EnergyPlus Transition Documentation

Release 0.95

US DOE

Sep 03, 2017

Contents

EnergyPlus Python Transition is a remake of the Fortran-based EnergyPlus file transition tool. The purpose of this tool is to transition an EnergyPlus input file from one version to the latest version. Because the input forms change between EnergyPlus versions so dramatically, having a tool like this is a mandatory piece of the EnergyPlus workflow.

The previous version, in Fortran, was difficult to maintain, as fewer and fewer Fortran developers remain. In addition, with the possibility of future input syntax changes (JSON), a new version transition tool was desired. This version, written in Python, is more modular in nature, with almost the entire code base written independent of any specific version of EnergyPlus, and only the rules themselves plus 2 other lines needing to be modified for adding another version. The rules themselves are simply derived classes in Python that give clear guidance on writing new rules.

Installation:

Each tagged release of the software is posted to PyPi. With this in place, installation of the library into a given Python installation is easy using pip:

```
pip install eptransition
```

Once this is installed, it will copy the library into Python's appropriate package folder, and also create an executable link to the main transition function, when possible, into the PATH, so that the eptransition script can be called directly from the command line. Usage of these two modes are described below.

Usage from Command Line:

Once installed, in order to execute the program from the command line, simply call the executable link created during installation and pass in the input file(s) to transition:

```
eptransition /path/to/idf /path/to/another/idf
```

Executing this command line will cause the tool to read the input file(s) first to find the start version for each transition process. The tool then checks whether this version is available and if so, reads appropriate dictionary files for the start/end version, processes the IDF, executes all transition rules, and writes out a transitioned input file.

Usage from Library:

Once installed, using from existing Python code is a simple matter. Simply create a new Python script, and start by importing the library:

```python
import eptransition
```

With the library imported, one can access all the underlying model structure, although the most likely usage will be to programmatically transition files. To do this, one can access the manager function directly:

```python
from eptransition.manager import TransitionManager
for idf in ['/path/to/idf', 'path/to/another/idf']:
   tm = TransitionManager("/path/to/idf")
   try:
       tm.perform_transition()
   except Exception as e:
       print(e)
```

This is equivalent to the command line call above.

Class Structure:

CHAPTER 1

Transition Module Documentation

`eptransition.transition.`**`main`**`(`*args=None*`)`

This is the highest level driving function for the transition process. This interprets either sys.argv directly, or a list of arguments that mimic sys.argv. (So that sys.argv can be passed in directly from other wrappers). This function is called from the command line via the pip installation.

Parameters **`args`** – An optional array of arguments, mimicking sys.argv. As such, item 0 must be a dummy program name, followed by real arguments. If this is not passed in, sys.argv is assumed.

Returns 0 on success, 1 for failure

Raises **`Exception`** – If the –raise flag is used, it will raise the underlying Exception at the first failure

CHAPTER 2

Manager Class Documentation

class `eptransition.manager.`**`TransitionManager`**(*original_input_file*)

Bases: `object`

This class is the main manager for performing transition of an input file to the latest version.

Developer note: This class raises many exceptions, so logging.exception is handled at the level of the code calling these functions within a try/except block. These functions do logging, but only the info/debug level.

Parameters **`original_input_file`** (*`str`*) – Full path to the original idf to transition

`perform_transition`()

This function manages the transition from one version to another by opening, validating, and writing files

Returns Final transitioned idf structure; raises exception for failures

Raises

- **`FileAccessException`** – if a specified file does not access
- **`FileTypeException`** – if a specified file type does not match the expected condition
- **`ManagerProcessingException`** – if there is a problem processing the contents of the files

`rvi_mvi_replace`(*original_file_path*, *new_file_path*, *output_rule*)

CHAPTER 3

Exceptions Class Documentation

exception `eptransition.exceptions.`**`FileAccessException`** (*file_path*, *problem_type*, *file_nickname*, *message=None*)

Bases: `exceptions.Exception`

This exception occurs when the transition tool encounters a problem accessing a prescribed input or output file.

Parameters

- **`file_path`** (*`str`*) – The file path which is causing the issue
- **`problem_type`** (*`str`*) – The type of problem occurring, from the constants defined in this class
- **`file_nickname`** (*`str`*) – The nickname of the file, from the constants defined in this class
- **`message`** (*`str`*) – An optional additional message to write out

`CANNOT_FIND_FILE` **= 'cannot find file'**

`CANNOT_READ_FILE` **= 'cannot read file'**

`CANNOT_WRITE_TO_FILE` **= 'cannot write to file'**

`FILE_EXISTS_MUST_DELETE` **= 'file exists, must delete'**

`ORIGINAL_DICT_FILE` **= 'original dictionary file'**

`ORIGINAL_INPUT_FILE` **= 'original input file'**

`TRIED_BUT_CANNOT_DELETE_FILE` **= "tried to delete file, but couldn't"**

`UPDATED_DICT_FILE` **= 'updated dictionary file'**

`UPDATED_INPUT_FILE` **= 'updated input file'**

exception `eptransition.exceptions.`**`FileTypeException`** (*file_path*, *file_nickname*, *message*)

Bases: `exceptions.Exception`

This exception occurs when the prescribed file types do not match the expected conditions.

ORIGINAL_DICT_FILE = 'original dictionary file'

ORIGINAL_INPUT_FILE = 'original input file'

UPDATED_DICT_FILE = 'updated dictionary file'

UPDATED_INPUT_FILE = 'updated input file'

exception `eptransition.exceptions.`**`ManagerProcessingException`**(*msg*, *issues=None*)

Bases: `exceptions.Exception`

This exception occurs when the transition tool encounters an unexpected issue when doing the transition.

exception `eptransition.exceptions.`**`ProcessingException`**(*message*, *line_index=None*, *object_name=''*, *field_name=''*)

Bases: `exceptions.Exception`

This exception occurs when an unexpected error occurs during the processing of an input file.

exception `eptransition.exceptions.`**`UnimplementedMethodException`**(*class_name*, *method_name*)

Bases: `exceptions.Exception`

This exception occurs when a call is made to a function that should be implemented in a derived class but is not, so the base class function is called. This is a developer issue.

Parameters

- **`class_name`** (*`str`*) – The name of the base class where the virtual function is defined
- **`method_name`** (*`str`*) – The method name which should be overridden in the derived class

CHAPTER 4

Versions Module Documentation

class `eptransition.versions.versions.`**`SingleTransition`**(*start_version*, *end_version*, *transitions*, *outputs*, *global_swap*)

Bases: `object`

Internal version information class

Parameters

- **`start_version`** (*`float`*) – The major.minor floating point version identifier for the start version of this transition
- **`end_version`** (*`float`*) – The major.minor floating point version identifier for the end version of this transition
- **`transitions`** (*`[TransitionRule]`*) – A list of class names that derive from TransitionRule as implemented for this version
- **`outputs`** (*`OutputVariableTransitionRule_or_None`*) – Name of a class that derives from OutputVariableTransitionRule, as implemented for this version
- **`global_swap`** (*`dict_or_None`*) – A dictionary of string:string that are used to globally search and replace within the idf prior to actual transition

Raises **`ManagerProcessingException`** – for any invalid inputs

class `eptransition.versions.versions.`**`TypeEnum`**

Bases: `object`

Simple enumeration style class laying out the possible file types available

`IDF` **= 'idf'**

`JSON` **= 'json'**

CHAPTER 5

IDD Object Module Documentation

class `eptransition.idd.objects.`**`IDDField`**(*an_index*)

A simple class that defines a single field for an IDD object. Relevant members are listed here:

Variables

- **`field_an_index`** (*`str`*) – Main identifier for this field
- **`meta_data`** (*`dict(str, [str])`*) – A dictionary, where each key is a string metadata type, such as "note", and each value is a list of strings for each entry in the metadata of the key type. So if the field has 3 note lines, the dictionary value for key "note" would be a 3 element list, holding the 3 note lines.
- **`field_name`** (*`str`*) – A convenience variable holding the field name, if it is found in the metadata

Constructor parameters:

Parameters **`an_index`** (*`str`*) – The Ai or Ni descriptor for this field in the IDD, where i is an integer 1-...

class `eptransition.idd.objects.`**`IDDGroup`**(*name*)

A simple class that defines a single IDD group. An IDD group is simply a container for IDD objects. Relevant members are listed here:

Variables

- **`name`** (*`str`*) – IDD Type, or name, of this group
- **`objects`** (*`list(IDDObject)`*) – A list of all objects found in the IDD within this group.

Constructor parameters:

Parameters **`name`** (*`str`*) – The group's name

class `eptransition.idd.objects.`**`IDDObject`**(*name*)

A simple class that defines a single IDD object. Relevant members are listed here:

Variables

- **`name`** (*`str`*) – IDD Type, or name, of this object

- **`meta_data`** (*`dict(str, [str])`*) – A dictionary, where each key is a string metadata type, such as "memo", and each value is a list of strings for each entry in the metadata of the key type. So if the object has 3 memo lines, the dictionary value for key "memo" would be a 3 element list, holding the 3 memo lines.
- **`fields`** (*`list(IDDField)`*) – A list of IDDField instances in order as read from the IDD

Constructor parameters:

Parameters **`name`** (*`str`*) – The object's type, or name

class `eptransition.idd.objects.`**`IDDStructure`**(*file_path*)

An IDD structure representation. This includes containing all the IDD objects (either inside groups or as standalone "single line objects"), as well as meta data such as the version ID for this IDD, and finally providing worker functions for accessing the IDD data

Relevant "public" members are listed here:

Variables

- **`file_path`** (*`str`*) – The path given when instantiating this IDD, not necessarily an actual path
- **`version_float`** (*`float`*) – The floating point representation of the version of this IDD (for 8.6.0 it is 8.6)
- **`build_string`** (*`str`*) – The abbreviated git SHA used when generating this IDD
- **`single_line_objects`** (*`[str]`*) – A list of strings, each representing a raw, single-token, name-only IDD object
- **`groups`** (*`list(IDDGroup)`*) – A list of all groups found in the IDD, each of which will contain IDD objects

Constructor parameters:

Parameters **`file_path`** (*`str`*) – A file path for this IDD; not necessarily a valid path as it is never used, just available for bookkeeping purposes.

`get_object_by_type`(*type_to_get*)

Given a type name, this returns the IDD object instance, or a single string if it is a single-line object

Parameters **`type_to_get`** – The name of the object to get, case-insensitive as it is compared insensitively inside

Returns If the object is a single-line object, simply the name; if the object is a full IDDObject instance, that instance is returned. If a match is not found, this returns None.

`get_objects_with_meta_data`(*meta_data*)

Given an object-level metadata string (required-object, e.g.), this returns objects that contain that metadata

Parameters **`meta_data`** – An object-level metadata string, such as required-object

Returns A list of IDDObjects that contain this metadata

CHAPTER 6

IDD Processor Module Documentation

class `eptransition.idd.processor.`**`CurrentReadType`**

Internal class containing constants for the different states of the actual IDD Processor engine

`EncounteredComment_ReadToCR` **= 0**

`LookingForFieldMetaDataOrNextField` **= 11**

`LookingForFieldMetaDataOrNextObject` **= 10**

`LookingForObjectMetaDataOrNextField` **= 4**

`ReadAnything` **= 1**

`ReadingFieldANValue` **= 7**

`ReadingFieldMetaData` **= 8**

`ReadingFieldMetaDataOrNextANValue` **= 9**

`ReadingGroupDeclaration` **= 2**

`ReadingObjectMetaData` **= 5**

`ReadingObjectMetaDataContents` **= 6**

`ReadingObjectName` **= 3**

class `eptransition.idd.processor.`**`IDDProcessor`**

The core IDD Processor class. Given an IDD via stream or path, this class has workers to robustly process the IDD into a rich IDDStructure instance.

The constructor takes no arguments but sets up instance variables. Relevant "public" members are listed here:

Variables

- **`idd`** (*`IDDStructure`*) – The resulting IDDStructure instance after processing the IDD file/stream
- **`file_path`** (*`str`*) – A file path for this IDD, although it may be just a simple descriptor

`peek_one_char`()

Internal worker function that reads a single character from the internal IDD stream but resets the stream to the former position

Returns A single character, the one immediately following the cursor, or None if it can't peek ahead.

`process_file`()

Internal worker function that reads the IDD stream, whether it was constructed from a file path, stream or string. This state machine worker moves character by character reading tokens and processing them into a meaningful IDD structure.

Returns An IDD structure describing the IDD contents

Raises **`ProcessingException`** – for any erroneous conditions encountered during processing

`process_file_given_file_path`(*file_path*)

This worker allows processing of an IDD file at a specific path on disk.

Parameters **`file_path`** – The path to an IDD file on disk.

Returns An IDDStructure instance created from processing the IDD file

Raises **`ProcessingException`** – if the specified file does not exist

`process_file_via_stream`(*idd_file_stream*)

This worker allows processing of an IDD snippet via stream. Most useful for unit testing, but possibly for other situations.

Parameters **`idd_file_stream`** (*`file-like-object`*) – An IDD snippet that responds to typical file-like commands such as read(). A common object would be the StringIO object.

Returns An IDDStructure instance created from processing the IDD snippet

`process_file_via_string`(*idd_string*)

This worker allows processing of an IDD snippet string. Most useful for unit testing, but possibly for other situations.

Parameters **`idd_string`** (*`str`*) – An IDD snippet string

Returns An IDDStructure instance created from processing the IDD string

`read_one_char`()

Internal worker function that reads a single character from the internal IDD stream, advancing the cursor.

Returns A single character, the one immediately following the cursor, or None if it can't read.

CHAPTER 7

IDF Object Module Documentation

class `eptransition.idf.objects.`**`IDFObject`**(*tokens*, *comment_blob=False*)

Bases: `object`

This class defines a single IDF object. An IDF object is either a comma/semicolon delimited list of actual object data, or a block of line delimited comments. Blocks of comment lines are treated as IDF objects so they can be intelligently written back out to a new IDF file after transition in the same location.

Relevant members are listed here:

Variables

- **`object_name`** (*`str`*) – IDD Type, or name, of this object
- **`fields`** (*`[str]`*) – A list of strings, one per field, found for this object in the IDF file

Constructor parameters:

Parameters

- **`tokens`** (*`[str]`*) – A list of tokens defining this idf object, the first token in the list is the object type.
- **`comment_blob`** (*`bool`*) – A signal that this list is comment data, and not an actual IDF object; default is False. indicating it is meaningful IDF data.

`object_string`(*idd_object=None*)

This function creates an intelligently formed IDF object. If the current instance is comment data, it simply writes the comment block out, line delimited, otherwise it writes out proper IDF syntax. If the matching IDD object is passed in as an argument, the field names are matched from that to create a properly commented IDF object.

Parameters **`idd_object`** (*`IDDObject`*) – The IDDObject structure that matches this IDFObject

Returns A string representation of the IDF object or comment block

`validate`(*idd_object*)

This function validates the current IDF object instance against standard IDD field tags such as minimum and maximum, etc.

Parameters **`idd_object`** (*`IDDObject`*) – The IDDObject structure that matches this IDFObject

Returns A list of ValidationIssue instances, each describing an issue encountered

`write_object` (*file_object*)
This function simply writes out the idf string to a file object

Parameters **`file_object`** – A file-type object that responds to a write command

Returns None

class `eptransition.idf.objects.`**`IDFStructure`** (*file_path*)
Bases: `object`

An IDF structure representation. This includes containing all the IDF objects in the file, as well as meta data such as the version ID for this IDD, and finally providing worker functions for accessing the IDD data

Relevant "public" members are listed here:

Variables

- **`file_path`** (*`str`*) – The path given when instantiating this IDF, not necessarily an actual path
- **`version_float`** (*`float`*) – The floating point representation of the version of this IDD (for 8.6.0 it is 8.6)
- **`objects`** (*`[IDFObject]`*) – A list of all IDF objects found in the IDF

Constructor parameters:

Parameters **`file_path`** (*`str`*) – A file path for this IDF; not necessarily a valid path as it is never used, just available for bookkeeping purposes.

`get_idf_objects_by_type` (*type_to_get*)
This function returns all objects of a given type found in this IDF structure instance

Parameters **`type_to_get`** (*`str`*) – A case-insensitive object type to retrieve

Returns A list of all objects of the given type

`global_swap` (*dict_of_swaps*)

`validate` (*idd_structure*)
This function validates the current IDF structure instance against standard IDD object tags such as required and unique objects.

Parameters **`idd_structure`** – An IDDStructure instance representing an entire IDD file

Returns A list of ValidationIssue instances, each describing an issue encountered

`whole_idf_string` (*idd_structure=None*)
This function returns a string representation of the entire IDF contents. If the idd structure argument is passed in, it is passed along to object worker functions in order to generate an intelligent representation.

Parameters **`idd_structure`** (*`IDDStructure`*) – An optional IDDStructure instance representing an entire IDD file

Returns A string of the entire IDF contents, ready to write to a file

`write_idf` (*idf_path*, *idd_structure=None*)
This function writes the entire IDF contents to a file. If the idd structure argument is passed in, it is passed along to object worker functions in order to generate an intelligent representation.

Parameters

- **`idf_path`** (*`str`*) – The path to the file to write
- **`idd_structure`** (*`IDDStructure`*) – An optional IDDStructure instance representing an entire IDD file

Returns None

class `eptransition.idf.objects.`**`ValidationIssue`**(*object_name*, *severity*, *message*, *field_name=None*)

This class stores information about any issue that occurred when reading an IDF file.

Parameters

- **`object_name`** (*`str`*) – The object type that was being validated when this issue arose
- **`severity`** (*`int`*) – The severity of this issue, from the class constants
- **`message`** (*`str`*) – A descriptive message for this issue
- **`field_name`** (*`str`*) – The field name that was being validated when this issue arose, if available.

`ERROR` = **2**

`INFORMATION` = **0**

`WARNING` = **1**

static **`severity_string`**(*severity_integer*)

Returns a string version of the severity of this issue

Parameters **`severity_integer`** (*`int`*) – One of the constants defined in this class (INFORMATION, etc.)

Returns A string representation of the severity

CHAPTER 8

IDF Processor Module Documentation

class `eptransition.idf.processor.`**`IDFProcessor`**

The core IDF Processor class. Given an IDF via stream or path, this class has workers to robustly process the IDF into a rich IDFStructure instance.

The constructor takes no arguments but sets up instance variables. Relevant "public" members are listed here:

Variables

- **`idf`** (*`IDFStructure`*) – The resulting IDFStructure instance after processing the IDF file/stream
- **`file_path`** (*`str`*) – A file path for this IDF, although it may be just a simple descriptor

`process_file`()

Internal worker function that reads the IDF stream, whether it was constructed from a file path, stream or string. This processor then processes the file line by line looking for IDF objects and comment blocks, and parsing them into a meaningful structure

Returns An IDF structure describing the IDF contents

Raises **`ProcessingException`** – for any issues encountered during the processing of the idf

`process_file_given_file_path`(*file_path*)

This worker allows processing of an IDF file at a specific path on disk.

Parameters **`file_path`** – The path to an IDF file on disk.

Returns An IDFStructure instance created from processing the IDF file

Raises **`ProcessingException`** – if the specified file does not exist

`process_file_via_stream`(*idf_file_stream*)

This worker allows processing of an IDF snippet via stream. Most useful for unit testing, but possibly for other situations.

Parameters **`idf_file_stream`** (*`file-like-object`*) – An IDF snippet that responds to typical file-like commands such as read(). A common object would be the StringIO object.

Returns An IDFStructure instance created from processing the IDF snippet

`process_file_via_string`(*idf_string*)

This worker allows processing of an IDF snippet string. Most useful for unit testing, but possibly for other situations.

Parameters `idf_string` (*str*) – An IDF snippet string

Returns An IDFStructure instance created from processing the IDF string

CHAPTER 9

Base Transition Rules Class Documentation

class `eptransition.rules.base_rule.`**`ObjectTypeAndName`**(*object_type*, *object_name*)

This is a simple class for defining an object type/name combination

Parameters

- **`object_type`** (`str`) – The object type
- **`object_name`** (`str`) – The name of the object (usually field[0]

class `eptransition.rules.base_rule.`**`OutputVariableTransitionRule`**

This class is a must-override base class for defining transition rules for output variable objects These objects are treated somewhat specially by the tool because a small change can affect so many objects, and it would be unwise to expect each version to include so much repeated code.

The structure of the output objects here is based on 8.5/8.6. In the future, if the objects didn't change much, it would make most sense to just keep using this class and making small tweaks as needed. If more major changes occur, it would be best to create a new base class to move forward.

The fields for each object are described next

•OV: Output:Variable

0.Key Value

1.Variable Name * * * *

2.Reporting Frequency

3.Schedule Name

•OM: Output:Meter, OMM: Output:Meter:MeterFileOnly

0.Name * * * *

1.Reporting Frequency

•OMC: Output:Meter:Cumulative, OMCM: Output:Meter:Cumulative:MeterFileOnly

0.Name * * * *

1.Reporting Frequency

- OTT: Output:Table:TimeBins
  0. Key Value
  1. Variable Name * * * *
  2. Interval Start
  3. Interval Size
  4. Interval Count
  5. Schedule Name
  6. Variable Type
- FMUI: ExternalInterface:FunctionalMockupUnitImport:From:Variable
  0. EnergyPlus Key Value
  1. EnergyPlus Variable Name * * * *
  2. FMU File Name
  3. FMU Instance Name
  4. FMU Variable Name
- FMUE: ExternalInterface:FunctionalMockupUnitExport:From:Variable
  0. EnergyPlus Key Value
  1. EnergyPlus Variable Name * * * *
  2. FMU Variable Name
- EMS: EnergyManagementSystem:Sensor
  0. Name
  1. Output:Variable or Output:Meter Key Name
  2. Output:Variable or Output:Meter Name * * * *
- OTM: Output:Table:Monthly
  0. Name
  1. Digits after Decimal
  2. Variable or Meter X Name * * * *
  3. Variable or Meter X Aggregation Type

  ... repeating with variable names for each 2, 4, 6, 8, ...
- OTA: Output:Table:Annual
  0. Name
  1. Filter
  2. Schedule Name
  3. Variable or Meter X Name * * * *
  4. Variable or Meter X Aggregation Type

  ... repeating with variable names for each 3, 5, 7, 9, ...
- MC: Meter:Custom

0.Name

1.Fuel Type

2.Key Name X

3.Output Variable or Meter Name X * * * *

... repeating with variable names for each 3, 5, 7, 9, ...

- MCD: Meter:CustomDecrement

  0.Name

  1.Fuel Type

  2.Source Meter Name ????

  3.Key Name X

  4.Output Variable or Meter Name X

  ... repeating with variable names for each 4, 6, 8, 10, ...

`EMS` = 'ENERGYMANAGEMENTSYSTEM:SENSOR'

`FMUE` = 'EXTERNALINTERFACE:FUNCTIONALMOCKUPUNITEXPORT:FROM:VARIABLE'

`FMUI` = 'EXTERNALINTERFACE:FUNCTIONALMOCKUPUNITIMPORT:FROM:VARIABLE'

`MC` = 'METER:CUSTOM'

`MCD` = 'METER:CUSTOMDECREMENT'

`OM` = 'OUTPUT:METER'

`OMC` = 'OUTPUT:METER:CUMULATIVE'

`OMCM` = 'OUTPUT:METER:CUMULATIVE:METERFILEONLY'

`OMM` = 'OUTPUT:METER:METERFILEONLY'

`OTA` = 'OUTPUT:TABLE:ANNUAL'

`OTM` = 'OUTPUT:TABLE:MONTHLY'

`OTT` = 'OUTPUT:TABLE:TIMEBINS'

`OV` = 'OUTPUT:VARIABLE'

`complex_output_operation` (*full_object*, *dependent_objects*)

This method should be overridden in derived classes and should perform the complex operations to transition the argument object passed in. The function should return a list because some complex operations may split the initial object into multiple objects. The object passed in will have any simple name swaps already performed.

Parameters

- **`full_object`** – The original object to be replaced.
- **`dependent_objects`** – A dictionary of dependent objects

Returns A list of new IDFObject instances, typically just one though

Raises **`UnimplementedMethodException`** – Raised if this method is called on the base class itself

`get_complex_operation_types``()`

This method should be overridden in the derived classes and return a list of object names that require more complex transition operations than a simple variable name swap

Returns A list of strings, each representing an object name that requires complex transition operations

Raises **`UnimplementedMethodException`** – Raised if this method is called on the base class itself

`get_dependent_object_names``()`

This method can be overridden in derived classes if any of the output variable name changes depend on other objects in the idf. Simply return a list of object names

Returns A list of object names that output variable name changes are dependent upon

`get_output_objects``()`

This method should be overridden in derived classes and return a list of all output-related object types in this version of EnergyPlus. A base version is available in the base class that can be used as a starter and if an object name changes, the derived class can change that name as needed in the return array.

Returns A list of strings, each representing an output object type name

Raises **`UnimplementedMethodException`** – Raised if this method is called on the base class itself

`get_simple_swaps``()`

This method should be overridden in derived classes and return a dictionary where each key is the name of an output variable, and the value of each key is the new variable name. This map is used when doing the simple variable name swaps.

Returns A dictionary of <old_variable_name, new_variable_name>

Raises **`UnimplementedMethodException`** – Raised if this method is called on the base class itself

`get_standard_indexes_from_object``(`*object_name*`)`

This method should be overridden in derived classes and return a list of the zero-based field indexes that include a variable name in the given object type. A base version is available in the base class that can be used as a starter and if the structure of any object types changes, the derived class can change that one as needed in the return list

Parameters **`object_name`** – The name of the object being inspected

Returns A list of zero-based indexes, each representing a field containing an output variable name

Raises **`UnimplementedMethodException`** – Raised if this method is called on the base class itself

`original_full_variable_type_list``()`

`original_standard_indexes_from_object``(`*object_name*`)`

This method returns the list of indexes where variable names are found. These are zero based indexes. This method returns a base version that can be used by a derived class directly, modified, or used as a template for future derived classes.

Parameters **`object_name`** – The upper case name of the object currently being transitioned.

Returns A list of zero-based indexes

`simple_name_swap``(`*variable_name*`)`

This method is a simple method that queries the *must-override get_simple_swaps* method in the derived class and either returns a new variable name to swap in place of the original name, or returns None as a signal that this original variable name does not need replacement

Parameters **`variable_name`** – The original variable name to potentially be replaced

Returns A new variable name, if a swap is to be performed, or None if not

`transition`(*core_object*, *dependent_objects*)

This method can be implemented by derived classes if necessary, but should capture the entire transition functionality just using the other required <must-override> methods in this class. This function first scans all the variable names in the current locations, and renames as needed. Then this function checks if this object type needs a complex transition, and if so, calls the appropriate derived method. This method then returns a full IDFObject instance.

Parameters

- **`core_object`** – The original object to be replaced
- **`dependent_objects`** – A dictionary of dependent objects

Returns A list of new IDFObject instances, typically just one though

class `eptransition.rules.base_rule.`**`TransitionReturn`**(*objects_to_write*, *objects_to_delete=None*)

This is a simple class for capturing the response from a transition call

Parameters

- **`objects_to_write`** (*[IDFObject]*) – The list of IDFObject instances to be written as a result of this transition
- **`objects_to_delete`** (*[ObjectTypeAndName]*) – The list of idf object type/name combinations to be deleted as a result of this transition

class `eptransition.rules.base_rule.`**`TransitionRule`**

This class is a must-override base class for defining transition rules for idf objects

`get_name_of_object_to_transition`()

This method should be overridden in derived classes and return a single name of an object that this rule handles the transition for.

Returns A string name of an object to transition

Raises **`UnimplementedMethodException`** – Raised if this method is called on the base class itself

`get_names_of_dependent_objects`()

This method should be overridden in derived classes and return a list of object names that the derived transition implementation is dependent upon.

Returns A list of string object names

Raises **`UnimplementedMethodException`** – Raised if this method is called on the base class itself

`transition`(*core_object*, *dependent_objects*)

This method is the core transition operation for this object.

Parameters

- **`core_object`** – The original idf object to be transitioned
- **`dependent_objects`** – A dictionary of {object_name: [idf_object, ...]} containing the idf object data in the original idf that have object names defined in this derived classes

get_names_of_dependent_objects method. Each key in this argument is a string object name, and each value is a list of all the idf objects in the file of that type.

Returns A list of new IDFObject instances, typically just one though

Raises **`UnimplementedMethodException`** – Raised if this method is called on the base class itself

CHAPTER 10

Generic Version Rule Class Documentation

class `eptransition.rules.version_rule.`**`VersionRule`**`(`*end_version*`)`

Bases: `eptransition.rules.base_rule.TransitionRule`

This class implements, in a generic fashion, the transition rule for the Version object. By passing in the identifier for the target version, the rules are set up so this doesn't have to change for each version.

Parameters **`end_version`** – The new value for the version object's single field: Version ID

`get_name_of_object_to_transition``()`

`get_names_of_dependent_objects``()`

`transition``(`*core_object*, *dependent_objects*`)`

CHAPTER 11

Indexes and tables

- genindex
- modindex
- search