cimpyorm Documentation

Thomas Offergeld

Jul 03, 2022

CONTENTS:

CHAPTER

ONE

INSTALLATION

CIMPyORM can be installed from pypi:

```
pip install cimpyorm
```

CIMPyORM comes pre-loaded with a CIM schema provided by ENTSO-E called Common Grid Model Exchange Specification (CGMES v. 2.4.15).

1.1 About

CIMPyORM is middleware to connect CIM datasets to a wide range of power system analysis tooling by providing comfortably navigable ORM representations of the datasets using SQLAlchemy and a variety of database backends.

It is developed and maintained by the Institute for High Voltage Equipment and Grids, Digitalization and Power Economics at RWTH Aachen University under a BSD-3-clause license.

1.2 Quickstart

1.2.1 Parsing datasets

The parser is invoked using the `parse` function:

`cimpyorm.parse`(*dataset*, *backend=<class 'cimpyorm.backends.SQLite'>*, *schema=None*, *log_to_file=False*, *silence_tqdm=False*)

Parse a database into a database backend and yield a database session to start querying on with the classes defined in the model namespace_name.

Afterwards, the database can be queried using SQLAlchemy query syntax, providing the CIM classes contained in the `Namespace` return value.

Parameters

- **dataset** (`Union[str, Path]`) – Path to the cim snapshot.
- **backend** – Database backend to be used (defaults to a SQLite on-disk database in the dataset location).
- **schema** (`Union[str, Path, None]`) – Location of the RDF schema to be used to parse the dataset (Folder of multiple RDF schemata or a single schema file).
- **log_to_file** (`Union[bool, Path, str]`) – Pass logging output to a file for this ingest only.
- **silence_tqdm** (`bool`) – Silence tqdm progress bars

Return type
Tuple[Session, Namespace]

Returns
sqlalchemy.orm.session.Session, argparse.Namespace

1.2.2 Loading datasets

Alternatively, an already parsed dataset can be loaded from on-disk files (using SQLite) or from a client-server database using the `load` function:

`cimpyorm.load`(*path_to_db*, *echo=False*)

Load an already parsed database from disk or connect to a server and yield a database session to start querying on with the classes defined in the model namespace_name.

Afterwards, the database can be queried using SQLAlchemy query syntax, providing the CIM classes contained in the `Namespace` return value.

Parameters

- **path_to_db** (Union[Engine, str]) – Path to the cim snapshot or a Engine.
- **echo** (bool) – Echo the SQL sent to the backend engine (SQLAlchemy option).

Return type
Tuple[Session, Namespace]

Returns
sqlalchemy.orm.session.Session, argparse.Namespace

1.2.3 Querying datasets

Queries for CIM objects are performed on the Session objects provided by the *parse()* and *load()* functions, e.g.:

```
db_session, model = parse(r"path_to_dataset")
acl = db_session.query(model.ACLineSegment).first()
```

to obtain the first CIM:ACLineSegment from the model.

The objects properties can subsequently be accessed as usual:

```
acl.r # Print the ACLineSegment's resistance
```

The CIM classes' inherited properties (to explore the model's classes and properties see *Exploring*) are also available:

```
acl.shortName # shortName is a property of CIM:IdentifiedObjects, which ACLineSegments␣
↪inherit from
```

Relationships can be accessed by their name:

```
bv = acl.BaseVoltage # Yields the CIM:BaseVoltage object associated with the␣
↪CIM:ACLineSegment
```

The model also allows for reverse lookup of relationships that are unidirectional in the CIM standard:

```
terminals = acl.Terminals # In the standard, a CIM:ACLineSegment (or rather its base␣
↪class
                          # CIM:ConductingEquipment) is referenced by a CIM:Terminal␣
↪definition,
                          # however, cimpyorm adds their inversion for convenience
                          # (the inverse property names are defined in the schema␣
↪definition)
```

1.3 Datasets

The internal representation of a CIM model is contained in the *dataset* object. A *dataset* wraps the functionality of a SQLAlchemy-session object for database operations (*query*, *add*, *commit* and so on).

Dataset objects are created by parsing or loading a CIM model, or by creating an empty dataset.

cimpyorm.**create_empty_dataset**(*version='16'*, *backend=<class 'cimpyorm.backends.SQLite'>*, *profile_whitelist=None*)

Create an empty dataset to be filled by instantiation of schema-objects.

Parameters

- **version** – The CIM version to be used.
- **backend** – The backend to be used.
- **profile_whitelist** – A list of allowed CIM-profiles. Profiles not contained in this list are not considered for schema generation. If empty or None, all schema elements defined by the CIM schema are generated.

Returns

An empty dataset/session object and the model namespace associated with the CIM schema.

1.3.1 Example Datasets

CIMPyORM comes preloaded with the ENTSO-E FullGrid and MiniGrid datasets.

They can be accessed through the datasets module:

```
from cimpyorm import datasets
db_session, model = datasets.ENTSOE_FullGrid()
db_session, model = datasets.ENTSOE_MiniBB()    # The Bus-Branch Model version
db_session, model = datasets.ENTSOE_MiniNB()    # The Node-Breaker Model version
```

1.4 Schema

The CIM Elements are organized in profiles and namespaces:

class cimpyorm.Model.Elements.Base.**CIMProfile**(*name*, *uri*, *short*)

A CIM Profile instance, usually contained in one file.

This class holds a profile found in a CIM Schema in a SQLAlchemy ORM.

Parameters
name – The profile's name.

properties
The CIM Properties defined in this profile.

classes
The CIM Classes used in this profile (not necessarily defined in it).

definitions = {'classes': <RelationshipProperty at 0x7f378411cd40; no key>}
The CIM Classes defined in this profile.

datatypes
The CIM Datatypes defined in this profile.

namespaces
The CIM namespaces contained in this profile.

class cimpyorm.Model.Elements.Base.**CIMNamespace**(*short*, *full_name*)
A CIM Namespace instance.

This class holds a namespace found in a CIM Schema in a SQLAlchemy ORM.

Parameters
- **short** – The namespace's short name.
- **full_name** – The namespace's full URI.

profiles
The CIM profiles that contain this namespace

1.4.1 SchemaElements

All elements described in a CIM Schema inherit from a common mixin (ElementMixin), that provides some common functionality, such as relationships to *CIMProfile* and *CIMNamespace* objects.

class cimpyorm.Model.Elements.Base.**ElementMixin**(*schema_elements=None*)
Mixin for schema entities.

This provides common functionality to associate individual SchemaElements (such as classes and properties) with namespaces and profiles. The namespace is read from the XMLS-description of the element, the profile is provided externally.

Parameters
- **schema_elements** – The XML-Description (an `etree.Element`) defining this ElementMixin.
- **profile** – Profile name the element is defined in.

profile
The profile this element was defined in.

namespace
The namespace this element is associated with.

CIMClass

The CIM Class represents a class defined in the CIM Schema.

class cimpyorm.Model.Elements.Class.**CIMClass**(*schema_elements=None*)

A CIM Schema Class (such as Terminal, IdentifiedObject, ...).

The class definition is read from its XMLS-description.

Parameters

- **schema_elements** – The XML-Description (an `etree.Element`) defining this CIMClass.
- **profile** – Profile name the element is defined in.

package

The package that contains this class definition.

parent

If this class inherits from a parent class, it is referenced here.

init_type(*base*)

Initialize ORM type using the CIMClass object :return: None

property all_props

Return all properties (native and inherited) defined for this CIMClass.

defined_in

name

namespace

namespace_name

profile

CIMProp

The CIM Class represents a class defined in the CIM Schema.

class cimpyorm.Model.Elements.Property.**CIMProp**(*schema_elements=None*)

Class representing a CIM Model property

cls

The class this property belongs to.

datatype

This property's datatype.

defined_in

name

namespace

namespace_name

profile

1.5 Serialization (Export)

CIMPyORM can be used to serialize datasets into XML-files as defined by the CIM standard. This way, datasets can be exported from CIMPyORM to facilitate import into other tools.

1.5.1 Dataset preparation

Before serialization, a dataset needs to be created and populated with CIM objects. To do this, either parse a dataset (as described in *Quickstart*), or start with an empty dataset and fill it with your own objects:

```
dataset, model = cimpyorm.create_empty_dataset(version="16")
```

Either add single objects:

```
term1 = model.Terminal(id=42, phases=model.enum.PhaseCode.v.AB)
dataset.add(term1)
```

Or add multiple at once:

```
dataset.add_all((model.Terminal(id=id) for id in range(10))
```

And finally commit the changes to the database:

```
dataset.commit()
```

1.5.2 Export

Once your dataset is ready to be exported, you can use the `export` function to generate the XML representation of the dataset as an InMemory-BytesIO object that can be written to disk or streamed.

`cimpyorm.export`(*dataset*, *mode='Single'*, *profile_whitelist=None*, *header_data=None*)

Returns a XML-serialization of the dataset within a zip-Archive.

Parameters

- **dataset** – The dataset/SQLAlchemy-Session object to export.
- **mode** (`str`) – {'Single', 'Multi'} - The export Mode, e.g. Single-File or Split by profiles.
- **profile_whitelist** (`Union[list, tuple, None]`) – The profiles to export. Mandatory if Export-mode is 'Multi'. Profiles can be specificed by their full name (e.g. 'EquipmentProfile' or their short name 'EQ')
- **header_data** (`Optional[dict]`) – Additional information for creating the file-headers (FullModel Objects). This is a dictionary of CIM-header supplements. Currently only the 'profile_header' field is supported, in which a list of profile identifiers (e.g. 'http://entsoe.eu/CIM/Topology/4/1') are defined.

Returns

A BytesIO-filehandle containing a zip-archive of the export.

The result can be written to disk by:

```
with open("outfile.xml", "wb+") as f:   # Open the file handle in binary mode with write␣
↪permissions
    f.write(result.getvalue())
```

1.6 Engines

Warning: This page is partially outdated as of 06/2020. Proceed with care.

The parser can use several database backend engines that serve as wrappers around SQLAlchemy Engines and sessionmakers.

1.6.1 Common functionality

1.6.2 Database engines

The available engines are:

SQLite

class cimpyorm.backends.**SQLite**(*path='out.db'*, *echo=False*, *driver=None*, *dataset_loc=None*)

__init__(*path='out.db'*, *echo=False*, *driver=None*, *dataset_loc=None*)

Default constructor for SQLite backend instance

Parameters

- **path** – Storage location for the .db-file (default: "out.db" in cwd)
- **echo** – SQLAlchemy "echo" parameter (default: False)
- **driver** – Python SQLite driver (default: sqlite3)
- **dataset_loc** – Dataset location used to automatically determine storage location (in the dataset folder)

For convenience, there is a named backend for In-Memory SQLite databases:

class cimpyorm.backends.**InMemory**(*echo=False*, *driver=None*)

__init__(*echo=False*, *driver=None*)

Default constructor for In-Memory-SQLite instances

Parameters

- **echo** – SQLAlchemy "echo" parameter (default: False)
- **driver** – Python SQLite driver (default: sqlite3)

Client-Server

MariaDB

class cimpyorm.backends.**MariaDB**(*username='root'*, *password=''*, *driver='pymysql'*, *host='127.0.0.1'*, *port=3306*, *path='cim'*, *echo=False*)

__init__(*username='root'*, *password=''*, *driver='pymysql'*, *host='127.0.0.1'*, *port=3306*, *path='cim'*, *echo=False*)

Default constructor for MariaDB backend instance

Parameters

- **username** – Username for the MariaDB database (default: root)
- **password** – Password for username (at) MariaDB database (default: "")
- **driver** – Python MariaDB driver (default: mysqlclient)
- **host** – Database host (default: localhost)
- **port** – Database port (default: 3306)
- **path** – Database name (default: "cim")
- **echo** – SQLAlchemy "echo" parameter (default: False)

MySQL

class cimpyorm.backends.**MySQL**(*username='root'*, *password=''*, *driver='pymysql'*, *host='127.0.0.1'*, *port=3306*, *path='cim'*, *echo=False*)

__init__(*username='root'*, *password=''*, *driver='pymysql'*, *host='127.0.0.1'*, *port=3306*, *path='cim'*, *echo=False*)

Default constructor for MySQL backend instance

Parameters

- **username** – Username for the MySQL database (default: root)
- **password** – Password for username (at) MySQL database (default: "")
- **driver** – Python MariaDB driver (default: pymysql)
- **host** – Database host (default: localhost)
- **port** – Database port (default: 3306)
- **path** – Database name (default: "cim")
- **echo** – SQLAlchemy "echo" parameter (default: False)

1.7 Linter

CIMPyORM supports linting CIM datasets against their schema definition using the `lint` function:

cimpyorm.**lint**(*session*, *model*)

Check the model for missing obligatory values and references and for invalid references (foreign key validation) and return the results in a pandas pivot-table.

Parameters

- **session** – The SQLAlchemy session object (obtained from parse/load).
- **model** – The parsed CIMPyORM model (obtained from parse/load).

Returns

Pandas pivot-table.

The function yields a pandas pivot_table containing information about schema violations and model inconsistencies, namely:

- Missing non-optional values (see *Exploring* on how to determine which fields are optional)
- Missing references
- Invalid references (foreign-key reference not found in referenced table)

Linting the ENTSO-E example *FullGrid* dataset against the CGMES 2.4.15 specification yields the following violations:

1.7.1 Note

As of version 0.6, the linter does not yet validate many-to-many relationships.

1.8 Exploring

The schema can be explored using *cimpyorm.describe()* for a description of elements.

describe() takes a format string to determine table layout (see the tabulate documentation).

cimpyorm.**describe**(*element*, *fmt='psql'*)

Give a description of an object.

Parameters

- **element** – The element to describe.
- **fmt** (str) – Format string for tabulate package (default postgres formatting).

Return type

None

For the CIM class ACLineSegment:

```
from cimpyorm import describe, load
s, m = load(r"path_to_db")
describe(m.ACLineSegment)
```

the following description is printed (the layout and information displayed has changed a bit over the months):

Hierarchy	Number of native properties
IdentifiedObject	8
PowerSystemResource	3
Equipment	3
ConductingEquipment	3
Conductor	1
ACLineSegment	9

Label	Domain	Multiplicity	Optional	Datatype	Unit	Multiplier	Inferred
DiagramObjects	IdentifiedObject	0..n	True	*DiagramObject	•	•	True
mRID	IdentifiedObject	0..1	True	String	•	•	False
name	IdentifiedObject	1..1	False	String	•	•	False
description	IdentifiedObject	1..1	False	String	•	•	False
entsoe_energyIdentCodeEic	IdentifiedObject	0..1	True	String	•	•	False
entsoe_shortName	IdentifiedObject	1..1	False	String	•	•	False
energyIdentCodeEic	IdentifiedObject	0..1	True	String	•	•	False
shortName	IdentifiedObject	0..1	True	String	•	•	False
Controls	PowerSystemResource	0..n	True	*Control	•	•	True
Measurements	PowerSystemResource	0..n	True	*Measurement	•	•	True
Location	PowerSystemResource	0..1	True	*Location	•	•	True
EquipmentContainer	Equipment	0..1	True	*EquipmentContainer	•	•	False
aggregate	Equipment	0..1	True	Boolean	•	•	False
OperationalLimitSet	Equipment	0..n	True	*OperationalLimitSet	•	•	True
Terminals	ConductingEquipment	0..n	True	*Terminal	•	•	True
BaseVoltage	ConductingEquipment	0..1	True	*BaseVoltage	•	•	False
SvStatus	ConductingEquipment	0..1	True	*SvStatus	•	•	True
length	Conductor	0..1	True	Length	m	k	False
bch	ACLineSegment	1..1	False	Susceptance	S	•	False
gch	ACLineSegment	0..1	True	Conductance	S	•	False
r	ACLineSegment	1..1	False	Resistance	ohm	•	False
x	ACLineSegment	1..1	False	Reactance	ohm	•	False
b0ch	ACLine	1..1	False	Suscep	S		False

1.8.1 CIM-Explorer

In addition the schema model for the CGMES v2.4.15 is provided by a related project.

INDEX

Symbols

A

C

D

E

I

L

M

N

P

S