OpenEnergyPlatform Documentation

Release 0.0.3

open_eGo

Nov 19, 2019

Contents

The OpenEnergyPlatform is a website that has three main targets:

1. Provide a language-independent interface that is a thin layer on top of the OpenEnergyDatabase
2. Implement an intuitive and easy-to use web interface on top of the OpenEnergyDatabase
3. Improve the visibility, communication and transparency of results from energy system modelling

CHAPTER 1

Mission statement

The transition to renewable energy sources is one of the huge goals of the last few decades. Whilst conventional energy generation provides a constant, generally available source of electricity, heat and so on, our environment pays a toll. Contrary, renewable energy generation is less environmentally demanding but more financially expensive or just locally or inconsistently available. Guaranteeing a steady and reliable, yet sustainable supply of energy requires still a lot of thorough research.

Expansion of the energy grid might imply measures that must be communicable in a transparent way. Hence, results from research of energy system studies should be publicly available and reproducible. This raises the need for publicly available data sources.

CHAPTER 2

App: dataedit

One aim ot the OpenEnergyPlatform is the visual and understandable presentation of such datasets. The underlying OpenEnergyDatabase (OEDB) stores datasets of different open-data projects. The visual presentation is implemented in the **dataedit** app.

2.1 Database visualisation

CHAPTER 3

App: api

The data stored in the OEDB is also used in several projects. In order to ease the access to required datasets the OEP provides a RESTful HTTP-interface in the **api** app:

3.1 Data interface (REST)

3.1.1 Data Structures

Constraint Definition

constraint_definition (Dictonary)

Specifies a definition of a constraint.

- `action` Action of constraint (e.g. ADD, DROP)
- `constraint_type` Type of constraint (e.g. UNIQUE, PRIMARY KEY, FOREIGN KEY)
- `constraint_name` Name of constraint.
- `constraint_parameter` Parameter of constraint.
- `reference_table` Name of reference table, can be None.
- `reference_column` Name of reference column, can be None.

Column Definition

column_definition (Dictonary)

Specifies a definition of a column.

- `name` Name of column.

- `new_name` New name of column, can be None.
- `data_type` New datatype of column, can be None.
- `is_nullable` New null value of column, can be None.
- `character_maximum_length` New data length of column, can be None.

Response Definition

response_dictonary (Dictonary)

Describes the result of an api action.

- `success (Boolean)` Result of Action
- `error (String)` Error Message
- `http_status (Integer)` HTTP status code (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes)

3.1.2 Table (RESTful)

URL: /schema/{schema}/table/{table}

GET

Reference needed.

PUT

Creates a new table in database. JSON should contain a constraint definition array and a column definition array.

Example:

```
{
  "constraints": [
    {
      "constraint_type": "FOREIGN KEY",
      "constraint_name": "fkey_schema_table_database_id",
      "constraint_parameter": "database_id",
      "reference_table": "example.table",
      "reference_column": "database_id_ref"
    },
    {
      "constraint_type": "PRIMARY KEY",
      "constraint_name": "pkey_schema_table_id",
      "constraint_parameter": "id",
      "reference_table": null,
      "reference_column": null
    }
  ],
  "columns": [
    {
      "name": "id",
      "data_type": "int",
```

(continues on next page)

(continued from previous page)

```
      "is_nullable": "YES",
      "character_maximum_length": null
    },
    {
      "name": "name",
      "data_type": "character varying",
      "is_nullable": "NO",
      "character_maximum_length": 50
    }
  ]
}
```

POST

JSON should contain a column or constraint definition. Additionally `action` and `type` should be mentioned.

- `type` can be `constraint` or `column`.
- `action` can be `ADD` and `DROP`.
- `constraint_type` can be every constraint type supported by Postgres.
- `reference_table` and `reference_column` can be null, if not necessary.

Example:

```
{
  "type" : "constraint",
  "action": "ADD",
  "constraint_type": "FOREIGN KEY",
  "constraint_name": "fkey_label",
  "constraint_parameter": "changed_name",
  "reference_table" : "reference.group_types",
  "reference_column" : "label"
}

{
  "type" : "column",
  "name" : "test_name",
  "newname" : "changed_name",
  "data_type": "character varying",
  "is_nullable": "NO",
  "character_maximum_length": 50
}
```

3.1.3 Rows (RESTful)

GET

URL: `/schema/<schema>/tables/<table>/rows/`

You can use this part to get information from the database.

You can specify the following parameters in the url:

- `columns (List)` List of selected columns, e.g. `id,name`

- **`where (List)` List of where clauses, e.g. `id+OPERATOR+1+CONNECTOR+name+OPERATOR+georg`**
  - OPERATORS could be EQUAL, GREATER, LOWER, NOTEQUAL, NOTGREATER, NOTLOWER
  - CONNECTORS could be AND, OR
- `orderby (List)` List of order columns, e.g. `name,code`
- `limit (Number)` Number of displayed items, e.g. `100`
- `offset (Number)` Number of offset from start, e.g. `10`

3.2 Deprecated Stuff

3.2.1 Create a table

Dictionary structure

schema (String) Specifies the schema name the table should be created in. If this schema does not exist it will be created.

table (String) Specifies the name of the table to be created.

fields (List) List specifying the columns of the new table (see *Field specification*).

constraints (List) List of additional constraints (see *Constraint specification*).

Field specification

name (String) Name of the field

type (String) Name of a valid Postgresql type

pk (Bool) Specifies whether this column is a primary key. Be aware of[1]

Constraint specification

Args:

name (String) Type of constraint. Possible values:

- `fk` (see *Foreign key specification*)

constraint (Dictionary) Dictionary as specified by the foreign key.

Foreign key specification

schema (String) Name of the schema the referenced table is stored in

table (String) Name of the referenced table

field (String) Name of the referenced column

[1] The OEP is currently only supporting a non-compound integer primary key labeled 'id'. Violation of this constraint might render the OEP unable to display the data stored in this table.

on_delete (String) Specifies the behaviour if this field is deleted. Possible values:

- `cascade`
- `no action`
- `restrict`
- `set null`
- `set default`

3.2.2 Insert data

schema (String) Specifies the schema name the table should be created in. If this schema does not exist it will be created.

table (String) Specifies the name of the table to be created.

fields (List) List specifying the column names the date should be inserted in.

values (List) Each element is a list of values that should be inserted. The number of elements must match the number of fields.

returning (Bool) An expression that is evaluated and returned as result. If this entry is present the result of this expression is returned as in *Select Data*.

3.2.3 Select data

all (Bool) Specifies whether all rows should be returned (default)

distinct (Bool) Specifies whether only unique rows should be returned

fields (List) The list of columns that should be returned (see **select_field_spec_**)

where (List) The list of condition that should be considered (see **select_condition_spec_**)

limit (Integer or 'all') Specifies how many results should be returned. If 'all' is set all matching rows will be returned (default).

offset (Integer) Specifies how many entries should be skipped before returning data

3.2.4 Binding the API to python

3.3 How to work with the API - An example

Note: The API is enable for the following schmemas only:

- model_draft
- sandbox

3.3.1 Authenticate

The OpenEnergy Platform API uses token authentication. Each user has a unique token assigned to it that will be used as an authentication password. You can access you token by visiting you profile on the OEP. In order to issue PUT or POST request you have to include this token in the *Authorization*-field of your request:

- Authorization: Token *your-token*

3.3.2 Create table

We want to create the following table with primary key *id*:

In order to do so, we send the following PUT request:

```
PUT oep.iks.cs.ovgu.de/api/v0/schema/sandbox/tables/example_table/
{
    "query": {
        "columns": [
            {
                "name":"id",
                "data_type": "Bigserial",
                "is_nullable": "NO"
            },{
                "name":"name",
                "data_type": "varchar",
                "character_maximum_length": "50"
            },{
                "name":"geom",
                "data_type": "geometry(point)"
            }
        ],
        "constraints": [
            {
                "constraint_type": "PRIMARY KEY",
                "constraint_parameter": "id",
            }
        ]
    }
}
```

and include the following headers:

- Content-Type: application/json
- Authorization: Token *your-token*

You can use any tool that can send HTTP-requests. E.g. you could use the linux tool **curl**:

```
curl
    -X PUT
    -H 'Content-Type: application/json'
    -H 'Authorization: Token <your-token>'
    -d '{
            "query": {
                "columns": [
                    {
                        "name":"id",
                        "data_type": "bigsersial",
```

(continues on next page)

(continued from previous page)

```
                        "is_nullable": "NO"
                    },{
                        "name":"name",
                        "data_type": "varchar",
                        "character_maximum_length": "50"
                    },{
                        "name":"geom",
                        "data_type": "geometry(point)"
                    }
                ],
                "constraints": [
                    {
                        "constraint_type": "PRIMARY KEY",
                        "constraint_parameter": "id",
                    }
                ]
            }
        }'
    oep.iks.cs.ovgu.de/api/v0/schema/sandbox/tables/example_table/
```

or **python**:

```
>>> import requests
>>> data = { "query": { "columns": [ { "name":"id", "data_type": "bigserial", "is_
↪nullable": "NO" },{ "name":"name", "data_type": "varchar", "character_maximum_length
↪": "50" },{ "name":"geom", "data_type": "geometry(point)" } ], "constraints": [ {
↪"constraint_type": "PRIMARY KEY", "constraint_parameter": "id" } ] } }
>>> requests.put(oep_url+'/api/v0/schema/sandbox/tables/example_table/', json=data,␣
↪headers={'Authorization': 'Token %s'%your_token} )
<Response [201]>
```

If everything went right, you will receive a 201-Resonse and the table has been created.

Note: The OEP will automatically grant the 'admin'-permissions on this table to your user.

```
>>> result = requests.get(oep_url+'/api/v0/schema/sandbox/tables/example_table/columns
↪')
>>> result.status_code
200
>>> json_result = result.json()
>>> json_result['id'] == {'character_maximum_length': None, 'maximum_cardinality':␣
↪None, 'is_nullable': False, 'data_type': 'bigint', 'numeric_precision': 64,
↪'character_octet_length': None, 'interval_type': None, 'dtd_identifier': '1',
↪'interval_precision': None, 'numeric_scale': 0, 'is_updatable': True, 'datetime_
↪precision': None, 'ordinal_position': 1, 'column_default': "nextval('sandbox.
↪example_table_id_seq'::regclass)", 'numeric_precision_radix': 2}
True
>>> json_result['geom'] == {'column_default': None, 'character_maximum_length': None,
↪'maximum_cardinality': None, 'is_nullable': True, 'data_type': 'USER-DEFINED',
↪'numeric_precision': None, 'character_octet_length': None, 'interval_type': None,
↪'dtd_identifier': '3', 'interval_precision': None, 'numeric_scale': None, 'is_
↪updatable': True, 'datetime_precision': None, 'ordinal_position': 3, 'column_default
↪': None, 'numeric_precision_radix': None}
True
>>> json_result['name'] == {'character_maximum_length': 50, 'maximum_cardinality':␣
↪None, 'is_nullable': True, 'data_type': 'character varying', 'numeric_precision':␣
↪None, 'character_octet_length': 200, 'interval_type': None, 'dtd_identifier': '2',
↪'interval_precision': None, 'numeric_scale': None, 'is_updatable': True, 'datetime_
↪precision': None, 'ordinal_position': 2, 'column_default': None, 'numeric_precision_
↪radix': None}
```

(continues on next page)

(continued from previous page)

```
True
```

Note: A table **must** have a column 'id' of type 'bigserial'.

```
>>> import requests
>>> data = { "query": { "columns": [ { "name":"name", "data_type": "varchar",
↪"character_maximum_length": "50" }]} }
>>> response = requests.put(oep_url+'/api/v0/schema/sandbox/tables/faulty_table/',␣
↪json=data, headers={'Authorization': 'Token %s'%your_token} )
>>> response.status_code
500
>>> response.json()['reason']
'Your table must have one column "id" of type "bigserial"'
```

```
>>> import requests
>>> data = { "query": { "columns": [ { "name":"id", "data_type": "integer"}]} }
>>> response = requests.put(oep_url+'/api/v0/schema/sandbox/tables/faulty_table/',␣
↪json=data, headers={'Authorization': 'Token %s'%your_token} )
>>> response.status_code
500
>>> response.json()['reason']
'Your column "id" must have type "bigserial"'
```

3.3.3 Insert data

You can insert data into a specific table by sending a request to its */rows* subresource. The *query* part of the sent data contians the row you want to insert in form of a JSON-dictionary::

```
{
    'name_of_column_1': 'value_in_column_1',
    'name_of_column_2': 'value_in_column_2',
    ...
}
```

If you the row you want to insert should have a specific id, send a PUT-request to the */rows/{id}/* subresource. In case the id should be generated automatically, just ommit the id field in the data dictionary and send a POST-request to the */rows/new* subresource. If successful, the response will contain the id of the new row.

In the following example, we want to add a row containing just the name "John Doe", **but** we do not want to set the the id of this entry.

curl:

```
curl
    -X POST
    -H "Content-Type: application/json"
    -H 'Authorization: Token <your-token>'
    -d '{"query": {"name": "John Doe"}}'
    oep.iks.cs.ovgu.de/api/v0/schema/sandbox/tables/example_table/rows/
```

python:

```
>>> import requests
>>> data = {"query": {"name": "John Doe"}}
>>> result = requests.post(oep_url+'/api/v0/schema/sandbox/tables/example_table/rows/
↪new', json=data, headers={'Authorization': 'Token %s'%your_token} )
>>> result.status_code
201
>>> json_result = result.json()
>>> json_result['data'] # Show the id of the new row
[[1]]
```

Alternatively, we can specify that the new row should be stored under id 12:

python:

```
>>> import requests
>>> data = {"query": {"name": "Mary Doe XII"}}
>>> result = requests.put(oep_url+'/api/v0/schema/sandbox/tables/example_table/rows/12
↪', json=data, headers={'Authorization': 'Token %s'%your_token} )
>>> result.status_code
201
```

Our database should have the following structure now:

id: int	name: varchar(50)	geom: geometry(Point)
1	John Doe	NULL
12	Mary Doe XII	NULL

Note: In order to insert new data, or perfom any other actions that alter the data state, you need the 'write'-permission for the respective table. Permissions can be granted by a user with 'admin'-permissions in the OEP web interface.

3.3.4 Select data

You can insert data into a specific table by sending a GET-request to its */rows* subresource. No authorization is required to do so.

curl:

```
curl
    -X GET
    oep.iks.cs.ovgu.de/api/v0/schema/sandbox/tables/example_table/rows/
```

The data will be returned as list of JSON-dictionaries similar to the ones used when adding new rows:

```
[
    {
        "name": "John Doe",
        "geom": null,
        "id": 1
    }
]
```

python:

```
>>> result = requests.get(oep_url+'/api/v0/schema/sandbox/tables/example_table/rows/',
↪ )
>>> result.status_code
200
>>> json_result = result.json()
>>> json_result ==  [{'id': 1, 'name': 'John Doe', 'geom': None}, {'id': 12, 'name':
↪'Mary Doe XII', 'geom': None}]
True
```

There are also optional parameters for these GET-queries:

- limit: Limit the number of returned rows
- offset: Ignore the specified amount of rows
- orderby: Name of a column to refer when ordering
- **column: Name of a column to include in the results. If not present, all** columns are returned
- where: Constraint fourmulated as *VALUE+OPERATOR+VALUE* with
  - VALUE: Constant or name of a column
  - OPERATOR: One of the following:
    - *EQUALS* or =,
    - *GREATER* or >,
    - *LOWER* or <,
    - *NOTEQUAL* or *!=* or <>,
    - *NOTGREATER* or <=,
    - *NOTLOWER* or >=,

```
>>> result = requests.get(oep_url+"/api/v0/schema/sandbox/tables/example_table/rows/?
↪where=name=John+Doe", )
>>> result.status_code
200
>>> json_result = result.json()
>>> json_result == [{'id': 1, 'name': 'John Doe', 'geom': None}]
True
```

```
>>> result = requests.get(oep_url+"/api/v0/schema/sandbox/tables/example_table/rows/1
↪", )
>>> result.status_code
200
>>> json_result = result.json()
>>> json_result == {'id': 1, 'name': 'John Doe', 'geom': None}
True
```

```
>>> result = requests.get(oep_url+"/api/v0/schema/sandbox/tables/example_table/rows/?
↪offset=1")
>>> result.status_code
200
>>> json_result = result.json()
>>> json_result == [{'id': 12, 'name': 'Mary Doe XII', 'geom': None}]
True
```

```
>>> result = requests.get(oep_url+"/api/v0/schema/sandbox/tables/example_table/rows/?
↪column=name&column=id")
>>> result.status_code
200
>>> json_result = result.json()
>>> json_result == [{'id': 1, 'name': 'John Doe'},{'id': 12, 'name': 'Mary Doe XII'}]
True
```

3.3.5 Add columns table

```
>>> data = {'query':{'data_type': 'varchar', 'character_maximum_length': 30}}
>>> result = requests.put(oep_url+"/api/v0/schema/sandbox/tables/example_table/
↪columns/first_name", json=data, headers={'Authorization': 'Token %s'%your_token})
>>> result.status_code
201
```

```
>>> result = requests.get(oep_url+"/api/v0/schema/sandbox/tables/example_table/
↪columns/first_name")
>>> result.status_code
200
>>> result.json() == {'numeric_scale': None, 'numeric_precision_radix': None, 'is_
↪updatable': True, 'maximum_cardinality': None, 'character_maximum_length': 30,
↪'character_octet_length': 120, 'ordinal_position': 4, 'is_nullable': True,
↪'interval_type': None, 'data_type': 'character varying', 'dtd_identifier': '4',
↪'column_default': None, 'datetime_precision': None, 'interval_precision': None,
↪'numeric_precision': None}
True
```

3.3.6 Alter data

Our current table looks as follows:

id: bigserial	name: varchar(50)	geom: geometry(Point)	first_name: varchar(30)
1	John Doe	NULL	NULL
12	Mary Doe XII	NULL	NULL

Our next task is to distribute for and last name to the different columns:

```
>>> result = requests.get(oep_url+'/api/v0/schema/sandbox/tables/example_table/rows/
↪') # Load the names via GET
>>> result.status_code
200
>>> for row in result.json():
...     first_name, last_name = str(row['name']).split(' ', 1) # Split the names at␣
↪the first space
...     data = {'query': {'name': last_name, 'first_name': first_name}} # Build the␣
↪data dictionary and post it to /rows/<id>
...     result = requests.post(oep_url+'/api/v0/schema/sandbox/tables/example_table/
↪rows/{id}'.format(id=row['id']), json=data, headers={'Authorization': 'Token %s'
↪%your_token})
...     result.status_code
200
200
```

Now, our table looks as follows:

id: int	name: varchar(50)	geom: geometry(Point)	first_name: varchar(30)
1	Doe	NULL	John
12	Doe XII	NULL	Mary

3.3.7 Alter tables

Currently, rows are allowed that contain no first name. In order to prohibit such behaviour, we have to set column *first_name* to *NOT NULL*. Such *ALTER TABLE* commands can be executed by POST-ing a dictionary with the corresponding values to the column's resource:

```
>>> data = {'query': {'is_nullable': False}}
>>> result = requests.post(oep_url+"/api/v0/schema/sandbox/tables/example_table/
↪columns/first_name", json=data, headers={'Authorization': 'Token %s'%your_token} )
>>> result.status_code
200
```

We can check, whether your command worked by retrieving the corresponding resource:

```
>>> result = requests.get(oep_url+"/api/v0/schema/sandbox/tables/example_table/
↪columns/first_name")
>>> result.status_code
200
>>> json_result = result.json()
>>> json_result['is_nullable']
False
```

After prohibiting null-values in the first name column, such rows can not be added anymore.

```
>>> import requests
>>> data = {"query": {"name": "McPaul"}}
>>> result = requests.post(oep_url+'/api/v0/schema/sandbox/tables/example_table/rows/
↪new', json=data, headers={'Authorization': 'Token %s'%your_token} )
>>> result.status_code
500
>>> result.json()['reason']
'Action violates not-null constraint on first_name. Failing row was (McPaul)'
```

Delete rows

In order to delete rows, you need the 'delete'-permission on the respective table. The permissions can be granted by an admin in the OEP web interface.

```
>>> import requests
>>> data = {"query": {"name": "McPaul"}}
>>> result = requests.delete(oep_url+'/api/v0/schema/sandbox/tables/example_table/
↪rows/1', json=data, headers={'Authorization': 'Token %s'%your_token} )
>>> result.status_code
200
>>> result = requests.get(oep_url+'/api/v0/schema/sandbox/tables/example_table/rows/1
↪')
>>> result.status_code
404
```

Delete tables

In order to delete rows, you need the 'admin'-permission on the respective table. The permissions can be granted by an admin in the OEP web interface.

```
>>> import requests
>>> requests.delete(oep_url+'/api/v0/schema/sandbox/tables/example_table', headers={
↪'Authorization': 'Token %s'%your_token} )
<Response [200]>
>>> requests.get(oep_url+'/api/v0/schema/sandbox/tables/example_table')
<Response [404]>
```

For more advanced commands read advanced

Handling Arrays

The underlying OpenEnergy Database is a Postgres database. Thus, it supports Array-typed fields.

```
>>> import requests
>>> data = { "query": { "columns": [ { "name":"id", "data_type": "bigserial", "is_
↪nullable": "NO" },{ "name":"arr", "data_type": "int[]"},{ "name":"geom", "data_type
↪": "geometry(point)" } ], "constraints": [ { "constraint_type": "PRIMARY KEY",
↪"constraint_parameter": "id" } ] } }
>>> requests.put(oep_url+'/api/v0/schema/sandbox/tables/example_table/', json=data,␣
↪headers={'Authorization': 'Token %s'%your_token} )
<Response [201]>
```

CHAPTER 4

App: modelview

Researchers or interested developers that just entered this field might be interested in an overview which open energy models already exists. This data is collected in so called fact sheets. Modellers can look through these, add their own models or enhance existing descriptions using the forms definied in the **modelview** app

4.1 Factsheet handler

Other apps are:

CHAPTER 5

App: login

5.1 User management

CHAPTER 6

Indices and tables

- genindex
- modindex
- search