tempalte input was provided.

```
- Object Type Reference (can be a regular expression):
    hvac_template_field:
      hvac_template_value:
        object_field: object_value

- Fan:.*:
    supply_fan_part_load_power_coefficients:
      None:
        fan_power_coefficient_1: 0.0015302446
        fan_power_coefficient_2: 0.0052080574
        fan_power_coefficient_3: 1.1086242
        fan_power_coefficient_4: -0.11635563
        fan_power_coefficient_5: 0
        fan_power_minimum_flow_rate_input_method: Fraction
        fan_power_minimum_flow_fraction: 0
```

Full Structure Example:

```
OptionTree:
  HVACTemplate:
    System:
      DedicatedOutdoorAir:
        BuildPath:
          BaseObjects:
          Actions:
            - ...
            - supply_fan_placement:
                None: &SystemCommonBuildPathActionsSupplyFanPlacementNone
                  Location: -1
                  ActionType: Insert
                  Objects:
                    - Fan:VariableVolume:
                        Fields:
                          << : *FanFields
                          maximum_flow_rate: Autosize
                      Connectors: *FanConnectors
        pressure_rise: 1000
                  Transitions: *FanCommonTransitionsSupply
                  Mappings: *FanCommonMappingsDedicatedOutdoorAir
                DrawThrough: *SystemCommonBuildPathActionsSupplyFanPlacementNone
                BlowThrough:
                  ObjectReference: OutdoorAir:Mixer
                  ActionType: Insert
                  Location: After
                  Objects:
                    - Fan:VariableVolume: *FanVariableVolumeSuperObject
                  Transitions: *FanCommonTransitionsSupply
                  Mappings: *FanCommonMappingsDedicatedOutdoorAir
      BaseObjects:
        Objects:
          - ...
          - Sizing:System:
              ...
```

(continues on next page)

(continued from previous page)

```
              100_outdoor_air_in_cooling: 'Yes'
              100_outdoor_air_in_heating: 'Yes'
              central_cooling_design_supply_air_humidity_ratio: 0.00924
              central_heating_design_supply_air_humidity_ratio: 0.003
              type_of_load_to_size_on: VentilationRequirement
              preheat_design_temperature: 2
        Transitions:
          - Sizing:System:
              ...
              cooling_coil_design_setpoint_temperature: central_cooling_design_supply_
↪air_temperature
              cooling_coil_design_setpoint: central_cooling_design_supply_air_
↪temperature
```

Complex References

References to object field values may take multiple forms. This feature is intended to provide greater flexibility for object definition and to link nodes without relying on static text fields. References may be specified as follows:

- **Static value: numeric or string.** Use a directly typed value. Note, the *.format()* method is applied to all string values. This allows template input values to be directly inserted into a string by simply applying the name within brackets '{}'. A set of blank brackets will have the unique template name inserte into that space.

  For a given HVACTemplate Object:

  ```
  "HVACTemplate:System:DedicatedOutdoorAir": {
    "DOAS": {
      "gas_heating_coil_efficiency": 0.8
    }
  }
  ```

  The inputs can be applied to the object in the YAML file

  ```
  Fields: &CoilHeatingFuelFields
    <<: *CoilHeatingCommonFields
    burner_efficiency: '{gas_heating_coil_efficiency}'
  ```

- ***BuildPath* Location References: string or integer** Reference a node by it's location in the *BuildPath*. This is only useful for System and Zone templates, not Plant or Loop templates.

  – Location: When given a string, a regular expression match is made to find an EnergyPlus object type in the *BuildPath*. When given an integer, the list index of the object in the *BuildPath* is used.

  – Occurrence: The number of matches to make before returning an object. A value of -1 will return the last matched object.

  – ValueLocation:

    ∗ self - The EnergyPlus object type

    ∗ key - The EnergyPlus unique object name

    ∗ Inlet - The connector inlet

    ∗ Outlet - The connector outlet

  ```
  AirLoopHVAC:
    supply_side_outlet_node_names:
      BuildPathReference:
  ```

(continues on next page)

(continued from previous page)

```
    Location: -1
    ValueLocation: Outlet
```

- General Reference Value Reference an epJSON object in the document. This is a key/value pair given to retrieve an EnergyPlus object. Note, only objects that are created within the scope of the given template can be referenced. The 'self' and 'key' options noted above are available as well.

```
Branch:
  name: '{} ChW Branch'
  components:
    - component_object_type:
        Chiller.*: self
      component_name:
        Chiller.*: key
      component_inlet_node_name:
        Chiller.*: chilled_water_inlet_node_name
      component_outlet_node_name:
        Chiller.*: chilled_water_outlet_node_name
```

1.1.4 Command Line Interface

-f, –file FILE_NAME : Specify file to expand

This argument may be omitted. A value passed to the program with no argument will be assumed to be a file name.

-h, –help : Display help information

-nb, –no_backup : Do no create backup files

It is not possible to comment sections of code in JSON formatted files. Therefore, the output expanded files do not have the ability to retain the HVACTemplate objects used to create the current document. If the original file were to be overwritten, then all template data would be lost. In an attempt to provide and additional layer of backups, the -nb option is set to False by default which means two files will be created: one with HVACTemplate objects, and one with all other objects. With these files, the original input file can be created, or specific objects can be copied and pasted.

-ns, –no_schema : Skip schema validation checks for pyExpandObjects. Note, this does not skip other schema validation operations within EnergyPlus itself.

One benefit of the JSON file format is that files can be validated before simulation. This means that erroneous inputs can be found before simulation, which saves time debugging output files and reading through logs, unsure of the error source. This includes syntax errors, values that are out of range, and missing required inputs. However, situations may occur when the user wishes to skip schema validation, in which case this flag should be used. By default, schema validation is enabled.

-o, –output_directory : Specify output directory. If not provided, then input file directory is used.

-l, –logger_level LOGGER_LEVEL: Set logging output level

Various levels of logging output are available for debugging, and other, purposes. A valid level, consistent with Python logging naming structure (i.e. DEBUG, INFO, WARNING, ERROR, CRITICAL), must be provided.

-v, –version : Display version information

-wl, –write-logs : Write logs to file

When this expansion tool is run from its source directory, the output can be written to a file, which is located in the logs directory (logs/base.log).

1.2 Logger

class `logger.`**`Logger`**(*logging_file_name='logging.conf', logger_name='console_only_logger', log_file_name='base', logger_level='WARNING', reset_stream=False*)

Bases: `object`

General logger

1.3 EPJSON

class `epjson_handler.`**`EPJSON`**(*no_schema=False, logger_level='WARNING', logger_name='console_only_logger', reset_stream=False*)

Bases: `logger.Logger`

Handle epJSON and JSON specific tasks

Attributes: Validator: schema validator from jsonschema

schema: loaded schema. Only validated schemas will be loaded.

input_epjson: input epjson file

schema_is_valid: initialized as None. False if failed, True if passed.

input_epjson_is_valid: initialized as None. False if failed, True if passed.

static epjson_genexp(*epjson*)

Create generator of individual epJSON objects in epJSON format from a dictionary of objects in epJSON format.

{object_type: {object_name: object_fields}, {…}} -> {object_type: {object_name: object_fields}}, {…}

Parameters **`epjson`** – epJSON object

Returns generator which returns one unique object in epJSON format for each object in an object_type.

epjson_process(*epjson_ref*)

Default loading and verification of epJSON file :param epjson_ref: epJSON in dictionary format or file location. :return: initialized class attributes and input_epJSON object

get_epjson_objects(*epjson: dict, object_type_regexp: str = '.*', object_name_regexp: str = '.*'*) → dict

Get objects from epJSON dictionary after filtering by object type and name.

Parameters

- **`epjson`** – epJSON formatted Dictionary to scan
- **`object_type_regexp`** – regular expression to match with object type
- **`object_name_regexp`** – regular expression to match with object_name

Returns epJSON dictionary of matched objects.

static merge_epjson(*super_dictionary: dict, object_dictionary: dict, unique_name_override: bool = False, unique_name_fail: bool = True*)

Merge a high level formatted dictionary with a sub-dictionary, both in epJSON format

Parameters

- **`super_dictionary`** – high level dictionary used as the base object

- **`object_dictionary`** – dictionary to merge into base object
- **`unique_name_override`** – allow a duplicate unique name to overwrite an existing object
- **`unique_name_fail`** – if override is set to False, choose whether to skip object or fail

Returns merged output of the two input dictionaries. Note, the super_dictionary is modified in this operation. A copy operation was not performed intentionally. If the user wants the original super_dictionary to remain unchanged then a copy.deepcopy() should be performed before running the function.

static purge_epjson(*epjson*, *purge_dictionary=None*)

Remove objects in an input epJSON object. :param epjson: input epJSON :param purge_dictionary: key-value pair of object_type and list of regular expressions to remove items

(.* removes all objects)

Returns epJSON with items referenced in purge_dictionary removed.

static summarize_epjson(*epjson*)

Provide summary of epJSON dictionary for comparisons and metrics.

Parameters **`epjson`** – epJSON formatted dictionary

Returns dictionary of count summaries

validate_epjson(*epjson*)

Validate json object as epJSON. Return object if valid

Parameters **`epjson`** – epJSON object

Returns validated epJSON object

1.4 HVAC Template

class `hvac_template.`**HVACTemplate**(*no_schema=False*, *logger_level='WARNING'*, *logger_name='console_only_logger'*, *reset_stream=True*)

Bases: `epjson_handler.EPJSON`

Handle HVACTemplate conversion process and connect created objects together.

Attributes: templates: HVACTemplate objects from epJSON file

base_objects: Non-HVACTemplate objects from epJSON file

templates_zones: HVACTemplate:Zone: objects

templates_systems: HVACTemplate:System: objects

templates_plant_equipment: HVACTemplate:Plant equipment objects

templates_plant_loops: HVACTemplate:Plant: loop objects

expanded_*: List of class objects for each template type

epjson: epJSON used to store connection objects

run(*input_epjson=None*)

Execute HVAC Template process workflow

Parameters **`input_epjson`** – input epJSON file

Returns epJSON containing expanded objects from templates

1.5 ExpandObjects

class expand_objects.**ExpandObjects**(*template=None*, *expansion_structure='/home/docs/checkouts/readthedocs.org/user expandobjects/checkouts/stable/src/resources/template_expansion_structures.yaml'* *logger_level='WARNING'*, *logger_name='console_only_logger'*)

Bases: `epjson_handler.EPJSON`

Class to contain general expansion functions as well as methods to connect template outputs.

Attributes: expansion_structure: file or dictionary of expansion structure details (from YAML)

template: epJSON dictionary containing HVACTemplate to expand

template_type: HVACTemplate object type

template_name: HVACTemplate unique name

epjson: dictionary of epSJON objects to write to file

unique_name: unique string used to modify to epJSON object names within the class

HVACTemplate fields are stored as class attributes

_apply_build_path_action(*build_path*, *action_instructions*)

Mutate a build path list based on a set of action instructions

Parameters

- **build_path** – Input build path list
- **action_instructions** – Formatted instructions to apply an action. Valid actions are 'Insert', 'Remove', and 'Replace' (case insensitive).

Returns mutated dictionary with action applied.

_apply_transitions(*option_tree_leaf: dict*) → list

Set object field values in an OptionTree leaf, which consist of a 'Objects', 'Transitions', and 'Mappings' keys using a supplied Transitions dictionary.

Transitions translates template input values to fields in Objects. A more direct method of transitioning template inputs to object values can be done using field name formatting (e.g. field_name: {template_field}) in the yaml file. This method of application is used to override default values if template inputs are provided.

Mappings maps values from templates to objects. This is necessary when the template input is not a direct transition to an object value.

Parameters **option_tree_leaf** – YAML loaded option tree end node with three keys: objects, transitions, mappings

Returns list of dictionary objects with transitions and mappings applied

_connect_and_convert_build_path_to_object_list(*build_path=None*, *loop_type='AirLoop'*)

Connect nodes in build path and convert to list of epJSON formatted objects

Parameters

- **build_path** – build path of EnergyPlus super objects
- **loop_type** – fluid flow path (only AirLoop available)

Returns object list of modified super objects. The build path is also saved as a class attribute for future reference

`_create_availability_manager_assignment_list`(*epjson: Optional[dict] = None*) → dict

Create AvailabilityManagerAssignmentList object from epJSON dictionary This list object is separated from the OptionTree build operations because it will vary based on the presence of AvailabilityManager:.* objects in the epJSON dictionary. Therefore, the list objects must be built afterwards in order to retrieve the referenced objects.

Parameters **`epjson`** – system epJSON formatted dictionary

Returns epJSON formatted AirLoopHVAC:ControllerList objects. These objects are also stored back to the input epJSON object.

`_create_objects`(*epjson=None*)

Create a set of EnergyPlus objects for a given template

Parameters **`epjson`** – epJSON object to use for merge and reference.

Returns epJSON dictionary of newly created objects. The input epJSON dictionary is also modified to include the newly created objects

`_flatten_list`(*nested_list: list*, *flat_list: list = []*, *clear: bool = True*) → list

Flattens list of lists to one list of items.

Parameters

- **`nested_list`** – list of nested dictionary objects
- **`flat_list`** – list used to store recursive addition of objects
- **`clear`** – Option to empty the recursive list

Returns flattened list of objects

`_get_option_tree`(*structure_hierarchy: list*) → dict

Retrieve structure from YAML loaded object and verify it is correctly formatted for an option tree :param structure_hierarchy: list representing structure hierarchy :return: structured object as dictionary

`_get_option_tree_leaf`(*option_tree: dict*, *leaf_path: list*) → dict

Return leaf from OptionTree with alternative options formatted in dictionary

Parameters

- **`option_tree`** – Yaml object holding HVACTemplate option tree
- **`leaf_path`** – path to leaf node of option tree

Returns Formatted dictionary with objects and alternative options to be applied.

`_get_option_tree_objects`(*structure_hierarchy: list*) → dict

Return objects from option tree leaves.

Parameters **`structure_hierarchy`** – list representing structure hierarchy

Returns epJSON dictionary with unresolved complex inputs

`_parse_build_path`(*object_list*, *epjson=None*)

Iterate over build path and check if each object is a super object. If not, commit the object to the input epJSON. Return a build path of only super objects.

Parameters

- **`object_list`** – BuildPath list of objects which may contain mixed combination of regular objects and super objects
- **`epjson`** – input epJSON dictionary

Returns build path of only super objects

`_process_build_path`(*option_tree*)
Create a connected group of objects from the BuildPath branch in the OptionTree. A build path is a list of 'super objects' which have an extra layer of structure. These keys are 'Fields' and 'Connectors'. The Fields are regular EnergyPlus field name-value pairs. The connectors are structured dictionaries that provide information on how one object should connect the previous/next object in the build path list. A branch of connected objects is also produced.

Parameters **`option_tree`** – OptionTree to use for build instructions

Returns list of EnergyPlus super objects. Additional EnergyPlus objects (Branch, Branchlist) that require the build path for their creation.

`_resolve_complex_input`(*field_name: str*, *input_value: Union[str, int, float, dict, list]*, *epjson: Optional[dict] = None*, *build_path: Optional[list] = None*, *recursions: int = 0*) → Generator[str, Dict[str, str], None]
Resolve a complex input reference into a field value

Parameters

- **`field_name`** – object field name
- **`input_value`** – field value input
- **`epjson`** – epJSON dictionary of objects
- **`build_path`** – BuildPath to reference for lookup
- **`recursions`** – cumulative count of recursive calls so that a cap can be made. This is a backup in case RecursionError misses an infinite loop.

Returns resolved field value

`_resolve_complex_input_from_build_path`(*build_path: list*, *lookup_instructions: dict*, *connector_path: str = 'AirLoop'*) → str
Resolve a complex input using a build path and location based instructions.

Parameters

- **`build_path`** – list of EnergyPlus super objects forming a build path
- **`lookup_instructions`** – instructions identifying the node location to return
- **`connector_path`** – fluid flow type path (AirLoop only available)

Returns Resolved field value

`build_compact_schedule`(*structure_hierarchy: list*, *insert_values: list*, *name: Optional[str] = None*) → dict
Build a Schedule:Compact schedule from inputs. Save epjJSON object to class dictionary and return to calling function.

Parameters

- **`structure_hierarchy`** – list indicating YAML structure hierarchy
- **`insert_values`** – list of values to insert into object
- **`name`** – (optional) name of object.

Returns epJSON object of compact schedule

`expansion_structure`
Verify expansion structure file location or object

get_structure(*structure_hierarchy: list*, *structure=None*) → dict

Retrieve structure from YAML loaded object. When retrieving TemplateObjects, the last item in the hierarchy will be matched via regex instead of a direct key call for TemplateObjects. This is done to allow for multiple template options in a single mapping.

Parameters

- **structure_hierarchy** – list representing structure hierarchy
- **structure** – YAML loaded dictionary, default is loaded yaml loaded object

Returns structured object as dictionary

rename_attribute(*old_attribute*, *new_attribute*)

Change attribute name to commonize variables. Do not perform the operation if the target attribute already has a value.

Parameters

- **old_attribute** – old attribute name
- **new_attribute** – new attribute name

Returns None. Old attribute is deleted and new attribute created in class

resolve_objects(*epjson*, *reference_epjson=None*)

Resolve complex inputs in epJSON formatted dictionary

Parameters

- **epjson** – epJSON dictionary with complex inputs
- **reference_epjson** – (optional) epJSON dictionary to be used as reference objects for complex lookups. If None, then the input epjson will be used

Returns epJSON dictionary with values replacing complex inputs

template

Verify if template dictionary is a valid type and structure

yaml_list_to_epjson_dictionaries(*yaml_list: list*) → dict

Convert list of YAML dictionaries into epJSON formatted dictionaries.

YAML dictionaries can either be regular or ‘super’ objects which contain ‘Fields’ and ‘Connectors’ :param yaml_list: list of yaml objects to be formatted. :return: epJSON formatted dictionary

1.6 ExpandThermostat

class expand_objects.**ExpandThermostat**(*template*, *logger_level='WARNING'*, *logger_name='console_only_logger'*, *epjson=None*)

Bases: `expand_objects.ExpandObjects`

Thermostat expansion operations

_create_and_set_schedules()

Create, or use existing, schedules. Assign schedule names to class variable

Returns Cooling and/or Heating schedule variables as class attributes

_create_thermostat_setpoints()

Create ThermostatSetpoint objects based on class setpoint_schedule_name attributes :return: Updated class epJSON dictionary with ThermostatSetpoint objects added.

run()
Perform all template expansion operations and return the class to the parent calling function. :return: Class object with epJSON dictionary as class attribute

1.7 ExpandZone

class expand_objects.**ExpandZone**(*template*, *logger_level='WARNING'*, *logger_name='console_only_logger'*, *epjson=None*, *system_class_objects=None*)
Bases: `expand_objects.ExpandObjects`

Zone expansion operations

Attributes from Descriptors:

system_transitions

zone_hvac_equipmentlist_object_type

design_specification_outdoor_air_object_status

design_specification_zone_air_distribution_object_status

heating_design_air_flow_method

cooling_design_air_flow_method

humidistat_object_type

fan_powered_reheat_type

cooling_design_air_flow_method
Set a class attribute to set cooling_design_air_flow_method for transitions in the YAML lookup. If the supply_air_maximum_flow_rate has been set to a value, then the cooling_design_air_flow_method field in Sizing:Zone should be 'Flow/Zone' with this value as the flow rate.

design_specification_outdoor_air_object_status
Set a class attribute to select the appropriate DesignSpecification:OutdoorAir from TemplateObjects in the YAML lookup.

design_specification_zone_air_distribution_object_status
Set a class attribute to select the appropriate DesignSpecification:ZoneAirDistribution from TemplateObjects in the YAML lookup.

fan_powered_reheat_type
Create a new class attribute from reheat_coil_type, for TemplateObjects in the YAML lookup for Fan-Powered zone objects. If the system flow_type is parallel then reheat_coil_type_parallel is set. For series, reheat_coil_type_series is set.

heating_design_air_flow_method
Set a class attribute to set heating_design_air_flow_method for transitions in the YAML lookup. If the supply_air_maximum_flow_rate has been set to a value, then the heating_design_air_flow_method field in Sizing:Zone should be 'Flow/Zone' with this value as the flow rate.

humidistat_object_type
Set a class attribute, humidistat_object_type, for TemplateObjects in the YAML lookup. Also set the dehumidification and humidification values to a default if not provided

run()
Process zone template :return: Class object with epJSON dictionary as class attribute

`system_transitions`
Set zone attributes based on system attributes

`vrf_type`
Create a new class attribute, vrf_type, for TemplateObjects in the YAML lookup for FanPowered zone objects.

`zone_hvac_equipmentlist_object_type`
Set a class attribute to select the appropriate ZoneHVAC:EquipmentList from TemplateObjects in the YAML lookup.

1.8 ExpandSystem

class `expand_objects.`**`ExpandSystem`**(*template*, *logger_level='WARNING'*, *logger_name='console_only_logger'*, *epjson=None*)
Bases: `expand_objects.ExpandObjects`

System expansion operations

Attributes from Descriptors: airloop_hvac_unitary_object_type

airloop_hvac_object_type

airloop_hvac_unitary_fan_type_and_placement

economizer_type_detailed

mixed_air_setpoint_control_type_detailed

cooling_coil_setpoint_control_type_detailed

heating_coil_setpoint_control_type_detailed

setpoint_control_type: concatenated string of cooling_coil_setpoint_control_type and heating_coil_setpoint_control_type

humidistat_type

outside_air_equipment_type

dehumidification_control_type

dehumidification_control_type_detailed

`_create_branch_and_branchlist_from_build_path`(*build_path: Optional[list] = None*, *loop_type: str = 'AirLoop'*, *epjson: Optional[dict] = None*, *include_branchlist: bool = True*, *modify_build_path: bool = True*)
Create Branch and BranchList objects from system build path These objects are separated from the OptionTree build operations because they vary based on the final build path. Also, the Branch object must reference an AirLoopHVAC:OutdoorAirSystem object that is also built after the OptionTree build operations.

Parameters

- **`build_path`** – system build path
- **`loop_type`** – string descriptor of the loop type, AirLoop by default
- **`epjson`** – epJSON dictionary
- **`include_branchlist`** – boolean flag to create branchlist or not

- **modify_build_path** – boolean flag to modify build path for outdoor air system

Returns epJSON formatted Branch and BranchList objects. These objects are also stored back to the input epJSON object.

_create_controller_list_from_epjson(*epjson: Optional[dict] = None*, *build_path: Optional[list] = None*, *controller_list: tuple = ('Controller:WaterCoil', 'Controller:OutdoorAir')*) → dict

Create AirLoopHVAC:ControllerList objects from epJSON dictionary These list objects are separated from the OptionTree build operations because they will vary based on the presence of Controller:WaterCoil and Controller:OutdoorAir objects in the epJSON dictionary. Therefore, the list objects must be built afterwards in order to retrieve the referenced Controller:.* objects.

Parameters

- **epjson** – system epJSON formatted dictionary
- **build_path** – List of epJSON super objects in build path order
- **controller_list** – List of controllers to be included in search

Returns epJSON formatted AirLoopHVAC:ControllerList objects. These objects are also stored back to the input epJSON object.

_create_outdoor_air_equipment_list_from_build_path(*build_path: Optional[list] = None*, *epjson: Optional[dict] = None*) → dict

Create AirLoopHVAC:OutdoorAirSystem:EquipmentList objects from system build path This object is separated from the OptionTree build operations because it varies based on the final build path. Therefore, this object must be built afterwards in order to retrieve the referenced outdoor air equipment objects.

Parameters

- **build_path** – system build path
- **epjson** – epJSON dictionary

Returns epJSON formatted AirLoopHVAC:OutdoorAirSystem:EquipmentList objects. These objects are also stored back to the input epJSON object.

_create_outdoor_air_system(*epjson: Optional[dict] = None*) → dict

Create AirLoopHVAC:OutdoorAirSystem object from epJSON dictionary This list object is separated from the OptionTree build operations because it must seek out the correct ControllerList containing the OutdoorAir:Mixer object.

Parameters **epjson** – system epJSON formatted dictionary

Returns epJSON formatted AirLoopHVAC:OutdoorAirSystem objects. These objects are also stored back to the input epJSON object.

_dual_duct_custom_edits()

Perform customized edits to typical ExpandSystem process for HVACTemplate:DualDuct :return: None. class attributes modified

_modify_build_path_for_equipment(*loop_type: str = 'AirLoop'*, *build_path: Optional[list] = None*) → list

Modify input build path to use special equipment objects in the build path, rather than individual components. This function loads a structured tuple that contains a list of instructions:

0 - HVACTemplate:System for which the instructions are applied 1 - Regular expression match of the super object to be manipulated. This object has a special format to

prevent it from being connected in earlier steps {Fields: ..., Connectors: {AirLoop: False}}

2 - Connectors to be applied to the object 3 - Tuple of objects to be removed from build path.

Parameters `build_path` – list of EnergyPlus Super objects

Returns build path

`_modify_build_path_for_outside_air_system`(*loop_type: str = 'AirLoop'*, *epjson: Optional[dict] = None*, *build_path: Optional[list] = None*) → list

Modify input build path to use AirLoopHVAC:OutdoorAirSystem as the first component, rather than individual outside air system components

Parameters `build_path` – list of EnergyPlus Super objects

Returns build path

`airloop_hvac_object_type`

Set a class attribute to select the appropriate AirLoopHVAC type from TemplateObjects in the YAML lookup.

`airloop_hvac_unitary_fan_type_and_placement`

Set a class attribute to select the appropriate fan type and placement from TemplateObjects in the YAML lookup.

`airloop_hvac_unitary_object_type`

Set a class attribute to select the appropriate unitary equipment type from TemplateObjects in the YAML lookup.

`cooling_coil_setpoint_control_type_detailed`

set cooling_coil_setpoint_control_type_detailed based on other template attributes for YAML TemplateObjects lookup.

`dehumidification_control_type`

Modify dehumidification_control_type based on other template attributes for YAML TemplateObjects lookup.

`dehumidification_control_type_detailed`

Create attribute dehumidification_control_type_detailed based on other template attributes for YAML TemplateObjects lookup.

`economizer_type_detailed`

set economizer_type_detailed based on other template attributes for YAML TemplateObjects lookup.

`heating_coil_setpoint_control_type_detailed`

create attribute heating_coil_setpoint_control_type_detailed based on other template attributes for YAML TemplateObjects lookup.

`humidistat_type`

Create class attribute humidistat_type to select Humidistat type based on other template attributes for YAML TemplateObjects lookup.

`mixed_air_setpoint_control_type_detailed`

set mixed_air_setpoint_control_type_detailed based on other template attributes for YAML TemplateObjects lookup.

`outside_air_equipment_type`

Create class attribute to select OA objects for YAML BuildPath actions.

`run`()

Process system template :return: class object with epJSON dictionary as class attribute

1.9 ExpandPlantLoop

class `expand_objects.`**`ExpandPlantLoop`**(*template*, *logger_level='WARNING'*, *logger_name='console_only_logger'*, *epjson=None*)

Bases: `expand_objects.ExpandObjects`

Plant loop expansion operations

Attributes from Descriptors: primary_pump_flow_and_type

secondary_pump_flow_and_type

`primary_pump_flow_and_type`

Create class attribute, primary_pump_flow_and_type to select the pump flow type, and configuration type for YAML TemplateObjects lookup.

`run`()

Process plant loop template :return: class object with epJSON dictionary as class attribute

`secondary_pump_flow_and_type`

Create class attribute, secondary_pump_flow_and_type to select the pump flow type, and configuration type for YAML TemplateObjects lookup.

1.10 ExpandPlantEquipment

class `expand_objects.`**`ExpandPlantEquipment`**(*template*, *logger_level='WARNING'*, *logger_name='console_only_logger'*, *plant_loop_class_objects=None*, *epjson=None*)

Bases: `expand_objects.ExpandObjects`

Plant Equipment operations

Attributes from Descriptors: template_plant_loop_type

pump_configuration_type

chiller_and_condenser_type

primary_pump_type

`chiller_and_condenser_type`

Set a class attribute to select the appropriate fan type and placement from TemplateObjects in the YAML lookup.

`primary_pump_type`

Set a class attribute pump_type that reflects the loop attribute primary_pump_type to use in TemplateObjects in the YAML lookup. This descriptor will only either be None, or 'PumpPerEquipment', since those are the only configurations needed to inform the equipment

`pump_configuration_type`

Set a class attribute pump_configuration_type that reflects the loop attribute of the same name to use in TemplateObjects in the YAML lookup.

`run`()

Process plant loop template :return: class object with epJSON dictionary as class attribute

`template_plant_loop_type`

Get and set the a priority list of plant loops for the equipment to serve.

CHAPTER

TWO

TESTING REPORTS

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX