otoole Documentation

Release unknown

Will Usher

Apr 29, 2021

Contents

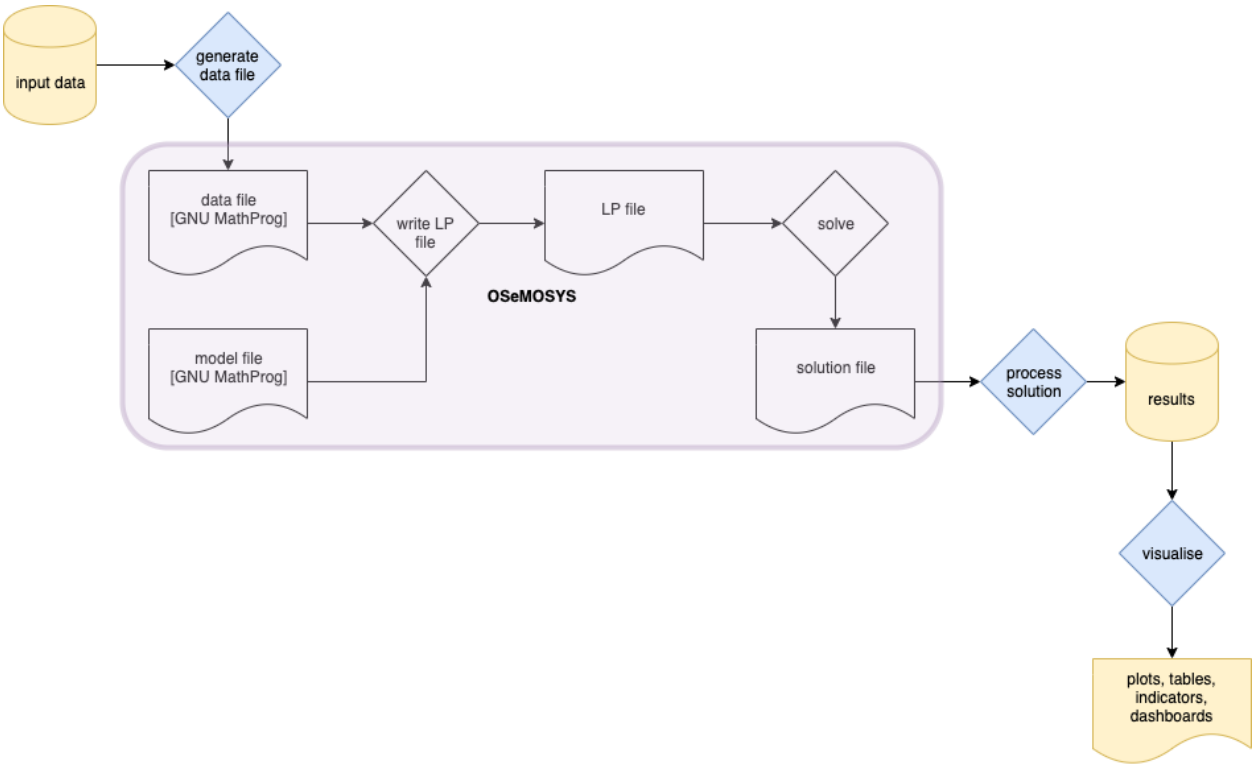
A Python toolkit to support use of OSeMOSYS

CHAPTER 1

Description

otoole is a Python package which provides a command-line interface for users of OSeMOSYS.

The aim of the package is to provide a community resource which centralises the commonly used pre- and post-processing steps around the use of OSeMOSYS.



otoole aims to support different ways of storing input data and results, including csv files, databases, datapackages and Excel workbooks, as well as different implementations of the OSeMOSYS model.

1.1 Dependencies

otoole requires a number of dependencies, including pygraphviz, which can be difficult to install on Windows.

The easiest way to install the dependencies is to use miniconda.

1. Obtain the miniconda package:
2. Add the **conda-forge** channel `conda config --add channels conda-forge`
3. Create a new Python environment `conda create -n myenv python=3.7 networkx datapackage pandas pulp graphviz xlrd`
4. Activate the new environment `conda activate myenv`
5. Use pip to install otoole `pip install otoole`

CHAPTER 2

Installation

Install **otoole** using pip:

```
pip install otoole
```

To upgrade **otoole** using pip:

```
pip install otoole --upgrade
```

CHAPTER 3

Usage

For detailed instructions of the use of the tool, run the command line help function:

```
otoole --help
```

CHAPTER 4

Contributing

New ideas and bugs are found on the repository Issue Tracker. Please do contribute by discussing and developing these ideas further, or by developing the codebase.

To contribute directly to the documentation of code development, you first need to install the package in *develop mode*:

```
git clone http://github.com/OSeMOSYS/otoole
cd otoole
git checkout <branch you wish to use>
python setup.py develop
```

Now, all changes made in the codebase will automatically be reflected in the installed Python version accessible on the command line or from importing otoole modules into other Python packages.

CHAPTER 5

Getting Started

Install `otoole` using pip:

```
pip install otoole
```

Check the version installed:

```
otoole -V
```

Download an OSeMOSYS datapackage and convert it to a modelfile:

```
otoole convert datapackage datafile https://zenodo.org/record/3479823/files/KTH-dESA/
↪simplicity-v0.1a0.zip simplicity.txt
```

Visualise the Reference Energy System:

```
otoole viz res https://zenodo.org/record/3479823/files/KTH-dESA/simplicity-v0.1a0.zip␣
↪res.png && open res.png
```

Alternatively, convert an OSeMOSYS datafile to a datapackage:

```
otoole convert datafile datapackage simplicity.txt simplicity
```

Validate the names of technologies and fuels against the standard naming convention and identify isolated fuels, emissions and technologies:

```
otoole validate datapackage https://zenodo.org/record/3479823/files/KTH-dESA/
↪simplicity-v0.1a0.zip
```

CHAPTER 6

Contents

6.1 License

6.2 Contributors

- Will Usher <wusher@kth.se>

6.3 Changelog

6.3.1 Version 0.5

- Add validation of names and fuels in datapackage - Adds `validate` command to the command-line interface - Define a validation config as a YAML file for names

6.3.2 Version 0.4

- Tidy up the command line interface
- Convert to/from SQLite database from/to datapackage
- Remove rotten pygraphviz dependency

6.3.3 Version 0.3

- Create a Tabular Data Package from an OSeMOSYS datafile

6.3.4 Version 0.2

- Visualise a reference energy system from a Tabular Data Package

6.3.5 Version 0.1

- Add CPLEX to csv or CBC solution file conversion script
- Create CSV files in a folder from an excel workbook
- Create a Tabular Data Package from a folder of CSVs
- Create an OSeMOSYS datafile from a Tabular Data Package
- Adds a command line interface to access these tools

6.4 otoole

6.4.1 otoole package

Subpackages

otoole.preprocess package

Submodules

otoole.preprocess.create_datapackage module

Creates a datapackage from a collection of CSV files of OSeMOSYS input data

- Uses Frictionless Data datapackage concept to build a JSON schema of the dataset

- Enforces relations between sets and indices in parameter files

otoole.preprocess.create_datapackage.**convert_datapackage_to_sqlite**(*path_to_datapackage*, *sqlite*)

Load and save table to SQLite

otoole.preprocess.create_datapackage.**generate_package**(*path_to_package*)

Creates a datapackage in folder `path_to_package`

[{'fields': 'REGION', 'reference': {'resource': 'REGION', 'fields': 'VALUE'}}]

otoole.preprocess.create_datapackage.**main**(*wide_folder*, *narrow_folder*)

otoole.preprocess.create_datapackage.**validate_contents**(*path_to_package*)

otoole.preprocess.datafile_to_datapackage module

otoole.preprocess.excel_to_osemosys module

Extract data from spreadsheets and write to an OSeMOSYS datafile

Reads and writes the following OSeMOSYS parameters and sets to and from files on disk.

Notes

Sets

These are the standard sets:

```
set YEAR;
set TECHNOLOGY;
set TIMESLICE;
set FUEL;
set EMISSION;
set MODE_OF_OPERATION;
set REGION;
set SEASON;
set DAYTYPE;
set DAILYTIMEBRACKET;
set FLEXIBLEDEMANDTYPE;
set STORAGE;
```

All sets are written in a CSV file in one column of values with header of `VALUE`. For example, the CSV file for set YEAR:

```
VALUE
2015
2016
2017
2018
2019
2020
```

Sets are written into the OSeMOSYS data file using the following syntax:

```
set YEAR := 2014 2015 2016 2017 2018 2019 2020 ;
```

param In general, parameters can be written into CSV files in narrow or wide

param formats. In narrow format, the CSV file should look as follows::

REGION,TIMESLICE,YEAR,VALUE SIMPLICITY,ID,2014,0.1667 SIMPLICITY,IN,2014,0.0833 SIMPLICITY,SD,2014,0.1667 SIMPLICITY,SN,2014,0.0833 SIMPLICITY,WD,2014,0.3333 SIMPLICITY,WN,2014,0.1667

In wide format, the final index is transposed:

```
REGION,TIMESLICE,2014,2015,...
SIMPLICITY,ID,0.1667,0.1667
SIMPLICITY,IN,0.0833,0.0833
SIMPLICITY,SD,0.1667,0.1667
SIMPLICITY,SN,0.0833,0.0833
SIMPLICITY,WD,0.3333,0.3333
SIMPLICITY,WN,0.1667,0.1667
```

Wide format is a bit nicer to use with spreadsheets, as it allows you to more easily plot graphs of values, but narrow format is a more flexible and easily manipulated data format.

Writing parameteters:

1-dimensional e.g. DiscountRate{r in REGION}

2-dimension e.g. YearSplit{l in TIMESLICE, y in YEAR}

n-dimensional e.g. DaysInDayType{ls in SEASON, ld in DAYTYPE, y in YEAR}

Global parameters:

```
param YearSplit{l in TIMESLICE, y in YEAR};
param DiscountRate{r in REGION};
param DaySplit{lh in DAILYTIMEBRACKET, y in YEAR};
param Conversionls{l in TIMESLICE, ls in SEASON};
param Conversionld{l in TIMESLICE, ld in DAYTYPE};
param Conversionlh{l in TIMESLICE, lh in DAILYTIMEBRACKET};
param DaysInDayType{ls in SEASON, ld in DAYTYPE, y in YEAR};
param TradeRoute{r in REGION, rr in REGION, f in FUEL, y in YEAR};
param DepreciationMethod{r in REGION};
```

Demand parameters:

```
param SpecifiedAnnualDemand{r in REGION, f in FUEL, y in YEAR};
param SpecifiedDemandProfile{r in REGION, f in FUEL, l in TIMESLICE, y in YEAR};
param AccumulatedAnnualDemand{r in REGION, f in FUEL, y in YEAR};
```

Performance parameters:

```
param CapacityToActivityUnit{r in REGION, t in TECHNOLOGY};
param TechWithCapacityNeededToMeetPeakTS{r in REGION, t in TECHNOLOGY};
param CapacityFactor{r in REGION, t in TECHNOLOGY, l in TIMESLICE, y in YEAR};
param AvailabilityFactor{r in REGION, t in TECHNOLOGY, y in YEAR};
param OperationalLife{r in REGION, t in TECHNOLOGY};
param ResidualCapacity{r in REGION, t in TECHNOLOGY, y in YEAR};
param InputActivityRatio{r in REGION, t in TECHNOLOGY, f in FUEL, m in MODE_OF_
↪OPERATION, y in YEAR};
param OutputActivityRatio{r in REGION, t in TECHNOLOGY, f in FUEL, m in MODE_OF_
↪OPERATION, y in YEAR};
```

Technology Costs parameters:

```
param CapitalCost{r in REGION, t in TECHNOLOGY, y in YEAR};
param VariableCost{r in REGION, t in TECHNOLOGY, m in MODE_OF_OPERATION, y in YEAR};
param FixedCost{r in REGION, t in TECHNOLOGY, y in YEAR};
```

Storage parameters:

```
param TechnologyToStorage{r in REGION, t in TECHNOLOGY, s in STORAGE, m in MODE_OF_
↪OPERATION};
param TechnologyFromStorage{r in REGION, t in TECHNOLOGY, s in STORAGE, m in MODE_OF_
↪OPERATION};
param StorageLevelStart{r in REGION, s in STORAGE};
param StorageMaxChargeRate{r in REGION, s in STORAGE};
param StorageMaxDischargeRate{r in REGION, s in STORAGE};
param MinStorageCharge{r in REGION, s in STORAGE, y in YEAR};
param OperationalLifeStorage{r in REGION, s in STORAGE};
param CapitalCostStorage{r in REGION, s in STORAGE, y in YEAR};
param ResidualStorageCapacity{r in REGION, s in STORAGE, y in YEAR};
```

Capacity Constraints parameters:

```
param CapacityOfOneTechnologyUnit{r in REGION, t in TECHNOLOGY, y in YEAR};
param TotalAnnualMaxCapacity{r in REGION, t in TECHNOLOGY, y in YEAR};
param TotalAnnualMinCapacity{r in REGION, t in TECHNOLOGY, y in YEAR};
```

Investment Constraints parameters:

```
param TotalAnnualMaxCapacityInvestment{r in REGION, t in TECHNOLOGY, y in YEAR};
param TotalAnnualMinCapacityInvestment{r in REGION, t in TECHNOLOGY, y in YEAR};
```

Activity Constraints parameters:

```
param TotalTechnologyAnnualActivityUpperLimit{r in REGION, t in TECHNOLOGY, y in YEAR}
↪;
param TotalTechnologyAnnualActivityLowerLimit{r in REGION, t in TECHNOLOGY, y in YEAR}
↪;
param TotalTechnologyModelPeriodActivityUpperLimit{r in REGION, t in TECHNOLOGY};
param TotalTechnologyModelPeriodActivityLowerLimit{r in REGION, t in TECHNOLOGY};
```

Reserve Margin parameters:

```
param ReserveMarginTagTechnology{r in REGION, t in TECHNOLOGY, y in YEAR};
param ReserveMarginTagFuel{r in REGION, f in FUEL, y in YEAR};
param ReserveMargin{r in REGION, y in YEAR};
```

RE Generation Target parameters:

```
param RETagTechnology{r in REGION, t in TECHNOLOGY, y in YEAR};
param RETagFuel{r in REGION, f in FUEL, y in YEAR};
param REMinProductionTarget{r in REGION, y in YEAR};
```

Emissions & Penalties parameters:

```
param EmissionActivityRatio{r in REGION, t in TECHNOLOGY, e in EMISSION, m in MODE_OF_
↪OPERATION, y in YEAR};
param EmissionsPenalty{r in REGION, e in EMISSION, y in YEAR};
param AnnualExogenousEmission{r in REGION, e in EMISSION, y in YEAR};
param AnnualEmissionLimit{r in REGION, e in EMISSION, y in YEAR};
```

(continues on next page)

(continued from previous page)

```
param ModelPeriodExogenousEmission{r in REGION, e in EMISSION};
param ModelPeriodEmissionLimit{r in REGION, e in EMISSION};
```

otoole.preprocess.excel_to_osemosys.**generate_csv_from_excel**(*input_workbook*, *output_folder*)

Generate a folder of CSV files from a spreadsheet

Parameters

- **input_workbook** (*str*) – Path to spreadsheet containing OSeMOSYS data
- **output_folder** (*str*) – Path of the folder containing the csv files

otoole.preprocess.excel_to_osemosys.**read_config**(*path_to_user_config: str = None*) → Dict[KT, VT]

Reads the config file holding expected OSeMOSYS set and parameter dimensions

Parameters **path_to_user_config** (*str, optional, default=None*) – Optional path to a user defined configuration file

Returns

Return type dict

otoole.preprocess.longify_data module

Read in a folder of irregular wide-format csv files and write them out as narrow csvs

otoole.preprocess.longify_data.**check_datatypes**(*narrow: pandas.core.frame.DataFrame, config_details: Dict[KT, VT], parameter: str*) → pandas.core.frame.DataFrame

Checks a parameters datatypes

Parameters

- **narrow** (*pandas.DataFrame*) – The parameter data
- **config_details** (*dict*) – The configuration dictionary
- **parameter** (*str*) – The name of the parameter

otoole.preprocess.longify_data.**check_parameter**(*df*, *config_details*, *name*)

otoole.preprocess.longify_data.**check_set**(*df*, *config_details*, *name*)

otoole.preprocess.longify_data.**check_set_datatype**(*narrow: pandas.core.frame.DataFrame, config_details: Dict[KT, VT], set_name: str*) → pandas.core.frame.DataFrame

Checks the datatypes of a set_name dataframe

Parameters

- **narrow** (*pandas.DataFrame*) – The set data
- **config_details** (*dict*) – The configuration dictionary
- **set_name** (*str*) – The name of the set

otoole.preprocess.longify_data.**main**(*output_folder*, *narrow_folder*)

Read in a folder of irregular wide-format files and write as narrow csvs

otoole.preprocess.longify_data.**write_out_dataframe**(*folder*, *parameter*, *df*)

Writes out a dataframe as a csv into the data subfolder of a datapackage

Parameters

- **folder** (*str*) –
- **parameter** (*str*) –
- **df** (*pandas.DataFrame*) –

otoole.preprocess.narrow_to_datafile module

otoole.preprocess.narrow_to_datafile.**main**(*datapackage: str*, *datafilepath: str*, *sql: bool = False*)

otoole.preprocess.narrow_to_datafile.**read_narrow_csv**(*filepath*)

otoole.preprocess.narrow_to_datafile.**write_parameter**(*filepath: TextIO*, *df: pandas.core.frame.DataFrame*, *parameter_name*, *default*)

Parameters

- **filepath** (*StreamIO*) –
- **df** (*pandas.DataFrame*) –
- **parameter_name** (*str*) –
- **default** (*int*) –

otoole.preprocess.narrow_to_datafile.**write_set**(*filepath: TextIO*, *df: pandas.core.frame.DataFrame*, *set_name*)

Parameters

- **filepath** (*StreamIO*) –
- **df** (*pandas.DataFrame*) –
- **parameter_name** (*str*) –

Module contents

otoole.results package

Submodules

otoole.results.convert module

Converts an OSeMOSYS solution file from CPLEX, CBC or GLPK into CBC or CSV format

class otoole.results.convert.**ConvertLine**(*data: List[T], start_year: int, end_year: int, output_format='cbc'*)

Bases: object

Abstract class which defines the interface to the family of convertors

Inherit this class and implement the _do_it() method to produce the data to be written out into a new format

Example

```
>>> cplex_line = "AnnualCost        REGION  CDBACKSTOP      1.0     0.0     ␣
↪137958.8400384134"
>>> convertor = RegionTechnology()
>>> convertor.convert()
VariableName(REGION,TECHCODE01,2015)       42.69         0\n
VariableName(REGION,TECHCODE01,2017)       137958.84         0\n
```

convert() → List[str]

convert_cbc() → List[str]
Format the data for writing to a CBC file

convert_csv() → List[str]
Format the data for writing to a csv file

class otoole.results.convert.**RegionTechnology**(*data: List[T], start_year: int, end_year: int, output_format='cbc'*)

Bases: *otoole.results.convert.ConvertLine*

class otoole.results.convert.**RegionTimeSliceTechnologyMode**(*data: List[T], start_year: int, end_year: int, output_format='cbc'*)

Bases: *otoole.results.convert.ConvertLine*

otoole.results.convert.**convert_cplex_file**(*cplex_filename: str, output_filename: str, start_year=2015, end_year=2070, output_format='cbc'*)

Converts a CPLEX solution file into that of the CBC solution file

Parameters

- **cplex_filename** (*str*) – Path to the transformed CPLEX solution file
- **output_filename** (*str*) – Path for the processed data to be written to

otoole.results.convert.**process_line**(*line: str, start_year: int, end_year: int, output_format: str*) → List[str]

Processes an individual line in a CPLEX file

A different ConvertLine implementation is chosen depending upon the variable name

Parameters

- **line** (*str*) –
- **start_year** (*int*) –
- **end_year** (*int*) –
- **output_format** (*str*) – The file format required - either `csv` or `cbc`

Module contents

otoole.visualise package

Submodules

otoole.visualise.res module

Visualise the reference energy system

`otoole.visualise.res.add_fuel`(*package_rows: List[List[T]]*) → List[Tuple[str, Dict[KT, VT]]]

Add fuel nodes

Parameters **package_rows** (*list of dict*) –

Returns A list of node names along with a dict of node attributes

Return type list of tuple

`otoole.visualise.res.build_graph`(*nodes: List[Tuple[str, Dict[KT, VT]]], edges: List[Tuple[str, str, Dict[KT, VT]]]*) → networkx.classes.digraph.DiGraph

Builds the graph using networkx

Parameters

- **nodes** (*list*) – A list of node tuples
- **edges** (*list*) – A list of edge tuples

Returns A directed graph representing the reference energy system

Return type networkx.DiGraph

`otoole.visualise.res.create_graph`(*datapackage: datapackage.package.Package*)

Creates a graph of technologies and fuels

Parameters **datapackage** (*datapackage.Package*) –

Returns networkx.DiGraph

Return type graph

`otoole.visualise.res.create_res`(*path_to_datapackage: str, path_to_resfile: str*)

Create a reference energy system diagram from a Tabular Data Package

Parameters

- **path_to_datapackage** (*str*) – The path to the `datapackage.json`
- **path_to_resfile** (*str*) – The path to the image file to be created

`otoole.visualise.res.draw_graph`(*graph, path_to_resfile*)

Layout the graph and write it to disk

Uses pygraphviz to set some graph attributes, layout the graph and write it to disk

Parameters **path_to_resfile** (*str*) – The file path of the PNG image file that will be created

`otoole.visualise.res.extract_edges`(*package_rows: List[Dict[KT, VT]], from_column: str, to_column: str, parameter_name: str, directed: bool = True*) → List[Tuple[str, str, Dict[KT, VT]]]

Add edges from a Tabular Data Table

Parameters

- **package_rows** (*list of dict*) –
- **from_column** (*str*) – The name of the column to use as source of the edge
- **to_column** (*str*) – The name of the column to use as a destination of the edge
- **parameter_name** (*str*) – The name of the parameter

- **directed** (*bool, default=True*) – Specifies whether the edge should have an arrow or not

Returns A list of edges with from/to nodes names and edge attributes

Return type list of tuple

otoole.visualise.res.**extract_nodes**(*package_rows: List[List[T]], node_type='technology', color='red', shape='circle'*) → List[Tuple[str, Dict[KT, VT]]]

Add nodes from a Tabular Data Table

Parameters

- **package_rows** (*list of dict*) –
- **node_type** (*str, default='technology'*) –
- **color** (*str, default='red'*) –
- **shape** (*str, default='circle'*) –

Returns A list of nodes with attributes

Return type list of tuple

otoole.visualise.res.**load_datapackage**(*path_to_datapackage: str*) → datapackage.package.Package

Module contents

Visualise different aspects of an OSeMOSYS model and data

Provides the following commands:

```
otoole viz res <path_to_datapackage.json> <path_to_res_image>
```

`otoole viz res` generates an image of the reference energy system in a datapackage

otoole.visualise.**create_res**(*path_to_datapackage: str, path_to_resfile: str*)

Create a reference energy system diagram from a Tabular Data Package

Parameters

- **path_to_datapackage** (*str*) – The path to the `datapackage.json`
- **path_to_resfile** (*str*) – The path to the image file to be created

Submodules

otoole.cli module

otoole.exceptions module

exception otoole.exceptions.**OtooleException**

Bases: `Exception`

Base class for all otoole exceptions.

exception otoole.exceptions.**OtooleRelationError**(*resource*, *foreign_resource*, *message*)

Bases: *otoole.exceptions.OtooleException*

Relations between input data is not correct

Parameters

- **resource** (*str*) – Name of the resource which is invalid
- **foreign_resource** (*str*) – Name of the resource which is invalid
- **message** (*str*) – Error message

exception otoole.exceptions.**OtooleValidationError**(*resource*, *message*)

Bases: *otoole.exceptions.OtooleException*

Input data is invalid

Parameters

- **resource** (*str*) – Name of the resource which is invalid
- **message** (*str*) – Error message

otoole.validate module

Ensures that technology and fuel names match the convention

For example, to validate the following list of names, you would use the config shown below:

```
theseUNIQUE_ENTRY1
are__UNIQUE_ENTRY2
all__UNIQUE_ENTRY1
validUNIQUE_ENTRY2
entryUNIQUE_ENTRY1
in__UNIQUE_ENTRY2
a____UNIQUE_ENTRY1
list_UNIQUE_ENTRY2
```

Create a yaml validation config with the following format:

```
codes:
  some_valid_codes:
    UNIQUE_ENTRY1: Description of unique entry 1
    UNIQUE_ENTRY2: Description of unique entry 2
schema:
  schema_name:
  - name: first_entry_in_schema
    valid: ['these', 'are__', 'all__', 'valid', 'entry', 'in__', 'a____', 'list_']
    position: (1, 5) # a tuple representing the start and end position
  - name: second_entry_in_schema
    valid: some_valid_codes  # references an entry in the codes section of the config
    position: (6, 19) # a tuple representing the start and end position
```

otoole.validate.**check_for_duplicates**(*codes: List[T]*) → bool

otoole.validate.**compose_expression**(*schema: List[T]*) → str

Generates a regular expression from a schema

Returns

Return type str

`otoole.validate.`**`compose_multi_expression`**(*resource: List[T]*) → str
Concatenates multiple expressions using an OR operator

Use to validate elements using an OR operation e.g. the elements must match this expression OR the expression

`otoole.validate.`**`create_schema`**(*config: Dict[KT, VT] = None*) → Dict[KT, VT]
Populate the dict of schema with codes from the validation config

Parameters **`config`**(*`dict, default=None`*) – A configuration dictionary containing `codes` and `schema` keys

`otoole.validate.`**`identify_orphaned_fuels_techs`**(*package*) → Dict[str, str]
Returns a list of fuels and technologies which are unconnected

Returns

Return type dict

`otoole.validate.`**`main`**(*file_format: str, filepath: str, config=None*)

`otoole.validate.`**`read_validation_config`**()

`otoole.validate.`**`validate`**(*expression: str, name: str*) → bool
Determine if `name` matches the `expression`

Parameters

- **`expression`**(*`str`*) –
- **`name`**(*`str`*) –

Returns

Return type bool

`otoole.validate.`**`validate_resource`**(*package, resource: str, schemas: List[Dict[KT, VT]]*)

Parameters

- **`package`** –
- **`resource`**(*`str`*) –
- **`schemas`**(*`List[Dict]`*) – The schema from which to create a validation expression

Module contents

`otoole.`**`read_datapackage`**(*filepath: str, sql: bool = False*)
Open an OSeMOSYS datapackage

Parameters

- **`filepath`**(*`str`*) –
- **`sql`**(*`bool, default=False`*) –

`otoole.`**`read_packaged_file`**(*filename: str, module_name: str = None*)

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

P

R

V

W